Decision Tree Classification using Scikit-Learn

Project Description

This project demonstrates the use of a **Decision Tree Classifier** for a supervised machine learning classification task. It highlights the process of loading and visualizing data, training the model, evaluating its performance, and interpreting the results through metrics and visualizations.

Why Decision Trees?

Decision trees are intuitive and interpretable models commonly used in classification tasks. They are robust to both numerical and categorical data and can handle missing values effectively. Decision trees are ideal for:

- Gaining insights into feature importance.
- Explaining predictions in layman's terms.
- Quick prototyping in classification tasks.

Prerequisites

Required Libraries

- **Python 3.7 or later**
- `pandas` : For data manipulation and analysis.
- `numpy` : For numerical computations.
- `scikit-learn` : For machine learning algorithms and evaluation metrics.
- `matplotlib` : For data visualization.
- `seaborn` : For advanced visualizations.

Installation

Run the following command to install the necessary libraries:

```
pip install pandas numpy scikit-learn matplotlib seaborn
```

Files Included

- `your_dataset.csv` : A placeholder dataset (replace with your actual dataset file).
- Python code for the Decision Tree Classifier.

Code Description

Steps in the Code

1. Dataset Loading:

```
import pandas as pd
data = pd.read_csv('your_dataset.csv')
```

The dataset is loaded using pandas. Replace `'your_dataset.csv'` with the actual path to your dataset.

2. Splitting the Data:

```
from sklearn.model_selection import train_test_split

X = data.drop(columns=['target'])  # Replace 'target' with the actual target column name
y = data['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
```

The dataset is split into training (80%) and testing (20%) sets.

3. Model Initialization and Training:

```
from sklearn.tree import DecisionTreeClassifier

dt_model = DecisionTreeClassifier(max_depth=5, random_state=42)
dt_model.fit(X_train, y_train)
```

- A `DecisionTreeClassifier` is initialized with a maximum depth of **5** to prevent overfitting and enhance interpretability.
- The model is trained on `X_train` (features) and `y_train` (target labels).

4. Model Predictions:

```
y_pred = dt_model.predict(X_test)
```

Predictions are made on the test dataset.

5. Evaluation Metrics:

Confusion Matrix:

```
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

Displays the counts of **True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN).**

Classification Report:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

Provides metrics such as **precision, recall, F1-score, and support** for each class.

Accuracy Score:

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy Score: {accuracy:.2f}")
```

Outputs the **overall accuracy** of the model.

6. Confusion Matrix Visualization:

```
 import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=["Class 0", "Cl
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

A heatmap representation of the confusion matrix is generated for easier interpretation.

Outputs

Metrics:

- **Confusion Matrix**
- **Classification Report (Precision, Recall, F1-Score)**
- **Accuracy Score**

Visualization:

- **Heatmap of the Confusion Matrix**

Example Output

For a hypothetical dataset, the output might look like:

Confusion Matrix:

```
[[50  2]
 [ 5 43]]
```

Classification Report:

```
              precision    recall  f1-score   support

           0       0.91      0.96      0.93        52
           1       0.96      0.90      0.93        48

    accuracy                           0.93       100
   macro avg       0.93      0.93      0.93       100
weighted avg       0.93      0.93      0.93       100
```

Accuracy Score:

```
Accuracy Score: 0.93
```

Use Cases

This project is useful for:

- **Binary or Multi-Class Classification Tasks**.
- **Visualizing and Understanding Model Performance**.
- **Quick Prototyping in Classification Problems**.

Future Enhancements

- **Hyperparameter tuning** for improved accuracy.
- **Cross-validation** to ensure model generalization.
- **Deployment of the model** for real-time predictions.
