DEBRE BERHAN UNIVERSITY

College of Computing

Department of Software Engineering

Individual Assignment

Course: Fundamentals of Machine Learning

Name	ID
Yenenesh Dabot	1500147

Submitted To : Derbewu

Felasman(MSc.

Submission Date : February 2, 2017 EC

Problem Definition: Credit Card Fraud Detection

The goal of this machine learning problem is to build a model that can accurately detect fraudulent transactions in a credit card transaction dataset. Given the nature of fraud detection, the model needs to distinguish between legitimate and fraudulent transactions based on a series of transaction-related features. This type of problem is typically formulated as a **binary classification problem**, where the two classes are:

- **Fraudulent** (fraud): The transaction is fraudulent.
- **Non-fraudulent** (non-fraud): The transaction is legitimate.

Problem Objective

- **Objective**: Develop a machine learning model that predicts whether a credit card transaction is fraudulent or not.
- **Challenges**:
    - The dataset is typically **imbalanced**, with a much larger number of non-fraudulent transactions than fraudulent ones.
    - Features that distinguish fraudulent from legitimate transactions might not always be directly interpretable or easy to identify, which will require careful feature engineering and analysis.

Key Features (Typical in Credit Card Fraud Datasets):

- **Transaction amount**: The amount of the transaction.
- **Transaction date/time**: When the transaction was made.
- **Cardholder ID**: An anonymized identifier for the person making the transaction.
- **Merchant information**: Identifiers related to the merchant.
- **Geographic location of the transaction**: The location where the transaction took place.
- **Transaction type**: Type of transaction (e.g., purchase, refund).
- **Previous fraud history**: Whether the account has had fraudulent transactions before.

Identifying and Obtaining the Dataset

The dataset used for credit card fraud detection can be found on platforms Kaggle. One commonly used dataset is the **Credit Card Fraud Detection dataset** from Kaggle, which contains transactions made by credit cards . It has been anonymized to protect users' identities.

Dataset Description:

Columns:

- **Time**: Elapsed time (in seconds) between this transaction and the first transaction in the dataset.

- **V1 to V28**: 28 anonymized features that are the result of a PCA transformation applied to the original features.
- **Amount**: The transaction amount.
- **Class**: The target variable indicating whether the transaction is fraudulent (1) or not (0).

- **Number of Records**: The dataset has 284,807 records.
- **Class Distribution**:
  - The dataset is highly imbalanced, with the number of fraudulent transactions being much smaller than the number of legitimate ones.

Data Understanding and Exploration

Steps for Data Exploration:

1. **Basic Information**: Check the number of records, columns, and data types of each feature using data.info().
2. **Missing Values**: Identify if there are any missing values in the dataset using data.isnull().sum().
3. **Class Distribution**: Since fraud detection is a binary classification problem, check the distribution of the target variable (Class) to identify class imbalance.
4. **Summary Statistics**: Use data.describe() to look at basic statistical summaries for the numeric columns, which can help in identifying any outliers or unusual values.

Documenting Observations from the EDA

Class Imbalance: There is a clear imbalance between fraudulent and non-fraudulent transactions. Most of the transactions are legitimate, and only a small fraction are fraudulent (around 0.17% of the total).

Feature Distributions:

- **Amount**: The distribution of transaction amounts is highly skewed, with a few large transactions standing out as potential outliers.
- **Time**: The **Time** feature has a relatively uniform distribution, but extreme values should be checked for potential anomalies.

Missing Values: There are no missing values in the dataset, as indicated by data.isnull().sum().

Outliers:

- **Amount**: There are some outliers in the transaction amounts (very large values), which may be legitimate high-value transactions or errors.
- **Time**: There may be outliers in the **Time** feature that could indicate errors or unusual transactions.

Feature Relationships:

- **Amount vs Class**: There's a higher concentration of fraudulent transactions among smaller amounts, but there are also a few high-value fraudulent transactions.
- **Time vs Class**: Fraudulent transactions seem to be distributed fairly evenly over time, but further analysis is needed to explore temporal patterns.

Data Preprocessing steps:

step	Why It's Needed?
Handling Missing Values	Avoids errors and ensures complete data for training.
Fixing Inconsistencies	Prevents invalid data from misleading the model.
Outlier Removal	Ensures extreme values don't distort model predictions.
Encoding Categorical Features	Converts categorical variables into a numerical format for machine learning.
Scaling & Normalization	Ensures numerical values are on a consistent scale for better model performance.

Model Evaluation and Analysis for Fraud Detection Model

we will assess the performance of our trained fraud detection model. Since fraud detection is a **binary classification problem** (fraudulent vs. legitimate transactions), we will focus on classification metrics like **Accuracy, Precision, Recall, F1-score, and AUC** (Area Under the Curve).

1. Making Predictions on Testing Data

After training your fraud detection model, you use the **test dataset** to see how well it performs. This dataset contains transactions that the model has never seen before.

In case:

- **Time**: How much time has passed since the first transaction.
- **V1, V2, V3, V4**: These are features that represent different aspects of the transaction, such as how unusual or different it is compared to past behavior.
- **Amount**: The transaction amount.

The model will predict whether each transaction is **fraudulent** or **legitimate** based on these features.

Where predictions will contain:

- 1 for fraudulent transactions
- 0 for legitimate transactions

2. Evaluating the Model's Performance

Next, we'll evaluate how well the model is performing using different metrics. The choice of metrics depends on the problem you're solving and the **imbalanced** nature of fraud detection (fraudulent transactions are usually much fewer than legitimate ones)

a. Accuracy tells you how often the model gets the prediction right overall.

b. Precision

Precision helps you understand how many of the transactions that your model **predicted as fraudulent** are actually **fraudulent**. It's essential because you want to minimize **false positives** (legitimate transactions flagged as fraudulent).

c. Recall

Recall measures how well your model identifies all fraudulent transactions. A model with high recall will find most of the fraudulent transactions, but may also flag legitimate ones as fraud (lower precision).

d. F1-Score

F1-Score is the balance between precision and recall. If your model is both good at detecting fraud and minimizing false positives, it will have a high F1-score.

Let's tailor the explanation to fit your **fraud detection system**. Here's a breakdown of how to evaluate your model after implementing it, including making predictions, evaluating performance, visualizing results, and interpreting what the metrics mean for your specific project.

1. Making Predictions on Test Data

Once you've trained your fraud detection model, you'll use **test data** (transactions that the model hasn't seen) to evaluate how well it predicts fraud.

In your case, you might use features like:

- **Time**: How much time has passed since the first transaction.
- **V1, V2, V3, V4**: These are features that represent different aspects of the transaction, such as how unusual or different it is compared to past behavior.
- **Amount**: The transaction amount.

The model will predict whether each transaction is **fraudulent** or **legitimate** based on these features.

Where predictions will contain:

- 1 for fraudulent transactions
- 0 for legitimate transactions

2. Visualizing Model Performance

a. Confusion Matrix

The confusion matrix helps you see the model's **errors** in a more visual form. It shows:

- **True Positives (TP)**: Correctly predicted frauds.
- **True Negatives (TN)**: Correctly predicted legitimate transactions.
- **False Positives (FP)**: Legitimate transactions incorrectly flagged as fraud.
- **False Negatives (FN)**: Fraudulent transactions incorrectly flagged as legitimate.

b. ROC Curve

A **ROC curve** compares the model's **True Positive Rate (TPR)** with the **False Positive Rate (FPR)**. A higher curve means the model is better at distinguishing between fraud and legitimate transactions.

3. Comparing Against a Baseline (Dummy Classifier)

It's useful to compare your model's performance with a **baseline** model that might just predict **the most frequent class** (e.g., always predicting that a transaction is legitimate).

4. Analyzing and Interpreting Results

let's look at how the different metrics help you analyze the model's performance:

- **Accuracy:** If the accuracy is high but the precision and recall are low, it means your model might be predicting too many legitimate transactions as fraudulent (low precision), or missing many fraudulent transactions (low recall).
- **Precision & Recall:** You might choose to adjust the threshold if precision or recall is too low. For example, you may want to **increase recall** (catch more frauds) even if it means sacrificing some precision.
- **F1-Score:** The F1-score is a good balance metric when you need both precision and recall to be high.
- **AUC:** If your AUC score is close to **1**, it indicates that your model is doing a great job of distinguishing fraud from legitimate transactions.

Summary of Metrics for Fraud Detection Model

Metric	Explanation
Accuracy	Overall percentage of correct predictions. Not always reliable with imbalanced data.
Precision	Percentage of predicted frauds that are actually fraud (important to reduce false positives).
Recall	Percentage of actual frauds identified by the model (important to catch as much fraud as possible).
F1-Score	Balance between precision and recall (best for when both are important).
AUC	Balance between precision and recall (best for when both are important).

Model Deployment: Deploying Your Fraud Detection Model Using FastAPI

Deploying a machine learning model as an API allows you to make predictions from any application (e.g., web apps, mobile apps) by sending a request with input data to the API. In your case, the goal is to deploy your **fraud detection model** using **FastAPI**. Here's how you can approach this deployment:

1. Setting Up FastAPI

FastAPI is a fast, modern framework for building APIs. It's easy to use with machine learning models, as it supports data validation with **Pydantic** and has good integration with **Python** libraries.

2.Install Required Libraries

To set up FastAPI, you first need to install the necessary libraries.

- fastapi: The web framework.
- uvicorn: An ASGI server to run the FastAPI app.
- joblib: For loading the trained model.
- pandas: For handling input data.

Explanation of the Code

- **FastAPI**: We initialize the FastAPI app and use **CORS middleware** to allow cross-origin requests (for example, if you are accessing the API from a frontend hosted elsewhere).
- **Transaction Class**: The Transaction class (based on Pydantic) is used to validate the input data. The user will send data in this format for prediction.
- **Model Loading**: We load the trained model (fraud_detection_model.joblib) using **joblib**.
- **Prediction**: The /predict endpoint processes the data, uses the model to make a prediction, and returns the result (fraudulent or legitimate).
- **Root Endpoint**: The / route serves a basic message for testing that the API is working.

3. Running the FastAPI App

To run the FastAPI application, use **Uvicorn** (ASGI server). This will allow the API to start and be ready to accept requests.

- python -m uvicorn main:app --reload
- main: The Python file containing the FastAPI app (main.py).
- --reload: Automatically reloads the server when code changes (ideal for development).

This will start the API on http://127.0.0.1:8000.

4. Testing the API

Once the server is running, you can test it by sending requests to the /predict endpoint. We use **Postman**, **curl**, or any HTTP client.

5. Deployment on a Cloud Service (Optional)

API is working locally, you can deploy it to a cloud platform for broader accessibility. Some common platforms for deploying FastAPI applications are:

- **AWS EC2**: A cloud-based virtual server where you can deploy FastAPI with a web server (like **Uvicorn** or **Gunicorn**).
- **Google Cloud Platform (GCP)** or **Microsoft Azure**: More complex options with additional configurations.

6. Instructions for Running and Testing Deployed API

Run Locally:

- Ensure FastAPI, Uvicorn, joblib, pandas installed.
- Run the FastAPI app with python -m uvicorn main:app --reload
- Access the API at http://127.0.0.1:8000.
- Test the API by sending POST requests with transaction data.