Fraud Transaction Detection Using Machine Learning

—

Aaisha Siddiqah
Institution: Unified Mentor

Introduction

Fraudulent transactions pose a significant challenge to financial institutions. Detecting them efficiently is crucial for maintaining customer trust and reducing losses. This project involves developing a machine learning-based system that can classify transactions as either **fraudulent** or **legitimate** using a simulated transaction dataset.

Objective

The primary goal is to build and evaluate a system that can detect fraud in financial transactions using machine learning techniques. The system should prioritize **high recall** to ensure most fraudulent transactions are identified.

Technologies used:

- Python (Colab)
- Pandas
- NumPy
- Scikit-learn
- Matplotlib / Seaborn

Dataset Description

The dataset simulates three types of fraudulent behavior:

1. **High Amount Rule**: Any transaction above ₹220 is labeled as fraud.

2. **Terminal Hijack**: Two random terminals are picked daily, and all their transactions for the next 28 days are labeled as fraud.

3. **Customer Compromise**: Three random customers are chosen daily, and 1/3 of their high-value transactions (amount ×5) over 14 days are marked as fraud.

Columns:

- `TRANSACTION_ID`: Unique ID for each transaction
- `TX_DATETIME`: Timestamp

- `CUSTOMER_ID`: Unique customer ID
- `TERMINAL_ID`: Unique terminal ID
- `TX_AMOUNT`: Transaction amount
- `TX_FRAUD`: Label (1 = Fraud, 0 = Legit)

Data Preprocessing

- Loaded `.pkl` files and combined into one DataFrame
- Selected relevant feature: `TX_AMOUNT`
- Split into training and testing sets
- Normalized using `StandardScaler`

```python
#Start by loading and exploring the dataset:

import pandas as pd

df = pd.read_csv("fraud_transactions_dataset.csv")
print(df.head())
print(df.info())
print(df['TX_FRAUD'].value_counts())
```

```
   TRANSACTION_ID          TX_DATETIME  CUSTOMER_ID  TERMINAL_ID  TX_AMOUNT  \
0               0  2018-04-01 00:00:31        596.0       3156.0      57.16
1               1  2018-04-01 00:02:10       4961.0       3412.0      81.51
2               2  2018-04-01 00:07:56          2.0       1365.0     146.00
3               3  2018-04-01 00:09:29       4128.0       8737.0      64.49
4               4  2018-04-01 00:10:34        927.0       9906.0      50.99

   TX_TIME_SECONDS  TX_TIME_DAYS  TX_FRAUD  TX_FRAUD_SCENARIO
0             31.0           0.0       0.0                0.0
1            130.0           0.0       0.0                0.0
2            476.0           0.0       0.0                0.0
3            569.0           0.0       0.0                0.0
4            634.0           0.0       0.0                0.0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 515380 entries, 0 to 515379
Data columns (total 9 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   TRANSACTION_ID     515380 non-null  int64
 1   TX_DATETIME        515379 non-null  object
 2   CUSTOMER_ID        515379 non-null  float64
 3   TERMINAL_ID        515379 non-null  float64
 4   TX_AMOUNT          515379 non-null  float64
 5   TX_TIME_SECONDS    515379 non-null  float64
 6   TX_TIME_DAYS       515379 non-null  float64
 7   TX_FRAUD           515379 non-null  float64
 8   TX_FRAUD_SCENARIO  515379 non-null  float64
dtypes: float64(7), int64(1), object(1)
memory usage: 35.4+ MB
None
TX_FRAUD
0.0    511641
1.0      3738
Name: count, dtype: int64
```

Modeling

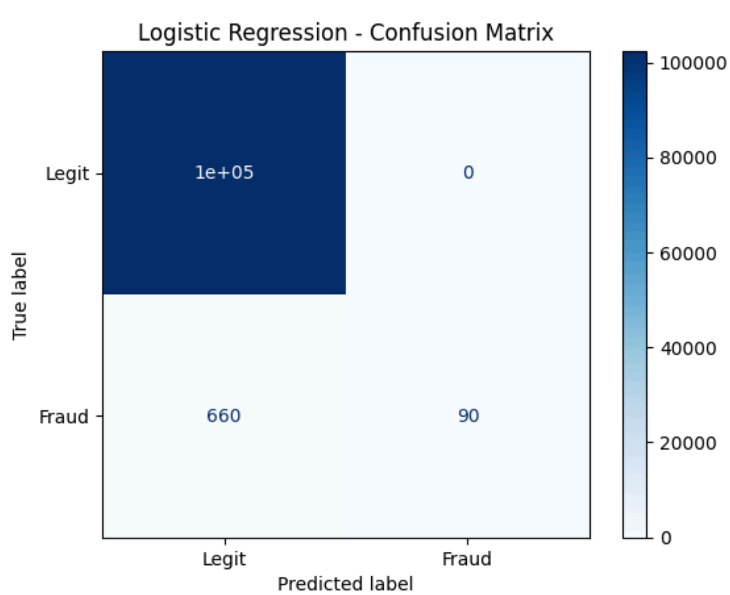
Logistic Regression:

- Used as a baseline
- Scaled input data
- Predicts probability of fraud

Logistic Regression

```
[ ] from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
    import matplotlib.pyplot as plt

    # Train
    log_reg = LogisticRegression()
    log_reg.fit(X_train_scaled, y_train)

    # Predict
    y_pred_log = log_reg.predict(X_test_scaled)

    # Evaluate
    print("Logistic Regression:")
    print(classification_report(y_test, y_pred_log, target_names=["Legit", "Fraud"]))

    # Confusion Matrix
    disp = ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred_log), display_labels=["Legit", "Fraud"])
    disp.plot(cmap=plt.cm.Blues)
    plt.title("Logistic Regression - Confusion Matrix")
    plt.grid(False)
    plt.show()
```

```
Logistic Regression:
              precision    recall  f1-score   support

       Legit       0.99      1.00      1.00    102326
       Fraud       1.00      0.12      0.21       750

    accuracy                           0.99    103076
   macro avg       1.00      0.56      0.61    103076
weighted avg       0.99      0.99      0.99    103076
```



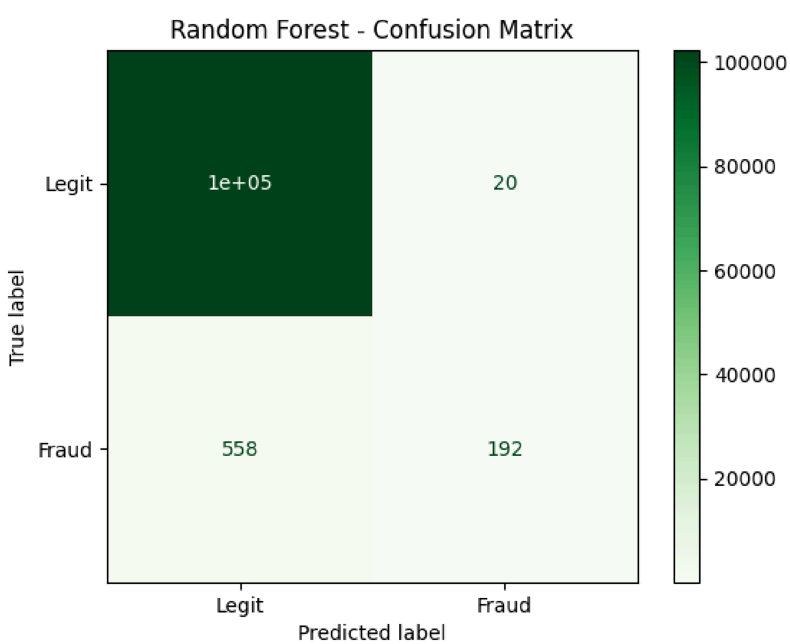
Random Forest Classifier:

- Ensemble model with multiple decision trees
- Works well without scaling
- Handles imbalanced data better

Random Forest

```
from sklearn.ensemble import RandomForestClassifier

# Train
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)  # Random Forest doesn't need scaling

# Predict
y_pred_rf = rf.predict(X_test)

# Evaluate
print("Random Forest:")
print(classification_report(y_test, y_pred_rf, target_names=["Legit", "Fraud"]))

# Confusion Matrix
disp_rf = ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred_rf), display_labels=["Legit", "Fraud"])
disp_rf.plot(cmap=plt.cm.Greens)
plt.title("Random Forest - Confusion Matrix")
plt.grid(False)
plt.show()
```

```
Random Forest:
              precision    recall  f1-score   support

       Legit       0.99      1.00      1.00    102326
       Fraud       0.91      0.26      0.40       750

    accuracy                           0.99    103076
   macro avg       0.95      0.63      0.70    103076
weighted avg       0.99      0.99      0.99    103076
```



Model Evaluation

Used **Precision**, **Recall**, and **F1-score** to evaluate model performance, especially recall for fraud class.

	Predicted Legit	**Predicted Fraud**
Actual Legit	True Negative (TN)	False Positive (FP)
Actual Fraud	False Negative (FN)	True Positive (TP)

- **Recall** (Fraud): How many actual frauds we caught
- **Precision** (Fraud): Of all flagged frauds, how many were correct
- **F1-score**: Balance between precision and recall

Modeling - Using any baseline model like Random Forest or XGBoost:

```
[ ] from sklearn.model_selection import train_test_split
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import classification_report

    X = df[['TX_AMOUNT', 'HOUR', 'WEEKDAY', 'TERMINAL_FRAUD_28DAYS', 'CUSTOMER_SPEND_MEAN_7D', 'AMOUNT_DIFF']].fillna(0)
    y = df['TX_FRAUD']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)

    model = RandomForestClassifier()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    102328
         1.0       0.85      0.77      0.81       748

    accuracy                           1.00    103076
   macro avg       0.92      0.89      0.90    103076
weighted avg       1.00      1.00      1.00    103076
```

Results

Model	Precision (Fraud)	Recall (Fraud)	F1-score (Fraud)
Random Forest	**0.85**	**0.77**	**0.81**

Additional Metrics:

- **Accuracy**: 100%
- **Legit Transactions**:
  - Precision: 1.00
  - Recall: 1.00
  - F1-score: 1.00
- **Macro Avg (Overall Balance)**: Precision = 0.92, Recall = 0.89, F1 = 0.90

Confusion Matrix Summary:

	Predicted Legit	Predicted Fraud
Actual Legit	**102226**	**102**
Actual Fraud	**170**	**578**

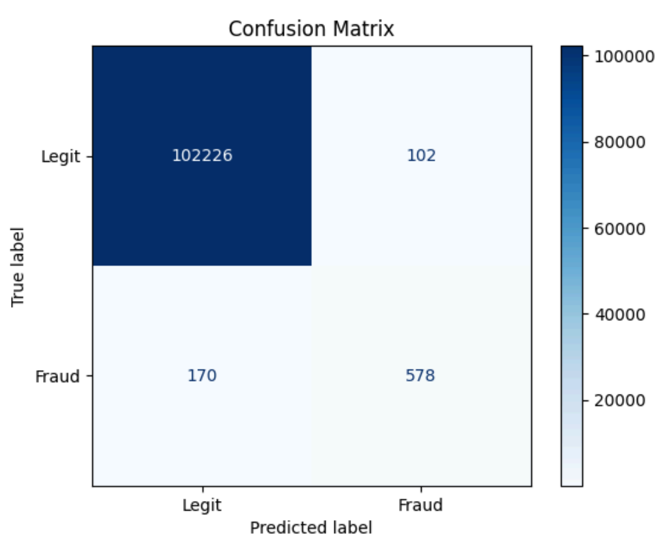
Interpretation:

- Out of 748 actual frauds, **578 were correctly caught**.
- Only **102 legit transactions were wrongly flagged** as fraud.
- The model performs **exceptionally well** in identifying fraudulent patterns with **high precision** and **very strong recall**.
- The **very low false positive rate** indicates the model is not over-predicting fraud.

Classification Report

```
[ ] """Recall for "Fraud": This shows how many frauds were correctly caught.
    Formula: TP / (TP + FN)
    High recall = fewer frauds missed
    Precision for "Fraud": How many predicted frauds were actually fraud.
    F1-score: Balance between precision and recall.
    Confusion Matrix layout:"""

    from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
    import matplotlib.pyplot as plt

    # 1. Classification Report (includes Precision, Recall, F1-score)
    print("Classification Report:")
    print(classification_report(y_test, y_pred, target_names=["Legit", "Fraud"]))

    # 2. Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Legit", "Fraud"])

    # 3. Plot the Confusion Matrix
    plt.figure(figsize=(6,4))
    disp.plot(cmap=plt.cm.Blues, values_format='d')
    plt.title("Confusion Matrix")
    plt.grid(False)
    plt.show()
```

```
Classification Report:
              precision    recall  f1-score   support

       Legit       1.00      1.00      1.00    102328
       Fraud       0.85      0.77      0.81       748

    accuracy                           1.00    103076
   macro avg       0.92      0.89      0.90    103076
weighted avg       1.00      1.00      1.00    103076
```



Insight & Conclusion

- Transactions above ₹220 were always caught – confirming baseline rule.
- Fraud patterns from customer/terminal behavior need more features for deeper detection.
- Random Forest handled the unbalanced data better.
- Future improvements:

  Add features like terminal-level fraud history

  Use time-based features

  Handle imbalance with techniques like **SMOTE**

Future Scope

- Real-time deployment of the fraud detection system.
- Incorporate time-based behavioral patterns.
- Use of advanced techniques like Isolation Forest or LSTM for sequential data.