

CREDIT CARD FRAUD DETECTION – ML ASSIGNMENT

Machine Learning - IT4060

Sajid Ahmed MJ

IT21294570

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.....	3
Problem Addressed	4
Dataset Used	5
Methodology.....	6
Results and Discussion	8
Limitations and Future Work	9
Appendix: Source Code	11

Introduction

Credit card fraud represents a growing threat in the era of digital finance, where millions of transactions are processed every second across the globe. As online payments become increasingly common, financial institutions face mounting pressure to ensure that transactions are not only fast and convenient but also secure. Fraudulent transactions not only lead to significant financial losses but also erode customer trust, damage institutional reputation, and may result in legal liabilities.

To address this challenge, **Machine Learning (ML)** has emerged as a powerful ally in the detection and prevention of credit card fraud. In particular, **supervised learning algorithms** are well-suited to this task, as they can be trained to distinguish between legitimate and fraudulent transactions based on historical data. Among the various supervised learning techniques, the **Support Vector Machine (SVM)** stands out for its effectiveness in handling high-dimensional data and its ability to create decision boundaries that maximize classification margins.

This report presents a complete machine learning pipeline for credit card fraud detection using an SVM-based classification model. The key focus areas include preprocessing steps tailored for imbalanced datasets, model training, evaluation using appropriate metrics, and a critical analysis of the results.

The dataset employed for this project is the **Credit Card Fraud Detection Dataset**, publicly available on Kaggle. It comprises 284,807 anonymized credit card transactions conducted by European cardholders in September 2013. Among these, only 492 transactions were labeled as fraudulent, representing just **0.172%** of the total data — highlighting a significant class imbalance.

The primary challenge with this dataset lies in the rarity of fraudulent transactions, which can severely affect the learning ability of traditional classifiers. This project tackles the problem by employing class-balancing techniques and evaluating performance using specialized metrics such as Precision-Recall AUC, F1-score, and ROC-AUC, which are more appropriate than accuracy for such imbalanced classification tasks.

The objective of this project is to develop and assess a robust machine learning model that can effectively classify transactions and serve as a foundation for real-world fraud detection systems.

Problem Addressed

The task addressed in this project is the development of a machine learning model capable of accurately detecting fraudulent credit card transactions. Credit card fraud poses a serious threat to the financial industry, leading to substantial monetary losses and diminished customer trust. Early and accurate detection of such fraud is crucial for minimizing risk and maintaining the integrity of financial systems.

The primary objective of this project is to leverage machine learning techniques to classify credit card transactions as either legitimate or fraudulent based on a set of derived features. These features, extracted from transactional data, are used to train a predictive model that can identify suspicious activities with high precision. A key challenge lies in achieving a balance between correctly identifying fraud (true positives) and minimizing false alarms (false positives), which can inconvenience customers and burden fraud investigation teams. The overarching goal is to build a robust, scalable model that can effectively support real-world fraud detection systems.

Dataset Used

Source and Overview

The dataset used in this project is publicly available from [Kaggle](#) and originates from research conducted by European credit card issuers. It contains anonymized data for transactions that took place over a period of two days in September 2013.

Characteristics of the Dataset

- **Total Transactions:** 284,807
- **Fraudulent Transactions:** 492 (0.172%)
- **Non-Fraudulent Transactions:** 284,315
- **Number of Features:** 30
 - V1 through V28: Principal components obtained via PCA
 - Time: Seconds elapsed between each transaction and the first transaction
 - Amount: Transaction amount
 - Class: Binary label (0 = legitimate, 1 = fraud)

Data Challenges

The primary challenge of this dataset lies in its **high class imbalance** — fraudulent transactions comprise less than 0.2% of the total records. Traditional classifiers may achieve high accuracy simply by predicting all transactions as non-fraudulent. Hence, using appropriate evaluation metrics and class-weighting strategies is crucial.

Methodology

Data Preprocessing

Data preprocessing plays a critical role in ensuring model accuracy and robustness.

Feature Scaling

Although the PCA-transformed features (V1–V28) are already standardized, the Amount and Time features were not. We applied **standard scaling** to these features using StandardScaler to normalize their values.

```
scaler = StandardScaler()
df['Amount'] = scaler.fit_transform(df[['Amount']])
df['Time'] = scaler.fit_transform(df[['Time']])
```

Train-Test Split

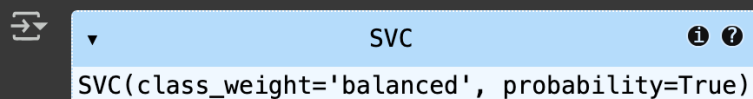
To preserve the class distribution across training and test sets, **Stratified Sampling** was used:

```
[31] X_train, X_test, y_train, y_test = train_test_split(
      X_scaled, y, test_size=0.3, stratify=y, random_state=42
    )
```

Imbalance Handling

To address the extreme class imbalance, the SVM model was configured with `class_weight='balanced'`, ensuring that the penalty for misclassifying the minority class (fraud) was increased during training.

```
svm_model = SVC(kernel='rbf', class_weight='balanced', probability=True)
svm_model.fit(X_train, y_train)
```

 The image shows a Jupyter Notebook cell with a dropdown menu for the `SVC` class. The dropdown is open, showing the configuration `SVC(class_weight='balanced', probability=True)`. The cell also contains a play button icon and a status bar indicating 15 minutes.

Support Vector Machine (SVM)

SVM is a well-known supervised ML algorithm designed to find the optimal hyperplane that separates data points of different classes with the maximum margin. For this classification task:

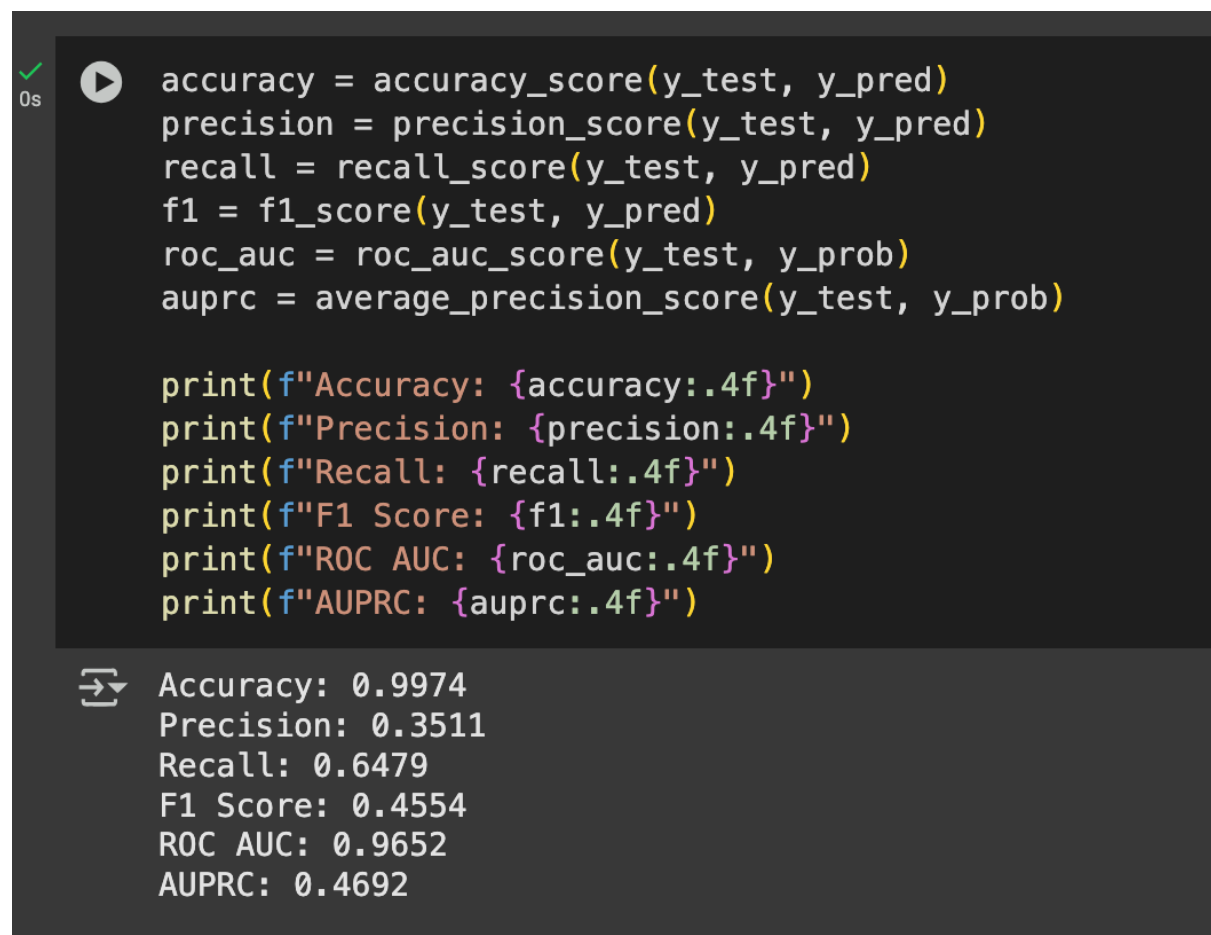
- **Kernel Used:** RBF (Radial Basis Function)
- **Regularization Parameter (C):** Default (tuned in future work)
- **Gamma:** Controls the curvature of the decision boundary

The non-linearity of the RBF kernel enables SVM to handle complex patterns in the data, making it suitable for this PCA-transformed dataset.

Evaluation Metrics

Since the dataset is highly imbalanced, typical metrics like accuracy can be misleading. Instead, the following were prioritized:

- **Precision:** Of all transactions predicted as fraud, how many were correct?
- **Recall (Sensitivity):** Of all actual fraud cases, how many were detected?
- **F1-Score:** Harmonic mean of precision and recall
- **ROC AUC:** Area under the Receiver Operating Characteristic curve
- **AUPRC:** Area under the Precision-Recall curve, more reliable for imbalanced data



```
0s ▶ accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_prob)
    auprc = average_precision_score(y_test, y_prob)

    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")
    print(f"ROC AUC: {roc_auc:.4f}")
    print(f"AUPRC: {auprc:.4f}")

⇒ Accuracy: 0.9974
   Precision: 0.3511
   Recall: 0.6479
   F1 Score: 0.4554
   ROC AUC: 0.9652
   AUPRC: 0.4692
```

Results and Discussion

The performance of the Support Vector Machine was evaluated using several metrics. Below are the results of the evaluation:

Confusion Matrix:

The confusion matrix showed the following values:

	Predicted: Not Fraud	Predicted: Fraud
Actual: Not Fraud	84,806	170
Actual: Fraud	50	92

Classification Report:

Classification Report:					
	precision	recall	f1-score	support	
0	1.00	1.00	1.00	84976	
1	0.35	0.65	0.46	142	
accuracy			1.00	85118	
macro avg	0.68	0.82	0.73	85118	
weighted avg	1.00	1.00	1.00	85118	

Performance Metrics:

- **Accuracy:** 99.74%
- **Precision (Fraud):** 35.11%
- **Recall (Fraud):** 64.79%
- **F1 Score (Fraud):** 45.54%
- **ROC AUC:** 0.9652
- **AUPRC:** 0.4692

While the model achieved high overall accuracy due to the class imbalance, the focus should be on performance for the fraud class. The recall value of **0.65** indicates that the model

detected approximately 65% of fraudulent transactions. However, a relatively low precision of **0.35** means that many flagged transactions were false positives.

Discussion:

Insights from Results

Despite a high overall accuracy, the key performance indicators for fraud detection (recall, F1, AUPRC) show room for improvement. The model was able to correctly identify 65% of fraudulent transactions. However, it misclassified legitimate transactions as fraud in 170 cases, as reflected in the confusion matrix.

This trade-off between **recall and precision** is a typical outcome when detecting rare events. Given the severe financial and reputational cost of missed fraud, higher recall is often preferred, even if it results in more false positives.

Strengths of the SVM Model

- **Robust to High-Dimensional Data:** SVM performed well with PCA-transformed features.
- **Good Recall and AUC:** Demonstrated a high ability to distinguish between classes (AUC = 0.9652).
- **Effective Handling of Imbalance:** Built-in class weighting proved useful.

Limitations and Future Work

Limitations

- **Low Precision:** A precision of 0.35 means many false alarms.
- **Scalability:** SVMs are computationally intensive for large datasets, and may not be optimal for real-time fraud detection in high-volume settings.
- **Hyperparameters Untuned:** The model used default hyperparameters; tuning could enhance performance.

Future Work:

The current SVM implementation lays the groundwork for further refinement and comparison. Future improvements could include:

- **Hyperparameter Tuning:** Use GridSearchCV or RandomizedSearchCV to optimize C, gamma, and kernel.

- **Synthetic Sampling Techniques:** Apply SMOTE or ADASYN to oversample fraud cases and train on a balanced dataset.
- **Ensemble Models:** Compare SVM with tree-based ensembles like Random Forest, XGBoost, and LightGBM.
- **Cost-Sensitive Learning:** Introduce cost-matrices to reflect the real-world impact of misclassification.
- **Feature Engineering:** Extract additional temporal features (e.g., transaction hour) or apply anomaly detection techniques.

Appendix: Source Code

Below is the source code for the project:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns


from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.metrics import (classification_report, confusion_matrix,
                              accuracy_score, precision_score, recall_score,
                              f1_score, roc_auc_score, average_precision_score,
                              precision_recall_curve)

df = pd.read_csv('creditcard.csv')

print("Shape:", df.shape)

print("Columns:", df.columns.tolist())

print("Class distribution:\n", df['Class'].value_counts())

# View first few rows

df.head()

print(df.head())

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

print(df.info())


# Check missing values

print(df.isnull().sum())

# Fraud ratio

fraud_ratio = df['Class'].sum() / len(df) * 100

print(f'Fraudulent transactions: {fraud_ratio:.4f}%')
```

```

plt.figure(figsize=(6, 4))
sns.countplot(x='Class', data=df)
plt.title("Class Distribution")
plt.xticks([0, 1], ['Non-Fraud (0)', 'Fraud (1)'])
plt.ylabel('Count')
plt.show()

plt.figure(figsize=(12, 8))
sns.heatmap(df.iloc[:, -10:].corr(), cmap='coolwarm', annot=True)
plt.title("Feature Correlation (Last 10 Columns)")
plt.show()

# Drop duplicates (if any)
df.drop_duplicates(inplace=True)

# Scale features (excluding 'Class')
X = df.drop('Class', axis=1)
y = df['Class']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.3, stratify=y, random_state=42
)

svm_model = SVC(kernel='rbf', class_weight='balanced', probability=True)
svm_model.fit(X_train, y_train)
y_pred = svm_model.predict(X_test)
y_prob = svm_model.predict_proba(X_test)[:, 1]
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
accuracy = accuracy_score(y_test, y_pred)

```

```
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_prob)
auprc = average_precision_score(y_test, y_prob)

print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1 Score: {f1:.4f}')
print(f'ROC AUC: {roc_auc:.4f}')
print(f'AUPRC: {auprc:.4f}')

precision_vals, recall_vals, _ = precision_recall_curve(y_test, y_prob)
plt.figure(figsize=(8, 5))
plt.plot(recall_vals, precision_vals, color='b', label=f'AUPRC = {auprc:.4f}')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.grid(True)
plt.show()
```