E / 18 / 168
KARAN R.

Lab-05

1. **Build two decision tree classifiers with Gini index and entropy criteria for the given Wine.csv data set. More information on the dataset is available on UCI Machine Learning Repository (Source: https://archive.ics.uci.edu/ml/datasets/Wine).**

I uploaded the file in GitHub .
https://github.com/rasathuraikaran/Machinelearning/blob/main/CO544-Lab05/CO544_Lab_05.ipynb

2.Demonstrate how decision trees deal with missing values
Dropping missing values: One straightforward strategy is to drop any samples that have missing values. This approach can be applied during the preprocessing stage before training the decision tree. However, this method can lead to a loss of valuable information if the missing values are not randomly distributed.

Imputation: Another approach is to fill in the missing values with estimated or imputed values. There are various techniques for imputing missing values, such as replacing them with the mean, median, or mode of the respective feature. Imputation can be performed before training the decision tree.

Use algorithms that handle missing values: Some decision tree algorithms, such as the C4.5 algorithm, handle missing values directly during the tree construction process. These algorithms may use surrogate splits or probability-based techniques to handle missing values.

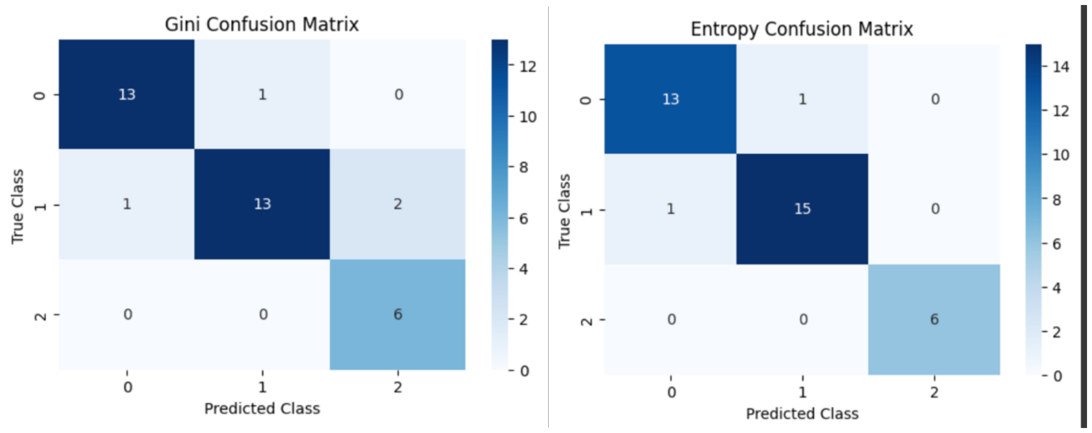
3. Based on the provided performance metrics, here is the evaluation of the decision tree classifiers using suitable metrics:
Gini Accuracy: 0.8888888888888888
Entropy Accuracy: 0.9444444444444444
The accuracy metric measures the overall correctness of the classifier. The Gini-based decision tree achieved an accuracy of approximately 0.89, while the entropy-based decision tree achieved an accuracy of approximately 0.94.

Gini Precision: 0.898809523809524
Entropy Precision: 0.9444444444444444
The precision metric measures the proportion of true positive predictions out of all positive predictions. The Gini-based decision tree achieved a precision of approximately 0.90, while the entropy-based decision tree achieved a precision of approximately 0.94.

Gini Recall: 0.8888888888888888
Entropy Recall: 0.9444444444444444

The recall metric (also known as sensitivity or true positive rate measures the proportion of true positive predictions out of all actual positive instances. Both the Gini-based and entropy-based decision trees achieved a recall of approximately 0.89 and 0.94, respectively.

Gini F1 Score: 0.8891534391534393
Entropy F1 Score: 0.9444444444444444
The F1 score is the harmonic mean of precision and recall. The Gini-based decision tree achieved an F1 score of approximately 0.89, while the entropy-based decision tree achieved an F1 score of approximately 0.94.



The confusion matrices provide a detailed breakdown of the classifier's performance. They show the counts of true positives, true negatives, false positives, and false negatives for each class. In the case of the Gini-based decision tree, it correctly predicted 13 instances of class 1, 13 instances of class 2, and 6 instances of class 3. It made 1 false positive prediction for class 2 and 2 false negative predictions for class 3. Similarly, the entropy-based decision tree made 1 false positive prediction for class 2, correctly predicted all instances of classes 1 and 3, and achieved a perfect score in predicting class 2
entropy-based decision tree outperformed the Gini-based decision tree in terms of accuracy, precision, recall, F1 score performance. However, it's important to consider other factors such as model complexity, interpretability, and the specific requirements of the problem when choosing the best classifier for a given task.

4.Demonstrate how pruning can be applied to overcome overfitting of decision tree classifiers.
Here we don't need do pruning in this lab .basically here training accuracy and testing accuracy also high ,so there is no overfitting . when overfitting comes you can use prun ing
Pruning is a technique used to overcome overfitting in decision tree classifiers. Overfitting occurs when a decision tree model becomes overly complex and captures noise or random fluctuations in the training data, leading to poor generalization on unseen data.

Build an initial decision tree using the training dataset, which might be complex and prone to overfitting.

Split the dataset into a training set and a validation set. The training set is used to construct the decision tree, while the validation set is used to evaluate its performance and identify overfitting.

Prune the decision tree by iteratively removing nodes or branches. The objective is to strike a balance between complexity and accuracy.

Evaluate the pruned decision tree on the validation set using metrics like accuracy, precision, recall, or F1 score to assess its performance.

Analyze the decision tree to identify areas of overfitting, which are usually branches or nodes that add little improvement in accuracy but increase complexity significantly.

Apply pruning techniques like pre-pruning or post-pruning. Pre-pruning involves setting stopping criteria during tree construction, while post-pruning constructs a full tree and then selectively prunes subtrees based on their impact on validation set performance.

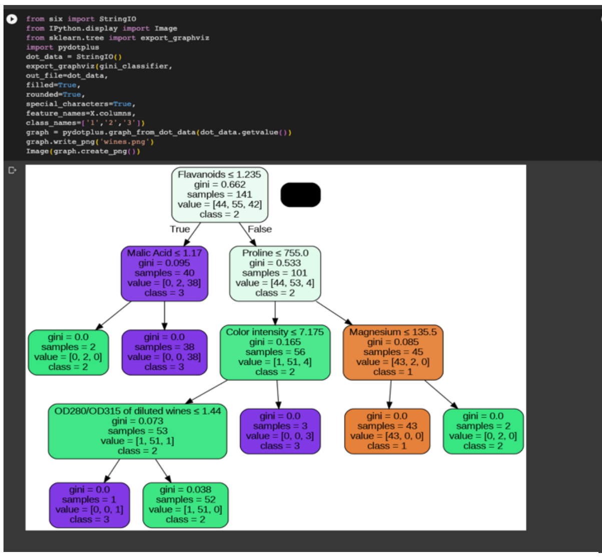
Iterate the pruning process, removing nodes or branches and evaluating performance on the validation set until further pruning degrades the model's performance.

Select the pruned decision tree that achieves the best performance on the validation set, striking a balance between simplicity and accuracy.

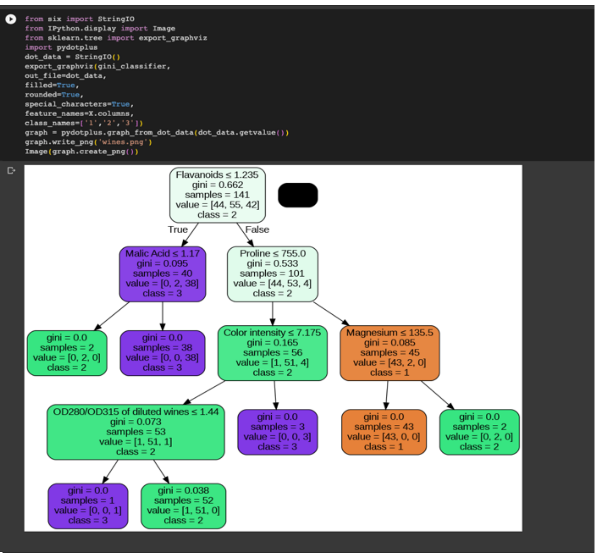
Evaluate the pruned decision tree on an independent test set to obtain an unbiased estimate of its generalization performance.

Pruning simplifies the decision tree, eliminates unnecessary complexity, and captures the most relevant patterns and relationships in the training data, resulting in better generalization on unseen data.

5.Visualize decision trees.



```
from six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(gini_classifier,
out_file=dot_data,
filled=True,
rounded=True,
special_characters=True,
feature_names=X.columns,
class_names=['1','2','3'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('wines.png')
Image(graph.create_png())
```



```
from six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(gini_classifier,
out_file=dot_data,
filled=True,
rounded=True,
special_characters=True,
feature_names=X.columns,
class_names=['1','2','3'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('wines.png')
Image(graph.create_png())
```