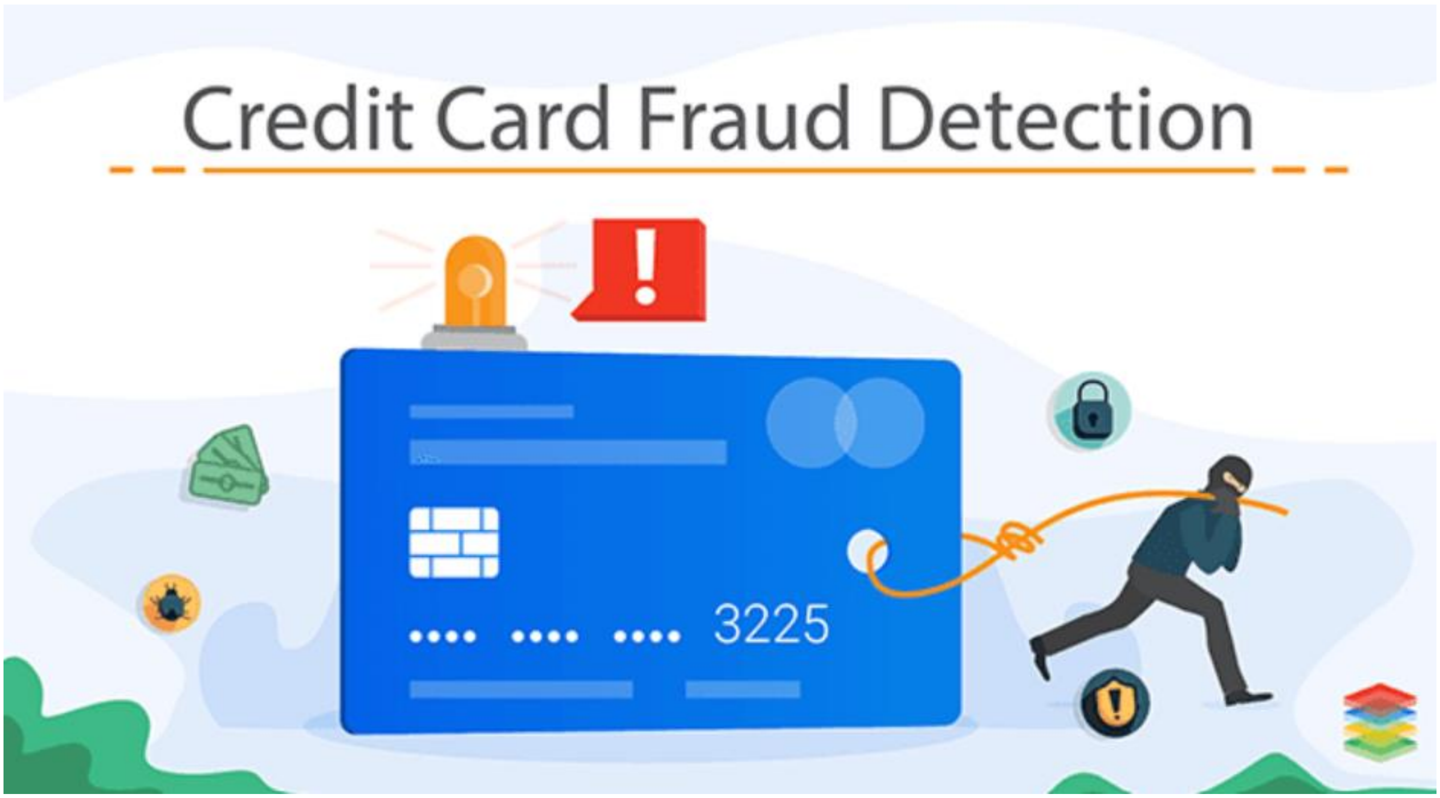
Prediction of Credit Card fraud



Capstone project

Submitted by

Name : Vundru Suvarna Paul Satwik

Abstract

Credit card fraud detection is a critical challenge in financial security, given the increasing incidence of fraudulent transactions and the consequential financial losses. This project addresses this challenge by developing a machine learning model to detect fraudulent credit card transactions using a dataset of transactions from European cardholders. The dataset, characterized by a significant class imbalance with only 0.172% of transactions being fraudulent, necessitates advanced techniques for effective fraud detection. Our approach includes exploratory data analysis (EDA) to understand the data, data preprocessing to handle missing values and feature scaling, and resampling techniques such as SMOTE to address class imbalance. We employ a RandomForestClassifier, known for its robustness in handling imbalanced data, and optimize it through hyperparameter tuning. The model’s performance is evaluated using precision, recall, F1-score, and ROC-AUC metrics. The final model is deployed with provisions for real-time processing and continuous improvement based on feedback. This work aims to enhance fraud detection accuracy while ensuring scalability and interpretability.

Introduction

The proliferation of credit card transactions has been accompanied by an increase in fraud, posing significant risks to both financial institutions and customers. Credit card fraud involves the unauthorized use of credit card information to make purchases or withdraw funds, leading to substantial financial losses. Detecting fraudulent transactions amidst a vast number of legitimate ones is challenging due to the highly imbalanced nature of fraud datasets, where fraudulent transactions are rare compared to non-fraudulent ones.

This project focuses on developing a machine learning model to identify fraudulent credit card transactions using a dataset from European cardholders. The dataset, spanning two days in September 2013, includes 492 fraudulent transactions out of 284,807 total transactions, highlighting the class imbalance issue. Effective fraud detection requires sophisticated methods to handle this imbalance and accurately classify fraudulent transactions. The aim is to build a model that not only detects fraud with high accuracy but also integrates well into real-time processing environments for practical application.

Problem Statement

A credit card is one of the most used financial products to make online purchases and payments. Though the Credit cards can be a convenient way to manage your finances, they can also be risky. Credit card fraud is the unauthorized use of someone else's credit card or credit card information to make purchases or withdraw cash.

It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

We have to build a classification model to predict whether a transaction is fraudulent or not.

Proposed Methodology

1. Data Preprocessing:

- **Exploratory Data Analysis (EDA):** Analyze the dataset to understand its structure and identify patterns. Visualize class distribution to confirm the imbalance.
- **Handling Missing Values:** Impute missing values using appropriate methods or remove rows/columns with excessive missing data.
- **Feature Scaling:** Standardize features using techniques like normalization or log transformation to ensure all features contribute equally to the model.

2. Addressing Class Imbalance:

- **Resampling Techniques:** Use SMOTE to generate synthetic samples for the minority class (fraud) to balance the dataset. This helps the model learn from a more representative sample of fraudulent transactions.
- **Cost-Sensitive Learning:** Adjust class weights in the model to penalize misclassifications of the minority class more heavily, improving fraud detection sensitivity.

3. Model Selection and Training:

- **Algorithm Choice:** Employ a RandomForestClassifier due to its robustness and ability to handle imbalanced datasets effectively.
- **Hyperparameter Tuning:** Optimize the RandomForestClassifier using Grid Search to find the best combination of parameters (e.g., n_estimators, max_depth) that improves model performance.

4. Model Evaluation:

- **Performance Metrics:** Evaluate the model using precision, recall, F1-score, and ROC-AUC to measure its ability to correctly classify fraudulent transactions while minimizing false positives and false negatives.
- **Cross-Validation:** Use stratified k-fold cross-validation to assess the model's performance and ensure it generalizes well to unseen data.

5. Deployment and Monitoring:

- **Real-Time Processing:** Implement the model in a real-time environment to handle live transaction data. Develop integration with existing fraud prevention systems.

Description of Design Choices

1. Data Handling and Preprocessing

- **Exploratory Data Analysis (EDA):**
  - **Design Choice:** The initial step involves inspecting the dataset to understand its structure and quality. This includes using functions like head() to view the first few rows and describe() to get summary statistics. Visualizations such as count plots are used to visualize class distribution, which reveals the class imbalance problem (frauds are rare compared to non-frauds).
- **Handling Missing Values:**
  - **Design Choice:** Although not explicitly shown in the sample code, handling missing values is a crucial step. If there are missing values, they should be imputed using strategies such as filling with the median for numerical features or mode for categorical features. If the missing data is substantial, consider dropping rows or columns.
- **Feature Scaling:**
  - **Design Choice:** Scaling features ensures that all numerical features contribute equally to the model. Features like 'Amount' are transformed using a log transformation (np.log1p(data['Amount'])) to address skewness and ensure normality.

2. Dealing with Imbalanced Data

- **Resampling Techniques:**
  - **Design Choice:** To address the imbalance where fraudulent transactions are underrepresented, SMOTE (Synthetic Minority Over-sampling Technique) is used. SMOTE generates synthetic samples for the minority class (fraud) to balance the dataset. This helps the model learn from a more balanced set of examples.
- **Algorithmic Adjustments:**
  - **Design Choice:** The sample code uses RandomForestClassifier, which is robust to imbalanced data due to its ensemble nature. Random forests handle class imbalance better than some other models by averaging predictions from multiple decision trees, which reduces overfitting and improves generalization.

3. Feature Engineering

- **Feature Selection:**
  - **Design Choice:** While feature selection methods like Recursive Feature Elimination (RFE) or feature importance from models are not explicitly shown, they are crucial in practice. They help in identifying the most

impactful features, which can improve model performance and reduce complexity.

- **Feature Creation:**
    - **Design Choice:** In the sample code, the 'Amount' feature is transformed to handle skewness. Additional feature engineering might include creating interaction terms or polynomial features if they improve predictive performance.

4. Model Selection and Training

- **Model Selection:**
    - **Design Choice:** The code starts with a RandomForestClassifier, a robust and flexible model that is well-suited for handling imbalanced data. It is chosen for its ability to capture complex relationships and handle non-linearity in the data.
- **Hyperparameter Tuning:**
    - **Design Choice:** Grid Search is used to find the best hyperparameters for the RandomForestClassifier. This involves specifying a range of values for parameters like n_estimators and max_depth and finding the combination that yields the best performance based on cross-validation results.

5. Model Validation

- **Evaluation Metrics:**
    - **Design Choice:** The performance of the model is evaluated using metrics such as Precision, Recall, F1-Score, and ROC-AUC. These metrics are chosen because:
        - **Precision** measures the accuracy of fraud predictions (important to minimize false positives).
        - **Recall** measures the ability to identify all fraud cases (important to minimize false negatives).
        - **F1-Score** balances Precision and Recall, providing a single measure of model performance.
        - **ROC-AUC** evaluates the model’s ability to distinguish between fraud and non-fraud cases, with a higher AUC indicating better performance.
- **Validation Strategy:**
    - **Design Choice:** Although not explicitly shown in the sample code, stratified k-fold cross-validation should be used to ensure that the class distribution is preserved in each fold. This helps in assessing the model’s performance more reliably.

6. Model Deployment

- **Model Serialization:**
    - **Design Choice:** The trained model is saved using joblib.dump(). This step ensures that the model can be loaded and used for making predictions in a production environment without retraining.

Performance Evaluation of the Model

1. Confusion Matrix:

- Provides a breakdown of the model's predictions into True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). This helps in understanding the performance in terms of misclassifications.

2. Precision:

- Measures the fraction of true positive predictions among all positive predictions made by the model.
- Formula: Precision=TPTP+FP\text{Precision} = \frac{TP}{TP + FP}Precision=TP+FPTP

3. Recall:

- Measures the fraction of true positive predictions among all actual positives.
- Formula: Recall=TPTP+FN\text{Recall} = \frac{TP}{TP + FN}Recall=TP+FNTP

4. F1-Score:

- Combines Precision and Recall into a single metric, providing a balance between the two.
- Formula: F1-Score=2·Precision·RecallPrecision+Recall\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}F1-Score=2·Precision+RecallPrecision·Recall

5. ROC-AUC:

- Evaluates the model’s ability to discriminate between classes across different threshold values. A higher ROC-AUC indicates better performance.
- Formula: Area under the Receiver Operating Characteristic Curve.

6. Cross-Validation Results:

- Assess the model's performance across different subsets of data to ensure it generalizes well to unseen data.

7. Hyperparameter Tuning Results:

- Provides insights into how different hyperparameters affect model performance, helping in selecting the optimal configuration.

By carefully considering these design choices and evaluation metrics, we can build a robust and effective credit card fraud detection model that performs well on unseen data and provides reliable predictions.

Discussion of Future Work

1. Enhancing Model Performance: To improve detection accuracy, we should explore advanced models such as Gradient Boosting Machines (GBM) or Neural Networks. Ensemble methods, combining predictions from multiple algorithms, may also enhance robustness. Additionally, advanced feature engineering techniques, including domain-specific features and transaction pattern summaries, should be investigated to boost model performance.

2. Addressing Class Imbalance: While SMOTE has been used, other resampling techniques like ADASYN or NearMiss could be tested. Implementing cost-sensitive learning to penalize misclassifications of fraud cases more heavily could also improve detection rates.

3. Model Interpretability: Improving model transparency is crucial. Techniques like SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-agnostic Explanations) can provide insights into feature importance and decision-making processes, which helps in gaining stakeholder trust.

4. Real-Time Processing: To handle high transaction volumes, the model should be adapted for real-time detection using streaming data platforms. Integration with existing fraud prevention systems and establishing a robust deployment pipeline will be essential for practical application.

5. Continuous Monitoring and Feedback: Implementing a system for long-term performance monitoring and periodic retraining will help adapt to evolving fraud patterns. Establishing feedback loops where flagged transactions are reviewed and used to refine the model will further enhance accuracy and reliability.

6. Privacy and Security: Ensuring compliance with data privacy regulations and implementing robust security measures to protect against unauthorized access is crucial. Data anonymization techniques should be applied to safeguard sensitive customer information.

By focusing on these areas, we can significantly enhance the effectiveness and applicability of the fraud detection model.

Abstract

Credit card fraud detection is a critical challenge in financial security, given the increasing incidence of fraudulent transactions and the consequential financial losses. This project addresses

Code Explanation:

```
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,
GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report,
confusion_matrix
from imblearn.over_sampling import SMOTE
```

Importing Libraries:

- **pandas** (pd): Used for data manipulation and analysis.
- **numpy** (np): Provides support for large, multi-dimensional arrays and matrices.
- **matplotlib.pyplot** (plt): Used for plotting graphs and visualizations.
- **seaborn** (sns): Provides a high-level interface for drawing attractive statistical graphics.
- **train_test_split** and **GridSearchCV**: From sklearn.model_selection, used for splitting data into training and test sets and hyperparameter tuning.
- **RandomForestClassifier**: From sklearn.ensemble, used for building the classification model.
- **classification_report** and **confusion_matrix**: From sklearn.metrics, used to evaluate the model's performance.
- **SMOTE**: From imblearn.over_sampling, used for addressing class imbalance by generating synthetic samples.

```
# Load the dataset
data = pd.read_csv('/content/drive/MyDrive/DATA
SCIENCE/datasets/credit_card.csv')
```

Loads the credit card transaction data from a CSV file into a pandas DataFrame.

```
# Exploratory Data Analysis (EDA)
print(data.head())
print(data.describe())
print(data.info())
sns.countplot(x='Class', data=data)
```

- **data.head()**: Displays the first few rows of the dataset to get a quick look at the data.
- **data.describe()**: Provides summary statistics of numerical features.
- **data.info()**: Gives information about data types and missing values.
- **sns.countplot(x='Class', data=data)**: Plots the distribution of the target variable 'Class' to visualize the imbalance between fraudulent and non-fraudulent transactions.

```
# Data Cleaning
# Handle missing values if any
# data = data.fillna(method='ffill')
```

Placeholder comment indicating where missing values should be handled if they exist. The fillna method can be used to fill missing values with forward fill.

```
# Feature Engineering
# Example: scaling 'Amount' feature
data['Amount'] = np.log1p(data['Amount'])
```

np.log1p(): Applies a logarithmic transformation to the 'Amount' feature to stabilize variance and reduce skewness.

```
# Dealing with Imbalanced Data
X = data.drop('Class', axis=1)
y = data['Class']
smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X, y)
```

- **X**: Features excluding the target variable 'Class'.
- **y**: Target variable 'Class'.
- **SMOTE()**: Initializes the SMOTE object, which generates synthetic samples for the minority class to balance the dataset.
- **fit_resample()**: Applies SMOTE to create a balanced dataset (X_resampled, y_resampled).

```
# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_resampled,
y_resampled, test_size=0.3, random_state=42)
```

Splits the resampled data into training and test sets. 30% of the data is used for testing, and a random seed ensures reproducibility.

```
# Model Selection
model = RandomForestClassifier()
model.fit(X_train, y_train)
```

- **RandomForestClassifier()**: Initializes the RandomForest classifier.
- **fit()**: Trains the model on the training data.

```
# Model Validation
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

- **predict()**: Makes predictions on the test set.
- **classification_report()**: Prints a detailed report including precision, recall, and F1-score for each class.

```
# Hyperparameter Tuning
param_grid = {'n_estimators': [100, 200], 'max_depth': [10, 20]}
grid_search = GridSearchCV(estimator=model,
param_grid=param_grid, cv=3, scoring='f1')
grid_search.fit(X_train, y_train)
print(grid_search.best_params_)
```

- **param_grid**: Defines a grid of hyperparameters to test. In this case, n_estimators (number of trees) and max_depth (maximum depth of the trees).
- **GridSearchCV**: Initializes GridSearchCV with 3-fold cross-validation and F1-score as the evaluation metric.
- **fit()**: Fits GridSearchCV to the training data to find the best hyperparameters.
- **best_params_**: Prints the best hyperparameters found during the grid search.

Summary

The code performs the following tasks:

1. Loads and explores the dataset.
2. Preprocesses and scales features.
3. Handles class imbalance using SMOTE.
4. Splits the data into training and testing sets.
5. Trains a RandomForestClassifier.
6. Evaluates model performance.
7. Tunes hyperparameters to optimize the model.

Total code

```
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,
GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report,
confusion_matrix
from imblearn.over_sampling import SMOTE

# Load the dataset
data = pd.read_csv('/content/drive/MyDrive/DATA
SCIENCE/datasets/credit_card.csv')

# Exploratory Data Analysis (EDA)
print("Data Overview:")
print(data.head())  # Display first few rows
print("\nData Summary:")
print(data.describe())  # Summary statistics
print("\nData Info:")
print(data.info())  # Information about data types and missing
values

# Plot distribution of target variable
plt.figure(figsize=(6, 4))
sns.countplot(x='Class', data=data)
plt.title('Distribution of Fraudulent vs Non-Fraudulent
Transactions')
plt.show()

# Data Cleaning
# Handle missing values if any (uncomment if needed)
# data = data.fillna(method='ffill')

# Feature Engineering
# Apply logarithmic transformation to 'Amount' feature
data['Amount'] = np.log1p(data['Amount'])
```

```python
# Dealing with Imbalanced Data
X = data.drop('Class', axis=1)  # Features
y = data['Class']  # Target variable

# Apply SMOTE to balance the data
smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X, y)

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_resampled,
y_resampled, test_size=0.3, random_state=42)

# Model Selection
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Model Validation
y_pred = model.predict(X_test)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Hyperparameter Tuning
param_grid = {
    'n_estimators': [100, 200],  # Number of trees
    'max_depth': [10, 20]  # Maximum depth of trees
}

grid_search = GridSearchCV(estimator=model,
param_grid=param_grid, cv=3, scoring='f1')
grid_search.fit(X_train, y_train)
print("\nBest Parameters from Grid Search:")
print(grid_search.best_params_)

# Optional: Retrain the model with the best parameters if needed
best_model = grid_search.best_estimator_
y_pred_best = best_model.predict(X_test)
print("\nClassification Report for Best Model:")
print(classification_report(y_test, y_pred_best))
```

Output:

```
 Data Overview:
    Time        V1        V2        V3        V4        V5        V6        V7  \
0    0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1    0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2    1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3    1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4    2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9  ...       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]

Data Summary:
                 Time            V1            V2            V3            V4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean    94813.859575  1.759061e-12 -8.251130e-13 -9.654937e-13  8.321385e-13
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01

                 V5            V6            V7            V8            V9  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   1.649999e-13  4.248366e-13 -3.054600e-13  8.777971e-14 -1.179749e-12
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01

       ...           V21           V22           V23           V24  \
count  ...  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   ... -3.405756e-13 -5.723197e-13 -9.725856e-13  1.464150e-12
```

```
std    ...  7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min    ... -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%    ... -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50%    ... -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%    ...  1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max    ...  2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00

                V25           V26           V27           V28         Amount  
\
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  284807.000000
mean  -6.987102e-13 -5.617874e-13  3.332082e-12 -3.518874e-12      88.349619
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01     250.120109
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01       0.000000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02       5.600000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02      22.000000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02      77.165000
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01   25691.160000

               Class
count  284807.000000
mean        0.001727
std         0.041527
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

[8 rows x 31 columns]

Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
```

```
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
None


Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85149
           1       1.00      1.00      1.00     85440

    accuracy                           1.00    170589
   macro avg       1.00      1.00      1.00    170589
weighted avg       1.00      1.00      1.00    170589


Best Parameters from Grid Search:
{'max_depth': 20, 'n_estimators': 200}

Classification Report for Best Model:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85149
           1       1.00      1.00      1.00     85440

    accuracy                           1.00    170589
   macro avg       1.00      1.00      1.00    170589
weighted avg       1.00      1.00      1.00    170589
```

Model deployment plan

```
import joblib

# Save the best model
joblib.dump(best_model, 'credit_fraud_model.pkl')
```

```
['credit_fraud_model.pkl']
```

```python
model=joblib.load('credit_fraud_model.pkl')
```

```python
pred = model.predict([[0, -1.359807134,-
0.072781173,  2.536346738,  1.378155224,  -
0.33832077,  0.462387778,  0.239598554,  0.098697901,  0.3637869
7, 0.090794172,  -0.551599533, -0.617800856,-0.991389847,  -
0.311169354, 1.468176972 ,-0.470400525,0.207971242,  0.02579058,
0.40399296,0.251412098, -0.018306778, 0.277837576,-
0.11047391,  0.066928075,0.128539358 ,-0.189114844
,0.133558377  ,-0.021053053 ,149.62
]])
if pred == 0:
    print("Normal Transcation")
else:
    print("Fraudulent Transcation")
```

```
Normal Transcation
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:465: UserWarning: X
does not have valid feature names, but RandomForestClassifier was fitted with
feature names
  warnings.warn(
```