[**Credit Card Fraud Detection - ML - GeeksforGeeks**](https://www.geeksforgeeks.org/machine-learning/ml-credit-card-fraud-detection/)

**Credit Card Fraud Detection Using Random Forest**

### Project Summary

This project aims to develop a machine learning model that can detect fraudulent credit card transactions. The goal is to automate fraud detection by training a model using historical data. The model is built using a Random Forest Classifier and is evaluated using several metrics to determine its effectiveness. After training, the model is saved to a file for future predictions. A separate script is also included to load this model and make predictions on new data.

### 1. Importing Required Libraries

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from matplotlib import gridspec

* numpy: This library is used for handling large arrays and performing mathematical operations efficiently.
* pandas: Provides powerful data structures (like DataFrames) for handling tabular data.
* matplotlib.pyplot: Used to generate visual plots such as bar graphs, histograms, and heatmaps.
* seaborn: A statistical plotting library that works well with pandas and matplotlib.
* gridspec: Allows customization of plot layouts (though not used later in the script).

### 2. Loading the Dataset

data = pd.read\_csv("creditcard.csv")

* Loads the dataset containing transaction records.
* The dataset includes multiple columns, with the most important one being Class.
* Class has two values: 0 for normal transactions and 1 for fraudulent ones.

### 3. Splitting Fraud and Valid Transactions

fraud = data[data['Class'] == 1]  
valid = data[data['Class'] == 0]  
outlierFraction = len(fraud)/(len(valid))  
print(outlierFraction)

* fraud: Extracts rows labeled as fraud.
* valid: Extracts rows labeled as normal.
* outlierFraction: Calculates how rare fraudulent transactions are compared to normal ones.
* This value is important because it highlights the class imbalance — a key challenge in fraud detection.

### 4. Separating Features and Target

X = data.drop(['Class'], axis=1)  
Y = data["Class"]  
print(X.shape)  
print(Y.shape)  
  
xData = X.values  
yData = Y.values

* X: All features used to make predictions (everything except Class).
* Y: The actual labels — 0 for normal and 1 for fraud.
* The .values method converts pandas DataFrames into NumPy arrays for easier processing.

### 5. Splitting Into Train and Test Sets

from sklearn.model\_selection import train\_test\_split  
  
xTrain, xTest, yTrain, yTest = train\_test\_split(  
 xData, yData, test\_size=0.2, random\_state=42)

* Splits the dataset into training and testing parts.
* test\_size=0.2: 20% of data is for testing, 80% for training.
* random\_state=42: Ensures the split is consistent across different runs.

### 6. Training the Random Forest Classifier

from sklearn.ensemble import RandomForestClassifier  
  
rfc = RandomForestClassifier()  
rfc.fit(xTrain, yTrain)

* Initializes the Random Forest Classifier.
* fit(): Trains the model on the training data.
* Random Forest uses multiple decision trees to make predictions. It reduces overfitting and improves accuracy.

### 7. Making Predictions

yPred = rfc.predict(xTest)

* Predicts whether each transaction in the test set is fraud or not.
* The result (yPred) is a list of 0s and 1s.

### 8. Evaluating the Model

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, matthews\_corrcoef, confusion\_matrix   
  
accuracy = accuracy\_score(yTest, yPred)  
precision = precision\_score(yTest, yPred)  
recall = recall\_score(yTest, yPred)  
f1 = f1\_score(yTest, yPred)  
mcc = matthews\_corrcoef(yTest, yPred)  
  
print("Model Evaluation Metrics:")  
print(f"Accuracy: {accuracy:.4f}")  
print(f"Precision: {precision:.4f}")  
print(f"Recall: {recall:.4f}")  
print(f"F1-Score: {f1:.4f}")  
print(f"Matthews Correlation Coefficient: {mcc:.4f}")

* accuracy: How many total predictions were correct.
* precision: Of all predicted frauds, how many were actually fraud.
* recall: Of all real frauds, how many were found.
* f1-score: Balances precision and recall.
* mcc: Gives a more balanced view, especially when classes are imbalanced.

### 9. Visualizing the Confusion Matrix

conf\_matrix = confusion\_matrix(yTest, yPred)  
plt.figure(figsize=(8, 6))  
sns.heatmap(conf\_matrix, annot=True, fmt="d", cmap="Blues",  
 xticklabels=['Normal', 'Fraud'], yticklabels=['Normal', 'Fraud'])  
plt.title("Confusion Matrix")  
plt.xlabel("Predicted Class")  
plt.ylabel("True Class")  
plt.show()

* The confusion matrix shows:
  + True Positives (fraud predicted as fraud)
  + True Negatives (normal predicted as normal)
  + False Positives (normal predicted as fraud)
  + False Negatives (fraud predicted as normal)

### 10. Saving the Trained Model

import joblib  
  
joblib.dump(rfc, "fraud\_model.pkl")

* Saves the trained model into a file named fraud\_model.pkl.
* Allows us to reuse the model later without retraining it.
* This file acts like a saved brain that has already learned how to detect fraud.

### 11. Prediction Script (predict\_from\_csv.py)

import pandas as pd  
import joblib  
  
model = joblib.load("fraud\_model.pkl")  
new\_data = pd.read\_csv("new\_transactions.csv")  
predictions = model.predict(new\_data)  
print(predictions)

* Loads the saved model.
* Reads a new file new\_transactions.csv that has new transactions without labels.
* Predicts if each transaction is fraud or not.
* Prints out a list of 0s and 1s.

### 12. Supporting Files

* creditcard.csv: The main dataset used to train the model.
* new\_transactions.csv: A sample of new transactions where we want to predict fraud.
* fraud\_model.pkl: The saved model file used for making future predictions.

### 13. Real-World Use Case

A bank or payment platform can use this trained model to screen every transaction in real time. If a transaction looks like fraud, it can alert the user or hold the payment for manual review. Since fraud is rare, precision and recall are especially important to avoid false alarms or missed frauds.

The model doesn’t need to be trained again and again — it can be reused with the .pkl file, which speeds up deployment and makes integration easier with other systems.