CREDIT CARD FRAUD DETECTION – ML ASSIGNEMENT

Machine Learning - IT4060

Saraf MMMS

IT21297854

B.Sc. (Hons) Degree in Information Technology specialized in

Software Engineering

Department of Software Engineering

Sri Lanka Institute of Information Technology

Sri Lanka

April 2025

Contents

Introduction

Problem Statement

Credit card fraud is a significant and growing challenge for financial institutions and consumers worldwide. The ability to detect fraudulent transactions in real-time is critical for minimizing financial losses and enhancing security. Traditional rule-based fraud detection systems struggle to keep up with the increasingly sophisticated and evolving nature of fraud patterns.
In contrast, machine learning techniques offer a promising solution by analyzing historical transaction data to uncover patterns and anomalies that may indicate fraudulent activity.

This project aims to apply machine learning to identify fraudulent credit card transactions. Specifically, the task involves predicting whether a given transaction is fraudulent or legitimate based on various transaction attributes. Accurate fraud detection models are crucial for reducing financial losses and ensuring the reliability and security of financial systems.

Objective

The primary objective of this project is to build a machine learning model that accurately classifies credit card transactions as either fraudulent or legitimate.
This project focuses on applying the **Random Forest Classifier**, a powerful ensemble learning algorithm, to handle the imbalanced dataset and generate reliable predictions.
Through this project, we demonstrate the application of machine learning techniques to fraud detection and evaluate the effectiveness of Random Forest in identifying fraudulent transactions.

Problem Addressed

The task at hand is to build a machine learning model that can detect fraudulent credit card transactions. Credit card fraud is a significant concern in the financial sector, and detecting it quickly can save businesses and individuals from financial loss.

The goal of this project is to use machine learning techniques to classify transactions as either legitimate or fraudulent, based on features extracted from the transactions. The ultimate objective is to develop a reliable model that can assist in detecting fraudulent behavior without causing too many false positives.

Dataset Used

The dataset used in this project is the **Credit Card Fraud Detection** dataset sourced from **Kaggle**, available here.
The dataset contains **284,807 transactions** recorded by European cardholders during September 2013. Out of these, only **492 transactions** are fraudulent, making the dataset highly imbalanced.

Key features of the dataset include:

- **Time:** Seconds elapsed between this transaction and the first transaction in the dataset.
- **Amount:** Transaction amount in local currency.
- **V1 to V28:** Anonymized features resulting from a Principal Component Analysis (PCA) transformation applied to the original data.
- **Class:** The target variable, where `0` represents a legitimate transaction and `1` represents a fraudulent transaction.

The highly imbalanced nature of the dataset presents a critical challenge when building an effective predictive model.

Methodology

In this project, we applied the **Random Forest Classifier**, a supervised learning algorithm, to detect fraudulent credit card transactions.
Random Forest was chosen because it is an **ensemble learning method** that utilizes multiple decision trees to make predictions, which improves the model's accuracy, robustness, and ability to generalize to unseen data.

Why Random Forest?

- **Ensemble Learning:**
  By combining multiple decision trees, Random Forest reduces overfitting and increases the generalization performance of the model.
- **Handling Imbalanced Datasets:**
  Random Forest is known to perform relatively well on imbalanced datasets without the need for extensive preprocessing or oversampling techniques.
- **Feature Importance:**
  Random Forest can provide insights into which features are most important for classification, helping to understand the behavior of the model.

The implementation was carried out using the **scikit-learn** library in Python, utilizing the RandomForestClassifier to train and evaluate the model.

The following steps were performed:

1. **Data Preprocessing:**
   - The Amount feature was standardized using **StandardScaler** to normalize its distribution.
   - The Time feature was dropped as it did not contribute significant information for fraud detection.
2. **Model Training:**
   - The dataset was split into training and testing sets using a **stratified 80-20 split** to maintain the class distribution.
   - The **Random Forest Classifier** was trained on the training set using the Class column as the target variable.
3. **Model Evaluation:**
   - The model's performance was evaluated using several metrics, including **Accuracy, Precision, Recall, F1-Score, Confusion Matrix**, and **ROC-AUC Score**.
   - Additionally, a **Confusion Matrix Heatmap** and **ROC Curve** were plotted to visually assess the performance.

Results and Discussion

The performance of the Random Forest Classifier was evaluated using several metrics. Below are the results of the evaluation:

Confusion Matrix:

The confusion matrix showed the following values:

- **True Negatives (TN): 6333**
- **False Positives (FP): 3**
- **False Negatives (FN): 4**
- **True Positives (TP): 16**

Classification Report:

- **Accuracy:** 99.89%
- **Precision (Class 1 - Fraud):** 0.84
- **Recall (Class 1 - Fraud):** 0.80
- **F1-score (Class 1 - Fraud):** 0.82
- **ROC-AUC Score:** 0.97

Discussion:

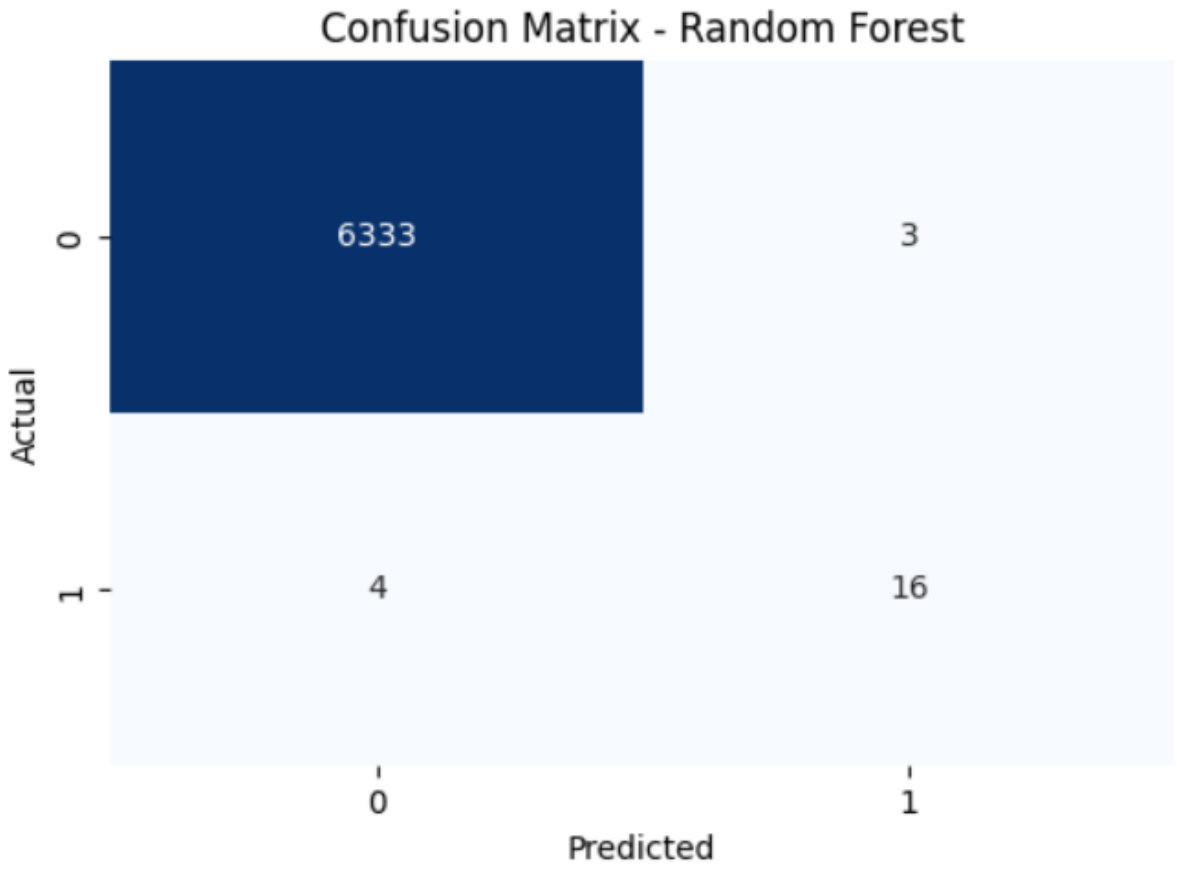
- **Accuracy:**
  The model achieved a high overall accuracy of 99.89%. However, due to the significant class imbalance in the dataset (very few fraudulent transactions compared to legitimate ones), accuracy alone is not a reliable metric. Even a model that predicts all transactions as legitimate would achieve high accuracy.
- **Precision and Recall:**
  The model obtained a precision of 0.84 for fraudulent transactions, meaning when it predicted a transaction as fraud, it was correct 84% of the time.
  The recall of 0.80 indicates that the model was able to detect 80% of actual frauds.
  This balance between precision and recall is critical for fraud detection, where identifying frauds is more important than just overall accuracy.
- **ROC-AUC Score:**
  The ROC-AUC score of 0.97 indicates that the Random Forest model has excellent discriminative power. The model can effectively differentiate between fraudulent and non-fraudulent transactions.

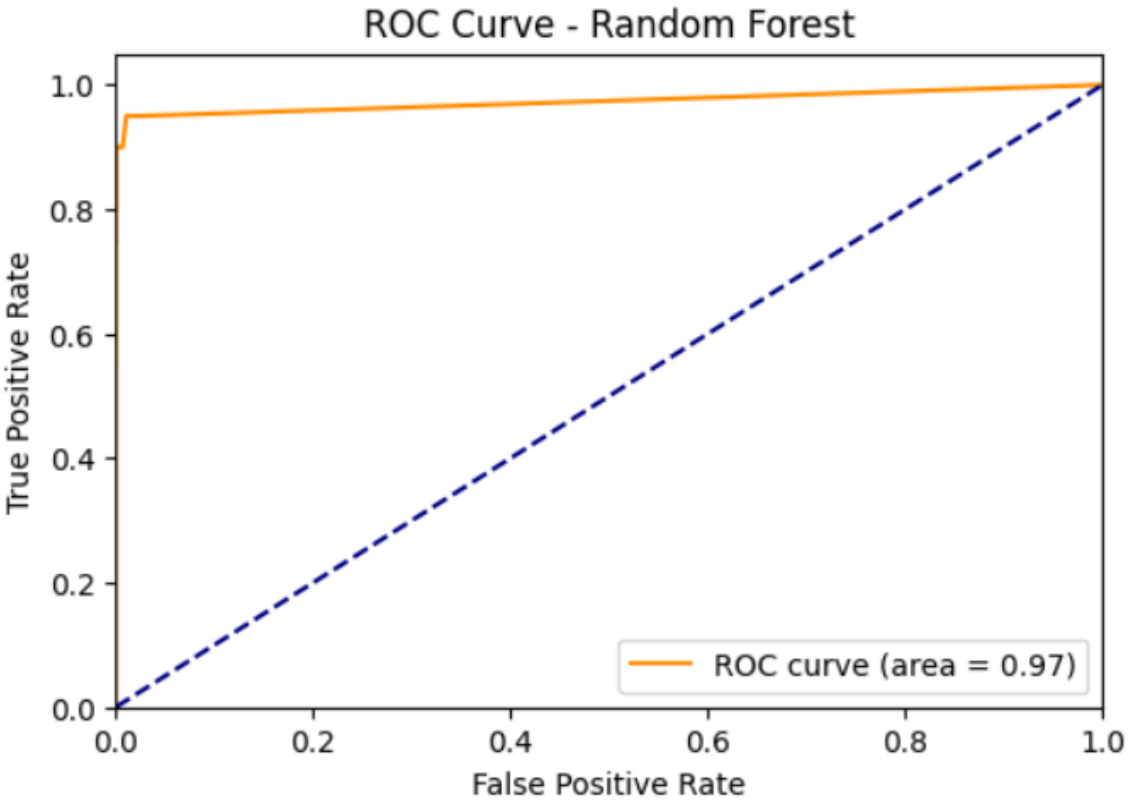
Screenshots:

Below are the graphical results included for better visualization:

- **ROC Curve:**
  The ROC curve shows that the model has a strong ability to distinguish between classes, with an AUC close to 1.
- **Confusion Matrix Heatmap:**
  The heatmap visualization clearly shows very few misclassifications, indicating good model performance.

(Screenshots of the ROC Curve and Confusion Matrix are attached below.)




ROC Curve - Random Forest
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.97)

Limitations and Future Work

Limitations:

- **Class Imbalance:** The dataset is highly imbalanced, with only about 0.17% of the transactions being fraudulent. This imbalance makes it difficult for the model to learn the characteristics of fraudulent transactions effectively.
- **Over-reliance on Majority Class:** The high accuracy reported by the model is primarily due to its ability to predict the majority class (legitimate transactions), while missing the minority class (fraudulent transactions).

Future Work:

1. **Handle Class Imbalance:**
    a. Techniques such as **SMOTE (Synthetic Minority Over-sampling Technique)** or **undersampling** could be used to balance the classes and help the model better learn the characteristics of fraudulent transactions.
2. **Try Other Models:**
    a. Explore more advanced algorithms like **XGBoost**, **LightGBM**, and **Neural Networks** to see if they can provide better performance on the imbalanced dataset.
3. **Feature Engineering:**
    a. Investigate the possibility of creating new features, such as aggregating transaction amounts over time, to provide additional context to the model.
    b. **PCA (Principal Component Analysis)** can also be applied to reduce dimensionality while retaining useful information.
4. **Real-time Fraud Detection:**
    a. Implement the model for **real-time fraud detection** in a production environment, where transactions are assessed in real time as they occur.
5. **Model Interpretability:**
    a. Use tools like **SHAP** (SHapley Additive exPlanations) or **LIME** (Local Interpretable Model-Agnostic Explanations) to understand the decision-making process of the model and identify the most important features.

Appendix: Source Code

Below is the source code for the project:

```
# -*- coding: utf-8 -*-
"""Credit Card Fraud Detection.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1Mq_2r2UHcr8k6U9MB5jIouaJYf1zKHNB

Import Necessary Libraries
"""

# Basic libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Machine Learning libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score

"""Load the Dataset"""

# Load the dataset
df = pd.read_csv('creditcard.csv')
```

```
# View first few rows
df.head()

print(df['Class'].value_counts())

"""Explore and Understand the Data"""

# Basic info
print(df.info())

# Check missing values
print(df.isnull().sum())

""" Data Preprocessing"""

# Scale the 'Amount' feature
scaler = StandardScaler()
df['Amount'] = scaler.fit_transform(df[['Amount']])

# Drop the 'Time' feature
df = df.drop(columns=['Time'])

# Split features and target
X = df.drop('Class', axis=1)  # Features
y = df['Class']               # Target

"""Data Splitting"""
```

```
# Impute or remove NaN values from the target variable 'y' before splitting
# Option 1: Remove rows with NaN values in 'y'
df = df.dropna(subset=['Class'])  # Assuming 'Class' is your target column

# Option 2: Impute NaN values with a suitable strategy (e.g., most frequent value)
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='most_frequent') # You can change the strategy
y_imputed = imputer.fit_transform(df[['Class']]) # Use df[['Class']] to keep it as a column vector
df['Class'] = y_imputed # Update the 'Class' column in your DataFrame

# Now proceed with the train_test_split
X = df.drop('Class', axis=1)  # Features
y = df['Class']               # Target

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

"""Apply Random Forest Classifier"""

# Create the Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
rf_model.fit(X_train, y_train)

"""Make Predictions"""
```

```
# Make predictions
y_pred = rf_model.predict(X_test)

"""Model Evaluation"""

# Classification Report
print("Classification Report:\n")
print(classification_report(y_test, y_pred))

# Confusion Matrix
print("Confusion Matrix:\n")
print(confusion_matrix(y_test, y_pred))

# ROC-AUC Score
roc_score = roc_auc_score(y_test, rf_model.predict_proba(X_test)[:,1])
print("ROC-AUC Score:", roc_score)

"""Confusion Matrix Heatmap"""

# Import
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

# Plot Confusion Matrix
```

```
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix - Random Forest')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

"""ROC Curve"""

from sklearn.metrics import roc_curve, auc

# Predict probabilities
y_probs = rf_model.predict_proba(X_test)[:, 1]

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_probs)
roc_auc = auc(fpr, tpr)

# Plot ROC Curve
plt.figure(figsize=(6,4))
plt.plot(fpr, tpr, color='darkorange', label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Random Forest')
plt.legend(loc="lower right")
plt.show()
```

```
"""# Pick a random sample from X_test

"""

sample = X_test.iloc[15]  # You can change the index (e.g., 10, 20, etc.)
print(sample)

"""Predict That Sample
python
Copy code

"""

# Reshape the sample
sample = sample.values.reshape(1, -1)

# Predict the class
predicted_class = rf_model.predict(sample)
print("Predicted Class:", predicted_class[0])

"""Check the actual value"""

actual_class = y_test.iloc[15]
print("Actual Class:", actual_class)

import joblib
joblib.dump(rf_model, 'credit_card_fraud_model.pkl') # Changed 'model' to 'rf_model'
```

```
import pickle


# Save using pickle
with open('credit_card_fraud_model.pickle', 'wb') as f:
    pickle.dump(rf_model, f)


# Load the model using pickle
with open('credit_card_fraud_model.pickle', 'rb') as f:
    loaded_model = pickle.load(f)
```

References

1. **Credit Card Fraud Detection Dataset**, Kaggle.
   Available at: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud
2. **Scikit-learn: Machine Learning in Python**, Pedregosa et al., Journal of Machine Learning Research, 2011.
   Available at: https://scikit-learn.org/stable/
3. **Random Forest Classifier Documentation**, Scikit-learn.
   Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html