Week 2: Decision Tree

Decision Trees are a widely used and interpretable algorithm for classifying or regressing data based on rules, mimicking human decision-making. The Iris dataset is a classic benchmark in machine learning, featuring measurements of four iris flower features (sepal length, sepal width, petal length, and petal width) across three species (Setosa, Versicolor, and Virginica). It's commonly used for educational purposes and algorithm evaluation.

```
import pandas as pd
import graphviz
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.metrics import classification_report
```

The above lines import the necessary libraries and modules, including **pandas** for data handling, **graphviz** for visualizing decision trees, **LabelEncoder** for encoding target classes, **train_test_split** for splitting the data into training and testing sets, **DecisionTreeClassifier** for creating a decision tree model, **export_graphviz** for exporting the decision tree to a DOT file, and **classification_report** for evaluating the model's performance.

```
df = pd.read_csv(r'C:\Fall 23\ENPM 808L\Week2 report\IrisNew.csv')
```

This line reads the Iris dataset from a CSV file located at the specified path and stores it in a Pandas DataFrame called df.

```
new_target = LabelEncoder()
df['target'] = new_target.fit_transform(df['Class'])
```

This code creates a LabelEncoder object and uses it to encode the 'Class' column in the dataset into numerical labels. These numerical labels are stored in a new column named 'target' in the DataFrame.

```
inputs = df.drop(['Class', 'target'], axis='columns')
inputs.columns = ['Sepal Length(cm)', 'Sepal Width(cm)', 'Petal Length(cm)', 'Petal Width(cm)']
```

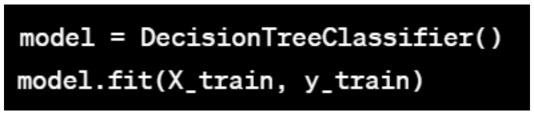
This code renames the columns in the 'inputs' DataFrame for better readability. It drops the 'Class' and 'target' columns and assigns more descriptive names to the columns representing the input features.

```
target = df['target']
```

This line creates a 'target' variable that stores the encoded target labels.

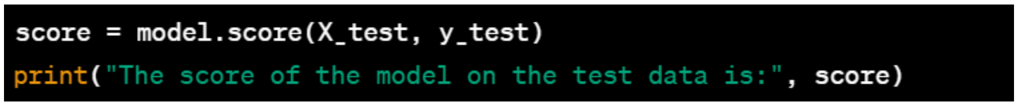
```
X_train, X_test, y_train, y_test = train_test_split(inputs, target, test_size=0.25, random_state=42)
```

This code splits the dataset into training and testing sets using the train_test_split function. It uses 75% of the data for training (X_train and y_train) and 25% for testing (X_test and y_test). The random_state parameter ensures reproducibility.



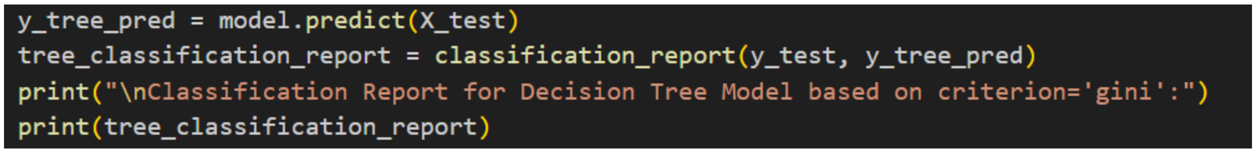
```
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
```

Here, a Decision Tree classifier model is created and trained using the training data.



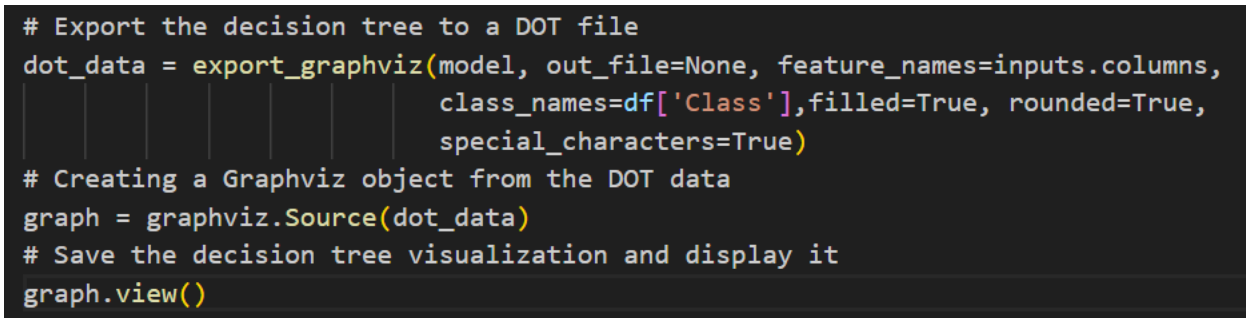
```
score = model.score(X_test, y_test)
print("The score of the model on the test data is:", score)
```

The model's accuracy score on the test data is calculated and printed, providing a measure of how well the model performed.



```
y_tree_pred = model.predict(X_test)
tree_classification_report = classification_report(y_test, y_tree_pred)
print("\nClassification Report for Decision Tree Model based on criterion='gini':")
print(tree_classification_report)
```

The code makes predictions on the test data using the trained decision tree model. It then generates a classification report that includes metrics such as precision, recall, and F1-score for each class and prints the report. This report provides a more detailed evaluation of the model's performance.



```
# Export the decision tree to a DOT file
dot_data = export_graphviz(model, out_file=None, feature_names=inputs.columns,
                           class_names=df['Class'],filled=True, rounded=True,
                           special_characters=True)
# Creating a Graphviz object from the DOT data
graph = graphviz.Source(dot_data)
# Save the decision tree visualization and display it
graph.view()
```

This part of the code exports the decision tree as a DOT file. The export_graphviz function takes the decision tree model, the feature names, class names, and various visualization settings.

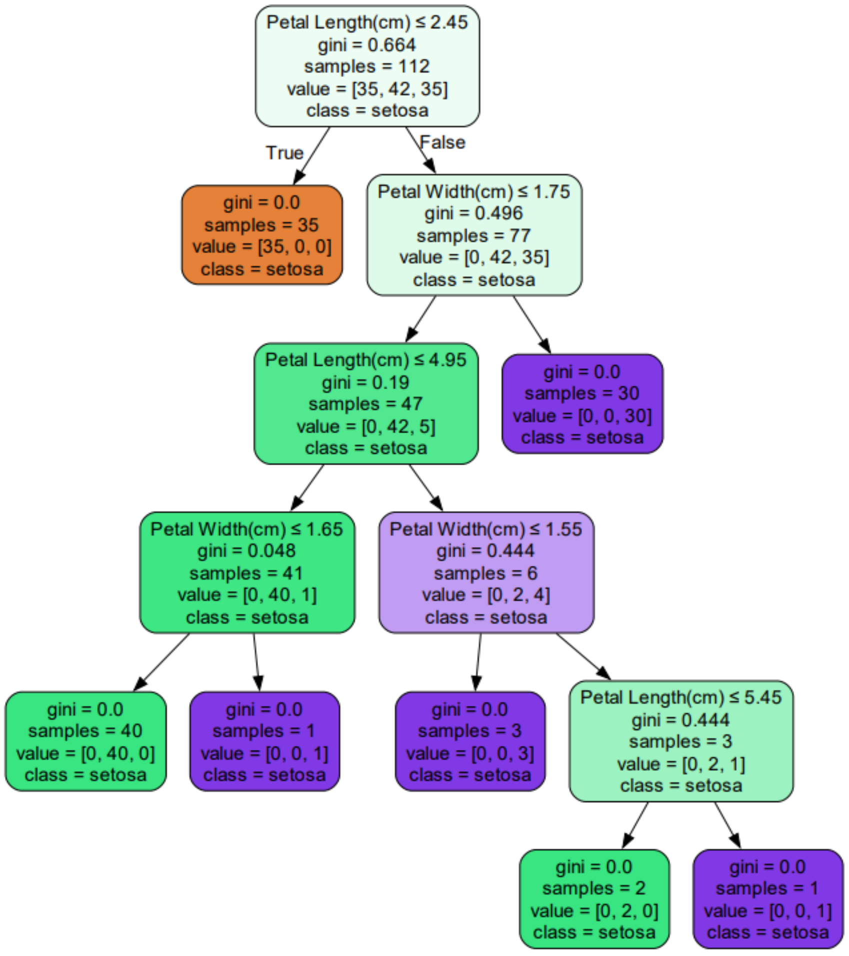
It creates a Graphviz object from the DOT data, which allows you to render and visualize the decision tree.

Finally, this code saves the decision tree visualization to a file and displays it. You'll find the generated visualization in the current working directory, and it will be saved with a ".pdf" extension, which you can open and view.

The output:

```
The score of the model on the test data is: 0.9736842105263158

Classification Report for Decision Tree Model based on criterion='gini':
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       1.00      0.88      0.93         8
           2       0.94      1.00      0.97        15

    accuracy                           0.97        38
   macro avg       0.98      0.96      0.97        38
weighted avg       0.98      0.97      0.97        38
```

Generated Decision Tree:



Result Analysis:
Accuracy (0.9737): The model's accuracy is approximately 97.37%, meaning it correctly predicted the class labels for nearly 97.37% of the samples in the test data.

Precision: Precision measures the proportion of true positive predictions out of all positive predictions made by the model.

For Class 0: Precision is 1.00, indicating that all the predictions for this class are accurate.

For Class 1: Precision is 1.00, meaning all positive predictions for this class are correct.

For Class 2: Precision is 0.94, indicating that 94% of positive predictions for this class are accurate.

Recall: Recall measures the proportion of true positive predictions out of all actual positive instances.

For Class 0: Recall is 1.00, meaning the model correctly identifies all actual instances of this class.

For Class 1: Recall is 0.88, indicating that 88% of actual instances of this class are correctly identified by the model.

For Class 2: Recall is 1.00, meaning the model correctly identifies all actual instances of this class.

F1-Score: The F1-score is the harmonic mean of precision and recall. It provides a balance between precision and recall.

For Class 0: F1-score is 1.00, indicating a perfect balance between precision and recall.

For Class 1: F1-score is 0.93, representing a good balance between precision and recall.

For Class 2: F1-score is 0.97, indicating a good balance between precision and recall.

Overall, the analysis of this classification report suggests that the Decision Tree model performs very well, with high precision, recall, and F1-scores for most classes, and an impressive overall accuracy of approximately 97.37%. It appears to be a robust model for classifying the Iris dataset.

Criterion parameter:

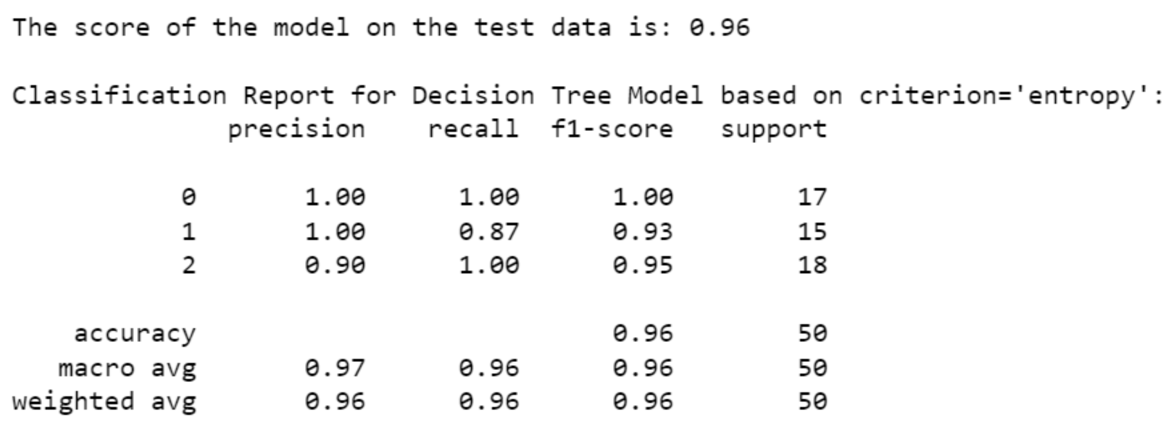
dtree_part1.py - Using Gini Index as Criterion

In the dtree_part1.py file, we create a DecisionTreeClassifier with the default criterion, which is 'Gini index. When criterion='gini', the Gini impurity criterion is used. The Gini impurity measures the probability of misclassifying a randomly chosen element if it were labeled randomly according to the distribution of classes in the node. A lower Gini impurity indicates a purer (homogeneous) node. In simple terms, 'Gini index' aims to make the class distribution within a node as homogeneous as possible.

dtree_part2.py - Using Entropy as Criterion

In the dtree_part2.py file, we create a DecisionTreeClassifier with the criterion explicitly set to 'entropy.' 'Entropy' is an alternative criterion used to measure node impurity. Instead of 'Gini index,' it uses 'information gain' to make decisions. The decision tree algorithm seeks to maximize information gain, which essentially means finding the split that results in the most informative separation of data.

The output while using Entropy as Criterion:

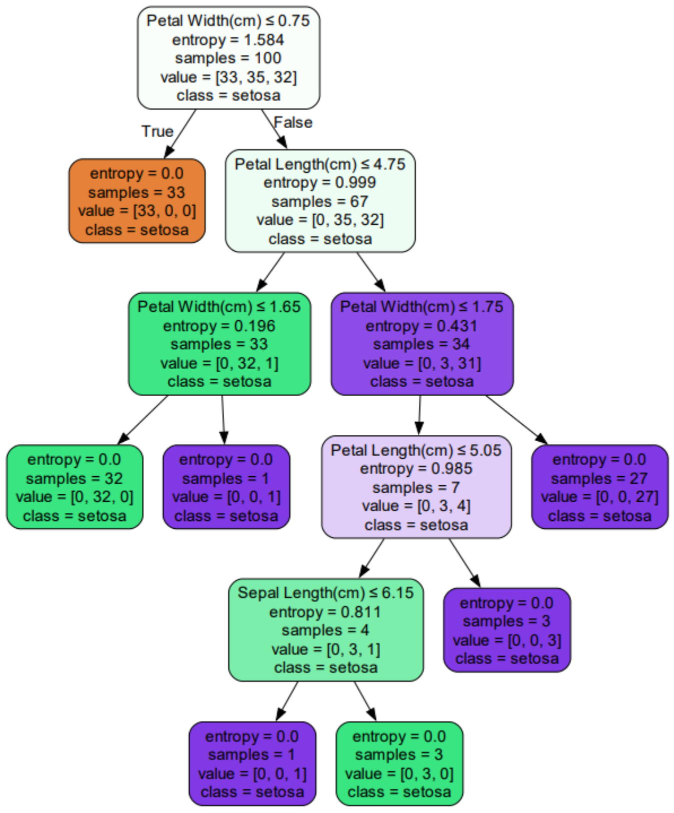


```
The score of the model on the test data is: 0.96

Classification Report for Decision Tree Model based on criterion='entropy':
             precision    recall  f1-score   support

           0       1.00      1.00      1.00        17
           1       1.00      0.87      0.93        15
           2       0.90      1.00      0.95        18

    accuracy                           0.96        50
   macro avg       0.97      0.96      0.96        50
weighted avg       0.96      0.96      0.96        50
```

Accuracy: The model with 'entropy' as the criterion achieves an accuracy of 96% on the test data.

Precision, Recall, and F1-Score: The classification report provides precision, recall, and F1-score for each class. In this case, it demonstrates high performance for all classes, with F1-scores ranging from 0.93 to 1.00, indicating a good balance between precision and recall. Notably, Class 0 has perfect precision and recall.

Tree generated:



Result comparison:

- The accuracy of the 'gini' criterion model is marginally higher than that of the 'entropy' criterion model.
- Both models perform exceptionally well, with high precision and recall, indicating strong predictive capabilities.
- The choice between 'entropy' and 'gini' as the criterion depends on the specific problem and dataset characteristics. In this case, either criterion appears to be effective, but 'gini' leads to a slightly higher accuracy.
- Keep in mind that these are relatively small datasets, and the difference in performance might be more pronounced in larger and more complex datasets.