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

Kaustuv Karki1, Undergraduate, IOE, Nikhil Pradhan2, Undergraduate, IOE

1Institute of Engineering Thapathali Campus, Tribhuvan University, Kathmandu, Nepal

2Institute of Engineering Thapathali Campus, Tribhuvan University, Kathmandu, Nepal

Corresponding author: Kaustuv Karki (e-mail: [karkikaustuv@gmail.com](mailto:karkikaustuv@gmail.com)),

Nikhil Pradhan (e-mail: [nikhilpradhan20b@gmail.com](mailto: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,

1. 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. Through a comparative analysis of the results obtained from different hyperparameter settings, we aim to identify the optimal configuration that maximizes the decision tree's classification accuracy for wine cultivar identification. Additionally, we assess the interpretability of the decision tree model, which can provide valuable insights into the underlying factors influencing wine cultivar classification. By applying decision tree classification on the dataset makes it easier to understand and analyze the patterns present in the dataset.

1. METHODOLOGY

A. BRIRF THEORY

The main concept of PCA[1] is removing any redundant dimensions and keeping the dimensions with the highest variance. PCA selects the principal components that capture all the major variations across the dataset encompassing most of the information present in the dataset

Suppose we have a dataset with features X and Y then we can fit a regression line onto the dataset. Drawing an orthogonal line to the best fit line we get,

Since data varies mostly along the best fit line. Now, we can project the points onto our new axis and get our new x-axis and y-axis. We can keep drawing lines perpendicular to both lines to get more axis.

These new axes are Principal Components. PC1 is generally used to denote the component that captures the most variation, PC2 the next and so on.

B. SYSTEM BLOCK DIAGRAM

A black background with a black square

Description automatically generated

1. INITIAL DATASET

Initially features are selected from which the target is interpreted. These features allow the model to make predictions on new data to assign classes.

1. PREPROCESSING THE DATASET

Before starting the PCA process, the feature vectors must be standardized so that they have zero mean and standard deviation of 1. This ensures that all the features have the same scale so that PCA gives equal importance to each feature preventing features with larger scale to disproportionately influence the result. Units are also removed during standardization making interpretation of principal components more meaningful.

1. EXPLORATORY DATA ANALYSIS

After standardization of features, covariance matrix of the standardized features is calculated. Covariance matrix shows the linear relationship between the features. For PCA decomposition, the diagonal elements of the covariance matrix i.e., Variance of a feature should be maximum, and the non-diagonal elements of the covariance matrix must be minimum. Covariance matrix allows PCA to find the direction of maximum variance.

1. TRAIN TEST SPLITTING OF THE DATASET

The covariance matrix is used of eigen decomposition to get the eigenvalues and eigenvectors. Eigenvalues represent the amount of variance captured by each principal component and eigenvectors represent the direction of each principal component. Each eigenvector is new basis vectors that are used to project the original data into new coordinates.

1. INIITIALIZING THE DECISION TREE CLASSIFIER

Eigen values are used to calculate the importance of each principal component. If n number of features is required, then eigen values are arranged in ascending order and eigenvectors with top n eigenvalue is selected. This ensures that maximum amount of variance is captured thus retaining essential information.

1. FITTING THE DECISION TREE

Selected eigenvectors are used to form a feature new feature matrix. The feature matrix acts as a new basis for the data, and the new features are orthogonal to each other.

1. VISUALIZING THE DECISION TREE

Change of basis is performed on the original dataset to project them onto new coordinates. This results in new dataset with reduced number of features.

1. MODEL EVALUATION

New data is then plotted to visualize the changes to study the result.

1. FEATURE IMPORTANCE
2. TUNING THE HYPERPARAMETES
3. MODEL EVALUATION WITH TUNED HYPERPARAMETERS
4. VISUALIZING THE TUNED DECISION TREE

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.

|  |  |
| --- | --- |
|  | (1) |

Where:

*k*= Number of classes

= Probability of a class

=Entropy of dataset Y.

1. 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.

|  |  |
| --- | --- |
|  | (2) |

Where:

= Information Gain

= Entropy

= Number of data points in Left node

= Number of data points

= Number of data points on Right node

=Entropy of Left node

=Entropy of Right node

1. 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 is shows higher purity.

|  |  |
| --- | --- |
|  | (3) |

Where:

= Probability of class

*k*= Number of classes

1. Change of basis

Basis of a vector space is the set of linearly independent vectors that span all the vector space. For e.g., Consider a 2-dimensional space with unit vector along x axis and along y axis. These unit vectors can then be scaled so that it can span all the possible points in the 2-dimensional space. In PCA, change on basis plays a vital role to as we map the original data into new one by changing the basis vector. Changing the basis does not change the data only its representation is changed. Changing the basis only projects the data vectors on the basis vectors.

Change of basis is achieved by:

|  |  |
| --- | --- |
|  | (5) |

A screenshot of a computer program

Description automatically generated

Where:

*Y* = Data points obtained after linear transformation

*P* = Basis vectors used for linear transformation

*X* = Original data points

C. INSTRUMENTATION

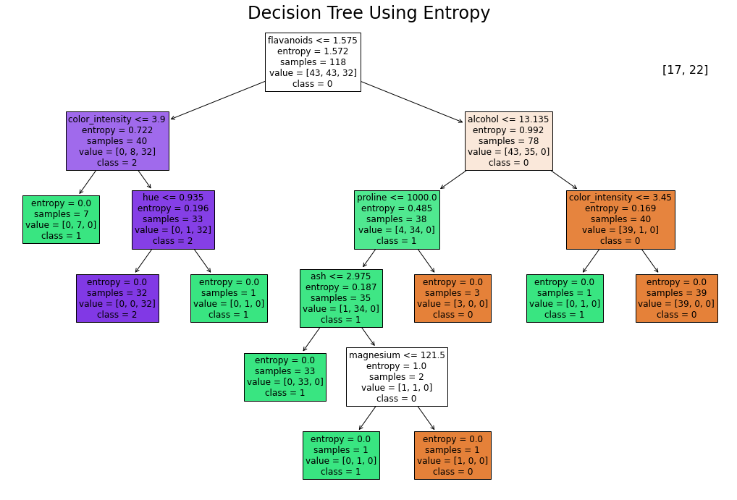
A blue squares with numbers and a blue square

Description automatically generatedIn the implementation of Principal Component Analysis (PCA) using Python, the following libraries and functions were utilized. Numpy, a powerful library for numerical computing, was employed for vector and matrix operations, providing efficient computation capabilities. Pandas, a popular data manipulation library, was used for storing and handling data in the form of dataframes. Scikit-learn, a comprehensive machine learning library, contributed the StandardScaler class for scaling the data, ensuring that all features have similar ranges. The np.linalg.eig function from Numpy was utilized to calculate the eigenvalues and eigenvectors, crucial components of PCA. Lastly, the PCA class from scikit-learn was employed to compare the results against the hand-coded implementation, facilitating a convenient and validated approach to PCA analysis. Also, for the visualization of the results using various 2D and 3D plots matplotlib and plotly were used.

RESULTS AND ANALYSIS

A. DECISION TREE USING ENTROPY

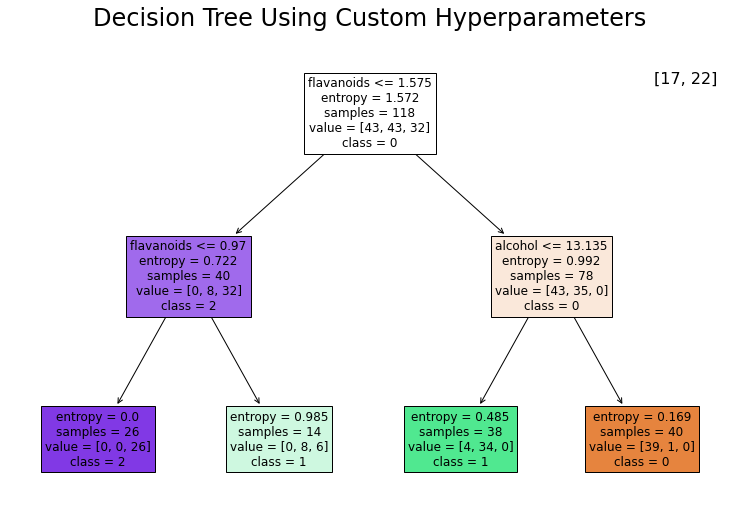
A screenshot of a computer

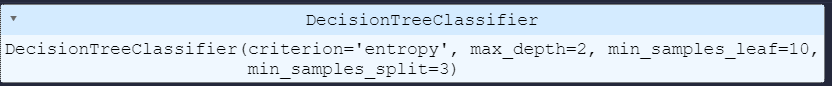
Description automatically generated

A graph with a bar graph

Description automatically generated

B. DECISION TREE USING CUSTOM HYPERPARAMETERS





A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

TABLE I

Explained variance for iris dataset

|  |  |  |
| --- | --- | --- |
| Principal Component | Explained Variance | Approx. (in %) |
| PC1 | 0.7296 | 72.96 |
| PC2 | 0.2285 | 22.85 |
| PC3 | 0.03669 | 3.66 |
| PC4 | 0.00517 | 0.51 |

TABLE II

Explained variance for diabetes prediction dataset

|  |  |  |
| --- | --- | --- |
| Principal Component | Explained Variance | Approx. (in %) |
| PC1 | 0.2150 | 21.5 |
| PC2 | 0.1380 | 13.8 |
| PC3 | 0.1322 | 13.22 |
| PC4 | 0.1196 | 11.96 |
| PC5 | 0.1099 | 10.99 |
| PC6 | 0.1056 | 10.56 |
| PC7 | 0.1041 | 10.41 |
| PC8 | 0.0752 | 7.52 |

C. DECISION TREE TUNING THE PRUNING VALUE

A graph with a line

Description automatically generated

C. DECISION TREE TUNING THE PRUNING VALUE

A computer screen shot of text

Description automatically generated

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A group of colorful squares with white text

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A blue squares with white text

Description automatically generated

A graph with blue bars

Description automatically generated

DISCUSSION AND ANALYSIS

This report includes the process of implementing PCA from scratch. The findings of this report have a strong impact on giving us in-depth intuition and insights on the workings of the algorithms used for PCA. The PCA from this report can be used for dimensionality reduction which is an important method for visualization and analysis of the data. During the analysis of plots for PCA for a randomly generated dummy dataset we notice that the at first the random generated from standard normal distribution had no correlation but after multiplying it with unform distribution matrix correlation was observed. This is because multiplying the matrix causes the linear combination of original variables. Similarly, PCA was applied on the Iris and Diabetes dataset and for Iris dataset adequate information was found to be found while reducing to two dimensions and the classes were totally separated after three dimensions. For the Diabetes Detection dataset, we found that even three dimensions could not retain all the necessary information about the dataset making the classes overlap in some cases.

The PCA from scikit-learn utilizes efficient linear algebra libraries resulting in excellent performance even for very large datasets. The PCA provided by the scikit learn library is also faster and efficient due to various optimization and parallelization techniques and can be implemented in a few lines of code with additional flexibility. Whereas the custom implementation of PCA requires extensive knowledge of the internal workings of the algorithm and the process is slower due to the lack of proper optimization.

CONCLUSION

In conclusion we implemented Principal Component Analysis (PCA) from scratch on three different datasets: Randomly generated Dummy Dataset, Iris Dataset and Diabetes Detection Dataset and gained valuable intuition and insights on the concept of dimensionality reduction using PCA. The comparison between our custom PCA and the PCA provided by the scikit -learn library showed that the mathematics involved in the implementation of PCA in scikit-learn library is slightly different from our own implementation as change in the sign of the eigen vectors were found to be present in the eigen vectors from the scikit-learn PCA causing the plots of the graph to reflect around some axis.

Therefore, our project implements PCA from scratch giving us the theoretical and mathematical intuition about a widely used process for dimensionality reduction and benefits as well as the limitations of PCA in representing the information present in the data. Also implementing PCA from scratch gave us valuable insights and intuition about the inner workings of the algorithm. By applying PCA on various datasets, we were able to reduce the dimensions of complex datasets while retaining the most important information. This dimensionality reduction enabled us to visualize the data in a lower-dimensional space, making it easier to interpret and understand. PCA facilitated the extraction of relevant information from the datasets, enabling us to make well-informed decisions by utilizing the transformed data.

REFERENCES

[1] J. Shlens, “A Tutorial on Principal Component Analysis,” Apr. 2014, [Online]. Available: http://arxiv.org/abs/1404.1100

[2] “The Iris Dataset — scikit-learn 1.3.0 documentation.” https://scikit-learn.org/stable/auto\_examples/datasets/plot\_iris\_dataset.html (accessed Jul. 03, 2023).

A person with glasses smiling

Description automatically generated**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.

A person in a suit

Description automatically generated with medium confidence**NIKHIL PRADHAN.**

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,"min\_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)