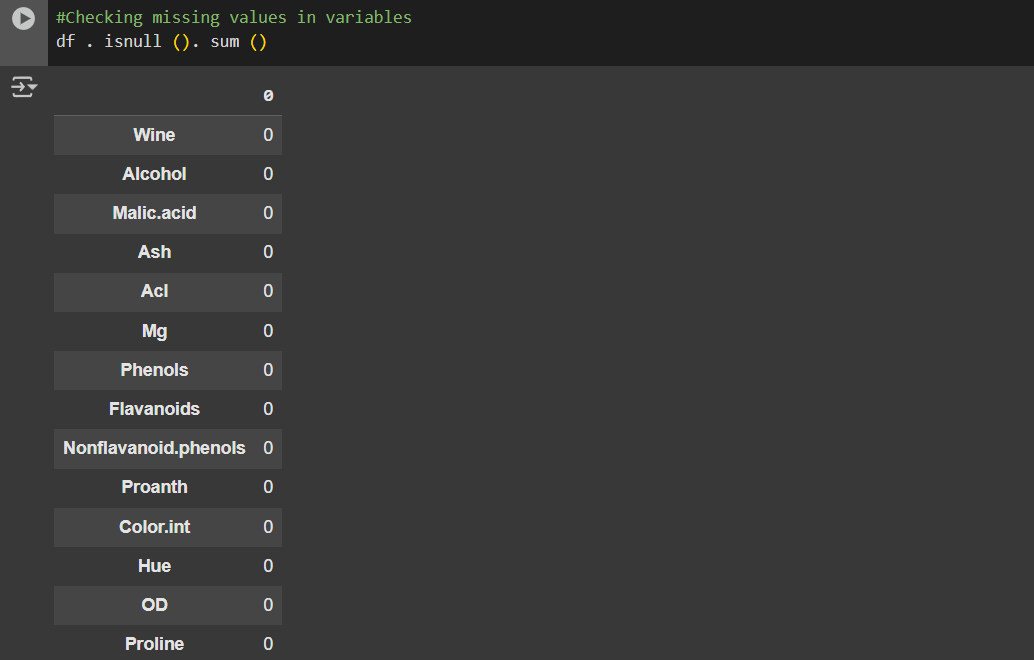
**CO544 – Machine Learning and Data Mining**

**Lab 03 - Decision Trees and k-Nearest Neighbors Classification**

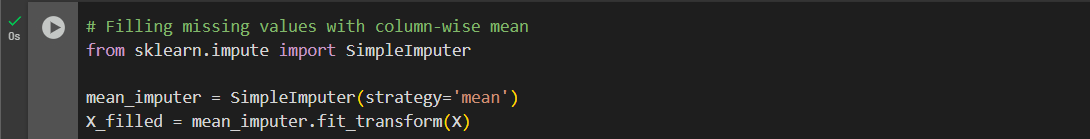
**E/20/212 Kumarasinghe R.M.S.H.**

1. Task 1: Build two decision tree classifiers with Gini index and entropy criteria for the given Wine.csv dataset.
2. Demonstrate how decision trees deal with missing values.

There are no attributes other than numbers in the Wine dataset. The dataset was checked for missing values using isnull().sum() and verified there were no values missing.

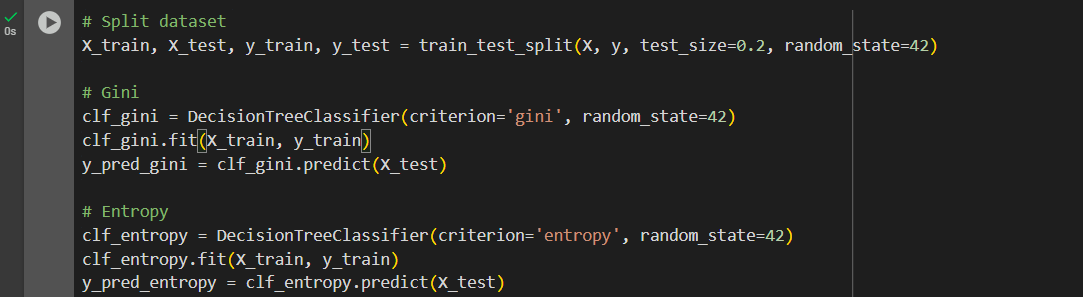


To simulate the handling of missing data in decision trees, mean imputation method is used via SimpleImputer(strategy='mean'). As scikit-learn's DecisionTreeClassifier does not natively support missing values, this preprocessing step is essential before model training.

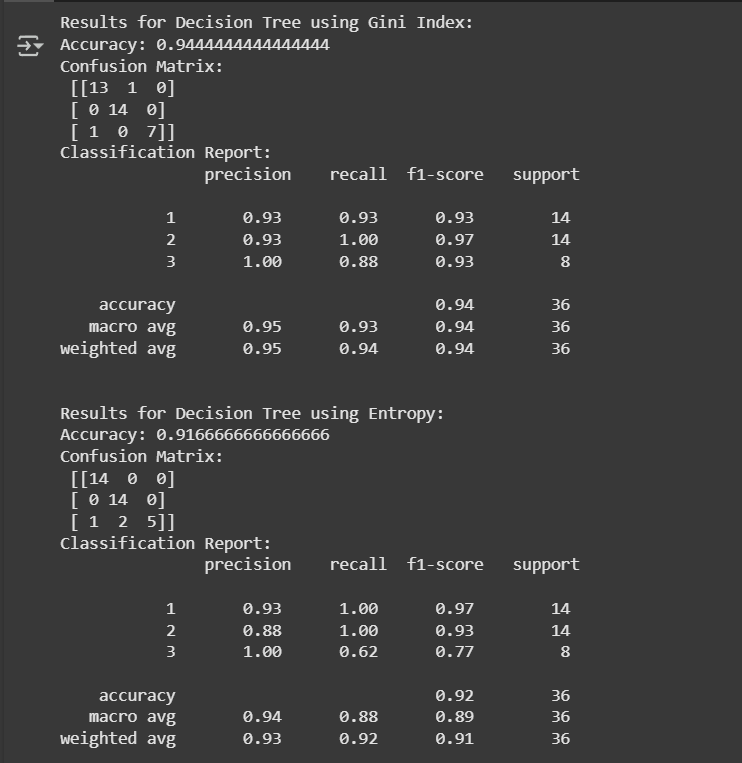


1. Evaluate the classifiers with suitable performance metrics.

Two decision tree models were developed, one using the Gini index and the other using entropy for splitting. The dataset was divided into 80% for training and 20% for testing.



Results for model evaluations:

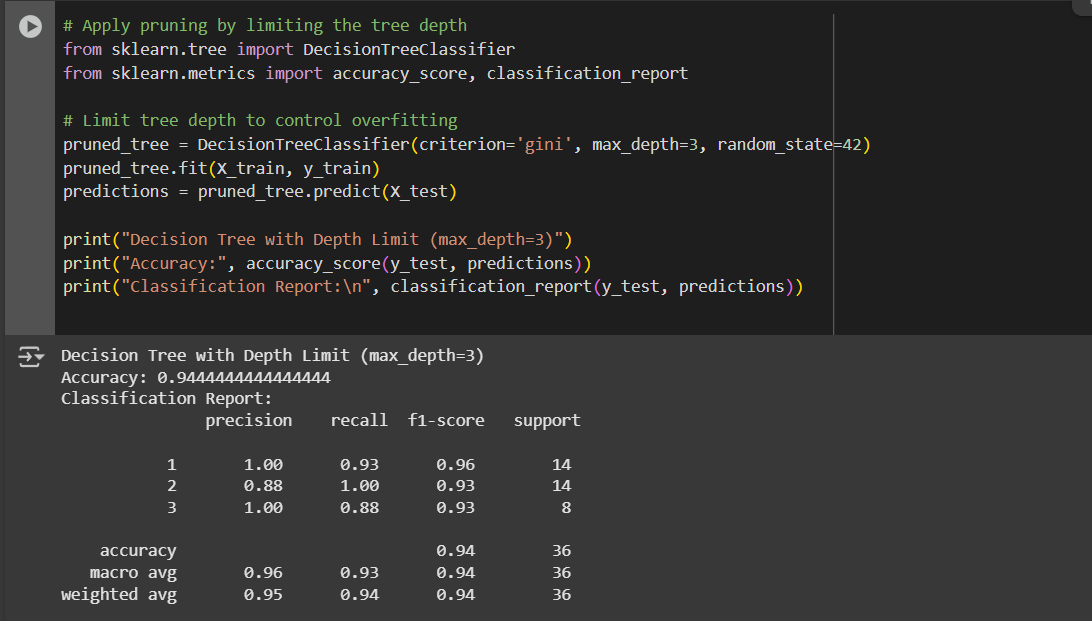


|  |  |
| --- | --- |
|  | Accuracy |
| Decision tree using Gini index | 0.9444444444444444 |
| Decision tree using Entropy | 0.9166666666666666 |

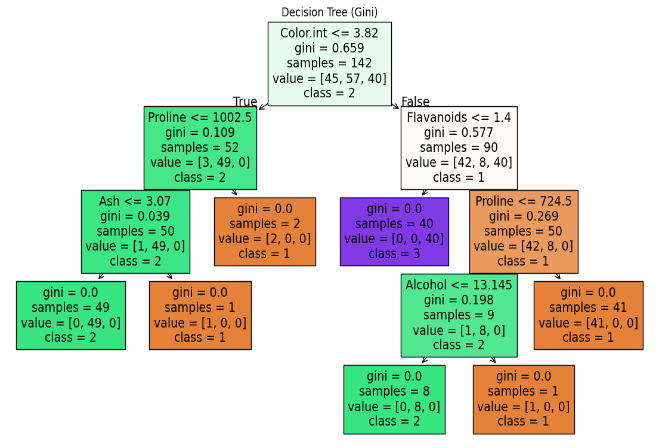
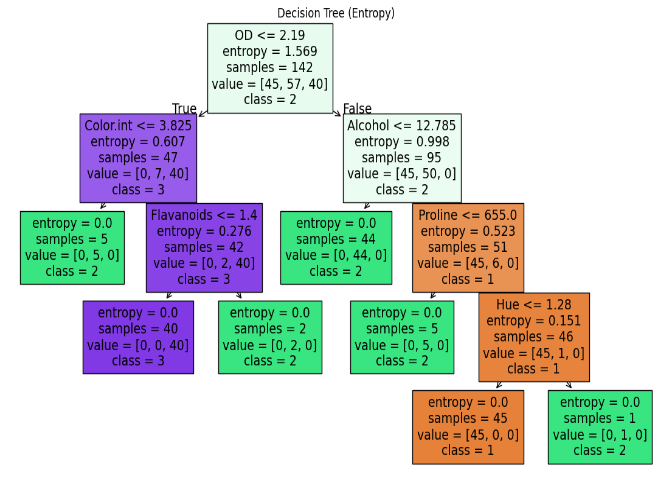
While the overall accuracy of both classifiers was comparable, there were minor variations in precision and recall values across individual classes.

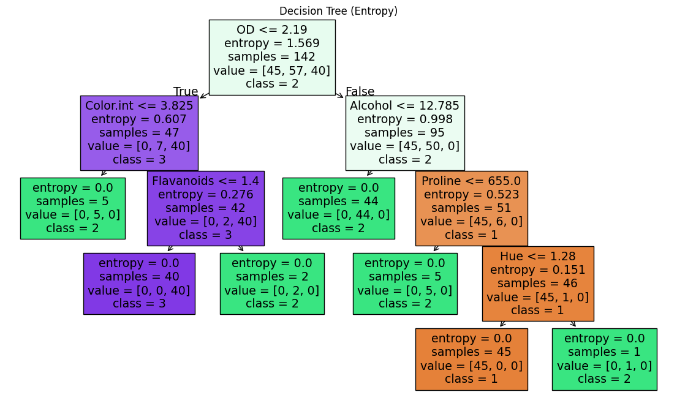
1. Demonstrate how pruning can be applied to overcome overfitting of decision tree classifiers.

In scikit-learn, decision tree optimization is achieved through pre-pruning techniques. One commonly used approach is restricting the maximum tree depth. To address overfitting, we applied pre-pruning by setting the max\_depth parameter to 3, effectively limiting the complexity of the mode



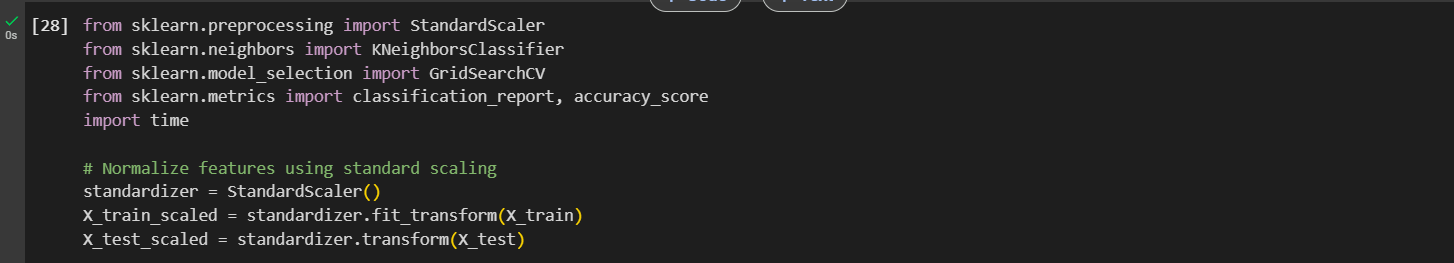
1. Visualize decision trees.



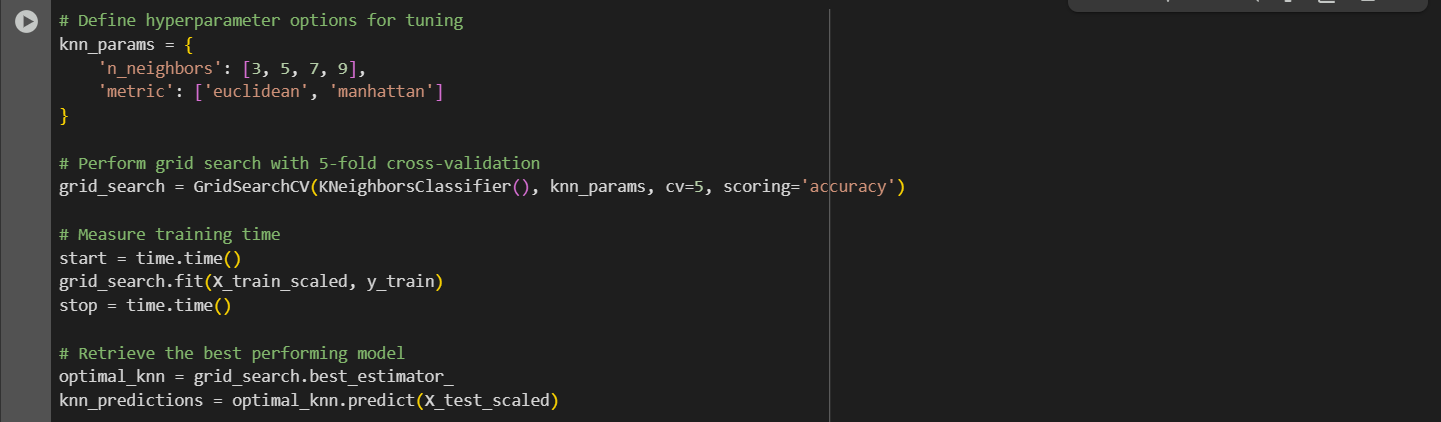
1. Task 2: Apply k-Nearest Neighbors to the same Wine.csv dataset.
2. Preprocess with feature scaling.

Since k-Nearest Neighbors relies on distance calculations, it's important that all features are on the same scale. To achieve this, we used StandardScaler() to standardize the input variables, ensuring that no single attribute dominates due to its magnitude.



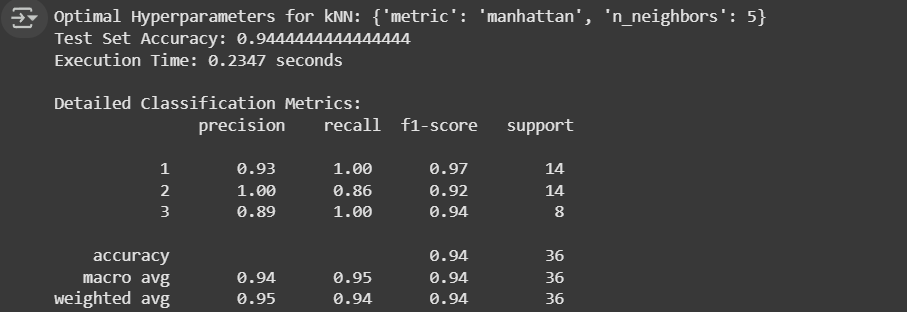
1. Tune k (and distance metric) via cross-validation.

To identify the most effective hyperparameters, we used GridSearchCV combined with 5-fold cross-validation. The tuning process explored different values for n\_neighbors and distance measures. The configuration that yielded the best performance was k=5 with the Manhattan distance.



1. Compare kNN’s accuracy, precision/recall, and runtime to your decision-tree results.

The following accuracy metrics were recorded for the kNN algorithm



|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **k-NN** | **Gini** | **Entropy** |
| Accuracy | 94.44% | 94.44% | 91.67% |
| Runtime | 0.3843s | - | - |
| Macro Avg F1- Score | 0.94 | 0.94 | 0.92 |
| Weighted Avg F1-Score | 0.94 | 0.94 | 0.91 |

Both k-NN and Gini-based decision trees achieved comparable accuracy. However, decision trees offered faster prediction times and greater interpretability, though they needed pruning to prevent overfitting. On the other hand, k-NN was straightforward to implement and performed well, but its prediction speed was slower because of the distance computations involved.