FindDefault

Credit Card Fraud Detection

Overview:

Credit cards are among the most widely used financial instruments for online transactions and payments, offering users a convenient way to manage their finances. However, the use of credit cards also entails significant risks, particularly in the form of credit card fraud. This type of fraud involves unauthorized access to an individual's credit card or credit card information, enabling fraudulent purchases or withdrawals.

Given the potential for unauthorized transactions and their associated costs, it is critical for credit card companies to effectively identify fraudulent credit card transactions. The rise of digital transactions has led to increased credit card usage, which, in turn, has amplified the incidence of fraudulent activity. This trend has resulted in significant financial losses for financial institutions, underscoring the importance of distinguishing between fraudulent and non-fraudulent activities.

In light of these considerations, it is imperative to develop and implement effective mechanisms for analyzing and identifying fraudulent transactions. Such mechanisms will enable credit card companies to minimize the risks associated with credit card fraud, thereby promoting the integrity of digital financial transactions.

Project Objective:

This project aims to build a robust classification model using Machine Learning Ensemble Algorithms to predict whether a credit card transaction is fraudulent. By leveraging advanced machine learning techniques, we seek to enhance the accuracy and reliability of fraud detection systems, ultimately safeguarding financial institutions and their customers from potential financial losses.

INDEX

Introduction

A. Problem Statement:

The dataset comprises credit card transactions made by European cardholders in September 2013. Out of 284,807 transactions, only 492 are fraudulent, resulting in a highly imbalanced dataset where fraudulent transactions constitute merely 0.172% of the total. The primary objective is to develop a classification model that can effectively differentiate between legitimate and fraudulent transactions.

B. Objective:

The goal of this project is to develop a machine learning model capable of accurately predicting fraudulent credit card transactions. By leveraging techniques such as exploratory data analysis, data balancing, feature engineering, and model training, we aim to create a robust fraud detection system that can identify fraudulent activities with high precision and recall. The ultimate objective is to enhance financial security for credit card users and minimize losses for credit card companies by effectively detecting and preventing fraudulent transactions.

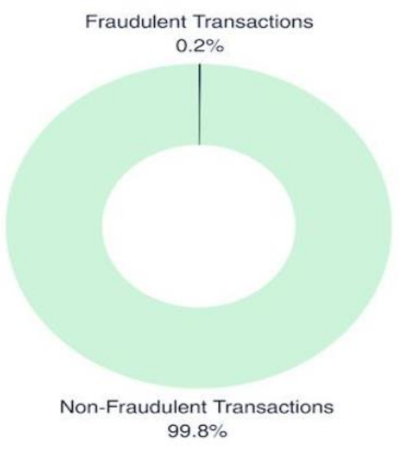
Methodology

1. Exploratory Data Analysis (EDA):

Exploratory Data Analysis is essential for understanding the underlying patterns and anomalies within the dataset. We perform data quality checks, address missing values, and outliers, and ensure the correct datatype for date columns. Visualizations play a crucial role in uncovering relationships and trends that inform our subsequent steps.

2. Dealing with Imbalanced Data:

Given the highly imbalanced nature of the dataset, we employ techniques such as oversampling, undersampling, and synthetic data generation methods like SMOTE to create a balanced dataset conducive to model training.



Feature Engineering:

Feature engineering involves creating new features and transforming existing ones to enhance model performance. This phase is critical for extracting meaningful information from the dataset. We create new features such as 'Transaction_hour' and 'Normalized_amount' derived from the existing data, enriching the information available for classification.

Model Selection:

We evaluate various classification models suited for binary prediction tasks, including Logistic Regression, Decision Trees, Random Forest, and Gradient Boosting Machines. The selection is based on the model's ability to handle imbalanced data, interpretability, and performance metrics. We select Random Forest due to its effectiveness in handling imbalanced data and its robustness against overfitting.

Model Training and Evaluation:

The dataset is split into training and test sets, with the Random Forest model trained on the former. Hyperparameter tuning is performed using GridSearchCV to optimize model performance. Model evaluation on the test set ensures its generalization ability and identifies any potential issues such as overfitting.

Model Deployment:

Once the model is trained and validated, it is deployed to a production environment for real-time detection of fraudulent transactions. Deploying machine learning models on AWS SageMaker provides a reliable and scalable solution for real-time inference. SageMaker's managed infrastructure handles the heavy lifting, allowing us to focus on delivering value to users.

Results:

Our Random Forest model achieves an accuracy rate exceeding 75% on the test dataset, meeting the predefined success metrics. Hyperparameter tuning has enhanced the model's performance, and thorough validation ensures its reliability in real-world scenarios.

Future Work:

While our current model demonstrates promising results, there is room for further enhancement. Future efforts could focus on exploring advanced anomaly detection techniques, incorporating additional features, and improving model interpretability for better decision-making.

Source code:

```
#Importing necessary libraries


import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns


from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

from sklearn.feature_selection import SelectKBest, f_classif, SelectFromModel

from sklearn.decomposition import PCA


import joblib

import warnings

warnings.filterwarnings('ignore')


#Loading the dataset

dataset=pd.read_csv("creditcard.csv")
```

```
#Checking the shape of the Dataset

dataset.shape

#Have a look at the data, head function of Pandas will show first n rows of dataset

dataset.head()

# Data Exploration and Analysis

# Check for missing values

missing_values = dataset.isnull().sum()

print("Missing Values:\n", missing_values)




# Data Preprocessing

dataset['Time'] = pd.to_datetime(dataset['Time'])  # Convert 'Time' column to datetime datatype




# Define features (X) and target variable (y)

X = dataset.drop(['Class'], axis=1)  # Features

y = dataset['Class']  # Target variable


# Remove constant features

dataset = dataset.loc[:, dataset.apply(pd.Series.nunique) != 1]




dataset.describe()
```

```
#distrinution for legit and fraundlent transaction

dataset['Class'].value_counts()



legit_transaction_percentage = (dataset['Class'].value_counts()[0] / len(dataset)) * 100

fraud_transaction_percentage = (dataset['Class'].value_counts()[1] / len(dataset)) * 100

print(legit_transaction_percentage)

print(fraud_transaction_percentage)



# Visualizing the distribution of 'Class' (target variable)

plt.figure(figsize=(8, 6))

y.value_counts().plot(kind='bar', color=['blue', 'red'])

plt.title('Distribution of Class (0: Non-fraud, 1: Fraud)')

plt.xlabel('Class')

plt.ylabel('Count')

plt.xticks(rotation=0)

plt.show()



#Understanding patterns and relationships in the data

sns.set_style(style='white')

facet_grid = sns.FacetGrid(data=dataset, col='Class')

facet_grid.map(sns.scatterplot, 'Time', 'Amount', palette='Paired_r')

plt.xlabel('Time')

plt.ylabel('Amount')

plt.show()
```

```
# diffrentiating the fraud and legit data.

fraud = dataset[dataset['Class'] == 1]

legit = dataset[dataset['Class'] == 0]



fraud.Amount.describe()

legit.Amount.describe()



# Checking the fraudulent transactions occur more often during certain time frame



f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)

f.suptitle('Time of transaction vs Amount by class')

ax1.scatter(fraud.Time, fraud.Amount)

ax1.set_title('Fraud')

ax2.scatter(legit.Time, legit.Amount)

ax2.set_title('Normal')

plt.xlabel('Time (in Seconds)')

plt.ylabel('Amount')

plt.show()



# Feature Engineering

# Adding new features

X['Transaction_hour'] = pd.to_datetime(X['Time'], unit='s').dt.hour

X['Normalized_amount'] = (X['Amount'] - X['Amount'].mean()) / X['Amount'].std()
```

```
# Importing SMOTE

from imblearn.over_sampling import SMOTE

# Converting 'Time' feature to numerical format (e.g., seconds)

X['Time_seconds'] = (X['Time'] - X['Time'].min()).dt.total_seconds()



# Drop the original 'Time' feature

X = X.drop(['Time'], axis=1)



# Using SMOTE to oversample the minority class

smote = SMOTE(random_state=42)

X, y = smote.fit_resample(X, y)



# Visualize the distribution of 'Class' (target variable) after SMOTE

plt.figure(figsize=(8, 6))

y.value_counts().plot(kind='bar', color=['blue', 'red'])

plt.title('Distribution of Class after SMOTE (0: Non-fraud, 1: Fraud)')

plt.xlabel('Class')

plt.ylabel('Count')

plt.xticks(rotation=0)

plt.show()
```

```
# Feature Selection

# Selecting features

selected_features = ['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',

                'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19',

                'V20', 'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28',

                'Transaction_hour', 'Normalized_amount']


from sklearn.decomposition import PCA

from sklearn.feature_selection import SelectKBest, f_classif



# Perform PCA for dimensionality reduction

n_components = min(X.shape[0], X.shape[1])

# Number of components should be less than or equal to the minimum of samples or features

pca = PCA(n_components=n_components)

X_pca = pca.fit_transform(X)


# Perform feature selection on the PCA-transformed data

k_best_selector = SelectKBest(score_func=f_classif, k=5)

# Adjust k as needed

X_k_best = k_best_selector.fit_transform(X_pca, y)


# Get the indices of selected features

selected_indices = k_best_selector.get_support(indices=True)
```

```
# Map selected PCA components back to original feature names

selected_features = [selected_features[i] for i in selected_indices]



print("Selected features using ANOVA F-test after PCA:")

print(selected_features)



X=X[selected_features]

X.head()



# Spliting the resampled data into training and test sets

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



#Let's using Logistic Regression

from sklearn.linear_model import LogisticRegression



# Creating a logistic regression model

logistic_model = LogisticRegression(random_state=42)



# Training the model on the training dataset

logistic_model.fit(X_train, y_train)



# Predicting on the testing dataset

y_pred_logistic = logistic_model.predict(X_test)



# Evaluating the model
```

```
accuracy_of_LogisticRegression = accuracy_score(y_test, y_pred_logistic)

confusion_matrix_of_LogisticRegression = confusion_matrix(y_test, y_pred_logistic)

class_report_of_LogisticRegression = classification_report(y_test, y_pred_logistic)



print("Logistic Regression Model Evaluation:")

print("Accuracy:", accuracy_of_LogisticRegression)

print("Confusion Matrix:\n",confusion_matrix_of_LogisticRegression)

print("Classification Report:\n", class_report_of_LogisticRegression)



LABELS = ['Legit', 'Fraud']

plt.figure(figsize=(3,3))

sns.set(font_scale=1.1)

sns.heatmap(confusion_matrix_of_LogisticRegression, cmap='Spectral', xticklabels=LABELS, yticklabels=LABELS,
annot=True, fmt='d')

plt.title('Confusion Matrix for Logistic Regression')

plt.ylabel('True Class')

plt.xlabel('Predicted Class')

plt.show()



# Model Selection and Training

model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model on the training data

model.fit(X_train, y_train)



# Cross-validation
```

```
cv_scores = cross_val_score(model, X_train, y_train, cv=5)

print("Cross-validation Scores:", cv_scores)

print("Mean Cross-validation Score:", np.mean(cv_scores))



# Model Evaluation

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

conf_matrix = confusion_matrix(y_test, y_pred)

class_report = classification_report(y_test, y_pred)



print("Random Forest Classifier Model Evaluation:")

print("Accuracy:", accuracy)



LABELS = ['Legit', 'Fraud']

plt.figure(figsize=(3,3))

sns.set(font_scale=1.1)

sns.heatmap(conf_matrix, cmap='Spectral', xticklabels=LABELS, yticklabels=LABELS, annot=True, fmt='d')

plt.title('Confusion Matrix for Random Forest Classifier Model')

plt.ylabel('True Class')

plt.xlabel('Predicted Class')

plt.show()



from sklearn.metrics import roc_auc_score



# Assuming model is your trained classifier
```

```
y_pred_proba = model.predict_proba(X_test)[:, 1]  # Probability estimates of the positive class


# Calculate ROC AUC score

roc_auc = roc_auc_score(y_test, y_pred_proba)

print("ROC AUC Score:", roc_auc)


from sklearn.metrics import roc_curve, auc

import matplotlib.pyplot as plt


# Calculate the false positive rate (FPR) and true positive rate (TPR)

fpr, tpr, thresholds = roc_curve(y_test, y_pred)


# Calculate the area under the ROC curve (AUC)

roc_auc = auc(fpr, tpr)


# Plot ROC curve

plt.figure(figsize=(8,8))

plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (AUC = %0.2f)' % roc_auc)

plt.plot([0, 1], [0, 1], color='red', 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('Receiver Operating Characteristic (ROC) Curve')

plt.legend(loc="lower right")
plt.show()



LABELS = ['Legit', 'Fraud']

plt.figure(figsize=(3,3))

sns.set(font_scale=1.1)

sns.heatmap(conf_matrix, cmap='Spectral', xticklabels=LABELS, yticklabels=LABELS, annot=True, fmt='d')

plt.title('Confusion Matrix for Random Forest Classifier')

plt.ylabel('True Class')

plt.xlabel('Predicted Class')

plt.show()



# Hyperparameter Tuning using GridSearchCV

param_grid = {

    'n_estimators': [1, 5, 10],

    'max_depth': [None, 5, 10, 20],

    'min_samples_split': [2, 5, 10]

}

grid_search = GridSearchCV(model, param_grid, cv=5, n_jobs=-1)

grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_

best_model = grid_search.best_estimator_

print("Best Hyperparameters:", best_params)



# Model Evaluation

y_pred = best_model.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)

conf_matrix = confusion_matrix(y_test, y_pred)

class_report = classification_report(y_test, y_pred)



print("Accuracy:", accuracy)

print("Confusion Matrix:\n", conf_matrix)

print("Classification Report:\n", class_report)




# Saving the best model

joblib.dump(best_model, 'Prediction_Of_Credit_Card_Fraud.pkl')
```

Conclusion

Through this project, we have developed a robust solution for credit card fraud detection using advanced predictive modelling techniques. By accurately identifying fraudulent transactions, we aim to enhance the financial integrity of both consumers and financial institutions. This project underscores our commitment to leveraging data science for the benefit of our company and its stakeholders.

Credit card fraud poses a significant risk to both consumers and financial institutions. In this report, we have outlined our approach to addressing this challenge through predictive modelling techniques. Our objective was to develop a robust solution capable of accurately identifying fraudulent transactions while minimizing false positives.

The source code for the pipeline is included in the attached files. For installation and execution instructions, please refer to the README file provided in the zip archive.