Exploring Decision Tree Classification for Wine Cultivar Identification and Optimizing the Hyperparameters (July 2023)

Kaustuv Karki[1], Undergraduate, IOE, Nikhil Pradhan[2], Undergraduate, IOE
[1]Institute of Engineering Thapathali Campus, Tribhuvan University, Kathmandu, Nepal
[2]Institute of Engineering Thapathali Campus, Tribhuvan University, Kathmandu, Nepal

Corresponding author: Kaustuv Karki (e-mail: karkikaustuv@gmail.com),
Nikhil Pradhan (e-mail: nikhilpradhan20b@gmail.com)

ABSTRACT In today's era, where vast amounts of data are generated and processed by numerous industries and individuals, manual analysis and processing of information can be exceedingly time-consuming. For huge data, classification tasks are helpful because they can help to make sense of large and complex datasets. By grouping data into categories simplifying the data to make it easier to understand and analyze the patterns and trends. The decision tree, a non-parametric supervised learning algorithm, is highly versatile, accommodating both classification and regression tasks. By employing the decision tree classifier from the scikit-learn library, this study aimed to assess the effectiveness of decision trees in accurately classifying data points, specifically in the context of wine cultivar identification. To achieve this, the dataset underwent multiple iterations of the decision tree algorithm with different hyperparameters which were fine tuned. The results obtained from these experiments were meticulously analyzed and compared, shedding light on the performance of decision trees in classifying wine cultivars and identifying the optimal hyperparameters for the task at hand. The findings from this research contribute to the understanding and application of decision tree classification in the context of wine cultivar identification, providing insights into its potential benefits for automating and enhancing data analysis processes in various domains.

INDEX TERMS Decision Tree Classification, Hyperparameter tuning, Supervised Learning, UCI Wine Dataset, Wine Cultivar Identification,

I. INTRODUCTION

The increasing availability of large and diverse datasets in various industries has led to a growing demand for efficient data analysis and mining techniques. Manual processing of such vast amounts of information can be time-consuming and error prone. In response to this challenge, machine learning algorithms, such as decision trees, have gained prominence for their ability to automate and optimize data analysis processes.

The objective of this research paper is to explore the application of decision tree classification for wine cultivar identification and investigate the optimization of hyperparameters to improve the algorithm's performance. The UCI Wine dataset serves as the basis for our experiments.

A decision tree is a predictive machine learning model that implements a hierarchical structure of nodes to partition the input database on its features, leading to a tree-like structure that is a flowchart of decisions. Internal node in the decision tree represents features whereas the leaf nodes represent the predicted class label. Decision trees explore the branches of the tree by constituting a series of questions. This allows the decision tree to identify the underlying patterns, relationships and rules governing the data. Such interpretability makes decision trees an invaluable asset for both analysts seeking insights and stakeholders requiring explanations for decision processes.

II. METHODOLOGY

A. BRIRF THEORY

A decision tree is a classification algorithm that represents a recursive partition of the instance space. The structure of the tree consists of a root node, several internal nodes and a number of leaf nodes. Each internal node in a decision tree partitions the data based on the input feature values. Certain criteria are created on each node on and on is basis of that criteria the data is divided into sub-spaces. Mostly, only one feature is considered in each test. Doing so creates partition based on the single feature value.

Classes are assigned to the leaf node of the decision tree. Each new test data traverses the tree from the root node to the leaf node. The class that the leaf node represents is assigned to the test data. In some cases, the leaf node may contain probability vector indicating the probability of belonging to a certain class.

For example, consider a scenario where the requirement to find if a person evades tax or not is based on the features income, marital status, and refund. The decision tree can have one of the three features as the root node. This root is selected based on some criterion that will be discussed later. Suppose the root node is marital status. Marital status has 2 values, married or single. Based on this value the dataset is further separated i.e., married, and single are separated on the root node. Similarly, we can choose the next internal node as refund or income. If the data can be classified only using the marital status, then no further internal nodes need to be created. So, from the above example we can see that the decision tree recursively partitions the data until leaf nodes or desired condition is reached.

Decision-makers prefer fewer complex trees as they are tedious for understanding and less comprehensible. It has also been found that tree complexity has a high impact on its accuracy. The complexity of a decision tree is measured by using the total number of nodes, total number of leaves, depth, and number of features. Each path in a decision tree can be treated as a rule which can be used to classify the given data to its respective class given by the leaf node. For example, in the figure given below, from root to the right most node rule "If Marital Status single and income less than 50,000 then the person falls on taxpayer" can be created.

During the creation of a decision tree few design issues can be found like:

- Classification of a leaf node:

This problem can be overcome by assigning the majority class to the leaf node or if the leaf is empty, the highest occurring class in the dataset can be assigned to the leaf.

- Splitting of record:

Splitting of records depends upon two conditions, choosing feature test condition and which is the best split. Feature test condition can be selected based on feature type and number of ways to split the feature. For example, Discrete features can be separated through multiway split (i.e. a node can has more than two children) or a binary split (i.e. a node has exactly two children). Continuous features can be converted into a binary decision of a form, or it can be discretized.

- Determination of Best Split:

Determination of best split is done by measuring the impurity of the nodes. Some of the techniques to measure the impurities of nodes are Gini Index, Entropy and Classification error. Based on the value of each of these metrics, we can determine the best split criterion.

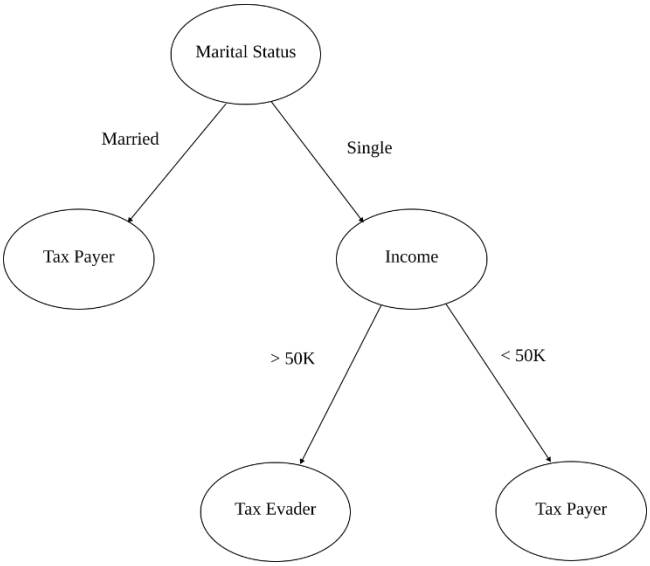

Figure 1 : Simple Decision Tree

B. SYSTEM BLOCK DIAGRAM

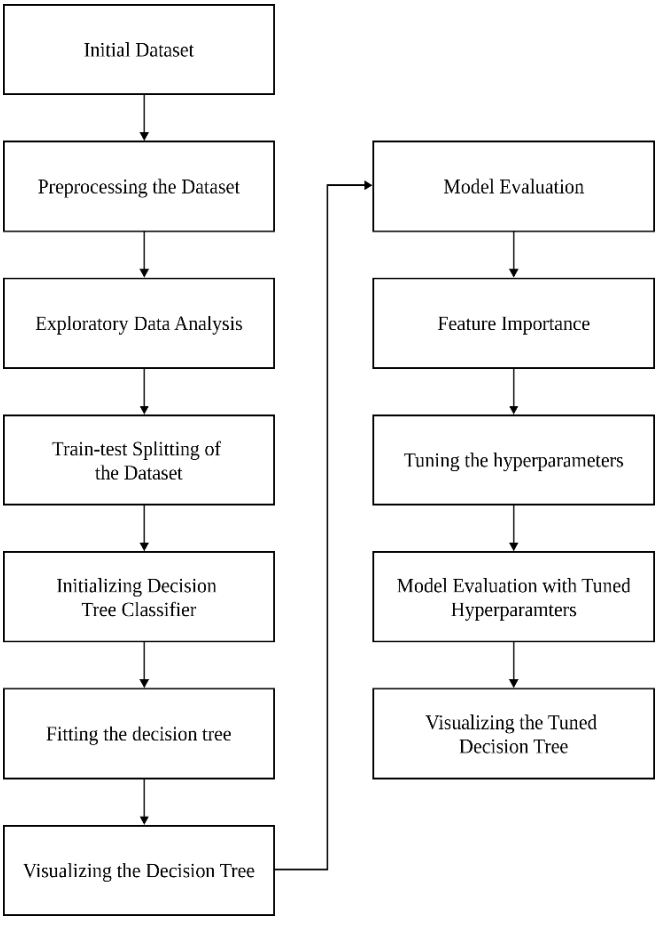

Figure 2 : System Block Diagram

1) INITIAL DATASET

For our study of decision tree classifier, we have chosen the wine classification dataset provided by scikit-learn. It is a copy of UCI ML Wine dataset which is modified to fit standard format. It consists of 3 classes having 178 records with 13 unique features that can be used to categorize the data.

2) PREPROCESSING THE DATASET

Initially the dataset is loaded into a data frame and its number of records is checked. A correlation matrix was plotted to find the quality of the dataset. Two extra data frame d_x and d_y was created. d_x was used to store features and d_y was used to store the target class.

3) EXPLORATORY DATA ANALYSIS

The dataset was checked for its value and its dimensions. Histplot was generated for each of the features to study its distribution in the dataset. As there were 13 features, 13 different histograms were generated and an additional one was also generated for the target.

4) TRAIN TEST SPLITTING OF THE DATASET

The dataset was then split into a training set and testing set. The train set consisted of 66.67 percent of the original dataset and test set consisted of 33.33 percent of the original dataset. After splitting the dataset into train and test, training data consisted of 118 records and test data consisted of 60 records. Train-test split was initialized with random state of 20 so as to produce consistent result.

5) INIITIALIZING THE DECISION TREE CLASSIFIER

After the creation of training and testing data, the decision tree was initialized with the criterion of entropy and that decision tree was trained on the train dataset.

6) FITTING THE DECISION TREE

After the decision tree was initialized, the training was used to fit the decision tree and generate the decision tree using fit function.

7) VISUALIZING THE DECISION TREE

The decision tree was then plotted to observe the root nodes, internal nodes, and the leaf nodes. Visualization of the decision tree allowed us to find the criterion upon which the data set was partitioned.

8) MODEL EVALUATION

A classification report of the created decision tree model was generated to find various evaluation metrics like accuracy, precision, f1-scoce. This step was essential as it gave us insights into the accuracy of the model.

9) FEATURE IMPORTANCE

The dataset was then analyzed based on the model to find out which features were important for the classification of the dataset. The importance of each feature was calculated and plotted using a bar graph.

10) TUNING THE HYPERPARAMETES

After the initial model was made then hyperparameters were tuned in two different occaison in one case the hyperparameters were taken randomly an in another instance the optimal hyperparameters were found using the gridsearchcv and used to create and define the decision tree.

11) MODEL EVALUATION WITH TUNED HYPERPARAMETERS

After using the hyperparameters found using gridsearchcv or the random hyperparameters the initialized decision tree was evaluated using the classification report and the performance of the tuned decision tree was discovered

12) VISUALIZING THE TUNED DECISION TREE

After tuning the decision tree, the final decision tree was plotted using the plot tree provide by the sci-kit learn library.

C. DATASET

The dataset was taken from the UCI dataset which is the wine classification dataset [1]. The features of this dataset are:

- Alcohol: Alcohol content in percentage
- Malic acid: Malic acid content in grams per liter
- Ash: Ash content in grams per liter
- Alkalinity of ash: Total alkalinity of ash in meq per liter
- Magnesium: Magnesium content in milligrams per liter
- Total phenols: Total phenols content in milligrams per liter present in the wine
- Flavanoids: Flavanoids content in milligrams per liter
- Nonflavanoid phenols: Nonflavanoid phenols content in milligrams per liter present in the wine
- Proanthocyanins: Proanthocyanins content in milligrams per liter present in the wine
- Color intensity: Color intensity of the wine
- Hue: Hue of the wine
- OD280/OD315 of diluted wines: OD280/OD315 of diluted wines
- Proline: Proline content in milligrams per liter
- Target: Class 0, 1 and 2 which gives the cultivars of wine

The example of the datasets can be seen as random 7 features were taken and shown below.

TABLE I
BASIC FORMAT OF THE DATASET

S.N	Alcohol	Ash	Alkalinity of ash	flavanoids	hue	target
0	14.23	2.43	15.6	3.06	1.04	0
1	13.2	2.14	11.2	2.76	1.05	0
2	11.96	2.3	21	2.14	0.99	1
3	11.66	1.92	16	1.57	1.23	1
4	13.45	2.6	23	0.92	0.85	2
5	12.82	2.3	19.5	0.66	0.72	2

D. MAJOR MATHEMATICAL FORMULAS

1) ENTROPY

Entropy is the measure of impurity of the dataset. It is calculated by summing the probability of each class in the dataset multiplied by their respective logarithms. When a dataset has high entropy, it signifies greater impurity, whereas a dataset with low entropy indicates higher purity.

$$H(Y) = -\sum_{i=1}^{k} p_i \log_2 p_i \quad (1)$$

Where:
k= Number of classes
p_i = Probability of a class
$H(Y)$ =Entropy of dataset Y.

2) INFORMATION GAIN

Information Gain measures how much information a feature gives us about a class. Also known as change in entropy. A higher Information Gain indicates that the feature provides more useful and distinct information about the class labels, making it a valuable attribute for classification.

$$\Delta H = H - \frac{m_L}{m} H_L - \frac{m_R}{m} H_R \quad (2)$$

Where:
ΔH = Information Gain
H = Entropy
m_L = Number of data points in Left node
m= Number of data points
m_L = Number of data points on Right node
H_L =Entropy of Left node
H_R =Entropy of Right node

3) GINI INDEX

Gini index is also a measure of the impurity of the dataset. It is calculated by summing the squared probabilities of each class in the dataset. A dataset with high Gini index is more impure, while a dataset with low Gini index shows higher purity.

$$GINI = 1 - \sum_{i=1}^{k} {p_i}^2 \quad (3)$$

Where:
p_i= Probability of class
k= Number of classes

C. INSTRUMENTATION

The pandas library was employed for efficient data handling and analysis, while numpy provided support for numerical operations. matplotlib.pyplot and seaborn were used for data visualization, allowing the creation of plots and charts.

The core functionality of constructing the decision tree classifier was facilitated by the DecisionTreeClassifier class from the sklearn.tree module. This class enabled the creation of a decision tree model with various customizable parameters.

The datasets module from sklearn was utilized to access built-in datasets, which were subsequently split into training and testing sets using the train_test_split function from sklearn.model_selection. This division ensured separate subsets for model training and evaluation.

To visualize the decision tree model, the tree module from sklearn was imported. This module provided the necessary functions for exporting the decision tree in a graphical format and for plotting the tree structure using matplotlib.pyplot.

Evaluation of the decision tree classifier's performance involved computing metrics such as classification report and accuracy score. The classification report and accuracy score functions from sklearn.metrics were used to assess the model's performance against the true labels. The confusion_matrix function was employed to generate the confusion matrix, which further aided in performance evaluation.

For cross-validation, the cross_val_score function from sklearn.model_selection was utilized. This function enabled the evaluation of the model's performance using a specific scoring metric across multiple cross-validation folds.

Lastly, hyperparameter tuning was performed using the GridSearchCV class from sklearn.model_selection. This class allowed for an exhaustive search over a specified parameter grid, facilitating the identification of optimal hyperparameters for the decision tree classifier.

The combination of these libraries and tools provided a comprehensive implementation of the decision tree classifier, encompassing data preprocessing, model construction, evaluation, visualization, cross-validation, and hyperparameter tuning.

III. RESULTS AND ANALYSIS

A. DATA ANALYSIS

The dataset from the scikit-learn library UCI wine classification dataset was taken and the dataset was explored to see its distribution and other things.

By performing EDA on the dataset, we know that there are no NaN values present in the dataset. So, there is no requirement of handling the values that are not present. There are 178 values present in the dataset along with 14 features

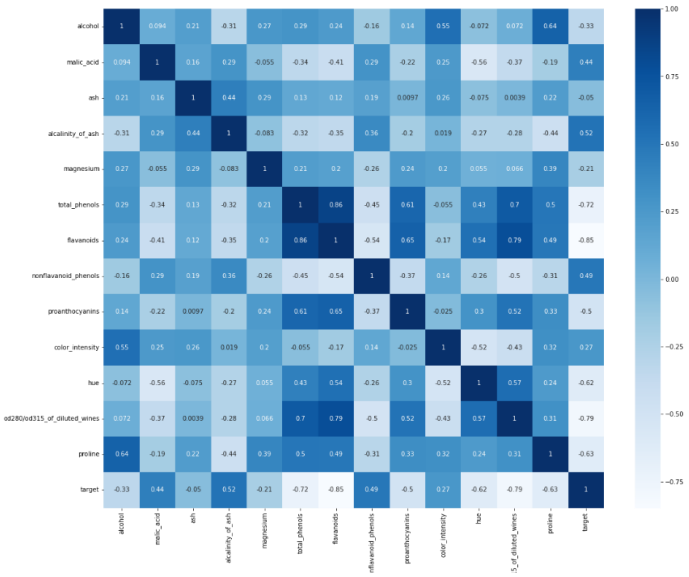

Figure 3 : Correlation Matrix

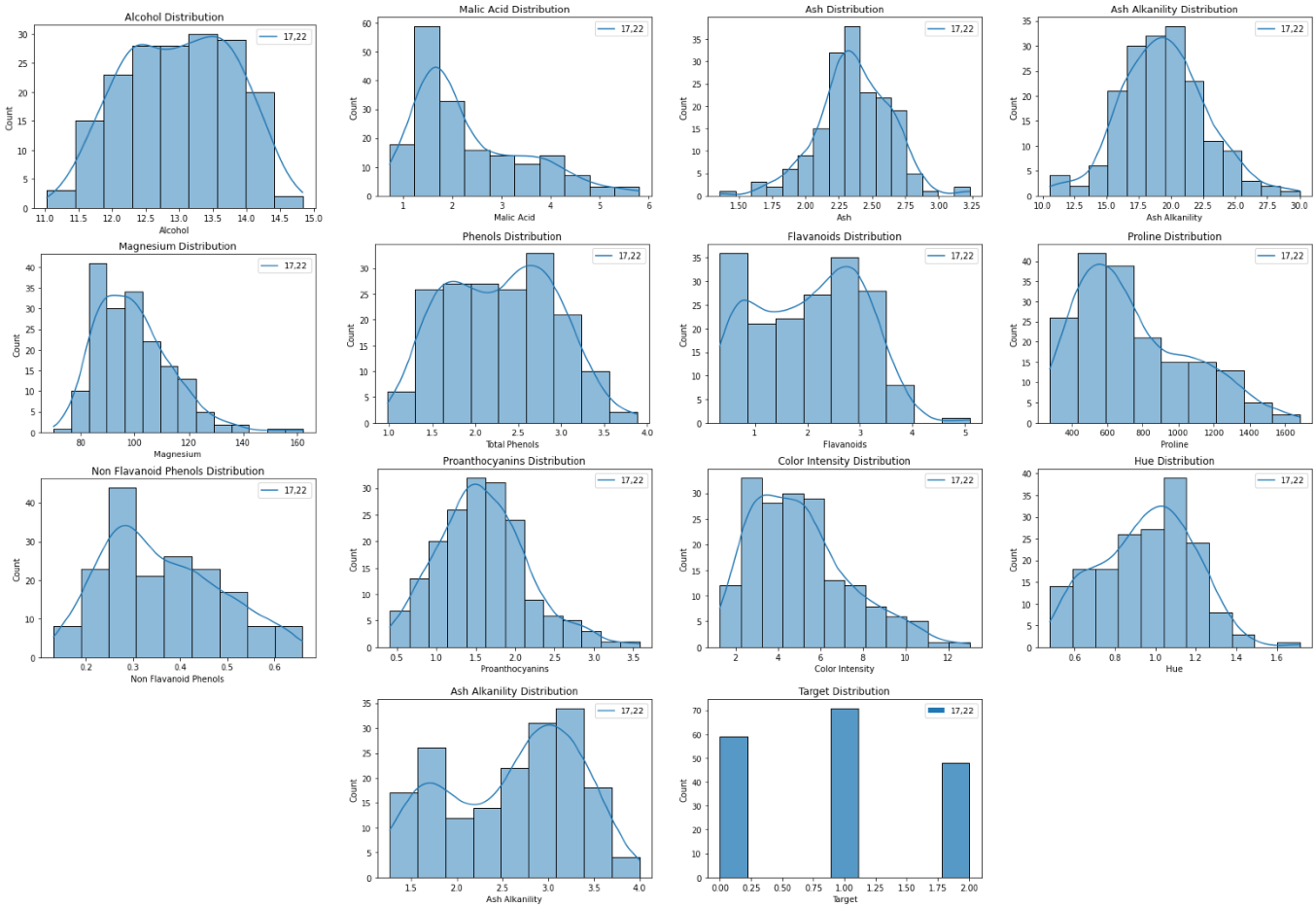

Figure 4 : Distribution Plots

where 13 are the input features and one of them is the target feature. All the features are continuous except the value for the target, which is classified into three classes 0, 1 and 2 of wine.

B. DECISION TREE USING ENTROPY

Initially the total datasets were broken into two parts the portion with the class labels to which the classifier needed to predict to and the other portion containing all the features.

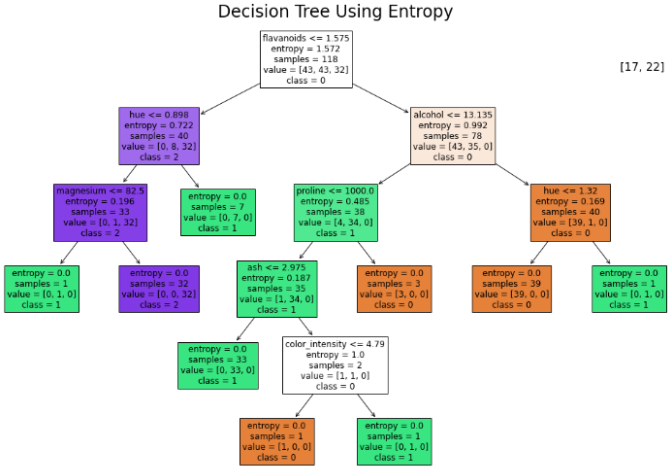

Figure 5 : Decision Tree Using Entropy

The features are used to predict the class label from the dataset.

For the initial decision tree, the hyperparameters were not specified or tuned for the only hyperparameter remotely given the indication was the criterion parameter which was taken as entropy. After the decision tree was initialized with entropy criterion the decision tree was fitted with the train dataset which was divided into the train set x which contained all the features of the dataset and test set y which contained the class labels for the dataset.

So initially after defining the criterion hyperparameter as entropy we get the decision tree by plotting using the tree function as provided by the scikit-learn library.

From the above decision tree, we can see that the base decision tree with not much hyperparameters tuning also give a pretty good result; Due to the few numbers of data points (178 datapoints) in the wine dataset from the UCI repository the decision tree does not have many nodes and branches because the wine cultivar classification dataset from UCI is especially made for classification tasks. The intuition for the importance of a feature for classifying the classes present in the decision is also known as the leaf node. The feature which is present in the leaf node has the highest importance in the classification task done by the decision tree. In this case the

flavonoids feature is the most important feature for the classification.

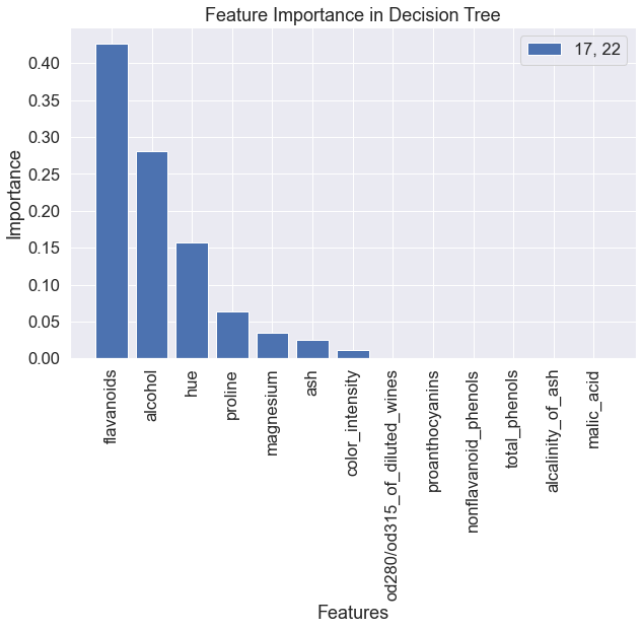

Figure 6 : Feature Importance in Decision Tree

More information about the feature importance can be found in the bar graph below.

From the above bar graph a similar conclusion as the decision tree diagram can be taken. We can see that it is clearly indicated that flavonoids have the highest importance for the process of classification which is the exact conclusion taken from the decision tree above. The flavonoids feature is followed by alcohol, hue proline, magnesium, ash and color intensity. They are the only features which are required for the classification task and the total importance they give are summed up and can be found to be equal to one. The other features like odd280/od315 of diluted wine, proanthocyanins, non-flavonoid phenols, total phenols, alkalinity of ash and malic acid have no importance in the process of classification of the decision tree, so their feature importance are found to be zero. This again can be verified by using the decision tree as the features whose importance are found to be zero are not used for the splitting process and they are omitted from the decision tree.

Finally, the performance of the classification model was taken out using classification report. The classification report provides a comprehensive overview of the model accuracy and effectiveness in classifying different categories. The classification report of the train dataset and the test dataset were taken.

TABLE II
CLASSIFICATION REPORT OF TRAIN SET FOR DECISION TREE WITH ENTROPY

Class	Precision	Recall	F1-score
0	1.00	1.00	1.00
1	1.00	1.00	1.00
2	1.00	1.00	1.00
Accuracy-score		1.00	

TABLE III
CLASSIFICATION REPORT OF TEST SET FOR DECISION TREE WITH ENTROPY

Class	Precision	Recall	F1-score
0	0.88	0.88	0.88
1	0.84	0.93	0.88
2	1.00	0.81	0.90
Accuracy-score		0.88	

The values for the table above were derived from a classification report on the test dataset to check the performance of the decision tree classifier using the predicted values produced by the decision tree. From the above information we can see that the model is performing with the accuracy of 88% which is pretty good, but it can be optimized by tuning various hyperparameters.

The results can also be visualized using the confusion matrix as shown below. As the diagonal values are high and the non-diagonal values are low the decision tree classifier correctly classifies most of the data in the dataset correctly.

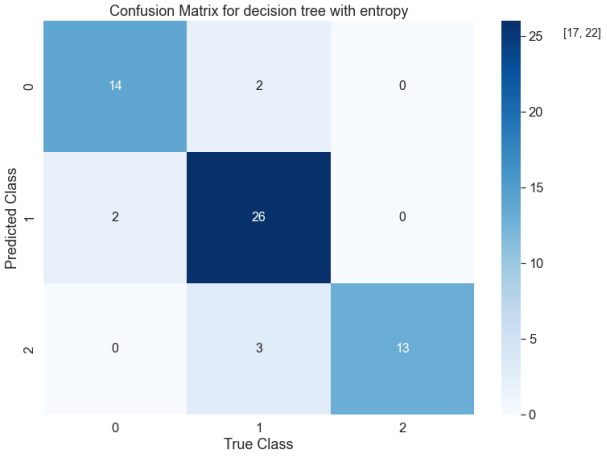

Figure 7 : Confusion Matrix for decision tree with entropy

C. DECISION TREE USING RANDOMLY SELECTED HYPERPARAMETERS

Initially just entropy was used as a hyperparameter while initializing the decision tree. But the decision tree not only has criterion as a hyperparameter it also has various other hyperparameters like max depth, minimum samples split, minimum samples leaf, cost complexity pruning value and so on. Random values for the hyperparameters were selected for the new decision tree to check how the decision tree behaves when random hyperparameters were taken when initializing the decision tree. After selecting the hyperparameters randomly various results were observed. Firstly, the instances of randomly selecting the hyperparameter resulted in suboptimal performance of the decision tree causing it to be less accurate than the decision tree with not specific hyperparameters selected. The randomly selected hyperparameter for the decision tree are: For the criterion the value of entropy was used, for the max depth the value of 2, for the minimum number of samples leaf the value of 2 was used and for minimum value of samples split the value of 3 was used.

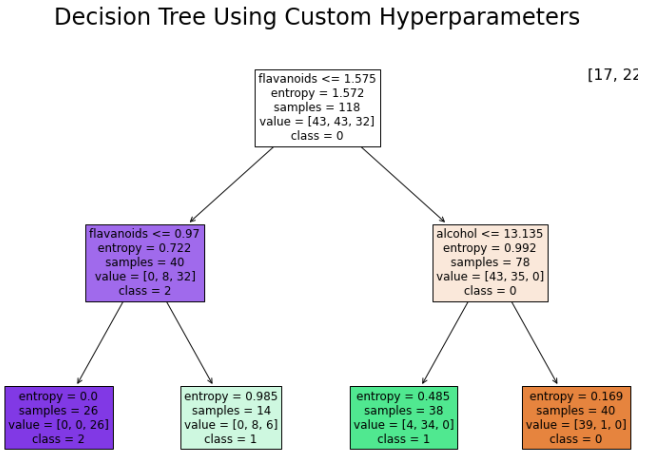

Figure 8 : Decision Tree Using Custom Hyperparameters

Various insights can be taken from the above decision tree. We can see that by specifying the value of max depth as 2 the depth of decision tree is smaller making it more readable and easily analyzable but it comes in the cost of the performance of the decision tree We can also observer that less features are being used than the initial decision tree with no hyperparameters making the visualization of this decision tree realistic if plotted in a two dimensional figure separating with the axes defined using feature flavonoids and alcohol.

The features which are used for splitting have certain importance assigned to them in the decision tree. The features at the top have more importance values than the features at the bottom. The feature having the highest importance is the flavonoid followed by alcohol. Other features are not present in the decision tree as they are not required for the task of splitting. The total importance of the feature alcohol and flavonoids add up to one and other features do not have any importance for the decision tree classification.

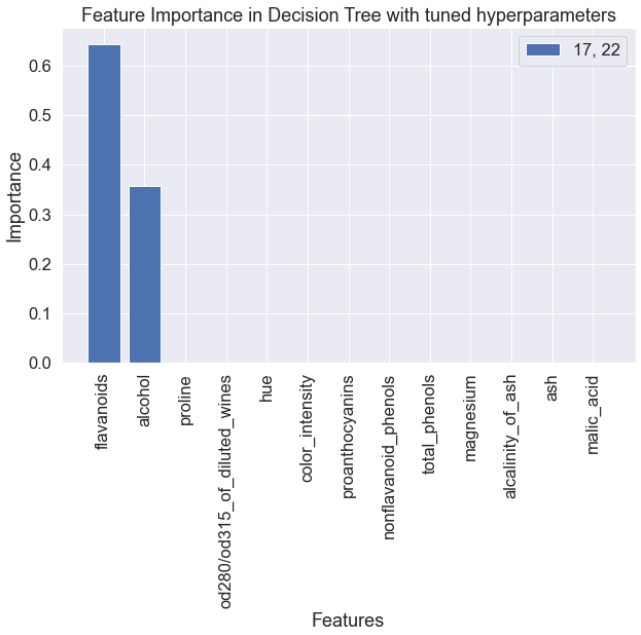

Figure 9 : Feature Importance in Decision Tree with tuned Hyperparameter

More information about the importance of the of the features of the dataset which impact the most in the formation of the decision tree can be gained from the above graph. We can see that the features flavonoids and alcohol are the only features which are important for the decision tree classification which is again reinforced by the figure of the decision tree as there are only separation using alcohol and flavonoids present in the decision tree. The other features proline, od280/od315 of diluted wines, hue, color intensity, proanthocyanins, nonflavonoid phenols, total phenols, magnesium, alkalinity of ash, ash and malic acid have no importance as these features are not used in the process of splitting of the decision tree.

TABLE IV
CLASSIFICATION REPORT OF TRAIN SET FOR DECISION TREE WITH RANDOM HYPERPARAMETERS

Class	Precision	Recall	F1-score
0	0.91	0.97	0.94
1	0.98	0.81	0.88
2	0.81	1.00	0.90
Accuracy-score		0.91	

TABLE V
CLASSIFICATION REPORT OF TRAIN SET FOR DECISION TREE WITH RANDOM HYPERPARAMETERS

Class	Precision	Recall	F1-score
0	0.87	0.81	0.84
1	0.78	0.89	0.83
2	0.92	0.75	0.83
Accuracy-score		0.83	

D. DECISION TREE TUNING THE PRUNING VALUE

Other ways of optimizing the decision tree are changing the cost complexity pruning value. The cost complexity pruning algorithm is used to find the optimal pruning strategy. In this case the cost complexity pruning process adds regularization term known as alpha parameter to the entropy criterion.

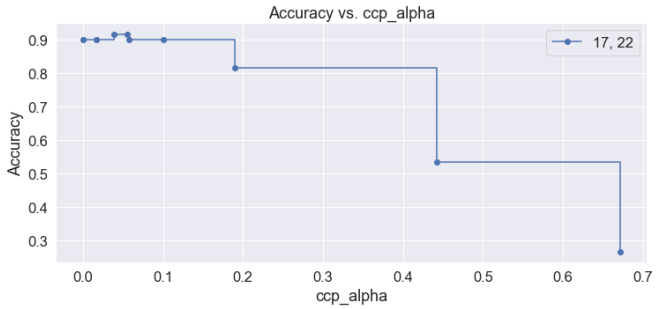

Figure 10 : Accuracy vs. ccp_alpha

From the above we can see that the highest accuracy can be found when the cost complexity pruning value is found to be 0.03857. This alpha parameter controls the trade off between the complexity of the decision tree and its fitting to the training data. Higher values of the alpha results in more aggressive pruning leading to simpler trees and lower values of the alpha results the decision tree to grow possibly capturing all the intricacies and pattern of the data.

Using the cost complexity pruning alpha value and using the criteria as the entropy we get the following decision tree.

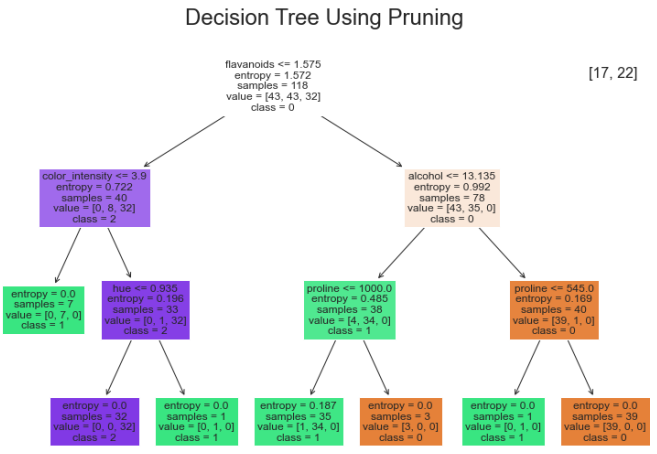

Figure 11 : Decision Tree Using Pruning

From the above decision tree, we can infer that the most important feature is found to be flavonoids followed by alcohol. The training accuracy was found to be 1.00. We can see that this tree is deeper and bigger than previous trees also capture most of the data as the test accuracy is found to be 92%. So, this kind of overfits the data as the training accuracy is greater than the test accuracy.

More information about the accuracy scores can be derived from the table below.

TABLE VI
CLASSIFICATION REPORT FOR DECISION TREE WITH PRUNING

Class	Precision	Recall	F1-score
0	0.93	0.88	0.90
1	0.87	0.96	0.92
2	1.00	0.88	0.93
Accuracy-score		0.92	

E. DECISION TREE AFTER TUNING THE HYPERPARAMETERS

Finally, we used GridSearchCV provided by the scikit-learn library to fine tune the hyperparameters initialized in the datasets. Using the function provided by the scikit-learn library and initializing the range of values that it can select from using the parameter grid we found the optimal hyperparameters for the decision tree fitting the data.

The parameter grid values taken for this process were:

- Criterion: Entropy, Gini and log loss were taken
- Max depth: The range of values from 1 to 10
- Minimum Samples Split: The range of values from 2 to 10.
- Minimum Samples Leaf: The range of values from 1 to 10

Then according to the metric accuracy, the best hyperparameters for the decision tree were chosen. The hyperparameters that the GridSearchCV returned were found to be “log loss”, max depth was found to be 5, minimum samples leaf was found to be 1 and the minimum samples split was found to be 4.

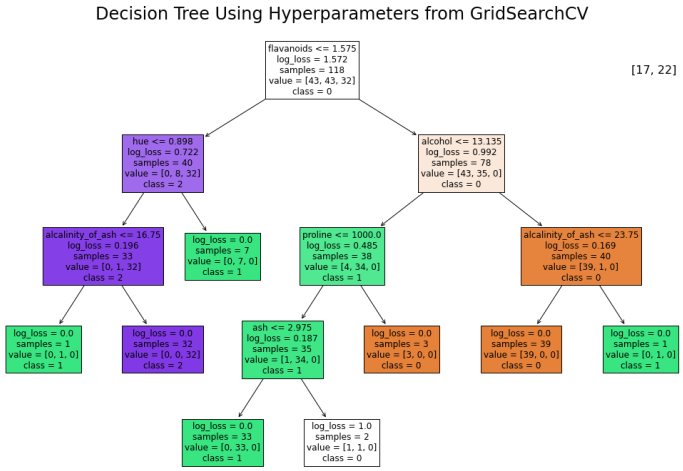

Figure 12 : Decision Tree Using Hyperparameters from GridSearchCV

Various insights can be taken from the above decision tree. We can see that as being in the root node of the decision tree the flavonoids feature is the most important followed by the other features. We can see the importance of each feature using the bar graph below.

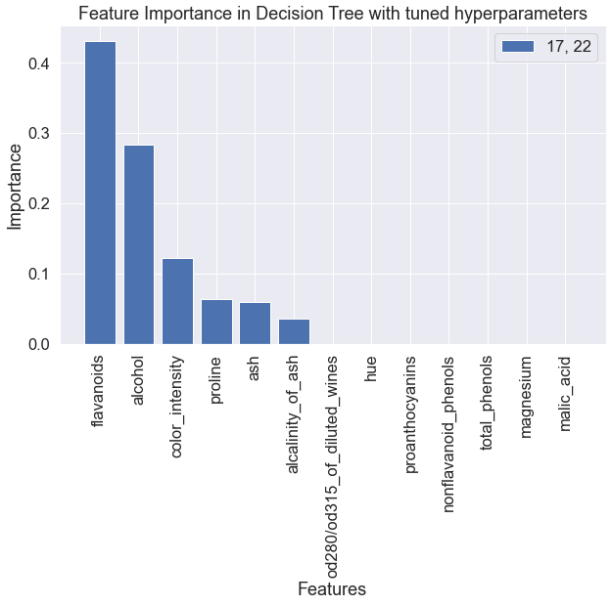

Figure 13 : Feature Importance in Decision Tree with Tuned hyperparameters

From the above bar plot we can see the only features affecting the splitting and the formation of the decision tree are flavanoids which is the most important followed by alcohol, color intensity, proline, ash, alcalinity of ash. These are the only features used in the decision tree. The features like hue, proanthocyanins, malic acid , toal phenols and so on are not important fot his instance of the decision tree.

Finally, the performance of the classification model was taken out using classification report. The classification report provides a comprehensive overview of the model accuracy and effectiveness in classifying different categories. The classification report of the train dataset and the test dataset were taken.

TABLE III
CLASSIFICATION REPORT OF TRAIN SET FOR DECISION TREE WITH RANDOM HYPERPARAMETERS

Class	Precision	Recall	F1-score
0	1.00	0.98	0.99
1	0.97	0.99	0.98
2	1.00	1.00	1.00
Accuracy-score		0.98	

The values for the table above were derived from a classification report on the train dataset. From the above table it can be inferred that the accuracy on the train dataset of the model was found to be 98%.

TABLE IV
CLASSIFICATION REPORT OF TRAIN SET FOR DECISION TREE WITH RANDOM HYPERPARAMETERS

Class	Precision	Recall	F1-score
0	0.93	0.88	0.90
1	0.90	0.96	0.93
2	1.00	0.94	0.97
Accuracy-score		0.93	

The values for the table above were derived from a classification report on the test dataset. From the above table it can be inferred that the accuracy on the train dataset of the model was found to be 93%.

Finally, the results from the predictions made by the decision tree can be visualized using the confusion matrix as shown below.

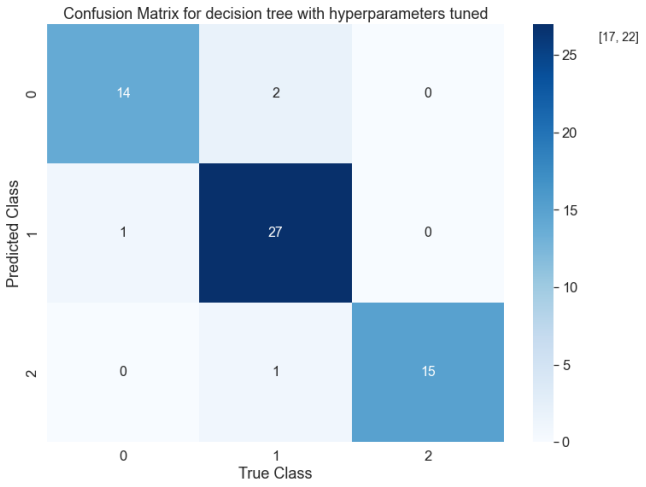

Figure 14 : Confusion Matrix for decision tree with hyperparameters tuned

IV. DISCUSSION AND ANALYSIS

This report presents the implementation process of a decision tree classifier. The aim of this implementation was to gain a deeper understanding of the inner workings of decision tree algorithms and to provide valuable insights and intuition into their functionality. Analysis of plot of tree suggested that even though there were 13 different features only 7 of them contributed to classification of a record. We have experimented with 3 different techniques of implementation of decision tree. The first one was to only use one hyperparameter i.e., entropy for generation of the classifier. This resulted in a tree that could perform with and accuracy of 88 percent. Such high accuracy was achieved because the dataset consisted of a small number of records and was specifically created for classification purposes. All the values in the dataset were standard values with no null value as a result no preprocessing on the dataset was required. But comparing to the accuracy on the train dataset when only the entropy was taken as a hyperparameter we found the train accuracy to be 100% which shows that the data is overfitting.

When we introduced randomly initialized hyperparameter, for the criterion the value of entropy, for the max depth the value of 2, for the minimum number of samples leaf the value of 2 and for minimum value of samples split the value of 3, we found that the model degraded and produced suboptimal result. As max depth was

taken as 2 most of the meaningful insights were lost, as a result the model could not infer from the data properly.

After that, we tuned the value of cost complexity pruning value. During this tuning a regularization term known as alpha was added to the entropy criterion. During our evaluation we found that the best value for alpha was 0.03857 which was then used as a hyperparameter value as ccp_alpha value then a decision tree was taken out which had a training accuracy of 100% and the test accuracy of 92% which shows overfitting but not as major as when only entropy was used.

Finally, GridSearchCv from scikit-learn was used to calculate the values of hyperparameters for the decision tree. The criterion as log loss, the max depth as 5, minimum samples leaf as 1 and minimum samples size of 4 was used to create a new decision tree. Applying the classification report on the new decision tree we found that the accuracy on the training data was found to be 98% and accuracy on the testing dataset was found to be 93% which has the values of the training and testing accuracy closer thus decreasing the overfitting but still not ideal.

From all the implementation of the decision tree we can also see that the feature falvanoids has the most impact in the formation of the decision tree as the root node was always found to be flavanoids followed by alcohol.

V. CONCLUSION

In conclusion, our implementation of the decision tree classifier using the scikit-learn library provided us with valuable learning experiences and insights into the application of decision trees for classification tasks. Throughout the implementation process, we focused on using the entropy criterion as the measure for splitting nodes in the decision tree. By applying the decision tree classifier to various datasets, we were able to observe the power of decision trees in capturing complex relationships and making accurate predictions. The scikit-learn library offered a comprehensive set of tools for constructing, training, and evaluating the decision tree classifier. During the implementation, we gained a deeper understanding of the decision tree algorithm and its decision-making process. We learned how the entropy criterion is used to evaluate the purity of the nodes and determine the optimal splitting point for classifying instances. This insight helped us grasp the importance of selecting informative features for improving the predictive performance of the decision tree classifier. Furthermore, we explored the different parameters and options provided by the scikit-learn library, such as pruning techniques. These features allowed us to fine-tune the decision tree classifier and improve its performance on the given datasets..

REFERENCES

[1] Aeberhard,Stefan and Forina,M.. (1991). Wine. UCI Machine Learning Repository. https://doi.org/10.24432/C5PC7J.

KAUSTUV KARKI is a devoted individual currently pursuing a graduate degree in Computer Engineering Program from Institute of Engineering Thapathali Campus under Tribhuvan University He is keen and highly motivated towards the field of Artificial Intelligence and Machine Learning. He actively seeks out various sources to learn more about these fields, including books, research papers, online courses, and tutorials. He believes in continuous learning and keeping up with the latest advancements in AI and ML.
Overall, his dedication, curiosity, and proactive approach make him a promising individual in the field of Artificial Intelligence and Machine Learning.
Kaustuv's curiosity serves as a driving force behind his pursuit of knowledge. He delves deep into complex concepts, asks thoughtful questions, and actively engages in discussions to gain a comprehensive understanding of AI and ML. This intellectual curiosity fuels his motivation and propels him to explore innovative solutions and approaches in the field.

NIKHIL PRADHAN is a dedicated and ambitious individual currently pursuing graduate studies in computer engineering from Institute of Engineering Thapathali Campus. With a strong passion for artificial intelligence (AI), he is actively engaged in expanding his knowledge and expertise in this rapidly evolving field. His academic pursuits provide him with a solid foundation in computer engineering and a deep understanding of programming languages and frameworks commonly used in AI applications. With his enthusiasm, academic background, and commitment to learning, He is well-positioned to make significant contributions to the AI industry.

APPENDIX A: CODE SNIPPETS

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
from sklearn import datasets
from sklearn.model_selection import
train_test_split
from sklearn import tree
from sklearn.metrics import classification_report,
accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import
cross_val_score
from sklearn.model_selection import GridSearchCV


# Features of the dataset
# Alcohol -> Alcohol content in percentage
# Malic acid -> Malic acid content in grams per
liter
# Ash -> Ash content in grams per liter
# Alkalinity of ash -> Total alkalinity of ash in
meq per liter
# Magnesium -> Magnesium content in milligrams per
liter
# Total phenols -> Total phenols content in
milligrams per liter
# Flavanoids -> Flavanoids content in milligrams
per liter
# Nonflavanoid phenols -> Nonflavanoid phenols
content in milligrams per liter
# Proanthocyanins  -> Proanthocyanins content in
milligrams per liter
# Color intensity -> Color intensity
# Hue -> Hue
# OD280/OD315 of diluted wines -> OD280/OD315 of
diluted wines
# Proline -> Proline content in milligrams per
liter
# Target -> Class 0, 1 and 2


# Loading the datasets from sklearn uci datasets
wine = datasets.load_wine()
wine_df = pd.DataFrame(wine.data, columns =
wine.feature_names)
wine_df['target'] = pd.Series(wine.target)
wine_df.shape
data = wine_df

# The target labels are class 0, class 1, class 2
cultivars

corr = data.corr()
f, ax = plt.subplots(figsize=(18, 15))
sns.heatmap(corr, cmap="Blues", annot=True)


data["target"].value_counts()


df_y = data["target"]
df_y = df_y.astype(int)
df_y.value_counts()
df_y.shape
df_y = df_y.values.reshape(-1, 1)

df_x = data.drop(columns=["target"])
data.shape,df_x.shape, df_y.shape
data["target"].value_counts()
```

```
# Exprolatory Data Analysis
data.describe()
column_name = wine.feature_names

# Alcohol distribution
sns.histplot(data["alcohol"], kde=True)
plt.title("Alcohol Distribution")
plt.xlabel("Alcohol")
plt.ylabel("Count")
plt.legend(["17,22"])

# Malic Acid Distribution
sns.histplot(data["malic_acid"], kde=True)
plt.title("Malic Acid Distribution")
plt.xlabel("Malic Acid")
plt.ylabel("Count")
plt.legend(["17,22"])

# Ash Distribution
sns.histplot(data["ash"], kde=True)
plt.title("Ash Distribution")
plt.xlabel("Ash")
plt.ylabel("Count")
plt.legend(["17,22"])

# Ash Alkanility Distribution
sns.histplot(data["alcalinity_of_ash"], kde=True)
plt.title("Ash Alkanility Distribution")
plt.xlabel("Ash Alkanility")
plt.ylabel("Count")
plt.legend(["17,22"])

# Magnesium Distribution
sns.histplot(data["magnesium"], kde=True)
plt.title("Magnesium Distribution")
plt.xlabel("Magnesium")
plt.ylabel("Count")
plt.legend(["17,22"])

# Phenols Distribution
sns.histplot(data["total_phenols"], kde=True)
plt.title("Phenols Distribution")
plt.xlabel("Total Phenols")
plt.ylabel("Count")
plt.legend(["17,22"])

# Flavanoids Distribution
sns.histplot(data["flavanoids"], kde=True)
plt.title("Flavanoids Distribution")
plt.xlabel("Flavanoids")
plt.ylabel("Count")
plt.legend(["17,22"])

# Non Flavanoid Phenols Distribution
sns.histplot(data["nonflavanoid phenols"],
kde=True)
plt.title("Non Flavanoid Phenols Distribution")
plt.xlabel("Non Flavanoid Phenols")
plt.ylabel("Count")
plt.legend(["17,22"])

# Proanthocyanins Distribution
sns.histplot(data["proanthocyanins"], kde=True)
plt.title("Proanthocyanins Distribution")
plt.xlabel("Proanthocyanins")
plt.ylabel("Count")
plt.legend(["17,22"])

# Color Intensity Distribution
sns.histplot(data["color_intensity"], kde=True)
plt.title("Color Intensity Distribution")
plt.xlabel("Color Intensity")
```

```
plt.ylabel("Count")
plt.legend(["17,22"])

# Hue Distribution
sns.histplot(data["hue"], kde=True)
plt.title("Hue Distribution")
plt.xlabel("Hue")
plt.ylabel("Count")
plt.legend(["17,22"])

# Ash Alkanility Distribution
sns.histplot(data["od280/od315_of_diluted_wines"],
kde=True)
plt.title("Ash Alkanility Distribution")
plt.xlabel("Ash Alkanility")
plt.ylabel("Count")
plt.legend(["17,22"])

# Proline Distribution
sns.histplot(data["proline"], kde=True)
plt.title("Proline Distribution")
plt.xlabel("Proline")
plt.ylabel("Count")
plt.legend(["17,22"])

# Target Distribution
sns.histplot(data["target"])
plt.title("Target Distribution")
plt.xlabel("Target")
plt.ylabel("Count")
plt.legend(["17,22"])


# Decision Tree without using any hyperparameters
df_y.shape

train_x, test_x, train_y, test_y =
train_test_split(df_x, df_y, test_size=1/3,
random_state=20)
train_x.shape, train_y.shape
clf_dt1 =
DecisionTreeClassifier(criterion="entropy")
clf_dt1.fit(train_x, train_y)
labels = ["0","1","2"]


fig = plt.figure(figsize=(18,12))
_ = tree.plot_tree(clf_dt1,

feature names=wine.feature names,
                    class_names=labels,
                    filled=True,
                    fontsize=12
                    )

# Accuracy score of train set
report_dt1_train =
classification_report(clf_dt1.predict(train_x),
train_y)
print(report_dt1_train)
# Accuracy score of test set
report_dt1_test = classification_report(test_y,
clf_dt1.predict(test_x))
print(report_dt1_test)

# For confusion matrix
y_pred =clf_dt1.predict(test_x)
cm = confusion_matrix(test_y, y_pred)
test_x.shape
plt.figure(figsize=(12, 9))
plt.title('Confusion Matrix')
sns.heatmap(cm, annot=True, fmt="", cmap='Blues')
plt.xlabel('Predicted Class')
plt.ylabel('True Class')

# Feature importance for the decision tree
feature_importance = clf_dt1.feature_importances_
feature_importance

# This sorts the values in the feature importance
in a descending order
sorted_indices = feature_importance.argsort()[::-
1]

# The sorted importance sorts the importance
values using the indices provided
sorted_importance =
feature_importance[sorted_indices]

# Sorts the features of the datset in the sorted
manner
sorted_features = train_x.columns[sorted_indices]
plt.figure(figsize=(10, 6))
plt.bar(range(len(sorted_importance)),
sorted_importance, tick_label=sorted_features)
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Feature Importance in Decision Tree')
plt.xticks(rotation=90)
plt.show()

scores = cross_val_score(clf_dt1, df_x, df_y,
cv=5)
scores.mean()
y_pred_train = clf_dt1.predict(train_x)
accuracy_score(train_y, y_pred_train)
<h3>Classification with hyperparameters
clf_dt2 =
DecisionTreeClassifier(criterion="entropy",
max_depth=3)
clf_dt2.fit(train_x, train_y)

fig = plt.figure(figsize=(18,12))
_ = tree.plot_tree(clf_dt2,

feature_names=wine.feature_names,
                    class_names=labels,
                    filled=True,
                    fontsize=12
                    )

# Prediction on Test And Train Set
report dt2 train =
classification_report(clf_dt2.predict(train_x),
train_y)
print(report_dt1_train)
report_dt2_test = classification_report(test_y,
clf_dt2.predict(test_x))
print(report dt2 test)

# Testing for More Hyperparameters
clf_dt3 =
DecisionTreeClassifier(criterion="entropy",
max_depth=3, min_samples_split=10,
min_samples_leaf=2)
clf_dt3.fit(train_x, train_y)
report_dt3_train =
classification_report(clf_dt3.predict(train_x),
train_y)
print(report_dt3_train)
report_dt3_test = classification_report(test_y,
clf_dt3.predict(test_x))
print(report_dt3_test)
fig = plt.figure(figsize=(20,10))
_ = tree.plot_tree(clf_dt3,
```

```
feature_names=wine.feature_names,
                    class_names=labels,
                    filled=True,
                    fontsize=12
                    )
path =
clf_dt1.cost_complexity_pruning_path(train_x,
train_y)
path
# Extract different values of ccp_alpha and
corresponding accuracy scores
ccp_alphas = path.ccp_alphas
scores = []
for ccp_alpha in ccp_alphas:
    clf =
DecisionTreeClassifier(ccp_alpha=ccp_alpha)
    clf.fit(train_x, train_y)
    y_pred = clf.predict(test_x)
    accuracy = accuracy_score(test_y, y_pred)
    scores.append(accuracy)

# Plot the accuracy scores as a function of
ccp_alpha
fig = plt.figure(figsize=(18,6))
plt.plot(ccp_alphas, scores, marker='.',
drawstyle="steps-post")
plt.xlabel("ccp_alpha")
plt.ylabel("Accuracy")
plt.title("Accuracy vs. ccp_alpha")
plt.show()
index_val = np.array(scores).argmax()
index_val
ccp_alpha_value = ccp_alphas[index_val]
ccp_alpha_value
scores_new = []
for max_depth_value in range(1,5):
    for min_samples_split_value in range(2, 10):
        for min_samples_leaf_value in range(1,10):
            clf =
DecisionTreeClassifier(criterion="entropy",
max_depth=max_depth_value,
min_samples_leaf=min_samples_leaf_value,
min_samples_split=min_samples_split_value)
            clf.fit(train_x, train_y)
            y_pred = clf.predict(test_x)
            accuracy = accuracy_score(test_y,
y_pred)

scores new.append({"max depth":max depth value,"mi
n_samples_split":min_samples_split_value,
"min_samples_leaf": min_samples_leaf_value,
"accuracy": accuracy})

criterion_name = ["entropy", "gini", "log_loss"]
parameters = dict(criterion=criterion name,
                      max_depth=range(1,10),

min_samples_split=range(2,10),

min_samples_leaf=range(1,10),
                      )
clf = DecisionTreeClassifier()
gscv = GridSearchCV(clf, param_grid=parameters,
scoring="accuracy", cv=5)
gscv.fit(train_x, train_y)
print(gscv.best_params_)
clf_dt4 =
DecisionTreeClassifier(criterion="entropy",
max_depth=7, min_samples_leaf=1,
min_samples_split=3)
clf_dt4.fit(train_x, train_y)
fig = plt.figure(figsize=(20,10))
_ = tree.plot_tree(clf_dt4,

feature_names=wine.feature_names,
                    class_names=labels,
                    filled=True,
                    fontsize=12
                    )
report_dt4_train =
classification_report(clf_dt4.predict(train_x),
train_y)
print(report_dt4_train)
report_dt4_test = classification_report(test_y,
clf_dt4.predict(test_x))
print(report_dt4_test)
```