CREDIT CARD FRAUD DETECTION

Report By: Mithun Kumar and K Guruprasad

1. Introduction:

Credit Card Fraud has become increasingly prevalent with the expansion of online shopping and digital payment systems. The need for robust fraud detection mechanisms has never been more critical. This project focuses on developing a systematic approach to detect credit card fraud by analyzing past transaction data. By employing data analysis techniques, we aim to draw meaningful insight that classifies transactions as fraudulent or legitimate. This analysis not only aids in immediate fraud detection but also enhances the overall performance of fraud detection systems through continuous learning and adaptation.

2. Dataset:

The dataset is taken from Kaggle which contains transactions made by credit cards in September 2013 by European cardholders. It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

3. Objectives:

- To explore and understand the features of the credit card fraud dataset.
- To perform data pre-processing, including handling missing values and outliers.
- To identify significant features influencing credit card fraud through statistical analysis.
- To build predictive models that accurately classify transactions as fraudulent or legitimate.
- To visualize results and present actionable insights to improve fraud detection mechanisms.

4. Methodology

The project will follow a structured approach:

- **Data Collection:** The dataset will be sourced from Kaggle, ensuring it is well-documented and reliable.
- **Exploratory Data Analysis (EDA):** Handle missing data using imputation techniques, Summarize the dataset using descriptive statistics and Create visualizations (e.g., histograms, box plots, correlation heatmaps) to understand feature distributions and interactions.
- **Modelling:** Split the dataset into training and testing subsets. Train various models (e.g., Logistic Regression, Decision Trees, Random Forest, etc.) and assess their performance using metrics such as accuracy, precision, recall, and F1-score. Optimize hyperparameters to enhance model performance.
- **Evaluation and Interpretation:** Compare model performances and select the best-performing model. Interpret results to comprehend the impact of different features on fraud detection.
- **Visualization:** Generate visual representations of findings to support conclusions.
- **Reporting:** Compile analysis, results, and insights into a detailed report.

5. Tools and Technologies

The project utilized the following tools and technologies:

- **Programming Language:** Python
- **Libraries:** Pandas, NumPy, Matplotlib, Seaborn, Scikit-learn
- **IDE:** Jupyter Notebook or any Python-compatible IDE
- **Data Source:** Kaggle dataset (Credit Card Fraud Detection Dataset)

6. Exploratory Data analysis (EDA):

- **Importing Necessary Libraries**

  The first step in the Exploratory Data Analysis (EDA) process typically involves importing essential libraries that facilitate data manipulation, visualization, and analysis.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

- **Loading the Dataset**

  After setting up the libraries, we load the dataset for analysis. The dataset typically contains anonymized features, transaction amounts, and a target class representing fraudulent and non-fraudulent transactions.

  data=pd.read_csv("creditcard.csv")

- **Basic Exploration**

  a) ***data.head()*** – First 5 Rows: This function is used to display the first 5 rows of the dataset. Provides a glimpse of how the dataset is structured.

  b) ***data.tail()*** – Last 5 Rows: Similar to head(), tail() shows the last 5 rows, ensuring there are no unexpected changes at the end of the dataset.

  c) ***data.shape*** – Dataset Shape: To understand how large the dataset is, we use shape() to check the number of rows and columns.

  d) ***data.info()*** – Dataset Information: This method displays a concise summary of the dataset, including data types, non-null counts, and memory usage.

- **Checking for Missing Values**

  We check for missing data to identify potential data quality issues.

  data.isnull().sum()

  Outputs the number of missing values for each column (if any). In a well-cleaned dataset, the result should ideally be 0 for all columns.

- **Descriptive Statistics**

  The describe() function provides essential descriptive statistics for all numeric columns in the dataset.

data.Amount.describe()

Returns count, mean, standard deviation, min, max, and quartiles for each numeric feature. Helpful in identifying the distribution, presence of outliers, and scale of different features like Amount and Time.

- **Identifying Duplicate Rows**
  Duplicate rows could lead to misleading results, especially in fraud detection, where each transaction is supposed to be unique.

  data.duplicated().any()

  This will give the number of duplicate rows in the dataset.

- **Analyzing Class Distribution**
  Since fraud detection is a binary classification problem, understanding the balance between classes is crucial. This can be done using the groupby() and mean() methods.

  data.groupby('Class').mean()

  Class Imbalance: Most datasets in fraud detection are highly imbalanced, with very few fraudulent cases compared to non-fraudulent ones.

7. Data Visualization:

- **Count Plot**
  A count plot visualizes the frequency of categories, making it easy to compare group sizes and identify imbalances in data, such as legitimate versus fraudulent transactions.

  plt.figure(figsize=(5,4))
  sns.countplot(x='Class',data=data)
  plt.title('Class Distribution (0: legit, 1: fraud)')
  plt.show()

- **Correlation Matrix**

  Correlations can provide insights into which features are related to each other and to the target variable (Class). We create a correlation matrix and focus on the features that are strongly correlated with fraud.

  plt.figure(figsize=(10, 6))
  correlation_matrix = data.corr()
  sns.heatmap(correlation_matrix, cmap='coolwarm', annot=False)
  plt.title('Correlation Matrix of the Features')
  plt.show()

- **Bar Plot**

  Visualizing the correlation of specific features with the target (Class) to gain insights into their potential importance for model building.

  plt.figure(figsize=(10,5))
  correlation_with_target =
  correlation_matrix['Class'].sort_values(ascending=False)
  sns.barplot(x=correlation_with_target.index, y=correlation_with_target.values,
  palette='viridis')
  plt.xticks(rotation=90)
  plt.title('Correlation of Features with Fraud Class')
  plt.show()

- **Scatter Plot**

  A scatter plot is created to visualize the relationship between the features 'V1' and 'V2' in the dataset, with points coloured based on their respective classes. The plot, titled "Scatter Plot of V1 vs. V2," helps identify patterns or clusters within the data.

  sns.scatterplot(x='V1', y='V2', hue='Class', data=data)
  plt.title('Scatter Plot of V1 vs. V2')
  plt.show()

- **Histogram**

  A histogram is generated to compare the distribution of the feature 'V1' for two categories: 'Legit' and 'Fraud.' Titled "Histogram of V1," the plot uses different colours for each category, facilitating a visual comparison of their distributions.

  plt.hist(legit['V1'], bins=50, alpha=0.5, label='Legit')
  plt.hist(fraud['V1'], bins=50, alpha=0.5, label='Fraud')
  plt.title('Histogram of V1')
  plt.legend()
  plt.show()

8. Modelling

- **Importing necessary libraries**

  The code imports necessary modules from the ‘sklearn’ library for machine learning tasks. Specifically, it imports functions for splitting data into training and testing sets, a logistic regression model for classification, and a metric to evaluate the model's accuracy.

  from sklearn.model_selection import train_test_split
  from sklearn.linear_model import LogisticRegression
  from sklearn.metrics import accuracy_score

- **Train test split**

  Train-test split is a method used to divide a dataset into two subsets: one for training a model and the other for testing its performance, ensuring that the model's effectiveness is evaluated on unseen data.

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

- **Model Training**

  Model training using logistic regression involves learning from historical transaction data to distinguish between legitimate and fraudulent transactions based on input features, enabling the model to predict the likelihood of fraud in new transactions.

  model = LogisticRegression()
  model.fit(X_train, y_train)

- **Model Prediction**

  This step involves using the trained logistic regression model to generate predictions for the test dataset, `X_test`. The resulting predicted labels are stored in the variable `y_pred`, allowing for comparison with the actual labels to evaluate the model's performance.

  y_pred = model.predict(X_test)

- **Model Evaluation Metrics**

  This section calculates and displays key performance metrics for the model, including accuracy, precision, recall, and F1-score, by comparing the predicted labels (y_pred) with the actual labels (y_test). The results are printed to provide insights into the model's effectiveness in classification tasks.

  from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
  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)
  print(f"Accuracy: {accuracy}")
  print(f"Precision: {precision}")
  print(f"Recall: {recall}")
  print(f"F1-score: {f1}")

- **Logistic Regression Model Evaluation**

  The performance of the logistic regression model is evaluated using several metrics, including accuracy, precision, recall, F1-score, and the area under the ROC curve

(AUC). These results provide a comprehensive assessment of the model's effectiveness in classifying the data.

accuracy_lr = accuracy_score(y_test, y_pred)

precision_lr = precision_score(y_test, y_pred)

recall_lr = recall_score(y_test, y_pred)

f1_lr = f1_score(y_test, y_pred)

auc_lr = roc_auc_score(y_test, y_pred)

- **Support Vector Machine (SVM) Model Evaluation**

  The performance of the Support Vector Machine (SVM) model is assessed using key metrics, including accuracy, precision, recall, F1-score, and the area under the ROC curve (AUC). The results are printed to provide a detailed evaluation of the model's effectiveness in classifying the data.

  accuracy_svm = accuracy_score(y_test, y_pred_svm)

  precision_svm = precision_score(y_test, y_pred_svm)

  recall_svm = recall_score(y_test, y_pred_svm)

  f1_svm = f1_score(y_test, y_pred_svm)

  auc_svm = roc_auc_score(y_test, y_pred_svm)

- **Random Forest Model Evaluartion**

  The performance of the Random Forest model is evaluated using essential metrics such as accuracy, precision, recall, F1-score, and the area under the ROC curve (AUC). The results provide a comprehensive assessment of the model's effectiveness in classifying the data.

  accuracy_rf = accuracy_score(y_test, y_pred_rf)

  precision_rf = precision_score(y_test, y_pred_rf)

  recall_rf = recall_score(y_test, y_pred_rf)

  f1_rf = f1_score(y_test, y_pred_rf)

  auc_rf = roc_auc_score(y_test, y_pred_rf)

- **XGBoost Evaluation**

  The performance of the XGBoost model is assessed using various evaluation metrics to determine its classification effectiveness. The results provide insights into the

model's strengths and weaknesses, guiding potential improvements or adjustments in future analyses.

accuracy_xgb = accuracy_score(y_test, y_pred_xgb)

precision_xgb = precision_score(y_test, y_pred_xgb)

recall_xgb = recall_score(y_test, y_pred_xgb)

f1_xgb = f1_score(y_test, y_pred_xgb)

auc_xgb = roc_auc_score(y_test, y_pred_xgb)

- **Confusion Matrix**

  Confusion matrices are created and displayed for each model, including Logistic Regression, SVM, Random Forest, and XGBoost. The heatmaps, annotated with counts of true and predicted classifications, offer a clear visual representation of each model's performance in distinguishing between classes.

  cm_lr = confusion_matrix(y_test, y_pred)

  cm_svm = confusion_matrix(y_test, y_pred_svm)

  cm_rf = confusion_matrix(y_test, y_pred_rf)

  cm_xgb = confusion_matrix(y_test, y_pred_xgb)

  fig, axes = plt.subplots(2, 2, figsize=(12, 8))

  sns.heatmap(cm_lr, annot=True, fmt='d', cmap='Blues', ax=axes[0, 0])

  axes[0, 0].set_title('Logistic Regression')

  sns.heatmap(cm_svm, annot=True, fmt='d', cmap='Blues', ax=axes[0, 1])

  axes[0, 1].set_title('SVM')

  sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Blues', ax=axes[1, 0])

  axes[1, 0].set_title('Random Forest')

  sns.heatmap(cm_xgb, annot=True, fmt='d', cmap='Blues', ax=axes[1, 1])

  axes[1, 1].set_title('XGBoost')

  plt.tight_layout()

  plt.show()

- **Model Evaluation Metrics Visualiztion**

  This section generates a bar plot to visualize the evaluation metrics—Accuracy, Precision, Recall, F1-score, and AUC—for four different models: Logistic Regression, SVM, Random Forest, and XGBoost. The plot provides a clear

comparison of the models' performance across these key metrics, facilitating an easy assessment of their relative strengths.

```
model_names = ['Logistic Regression', 'SVM', 'Random Forest', 'XGBoost']
metrics = ['Accuracy', 'Precision', 'Recall', 'F1-score', 'AUC']
model_metrics = [
    [accuracy_lr, precision_lr, recall_lr, f1_lr, auc_lr],
    [accuracy_svm, precision_svm, recall_svm, f1_svm, auc_svm],
    [accuracy_rf, precision_rf, recall_rf, f1_rf, auc_rf],
    [accuracy_xgb, precision_xgb, recall_xgb, f1_xgb, auc_xgb]
]
fig, ax = plt.subplots(figsize=(8, 5))
x = np.arange(len(model_names))
width = 0.15
for i, metric in enumerate(metrics):
    ax.bar(x + i * width, [model[i] for model in model_metrics], width, label=metric)
ax.set_ylabel('Score')
ax.set_title('Model Evaluation Metrics')
ax.set_xticks(x + width * (len(metrics) / 2))
ax.set_xticklabels(model_names)
ax.legend(loc='lower right')
plt.tight_layout()
plt.show()
```

Conclusion

This project aims to enhance the understanding of credit card fraud detection by applying data analysis techniques. The insights gained from this analysis will provide valuable information to financial institutions, aiding in the development of more effective fraud prevention strategies, ultimately leading to increased customer trust and safety in digital transactions.