FindDefault (Prediction of Credit Card fraud)

Welcome to Prediction of Credit Card fraud notebook!

Greeting and welcome to this comprehesive notebook detailing the analysis and implementation of a credit card fraud detection system.

Dependencies and Imports

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve, auc
```

Load Dataset

Read the dataset for the specific path

(s3://ccfraud01/creditcard.csv) into a Pandas DataFrame (df)

```
# Load Dataset
df = pd.read_csv('s3://ccfraud01/creditcard.csv')
```

Exploratory Data Analysis (EDA)

```
# Display the first few rows of the dataset
df.head

# Display the last few rows of the dataset
df.tail()

# Display dataset information
df.info()

# Check for missing values in each column
df.isnull().sum()

# Display the shape of the dataset (number of rows and columns)
df.shape
```

```
# Check class distribution to determine balance or imbalance
df['Class'].value_counts()
```

The dataset highly unblanced

0 = Normal Transaction

1 = Fraud Transaction

```
# Generate summary statistics for numerical features in the dataset
df.describe()

# Balance the dataset by separating legitimate (Class 0) and
fraudulent (Class 1) transactions
legit = df[df.Class == 0]
fraud = df[df.Class == 1]

# DataFrame containing legitimate (non-fraudulent) transactions
legit

# Display the first few rows of the DataFrame containing fraudulent
transactions
fraud.head()

# Display the shape of the DataFrame containing fraudulent
transactions
print(fraud.shape)

# Display the shape of the DataFrame containing legitimate
transactions
print(legit.shape)

# Generate summary statistics for the 'Amount' column in the DataFrame
containing legitimate transactions
legit.Amount.describe()

# Generate summary statistics for the 'Amount' column in the DataFrame
containing fraudulent transactions
fraud.Amount.describe

# Compare the mean values of each numerical feature for both normal
(Class 0) and fraudulent (Class 1) transactions
df.groupby('Class').mean()
```

Building a Sample Dataset with Balanced Distribution

To ensure the robustness of our model in detecting fraudulent transactions, it's essential to train it on a dataset with a balanced distribution of normal (legitimate) and fraudulent transactions. This helps prevent bias towards the majority class and improves the model's ability to generalize.

The following code snippet demonstrates the construction of a sample dataset with a similar distribution of normal and fraudulent transactions:

```
# Build a sample dataset containing a similar distribution of normal
(legitimate) and fraudulent transactions
legit_sample =  legit.sample(n = 492)
df = pd.concat([legit_sample, fraud],axis = 0)

# Display the shape of the sample dataset
df.shape

# Display the class distribution in the sample dataset
df['Class'].value_counts()

# Calculate the mean of each numerical feature grouped by the 'Class'
column in the sample dataset
df.groupby('Class').mean()
```

Splitting Data into Features and Target Variables

Before training a machine learning model, it's necessary to separate the dataset into input features (independent variables) and the target variable (dependent variable). This allows us to train the model to predict the target variable based on the input features.

The following code snippet demonstrates the process of splitting the dataset into input features (X) and the target variable (Y):

```
# Splitting the data into features and target variables
# X = input features (independent features)
# Y = dependent feature
X = df.drop('Class', axis=1)
Y = df['Class']
print(X)

# Display the shape of the input features (X)
X.shape

# Display the shape of the target variable (Y)
Y.shape
```

Splitting the Dataset into Training and Testing Sets

To evaluate the performance of our machine learning model, it's crucial to split the dataset into training and testing sets. This allows us to train the model on a portion of the data and evaluate its performance on unseen data.

The following code snippet demonstrates the splitting of the dataset into training and testing sets using the train_test_split function:

```
# Split the dataset into training and testing sets using
train_test_split
# X_train: training set of input features
# X_test: testing set of input features
# Y_train: training set of target variable
# Y_test: testing set of target variable
# test_size: proportion of the dataset to include in the testing split
(here, 20%)
# stratify: ensures that the class distribution is preserved in the
splits
# random_state: seed for random number generation for reproducibility

X_train, X_test, Y_train , Y_test = train_test_split(X, Y, test_size =
0.2,stratify = Y,  random_state=2)

# Print the shapes of the input features in the original dataset,
training set, and testing set
print( X.shape, X_train.shape, X_test.shape)
```

Logistic Regression Model Initialization and Training

Logistic Regression is a widely used classification algorithm that predicts the probability of a binary outcome. In our case, it's employed to predict whether a transaction is fraudulent or not based on input features.

The following code snippet demonstrates the initialization and training of a Logistic Regression model:

```
# Logistic Regression model initialization and training

# Initialize the Logistic Regression model
model = LogisticRegression()

# Fit the model on the training data
model.fit(X_train, Y_train)
```

Model Evaluation

Model evaluation is crucial to assess the performance of the trained model on unseen data. Accuracy score is one of the commonly used metrics to evaluate classification models, which measures the proportion of correctly classified instances.

The following code snippet demonstrates the calculation of the accuracy score on the training data:

```
# Model Evaluation
# Calculate accuracy score on the training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

```
# Print the accuracy score on the training data
print('Accuracy on Training data: ', training_data_accuracy)

# Calculate accuracy score on the test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)

# Print the accuracy score on the testing data
print('Accuracy on testing data :' , test_data_accuracy)
```

Confusion Matrix

The confusion matrix is a performance measurement for classification problems where output can be two or more classes. It is a table with four different combinations of predicted and actual values: true positive (TP), true negative (TN), false positive (FP), and false negative (FN).

The following confusion matrix represents the performance of the model on the test data:

```
# Calculate confusion matrix
conf_matrix = confusion_matrix(Y_test, X_test_prediction)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```

Model Evaluation: Precision, Recall, and F1-score

In addition to accuracy, precision, recall, and F1-score are commonly used metrics to evaluate classification models. Precision measures the proportion of true positive predictions among all positive predictions. Recall (also known as sensitivity) measures the proportion of true positive predictions among all actual positive instances. F1-score is the harmonic mean of precision and recall, providing a balance between the two metrics.

The following code snippet calculates precision, recall, and F1-score on the test data:

```
# Calculate precision, recall, and F1-score
print(classification_report(Y_test, X_test_prediction))
```

Receiver Operating Characteristic (ROC) Curve and AUC

The ROC curve is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. It plots the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The area under the ROC curve (AUC) represents the performance of the classifier.

The following ROC curve visualizes the performance of the model on the test data:

```
# Calculate ROC curve and AUC
fpr, tpr, thresholds = roc_curve(Y_test, model.predict_proba(X_test)
[:,1])
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area =
%0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, 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()
```

Conclusion

In this project, we developed a machine learning model to detect fraudulent transactions in credit card data. We began by exploring the dataset and understanding its characteristics through exploratory data analysis. We then preprocessed the data, balanced the class distribution, and split it into training and testing sets. A Logistic Regression model was trained on the training data and evaluated using various metrics including accuracy, precision, recall, and F1-score.

Additionally, we analyzed the Receiver Operating Characteristic (ROC) curve and calculated the Area Under the Curve (AUC) to assess the model's performance in distinguishing between positive and negative classes. The ROC curve indicated good performance with an AUC score of 0.92.

Overall, the developed model demonstrates promising results in detecting fraudulent transactions, which can aid credit card companies in preventing financial losses and ensuring customer security.

Thank You

Riddhi Sharma