ML Lab-3

Praneesh R V

CB.SC.U4CYS23036

Installing Creditcardfraud.csv

```
! pip install kaggle
! mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle/
! chmod 600 ~/.kaggle/kaggle.json
!kaggle datasets download -d mlg-ulb/creditcardfraud
!unzip creditcardfraud.zip
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (1.6.17)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.17.0)
Requirement already satisfied: certifi>=2023.7.22 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2024.12.14)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from kaggle) (4.67.1)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle) (8.0.4)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.2.3)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle) (6.2.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->kaggle) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.10)
cp: cannot stat 'kaggle.json': No such file or directory
chmod: cannot access '/root/.kaggle/kaggle.json': No such file or directory
Dataset URL: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud
License(s): DbCL-1.0
Downloading creditcardfraud.zip to /content
 92% 61.0M/66.0M [00:00<00:00, 88.3MB/s]
100% 66.0M/66.0M [00:00<00:00, 90.8MB/s]
Archive:  creditcardfraud.zip
  inflating: creditcard.csv
```

splitting the data into training data and testing data

```
[2]
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

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

cc_train, cc_test = train_test_split(cc, test_size=0.2, random_state=42)

# Display dataset sizes
print("Original dataset size:", cc.shape)
print("Iris train dataset size:", cc_train.shape)
print("Iris test dataset size:", cc_test.shape)

# Resetting index for train and test sets
cc_train = cc_train.reset_index(drop=True)
cc_test = cc_test.reset_index(drop=True)
```

```
Original dataset size: (284807, 31)
Iris train dataset size: (227845, 31)
Iris test dataset size: (56962, 31)
```

The code applies a Decision Tree Classifier to detect anomalies, such as fraudulent transactions, in a dataset. It prepares the data by splitting it into training and testing subsets, separating the features from the target column (Class). The classifier is initialized, trained on the training data, and used to make predictions on the test data.

The model's performance is evaluated using metrics like accuracy, precision, recall, and F1-score, presented in a classification report. Additionally, the decision tree is visualized to illustrate how features are used for classification, with node colors and labels providing insights into the model's decision-making process. This approach effectively combines training, evaluation, and interpretability for anomaly detection.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
from sklearn import tree

X_train = cc_train.drop('Class', axis=1)  # Replace 'Class' with the actual target column name
y_train = cc_train['Class']
X_test = cc_test.drop('Class', axis=1)
y_test = cc_test['Class']

# Initialize the Decision Tree Classifier
dtc = DecisionTreeClassifier(random_state=42)

# Train the model
dtc.fit(X_train, y_train)

# Make predictions
y_pred = dtc.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Display results
print(f"Accuracy: {accuracy:.2f}")
print("\nClassification Report:")
print(class_report)

# Optional: Visualize the Decision Tree
plt.figure(figsize=(20, 10))
tree.plot_tree(dtc, feature_names=X_train.columns, class_names=['0', '1'], filled=True)
plt.show()
```

Output:

Accuracy: Displays the overall percentage of correct predictions.

Classification Report: Lists precision, recall, and F1-score for fraud (Class 1) and non-fraud (Class 0) transactions.

```
Accuracy: 1.00

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.70      0.80      0.74        98

    accuracy                           1.00     56962
   macro avg       0.85      0.90      0.87     56962
weighted avg       1.00      1.00      1.00     56962
```

Tree Visualization: Shows the structure of the decision tree, illustrating how features are used to classify transactions.