Project 2. Decision Tree

Artificial Intelligence

CS420

Group's member:

Le Van Cuong - 22125013

Nguyen Minh Quan - 22125079

Bui Danh Nghe - 22125063

Phung Khanh Vinh - 22125122

Contents

1 Task Distribution

Table 1: Task Distribution & Completion Rate

	(22125013)	(22125063)	(22125079)	(22125122)
Dataset (10% Total)	100%			
Report (30% Total)		100%	100%	100%
Notebook Framework (10% Total)	100%			
Notebook 1 (5% Total)			100%	
Notebook 2 (5% Total)				100%
Notebook 3 (10% Total)		100%		
Proofread + Notebook test (10% Total)	100%	100%		
Analysis + Interpretation (20% Total)			100%	100%
Total	100%	100%	100%	100%

2 Important stuff

2.1 Notebook

You can read the attachment notebook with this report or access them through this link on Google Colab.

Dataset 1: https://colab.research.google.com/drive/1AlRs7HZAwWVG-DvkRGYF0tMCd-n_kptC?usp=sharing

Dataset 2: https://colab.research.google.com/drive/1I_DwqVZVO62fkMzP9tBxE2MB7hEWgyPu?usp=sharing

Dataset 3: https://colab.research.google.com/drive/1dSyxHdpu1AzIx_MrOVBF1bINJbxGbxKW?usp=sharing

2.2 Dataset

Our custom dataset is created by us and can be accessed through this link.

Dataset 3: https://drive.google.com/file/d/108GPZ6REDp5EQi5r4-QszNqxSdAC6Ksz/view?usp=drive_link

The dataset above is created using a the tool modified by us that can be accessed through this link.

Tool for dataset creation: https://drive.google.com/file/d/1gcNcJ2FiifAvEHoz9-IndVf4mNg2NhM1/view?usp=sharing

You can read the description of the tool here, but when you use it, please run our version, the data set will be downloaded as a .csv file to your machine.
https://github.com/NetroScript/TeamFortressMatchHistoryAnalyzer

3 Introduction

This report details our development and implementation of decision tree models to analyze and classify data across various real-world datasets. The task involves building decision tree using scikit-learn.
The binary classification data set focuses on the Wisconsin UCI Breast Cancer Data Set (diagnostic), which aims to distinguish between malignant and benign tumors based on imaging characteristics. The multi-class dataset employs the UCI Wine Quality dataset, which classifies wine samples into quality levels using physicochemical attributes such as acidity and alcohol content. Additionally, a third data set has been curated, and will be analyze upon in this report.

4 Self-evaluation

We successfully implemented and tested the decision tree for all three dataset with clear explanation and analysis for each of them. Moreover, instead of using an existing dataset, we have created our own dataset using the information we collected ourself. Hopefully, this can also be a source of data that can be utilized in the future.

Accomplishments

- Meet all project objectives
- Finish all notebook with user guide
- A interesting dataset that can be reuse in the future

Possible Improvement

- The dataset, while is taken from real life may have few practical use and may be more suitable as data science or statistical analysis problem.

5 The UCI Breast Cancer Wisconsin (Diagnostic) dataset

5.1 Brief Overview

The UCI Breast Cancer Wisconsin (Diagnostic) dataset is a widely used binary classification dataset aimed at distinguishing between malignant (M) and benign (B) tumors. It contains 569 samples, each characterized by 30 numerical features derived from digitized images of fine needle aspirate (FNA) tests of breast tissue. These features capture details such as texture, perimeter, smoothness, and symmetry of the cell nuclei.
Key aspects of the dataset include:

- **Number of Samples**: 569
- **Classes**: Two (Malignant and Benign) (B or M)
- **Features**: 30 continuous attributes
- **Class Distribution**: Slightly imbalanced, with more benign cases than malignant.

This dataset is an excellent benchmark for evaluating classification algorithms due to its real-world relevance, clean structure, and manageable size.

5.2 Note

The result of each cell is shown in the notebook, if you don't need to re-run the notebook, this section can be ignored.
The process of making this notebook took quite a while, we had encountered multiple instances of the ics server dying. The last iteration has been run by the dataset in the archive.ics.uci.edu, however, if you rerun this code and find the server dying, in the end of the notebook we provide a section to take the dataset from a zip file in google drive, run that section and ignore the import datasets section of 2.1. The rest can be execute as normal.

5.3 Dataset preparation

In this section, we detail the process of preparing the training and test datasets for our experiments. The UCI Breast Cancer Wisconsin (Diagnostic) dataset was used, with features and labels extracted

as described previously. The dataset was shuffled and split in a stratified manner to maintain the distribution of classes across all subsets.

The following subsets were created:

- **feature_train**: A set of training samples.
- **label_train**: A set of labels corresponding to the samples in **feature_train**.
- **feature_test**: A set of test samples with a structure same as **feature_train**.
- **label_test**: A set of labels corresponding to the samples in **feature_test**.

Training and test sets were prepared in four different proportions and group together: 40/60, 60/40, 80/20, and 90/10 (train/test). This resulted in a total of 16 subsets group into 4 group:

- **40/60 Split**
- **60/40 Split**
- **80/20 Split**
- **90/10 Split**

The dataset in shuffle randomly in order to ensure the dataset is split in a stratified fashion.

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt

label_encoder = LabelEncoder()
labels = label_encoder.fit_transform(labels)

# Function to create stratified train/test splits
def create_splits(features, labels, test_sizes):
    splits = []
    for test_size in test_sizes:
        features_train, features_test, labels_train, labels_test =
   train_test_split(
```

```
            features, labels, test_size=test_size, stratify=labels, random_state=42
        )
        splits.append((features_train, features_test, labels_train, labels_test))
    return splits

# Define train/test proportions
test_sizes = [0.6, 0.4, 0.2, 0.1]
splits = create_splits(features, labels, test_sizes)

# Function to visualize class distributions
def plot_class_distribution(labels, title):
    unique, counts = np.unique(labels, return_counts=True)
    class_names = ['B','M']
    plt.bar(class_names, counts, color=['blue', 'orange'])
    plt.title(title)
    plt.xlabel("Class")
    plt.ylabel("Count")
    plt.show()

# Plot class distribution for the original dataset
plot_class_distribution(labels, "Original Dataset Class Distribution")
```

The distribution of the classes are as followed:

40/60 Split:

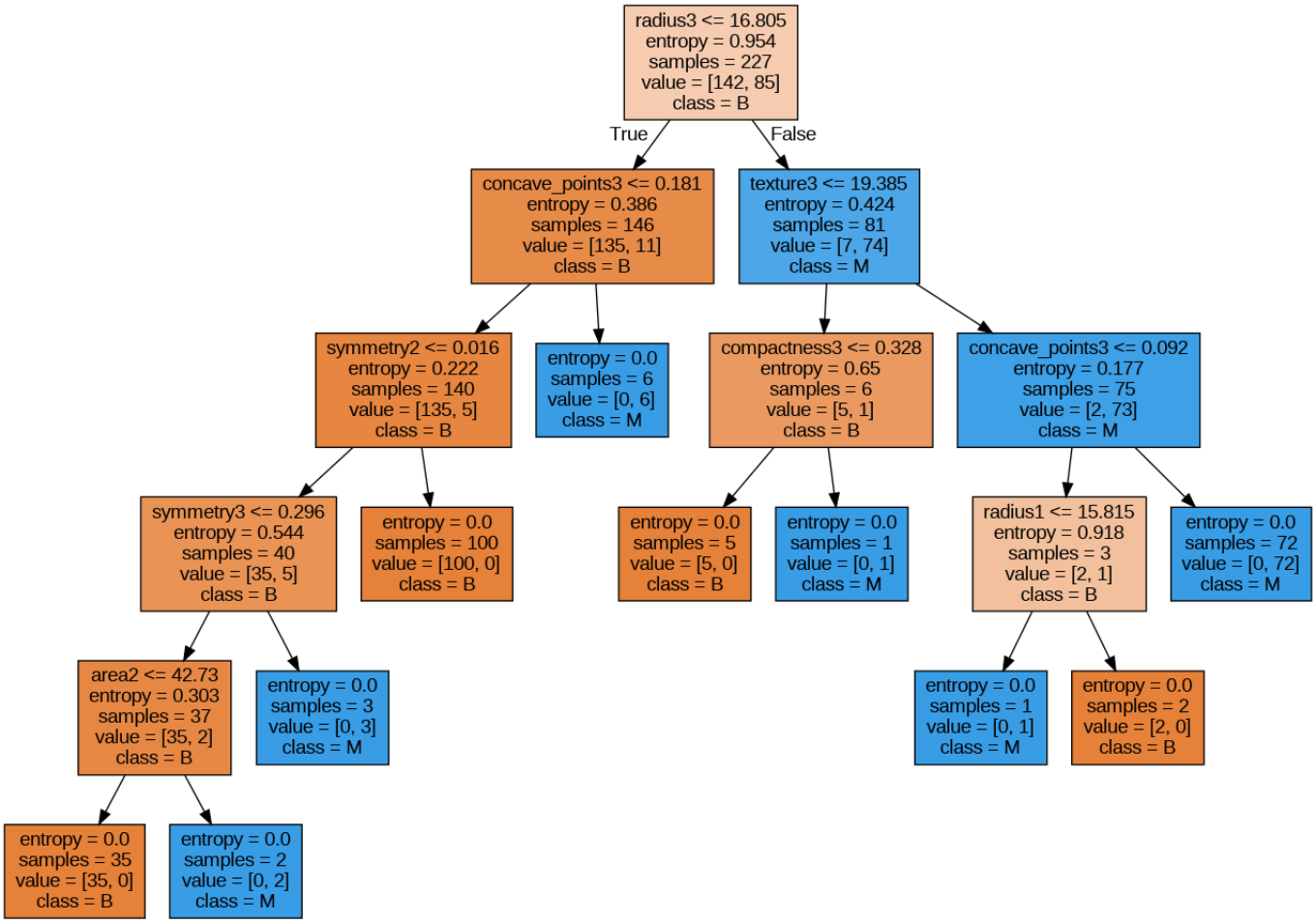

Figure 1: Decision tree for the 40/60 Dataset

The implementation of the process is as follow:

Original Dataset

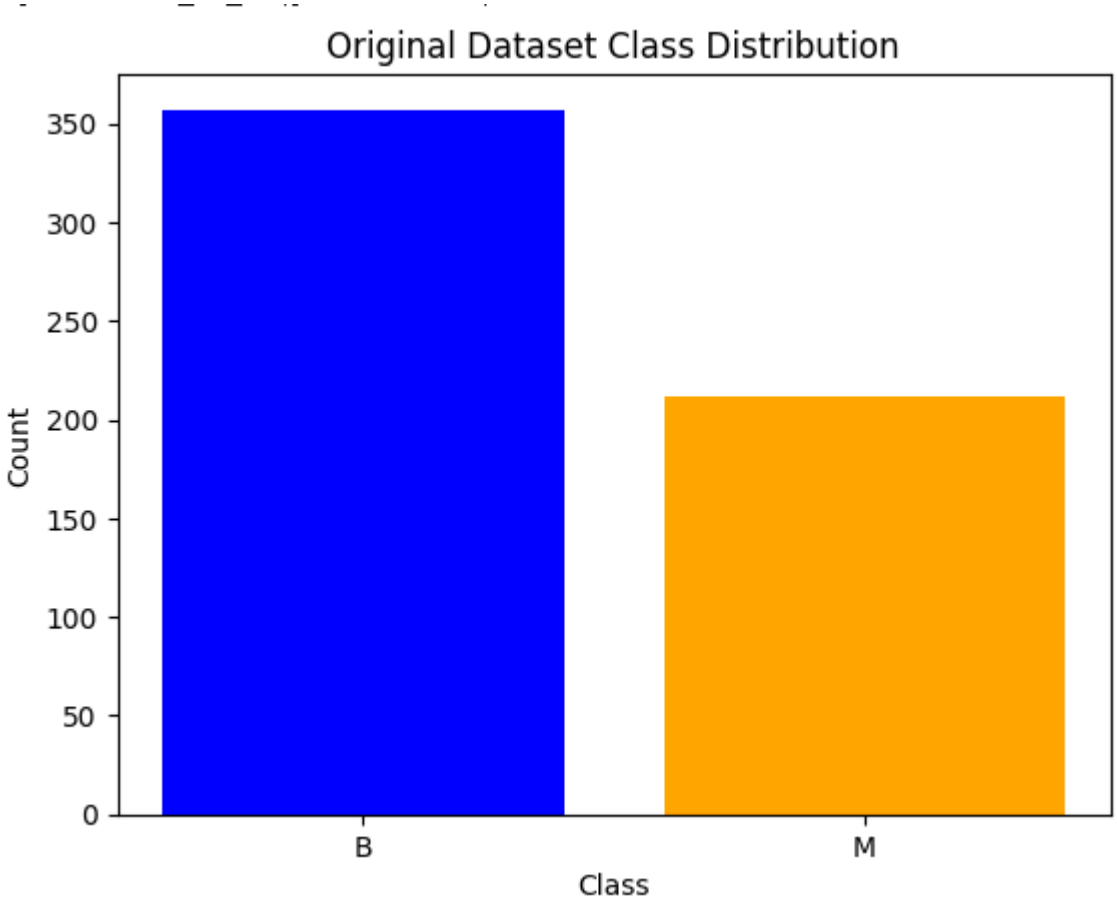

Figure 2: Class distribution of the original dataset

Training set 40/60 split

Figure 3: Class distribution of the 40/60 split training set

Testing set 40/60 split

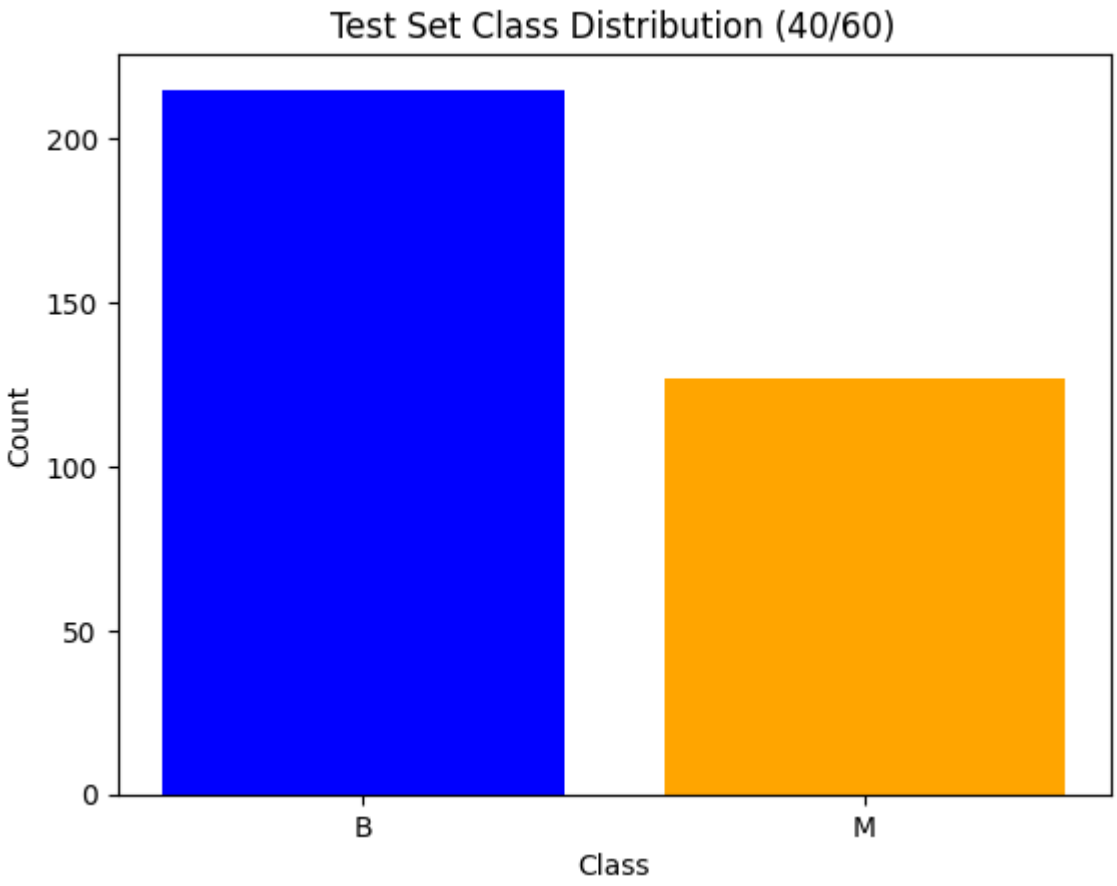

Figure 4: Class distribution of the 40/60 split testing set

Training set 60/40 split

Figure 5: Class distribution of the 60/40 split training set

Testing set 60/40 split

Figure 6: Class distribution of the 60/40 split testing set

Training set 80/20 split

Figure 7: Class distribution of the 80/20 split training set

Testing set 80/20 split

Figure 8: Class distribution of the 80/20 split testing set

Training set 90/10 split

Figure 9: Class distribution of the 90/10 split training set

Testing set 90/10 split

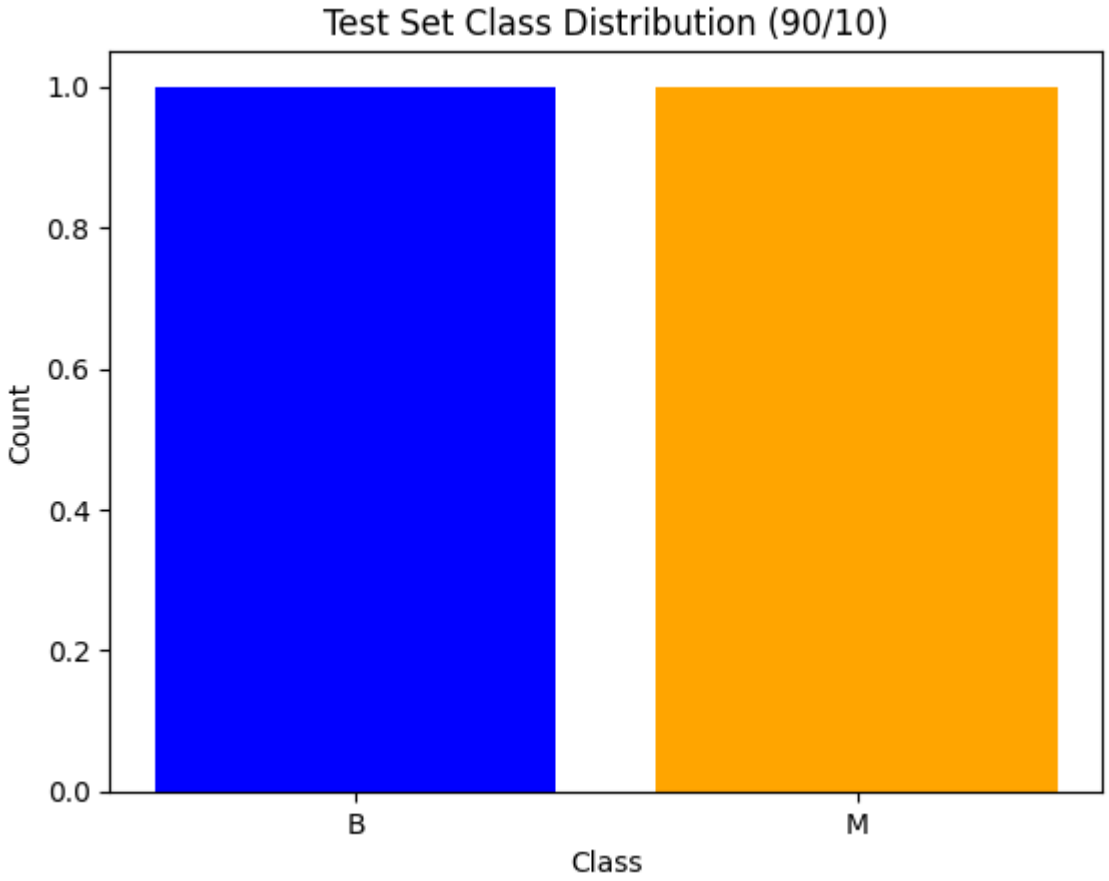

Figure 10: Class distribution of the 90/10 split testing set

5.4 Building the Decision Tree Classifiers

This task involved conducting experiments on the train/test proportions prepared earlier (40/60, 60/40, 80/20, and 90/10). For each training set, an instance of `sklearn.tree.DecisionTreeClassifier` was fitted using information gain.

The resulting decision trees were visualized using Graphviz for each experiment. These visualizations provide a clear understanding of the tree structure and the decision rules derived from the data.

The decision trees of the data set are as follow, a higher resolution image can be obtain in the notebook:

40/60 Split:

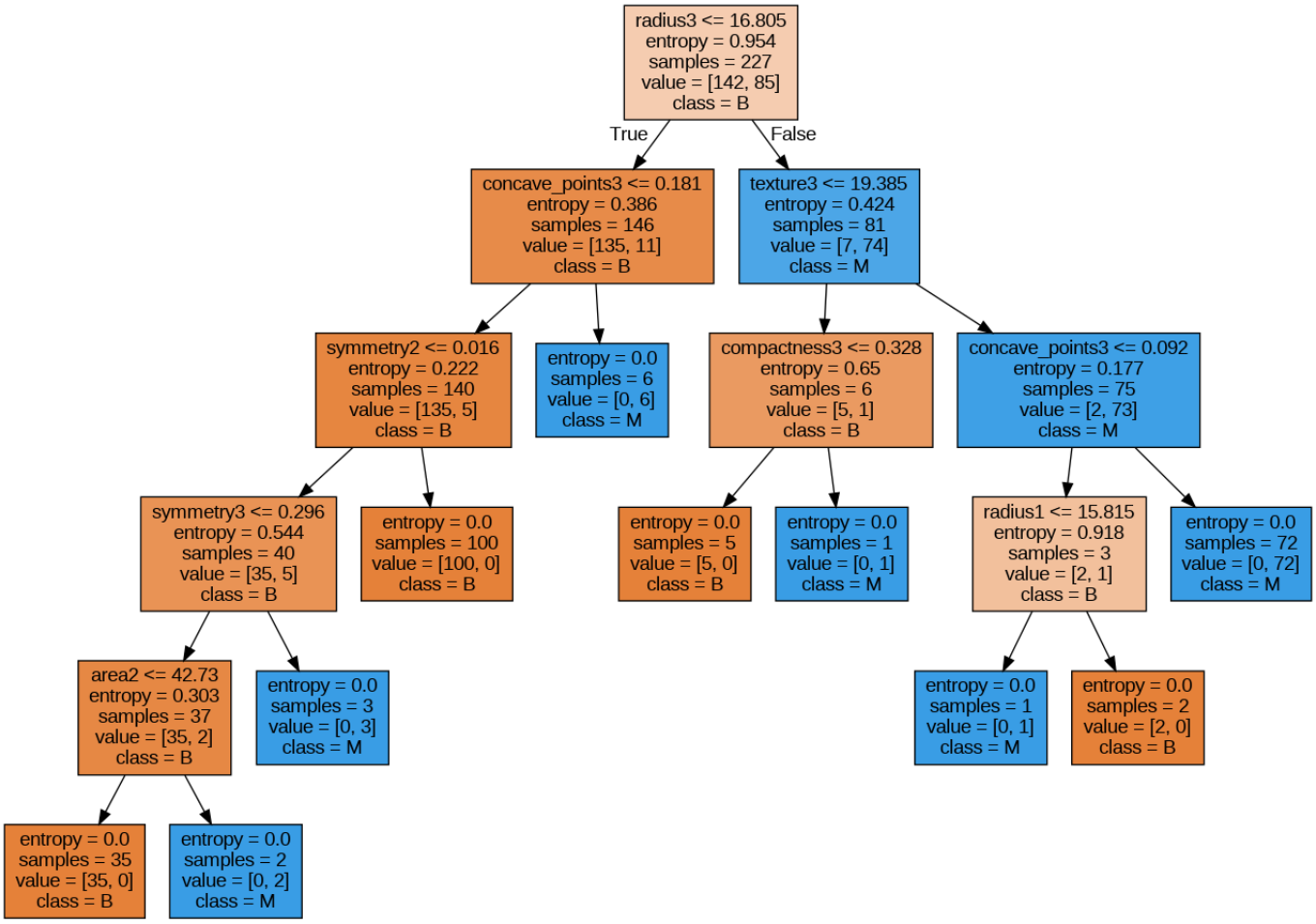

Figure 11: Decision tree for the 40/60 Dataset

60/40 Split:

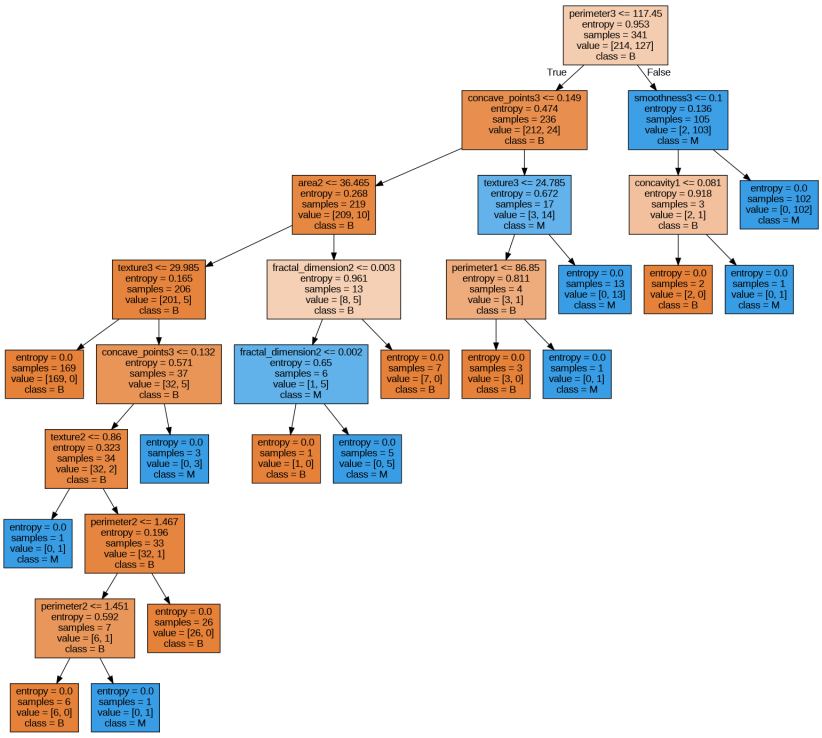

Figure 12: Decision tree for the 60/40 Dataset

80/20 Split:

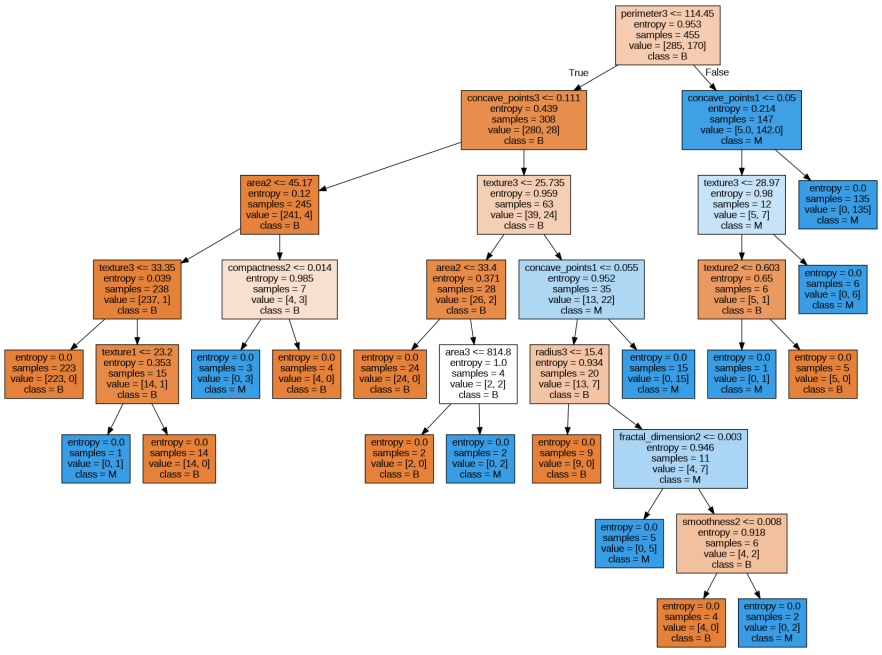

Figure 13: Decision tree for the 80/20 Dataset

90/10 Split:

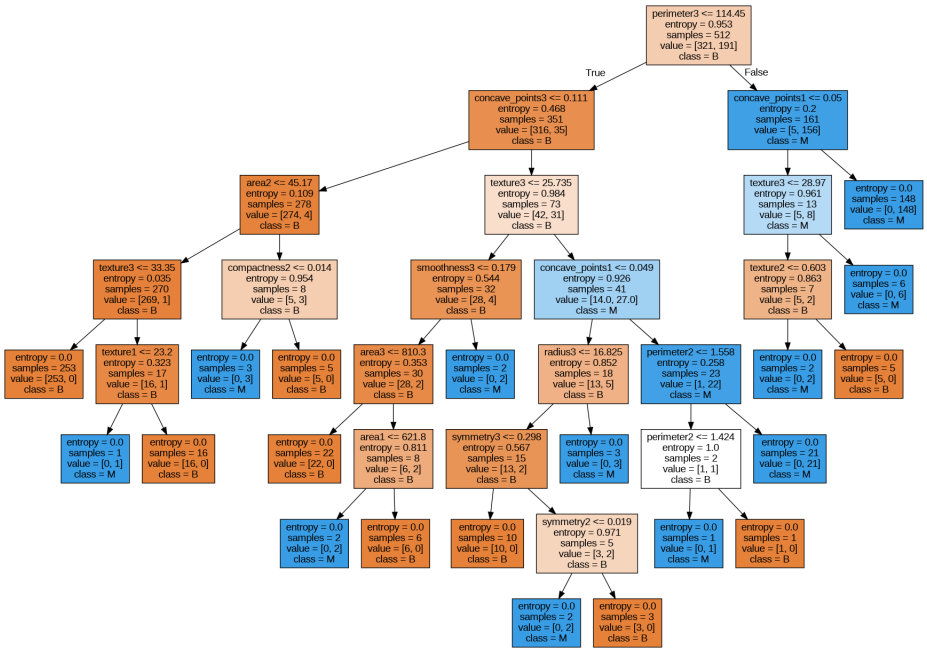

Figure 14: Decision tree for the 90/10 Dataset

5.5 Evaluating the Decision Tree Classifiers

The classification report and confusion matrix for the split are as followed:

40/60 Split:

	precision	recall	f1-score	support
B	0.91	0.96	0.93	215
M	0.92	0.83	0.88	127
accuracy			0.91	342
macro avg	0.91	0.90	0.90	342
weighted avg	0.91	0.91	0.91	342

Figure 15: Classification report for the 40/60 Dataset

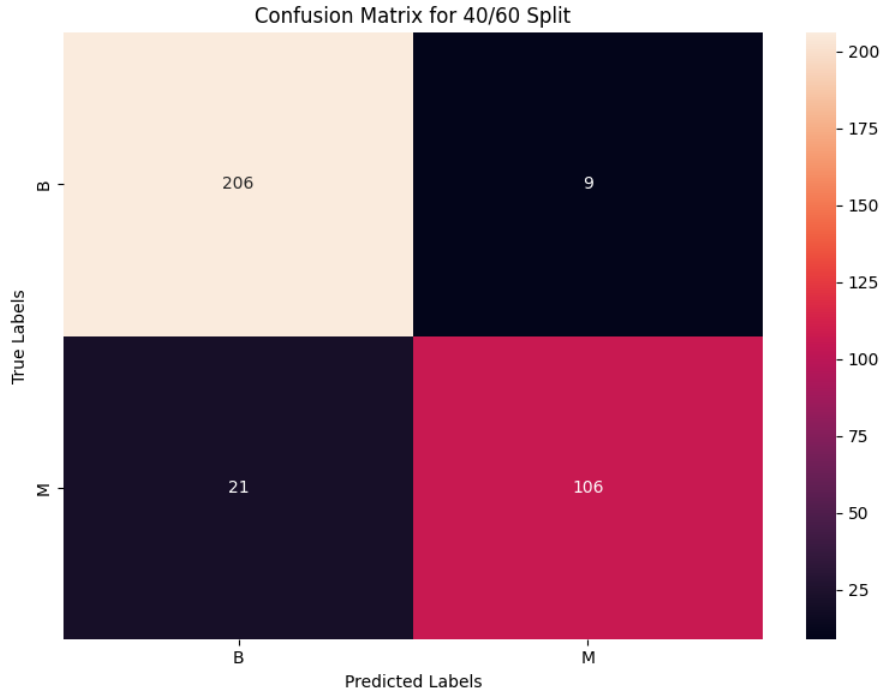

Figure 16: Confusion matrix for the 40/60 Dataset

60/40 Split:

Classification Report for {'60/40'} Split :

	precision	recall	f1-score	support
B	0.94	0.96	0.95	143
M	0.93	0.91	0.92	85
accuracy			0.94	228
macro avg	0.94	0.93	0.93	228
weighted avg	0.94	0.94	0.94	228

Figure 17: Classification report for the 60/40 Dataset

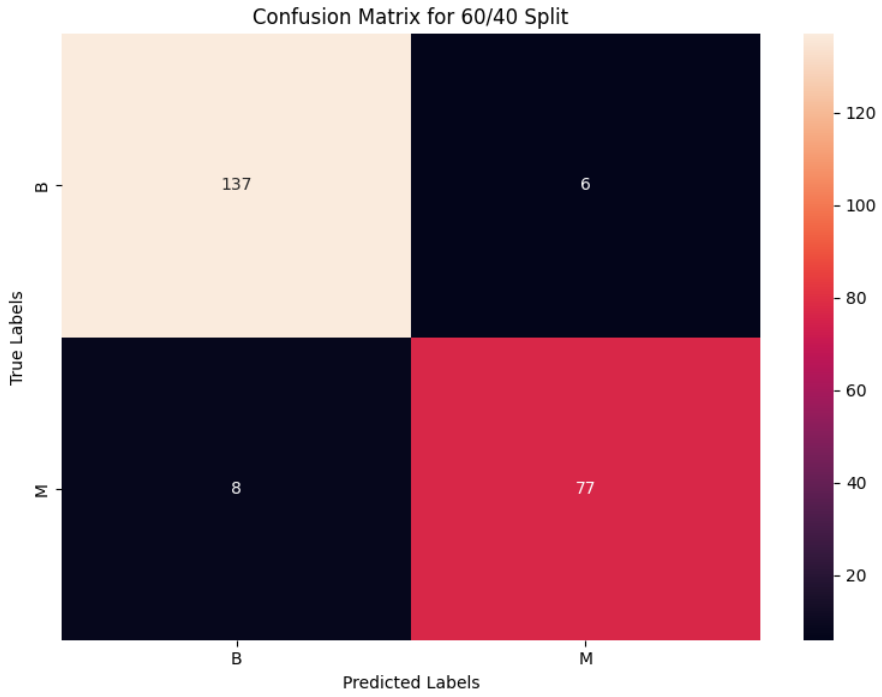

Figure 18: Confusion matrix for the 60/40 Dataset

80/20 Split:

Classification Report for {'80/20'} Split :

	precision	recall	f1-score	support
B	0.95	0.99	0.97	72
M	0.97	0.90	0.94	42
accuracy			0.96	114
macro avg	0.96	0.95	0.95	114
weighted avg	0.96	0.96	0.96	114

Figure 19: Classification report for the 80/20 Dataset

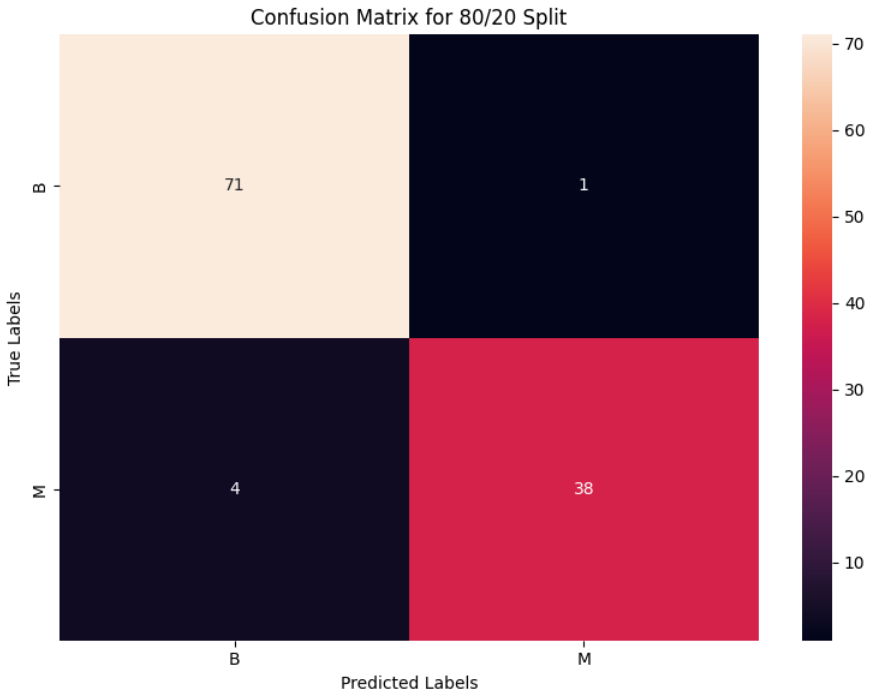

Figure 20: Confusion matrix for the 80/20 Dataset

90/10 Split:

Classification Report for {'90/10'} Split :

	precision	recall	f1-score	support
B	0.95	0.97	0.96	36
M	0.95	0.90	0.93	21
accuracy			0.95	57
macro avg	0.95	0.94	0.94	57
weighted avg	0.95	0.95	0.95	57

Figure 21: Classification report for the 90/10 Dataset

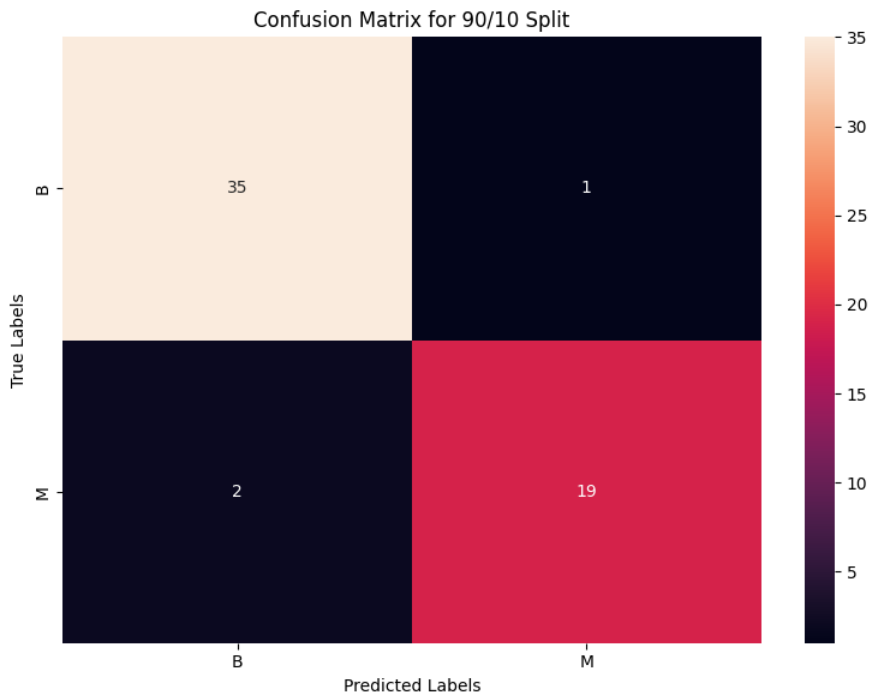

Figure 22: Confusion matrix for the 90/10 Dataset

5.5.1 Classifications report interpretation + insight

Overall, this confusion report show that the decision tree has high precision when predicting both classes. More than 90% for both, this show that the model has minimized the number of false positive.

However, the model has a lower recall, especially for malignant tumor where it drop to below 90% in split 40/60. While still is a high number, this infer that the model may occasionally misses malignant tumors (false negatives).

About F-1 score, for B: 0.97, and for M: 0.94, reflecting a strong balance between precision and recall for both classes. The slightly lower F1 score for M is due to its lower recall.

Overall accuracy is 96%, indicating that the model correctly classified 96% of all test samples. This high accuracy suggests the decision tree is well-suited for this dataset.

Averages precision, recall, and F1 scores equally across both classes. At 0.95, it confirms balanced performance. Weighted Average (weighted by support for each class): 0.96, further emphasizing good overall performance.

5.5.2 Confusion matrix interpretation + insight

From the confusion matrix, it seem that the model is slightly better at predicting benign tumor than malignant one. However, this is likely due to the fact that their is a different between the two classes distribution. Hence, with more benign tumor to learn from, it is likely the model will find it easier to identify benign tumor. Still, with more malignant tumor, this may pose a problem when using this model in practice as for malignant cases are of greater concern in a medical context since missing a malignant tumor could have serious consequences.

5.6 Depth and accuracy for decision tree of 80/20 split dataset

5.6.1 Decision tree

The decision tree is as followed for each depth:

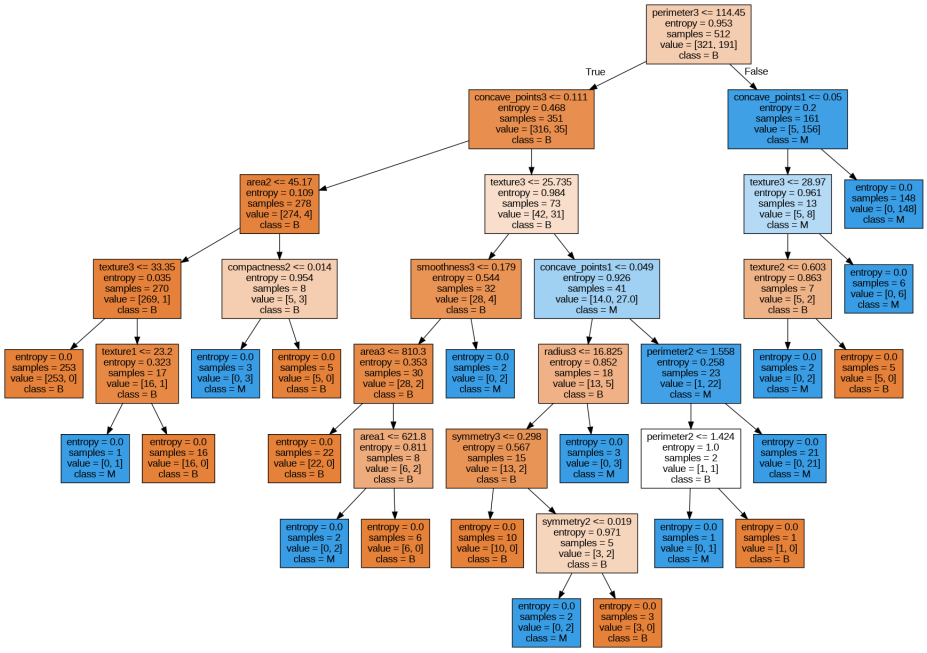

Figure 23: Decision tree for depth = None

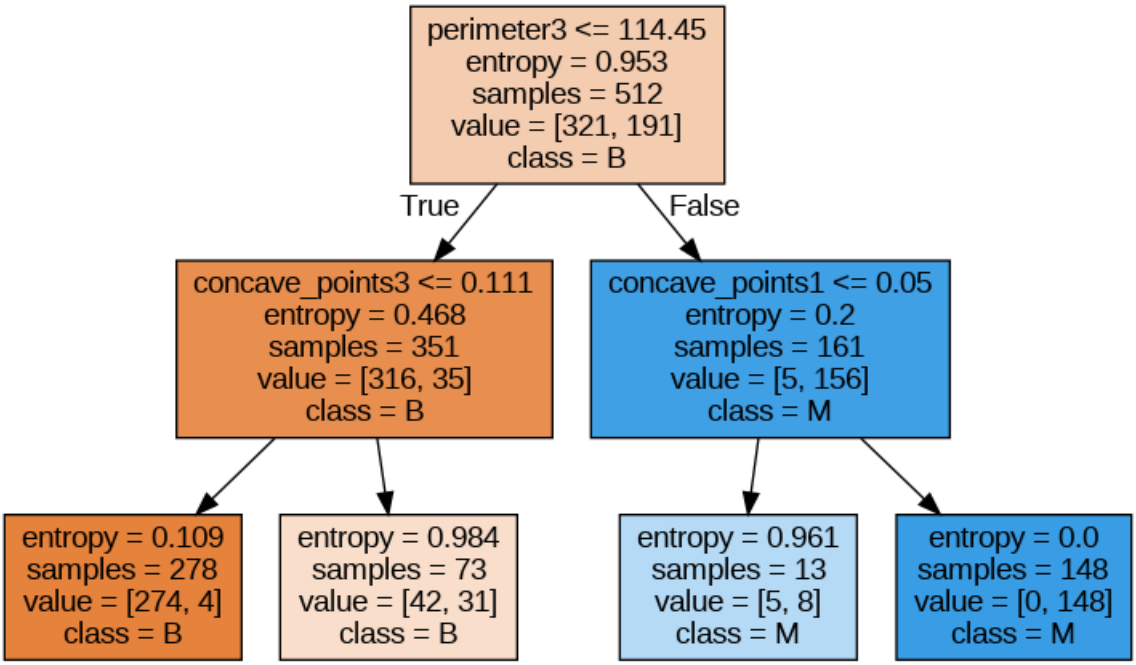

Figure 24: Decision tree for depth = 2

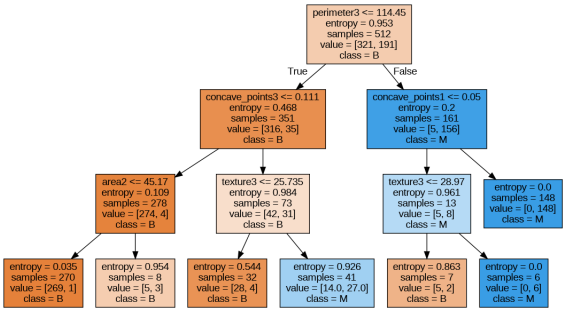

Figure 25: Decision tree for depth = 3

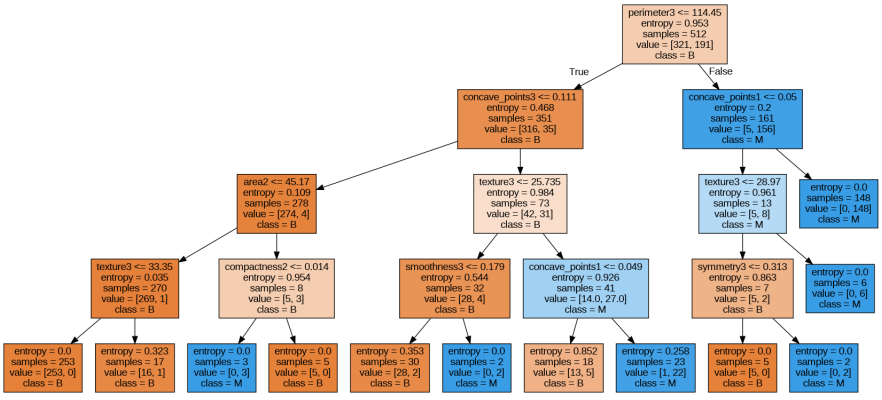

Figure 26: Decision tree for depth = 4

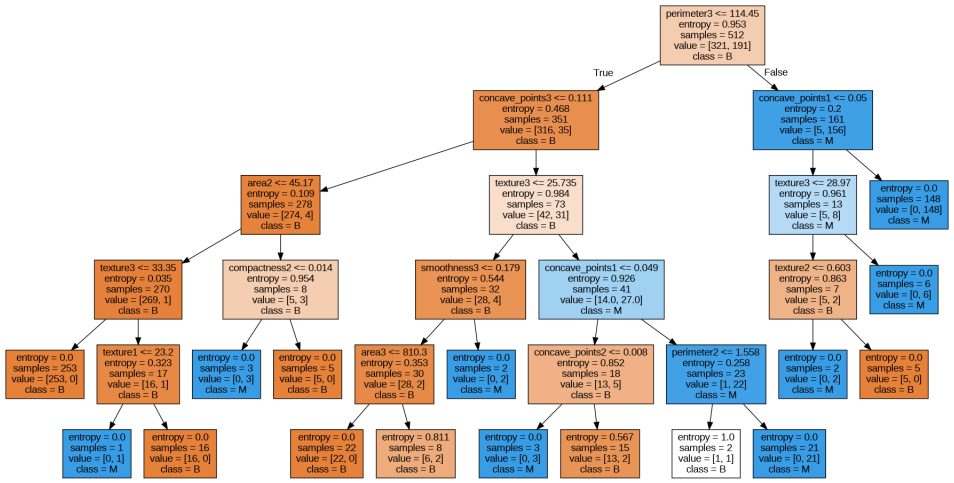

Figure 27: Decision tree for depth = 5

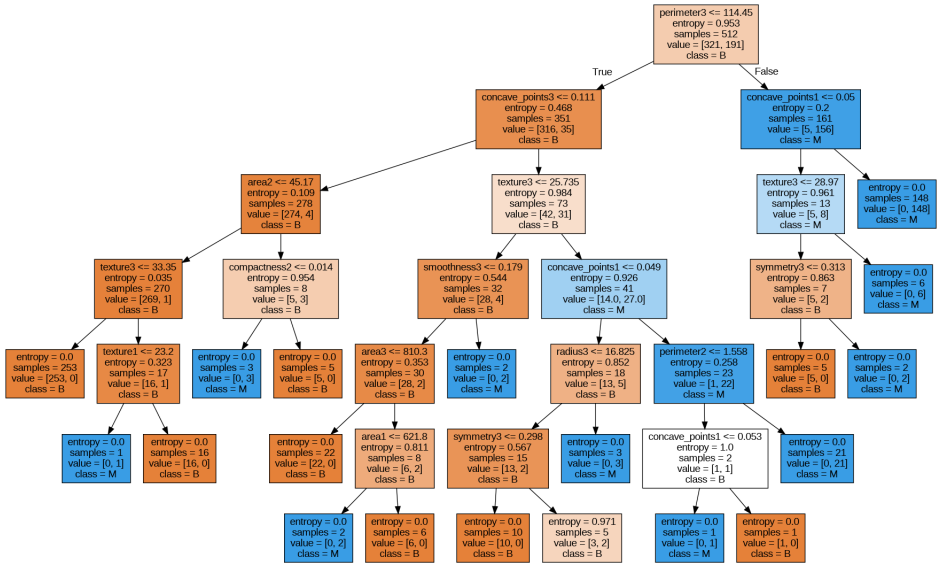

Figure 28: Decision tree for depth = 6

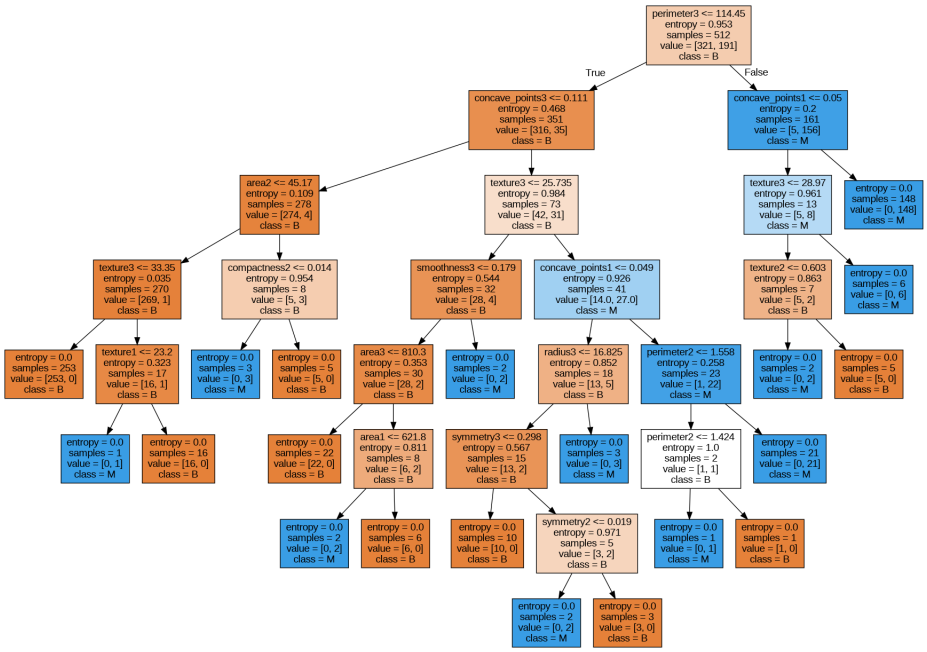

Figure 29: Decision tree for depth = 7

The result of the depth is as below:

Table 2: Accuracy of Decision Tree Classifier for Different max_depth Values for 80/20 Dataset

max_depth	None	2	3	4	5	6	7
Accuracy	0.9561	0.8859	0.9386	0.9298	0.9561	0.9298	0.9561

This can be visualize in this chart:

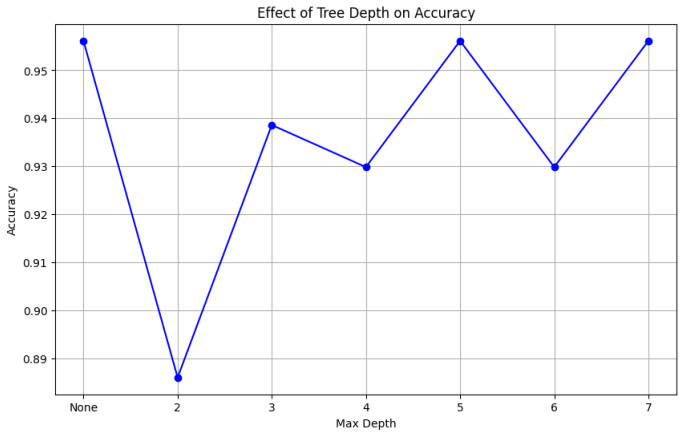

Figure 30: Accuracy of different decision tree depth on the 80/20 dataset

5.6.2 Insights on the Statistics Reported

The highest accuracy of **95.61%** was observed for *max_depth = None, 5, and 7*, indicating that deeper trees or no restriction on depth effectively capture the complexity of the dataset. The lowest accuracy of **88.60%** for *max_depth = 2* suggests that overly shallow trees lack sufficient complexity to make accurate predictions. Accuracy increases as depth grows from 2 to 3, demonstrating that adding more depth allows the model to better capture relationships in the data. However, there are slight fluctuations as depth increases further (e.g., **93.98%** for *depth = 4* vs. **95.61%** for *depth = 5*), likely due to overfitting on the training data for deeper trees. The consistent accuracy at *max_depth ¿= 5* suggests diminishing returns in predictive performance for increasing tree depth. Hence a tree of size of 3 or 4 is likely to be the most suitable tree depth for this problem. For practical application, this mean a relatively simple model can be used.

5.7 For other dataset:

For simplicity, we will show only the accuracies of the decision tree:

40/60 Split

Table 3: Accuracy of Decision Tree Classifier for Different max_depth Values for 40/60 Dataset

max_depth	None	2	3	4	5	6	7
Accuracy	0.9123	0.9327	0.9327	0.9123	0.9123	0.9123	0.9123

Figure 31: Accuracy of different decision tree depth on the 40/60 dataset

60/40 Split

Table 4: Accuracy of Decision Tree Classifier for Different max_depth Values for 60/40 Dataset

max_depth	None	2	3	4	5	6	7
Accuracy	0.9386	0.9298	0.9386	0.9254	0.9605	0.9430	0.9430

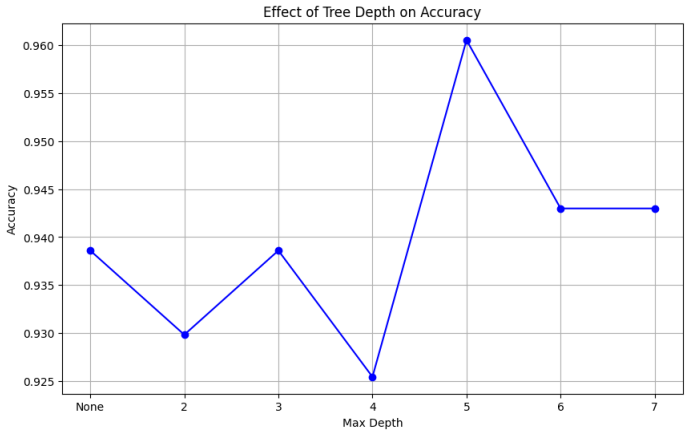

Figure 32: Accuracy of different decision tree depths on the 60/40 dataset

5.7.1 Ínterpretation + Insight

For most splits, the accuracy of the classifier is stable, this mean that increase the depth further may be unlikely to provide much more improvement.

For larger training dataset, it seem to provide better accuracy. However the performance tend to stagnate when *max_depth* is around 3 to 5, further reinforcing our analysis above.

6 The UCI Wine Quality dataset

6.1 Brief Overview

The UCI Wine Quality dataset is used for classifying wine samples into quality levels based on physicochemical properties such as acidity, alcohol content, etc. This dataset contains two datasets, one for red wine, and one for white wine, total to more than 6000 samples. Here, we will only care about white wine, which only have 4898 samples.

Key aspects of the dataset include:

- **Number of Samples**: 4898

- **Classes**: 11 levels of wine quality, ranging from 0 (lowest) to 10 (highest)
- **Features**: 11 continuous attributes, including properties like alcohol content, pH, and acidity
- **Class Distribution**: Imbalanced, with more wines categorized as average quality (5, 6, 7)

This dataset is commonly used in machine learning experiments for classification tasks and helps in the prediction of wine quality based on various chemical attributes.

6.2 Note

The result of each cell is shown in the notebook, if you don't need to re-run the notebook, this section can be ignored.
The process of making this notebook took quite a while, we had encountered multiple instances of the ics server dying. The last iteration has been run by the dataset in the archive.ics.uci.edu, however, if you rerun this code and find the server dying, in the end of the notebook we provide a section to take the dataset from a zip file in google drive, run that section and ignore the import datasets section of 2.1. The rest can be execute as normal.
Image size can also be a problem, this dataset created big decision tree, hence we have put guide on how to customize the size of the image in the notebook for your personal preferences with just one click.

6.3 Dataset preparation

6.3.1 Class grouping

For clarity and easier interpretation, the classes are group into 3 different group. Low quality (classes 0-4), Standard quality (classes 5-6), and High quality (classes 7-10). This will be our targets for analysis later.

6.3.2 Splitting dataset

In this section, we detail the process of preparing the training and test datasets for our experiments. The UCI Wine Quality dataset was used, with features and labels extracted as described previously.

The dataset was shuffled and split in a stratified manner to maintain the distribution of classes across all subsets.

The following subsets were created:

- **feature_train**: A set of training samples.
- **label_train**: A set of labels corresponding to the samples in **feature_train**.
- **feature_test**: A set of test samples with a structure same as **feature_train**.
- **label_test**: A set of labels corresponding to the samples in **feature_test**.

Training and test sets were prepared in four different proportions and group together: 40/60, 60/40, 80/20, and 90/10 (train/test). This resulted in a total of 16 subsets group into 4 group:

- **40/60 Split**
- **60/40 Split**
- **80/20 Split**
- **90/10 Split**

The dataset in shuffle randomly in order to ensure the dataset is split in a stratified fashion.

The implementation of the process is as follow:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt

label_encoder = LabelEncoder()
labels = label_encoder.fit_transform(labels)

# Function to create stratified train/test splits
def create_splits(features, labels, test_sizes):
    splits = []
    for test_size in test_sizes:
```

```
        features_train, features_test, labels_train, labels_test =
    train_test_split(
            features, labels, test_size=test_size, stratify=labels, random_state
    =42
        )
        splits.append((features_train, features_test, labels_train, labels_test)
    )
    return splits

# Define train/test proportions
test_sizes = [0.6, 0.4, 0.2, 0.1]
splits = create_splits(features, labels, test_sizes)

# Function to visualize class distributions
def plot_class_distribution(labels, title):
    unique, counts = np.unique(labels, return_counts=True)
    class_names = ['Low Quality', 'Standard  Quality', 'High Quality']
    plt.bar(class_names, counts, color=['blue', 'orange', 'red'])
    plt.title(title)
    plt.xlabel("Class")
    plt.ylabel("Count")
    plt.show()

# Plot class distribution for the original dataset
plot_class_distribution(labels, "Original Dataset Class Distribution")

# Visualize class distributions for each train/test split
#for i, (features_train, features_test, labels_train, labels_test) in enumerate(
    splits):
#    proportion = f"{int((1 - test_sizes[i]) * 100)}/{int(test_sizes[i] * 100)}"
#    plot_class_distribution(labels_train, f"Training Set Class Distribution ({
    proportion})")
#    plot_class_distribution(labels_test, f"Test Set Class Distribution ({
    proportion})")
```

The distribution of the classes are as followed:

Original Dataset

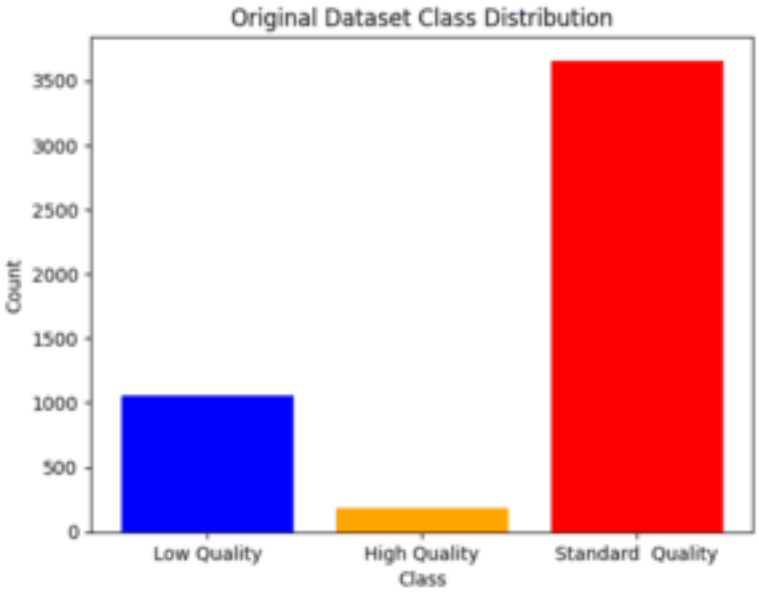

Figure 33: Class distribution of the original dataset

Training set and Testing set 40/60 split

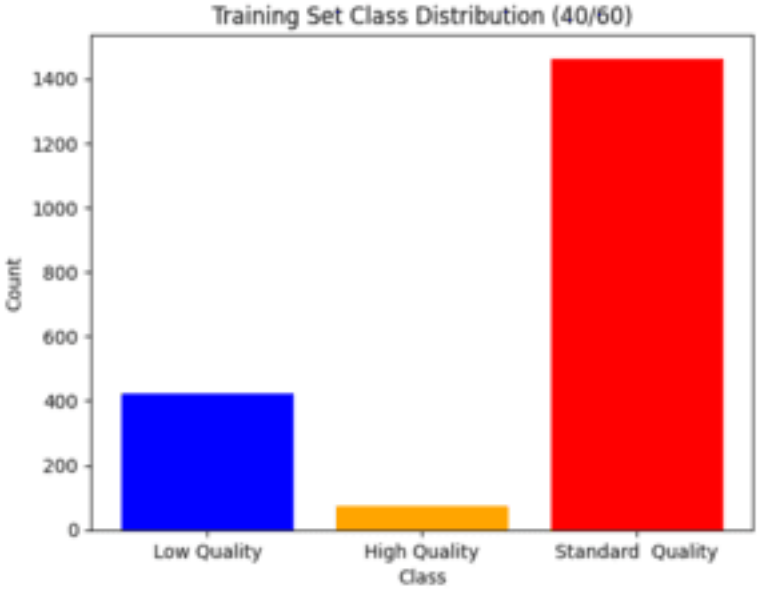

Figure 34: Class distribution of the 40/60 split training set and testing set

Training set and Testing set 60/40 split

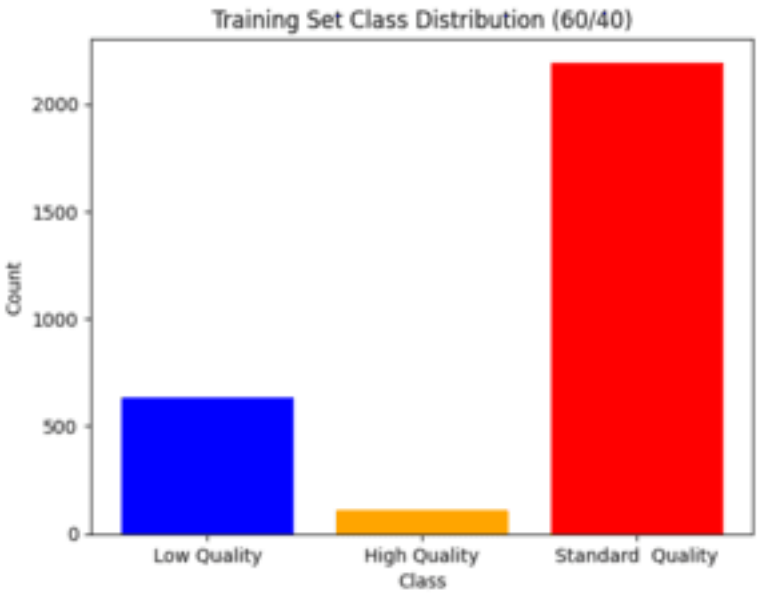

Figure 35: Class distribution of the 60/40 split training set and testing set

Training set and Testing set 80/20 split

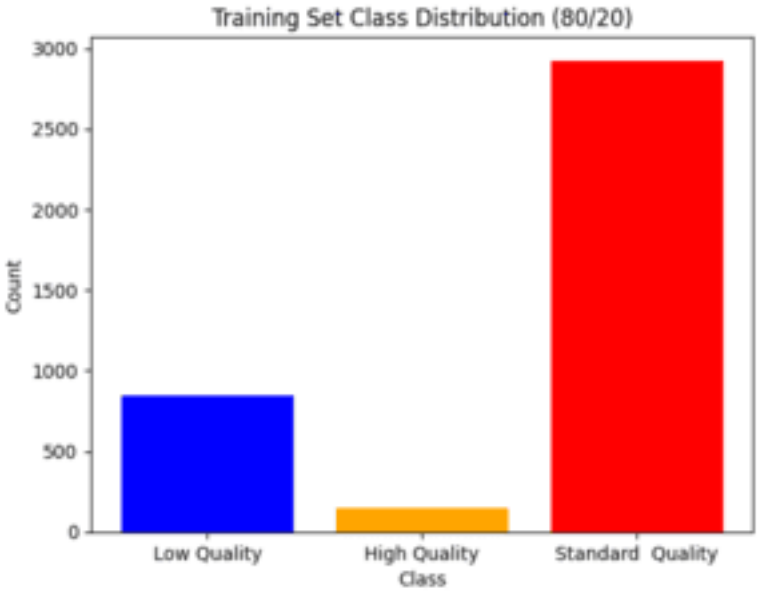

Figure 36: Class distribution of the 80/20 split training set and testing set

Training set and Testing set 90/10 split

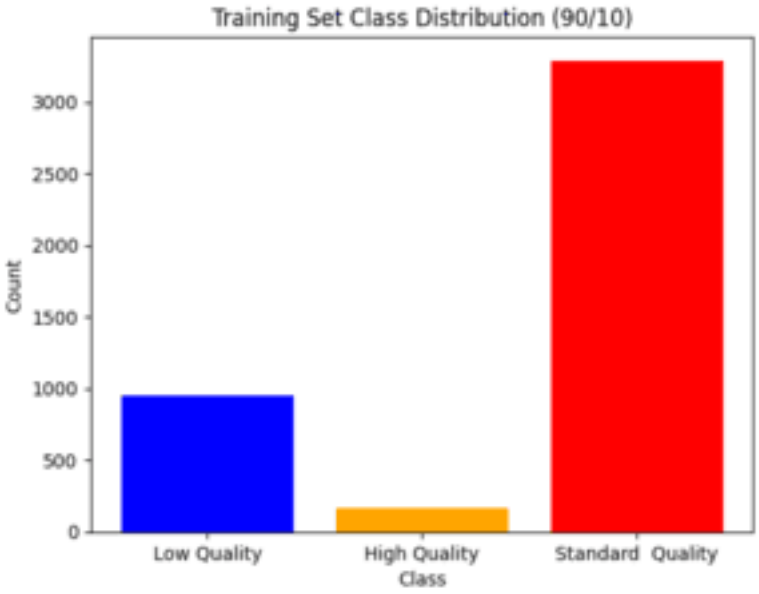

Figure 37: Class distribution of the 90/10 split training set and testing set

6.4 Building the Decision Tree Classifiers

This task involved conducting experiments on the train/test proportions prepared earlier (40/60, 60/40, 80/20, and 90/10). For each training set, an instance of `sklearn.tree.DecisionTreeClassifier` was fitted using information gain.

The resulting decision trees were visualized using Graphviz for each experiment. These visualizations provide a clear understanding of the tree structure and the decision rules derived from the data.

The decision trees of the data set are as follow, a higher resolution image can be obtain in the notebook:

40/60 Split:

Figure 38: Decision tree for the 40/60 Dataset

60/40 Split:

▫`img/Dataset2/decision_tree_6.png`

Figure 39: Decision tree for the 60/40 Dataset

80/20 Split:

▫img/Dataset2/decision_tree_8.png

Figure 40: Decision tree for the 80/20 Dataset

90/10 Split:

▫img/Dataset2/decision_tree_9.png

Figure 41: Decision tree for the 90/10 Dataset

6.4.1 Classification report Interpretation + Insight

The classification report for the decision tree model on the **UCI Wine Quality Dataset** reveals some important insights regarding the performance of the model on different wine quality classes.

- **Low Quality:** The model has a precision of 0.52, recall of 0.53, and an F1-score of 0.53. This indicates that the model is relatively balanced in its predictions for low-quality wines, but there is room for improvement. It does not perform exceptionally well, but it is reasonably effective at identifying low-quality wines.
- **High Quality:** The model performs poorly for high-quality wines, with a precision of 0.22, recall of 0.20, and F1-score of 0.21. These low values suggest that the decision tree is not particularly effective at identifying high-quality wines, likely due to the class imbalance, as there are fewer high-quality wines compared to low and standard-quality wines.
- **Standard Quality:** The model performs best for standard-quality wines, with a precision of 0.83, recall of 0.83, and F1-score of 0.83. This indicates strong performance, reflecting that the majority of wines in the dataset are of standard quality. The model is effective at identifying wines of this class.
- **Overall Accuracy:** The model achieves an accuracy of **74%**, meaning that it correctly classifies 74% of the wines across all classes. This indicates that while the decision tree does a decent job of classifying wines, there is still room for improvement, especially with the prediction of high-quality wines.
- **Macro Average:** The macro average for precision, recall, and F1-score is **0.52**, which reflects the model's overall performance across all classes without considering the class distribution.

This suggests that the model is performing poorly for some classes (e.g., high quality) while performing better for others (e.g., standard quality).

- **Weighted Average:** The weighted average precision, recall, and F1-score are **0.74**, indicating that, when accounting for the class imbalance (more standard-quality wines), the model performs fairly well. The weighted average reinforces that the model is more successful with the more frequent classes.

6.4.2 Confusion matrix interpretation + insight

From the confusion matrix, it seem that the model is struggling when trying to distinguish standard wine to lower and higher quality type, this is likely because of the class imbalance in this dataset where standard quality outnumbered the rest.

6.5 Depth and accuracy for decision tree of 80/20 split dataset

6.5.1 Decision tree

The result of the depth is as below:

Table 5: Accuracy of Decision Tree Classifier for Different max_depth Values for 80/20 Dataset

max_depth	None	2	3	4	5	6	7
Accuracy	0.7755	0.7459	0.7612	0.7724	0.7816	0.7867	0.7878

This can be visualize in this chart:

Figure 42: Accuracy of different decision tree depth on the 80/20 dataset

6.5.2 Insights on the Statistics Reported

The highest accuracy of 0.7878 was observed for depth = 7, indicating that deeper trees effectively capture the complexity of the dataset. The lowest accuracy of 0.7459 for depth = 2 suggests that overly shallow trees lack sufficient complexity to make accurate predictions. Accuracy improves as depth increases from 2 to 7, demonstrating that adding more depth helps the model capture more complex relationships in the data.

Thus, a tree depth of 6 or 7 appears to be the most suitable for this problem. This mean the tree will be quite complicated.

6.6 For Other Dataset:

For simplicity, we will show only the accuracies of the decision tree:

40/60 Split

Table 6: Accuracy of Decision Tree Classifier for Different max_depth Values for 40/60 Dataset

max_depth	None	2	3	4	5	6	7 height**Accuracy**
0.9123	0.9327	0.9327	0.9123	0.9123	0.9123	0.9123 height	

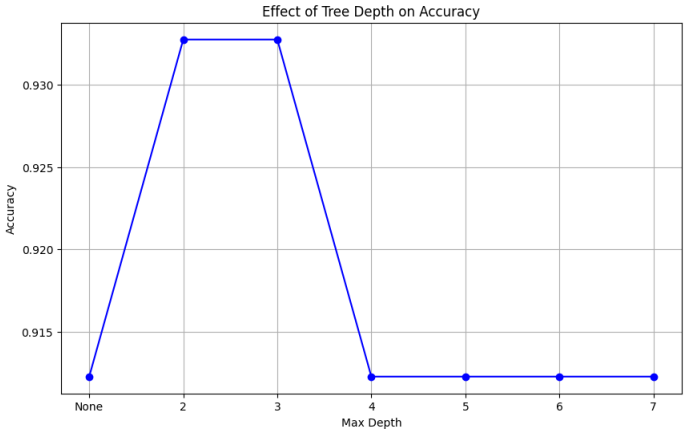

Figure 43: Accuracy of different decision tree depth on the 40/60 dataset

60/40 Split

Table 7: Accuracy of Decision Tree Classifier for Different $\max_{d}epthValuesfor60/40Dataset$

max_depth	None	2	3	4	5	6	7 height**Accuracy**
0.9386	0.9298	0.9386	0.9254	0.9605	0.9430	0.9430 height	

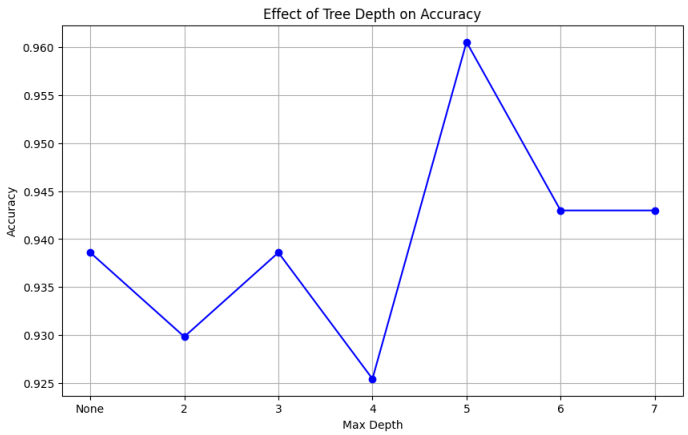

Figure 44: Accuracy of different decision tree depths on the 60/40 dataset

6.6.1 Interpretation + Insight

For most splits, the accuracy of the classifier is stable, indicating that further increases in the depth of the decision tree may not provide significant improvement in performance.
For larger training datasets, it seems to provide better accuracy, especially for the 60/40 split. However, the performance tends to stagnate when $max_depthisaround3to5, reinforcingouranalysisthatdeepertrees$
The accuracy of the decision tree for the 40/60 split remained steady across all depths, ranging from ***91.23%*** *to* ***93.27%****. For the 60/40 split, the accuracy improved slightly with deeper trees, with* ***96.05%*** *observed for* $max_depth = 5and$

6.7 Brief Overview

This is a dataset that is created by us, a member of our team (We will refer to him as Mr. C) is addicted to the game Team Fortress 2, created by Valve and release in 2007.
Key aspects of the dataset include:

- ***Number of Samples****: 3340*
- ***Attributes****: 48 attributes of many different type*
- ***Class Distribution****: The score seem to follow a normal distribution*

6.8 Team Fortress 2 Match History Attributes

6.8.1 Match Details

- ***conclusion***: *Indicates whether the match reached a conclusion (true for completed matches, false for abandoned).*
- ***is_comp***: *Specifies whether the match was played in competitive mode (true) or casual mode (false).*
- ***map_index***: *Numerical ID representing the map where the match was played.*
- ***creationtime***: *Timestamp when the match was created or started.*
- ***ip***: *The IP address of the game server hosting the match.*
- ***match_port***: *The port on the server used for the match.*
- ***datacenter***: *The physical or logical server location (data center) where the match was hosted.*

6.8.2 Match Participation Details

- ***match_size***: *Number of players involved in the match (e.g., 6v6, 12v12).*
- ***join_time***: *Timestamp of when the player joined the match.*
- ***party_id_at_join***: *Identifier for the party the player was in when they joined the match (if applicable).*
- ***team_at_join***: *The team the player was assigned to upon joining (e.g., RED or BLU).*
- ***ping_estimate_at_join***: *The estimated network latency (ping) when the player joined the match, measured in milliseconds.*
- ***joined_after_match_start***: *Indicates whether the player joined after the match had already started (true or false).*

6.8.3 Match Timing and Outcome

- ***time_in_queue****: The duration the player waited in the matchmaking queue, measured in seconds.*
- ***match_end_time****: Timestamp indicating when the match ended.*
- ***season_id****: Identifies the competitive season during which the match took place (for competitive matches).*
- ***match_status****: Numerical code representing the match's status (e.g., 0 for completed, 3 for abandoned).*
- ***match_duration****: The total duration of the match, measured in seconds.*

6.8.4 Team and Score Information

- ***red_team_final_score****: The final score of the RED team.*
- ***blu_team_final_score****: The final score of the BLU team.*
- ***winning_team****: Indicates which team won the match (e.g., RED, BLU, or DRAW).*
- ***game_mode****: Numerical or categorical identifier for the game mode (e.g., payload, capture the flag).*
- ***win_reason****: Numerical or textual code describing why the match concluded in favor of the winning team.*
- ***match_flags****: Special flags or tags associated with the match (e.g., test matches, bot matches).*
- ***match_included_bots****: Indicates the number of bots present in the match.*

6.8.5 Player-Specific Details

- ***time_left_match****: Timestamp when the player left the match (if they did before the conclusion).*
- ***result_partyid****: Identifier for the player's party at the conclusion of the match.*

- ***result_team****: The team the player was on at the end of the match (e.g., RED or BLU).*
- ***result_score****: The player's score at the end of the match.*
- ***result_ping****: The player's ping at the end of the match.*
- ***result_player_flags****: Flags describing special conditions for the player during the match.*
- ***result_displayed_rating****: The player's visible rank or rating at the end of the match (for competitive matches).*
- ***result_displayed_rating_change****: The change in the player's rating due to the match's outcome.*
- ***result_rank****: The player's rank in competitive mode (if applicable).*

6.8.6 Player Performance Details

- ***classes_played****: Binary-coded representation of the classes the player used during the match (e.g., Scout, Medic, etc.).*
- ***kills****: The number of kills the player achieved during the match.*
- ***deaths****: The number of times the player died during the match.*
- ***damage****: Total damage dealt by the player during the match.*
- ***healing****: Total healing provided by the player during the match.*
- ***support****: Total support points or other assistance actions (e.g., capturing objectives).*

6.8.7 Medals and Rewards

- ***score_medal****: The medal earned for the player's score (e.g., Bronze, Silver, Gold).*
- ***kills_medal****: The medal earned for the number of kills.*
- ***damage_medal****: The medal earned for the amount of damage dealt.*
- ***healing_medal****: The medal earned for the amount of healing provided.*
- ***support_medal****: The medal earned for support actions.*

6.8.8 Additional Match Information

- ***leave_reason***: *Code or description of why the player left the match early (if applicable).*
- ***connection_time***: *Timestamp of when the player connected to the game server.*
- ***id***: *Unique identifier for the match.*

This dataset is chosen in honor of Mr. C postponing our group project so that he can play more Team Fortress 2.

6.9 Note

Image size can also be a problem, this dataset created big decision tree, hence we have put guide on how to customize the size of the image in the notebook for your personal preferences with just one click.

6.10 Target and Features

As our purpose is to analyze how badly he played, we will focus on score as our target. As in Team Fortress 2, score is calculated from K/D/A healing, damage and support we will ignore the for a more interesting dataset. As such, we will analyze score base on these following external features.

- ***day_of_week***: *The day of the week when the match took place.*
- ***hour***: *The hour of the day when the match was played.*
- ***game_mode***: *The mode in which the game was played (e.g., payload, capture the flag).*
- ***result_ping***: *The player's ping at the end of the match.*
- ***result_displayed_rating***: *The visible rating or rank of the player at the conclusion of the match (for competitive matches).*

However as you may already noticed, score is largely affected by time played, hence, we will change our targets a little bit. We introduce this variables.
time_difference_seconds: *The time difference, in seconds, between a specific event or action in the match and another reference point which is match end times.*

Then we get our final target:

***performance**: A derived metric representing the player's performance, calculated as the ratio of the player's score (`result_score`) to the time difference in seconds (`time_difference_seconds`). For clarity and easier interpretation, the classes are group into 3 different group. Low Performance (classes 0-0.025), Standard quality (classes 0.025-0.055), and High quality (classes ¿ 0.055). This will be our targets for analysis later.*

6.11 Dataset splitting

The following subsets were created:

- ***feature_train**: A set of training samples.*
- ***label_train**: A set of labels corresponding to the samples in **feature_train**.*
- ***feature_test**: A set of test samples with a structure same as **feature_train**.*
- ***label_test**: A set of labels corresponding to the samples in **feature_test**.*

Training and test sets were prepared in four different proportions and group together: 40/60, 60/40, 80/20, and 90/10 (train/test). This resulted in a total of 16 *subsets group into* 4 *group:*

- ***40/60 Split***
- ***60/40 Split***
- ***80/20 Split***
- ***90/10 Split***

The dataset in shuffle randomly in order to ensure the dataset is split in a stratified fashion.

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt

label_encoder = LabelEncoder()
```

```
labels = label_encoder.fit_transform(labels)

# Function to create stratified train/test splits
def create_splits(features, labels, test_sizes):
    splits = []
    for test_size in test_sizes:
        features_train, features_test, labels_train, labels_test =
    train_test_split(
            features, labels, test_size=test_size, stratify=labels, random_state
    =42
        )
        splits.append((features_train, features_test, labels_train, labels_test)
    )
    return splits

# Define train/test proportions
test_sizes = [0.6, 0.4, 0.2, 0.1]
splits = create_splits(features, labels, test_sizes)

# Function to visualize class distributions
def plot_class_distribution(labels, title):
    unique, counts = np.unique(labels, return_counts=True)
    class_names = ['B','M']
    plt.bar(class_names, counts, color=['blue', 'orange'])
    plt.title(title)
    plt.xlabel("Class")
    plt.ylabel("Count")
    plt.show()

# Plot class distribution for the original dataset
plot_class_distribution(labels, "Original Dataset Class Distribution")
```

The distribution of the classes are as followed:

40/60 Split:

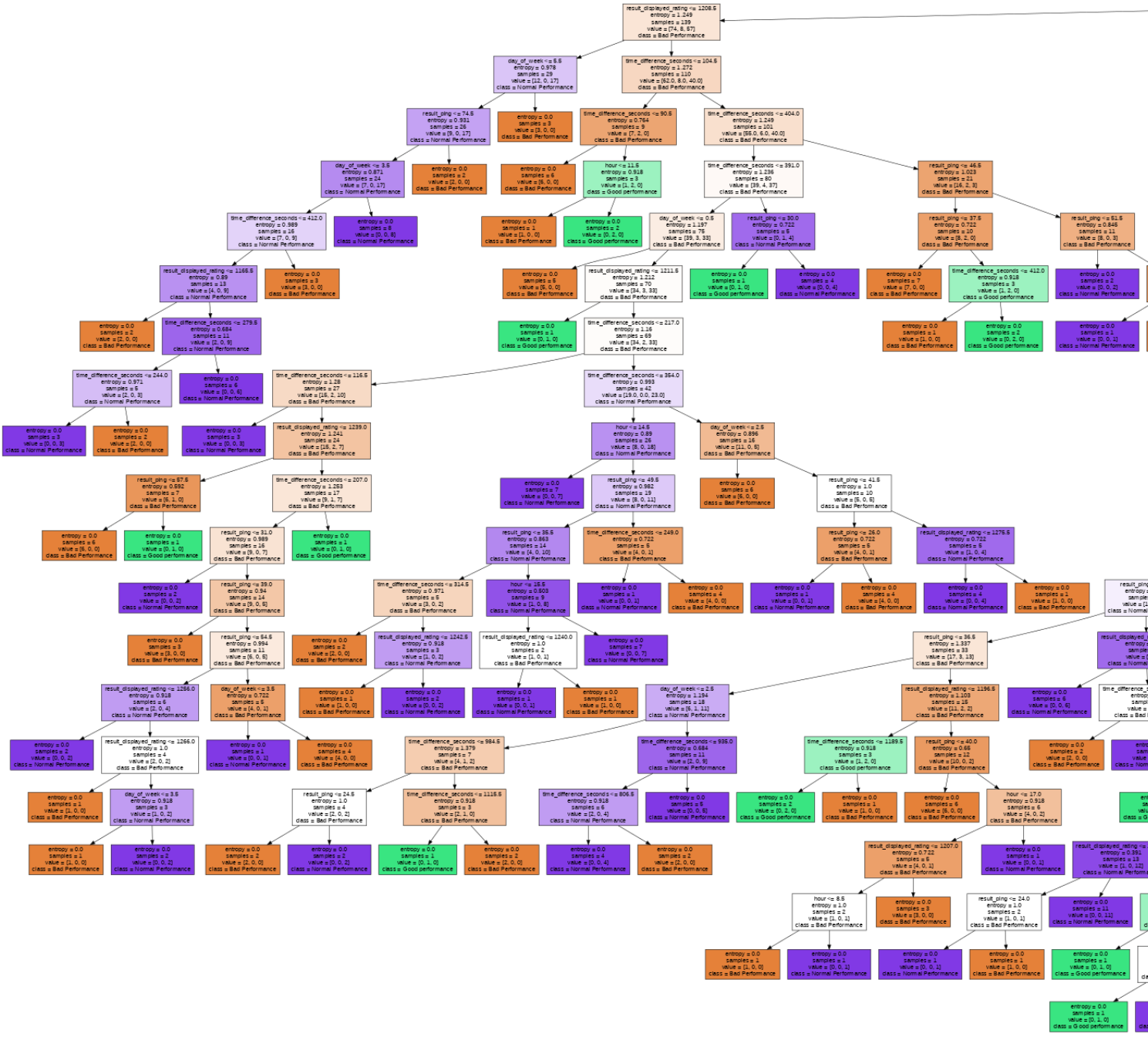

Figure 45: Decision tree for the 40/60 Dataset

The implementation of the process is as follow:

Original Dataset

Figure 46: Class distribution of the original dataset

Training set and Testing set 40/60 split

Figure 47: Class distribution of the 40/60 split training set and testing set

Testing set 40/60 split

Training set and Testing set 60/40 split

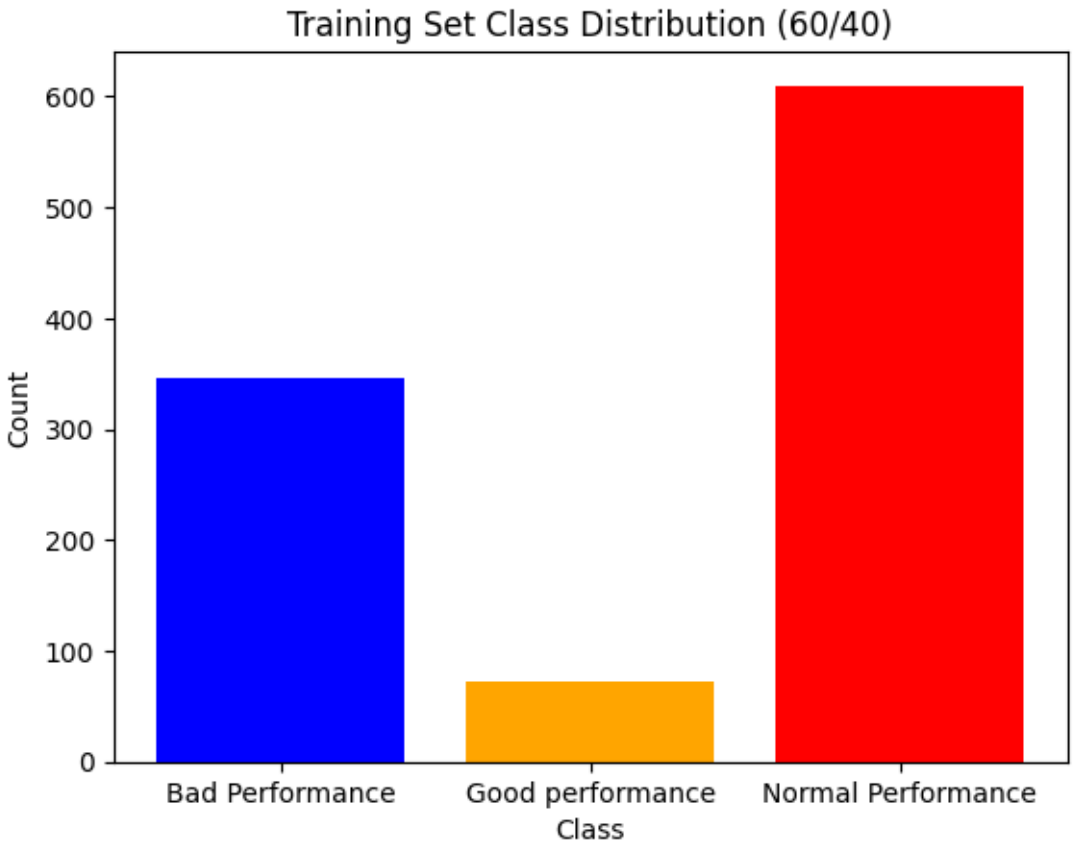

Figure 48: Class distribution of the 60/40 split training set and testing set

Training set and Testing set 80/20 split

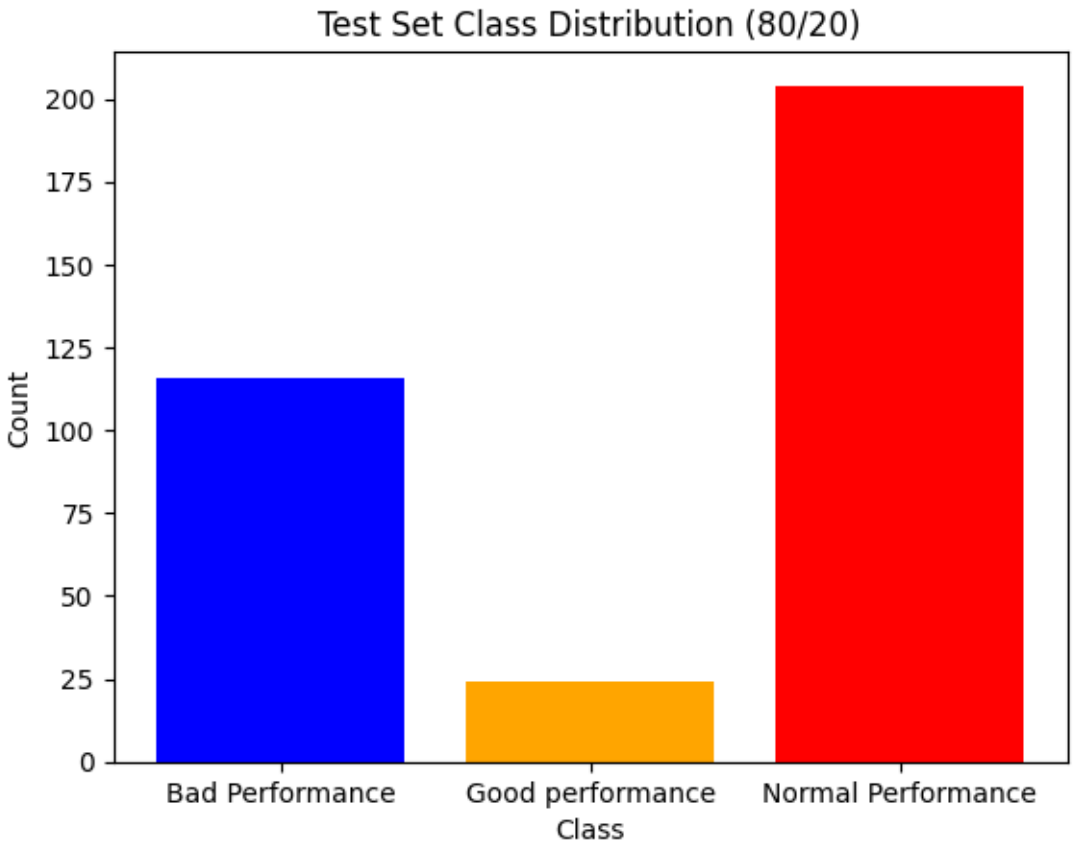

Figure 49: Class distribution of the 80/20 split training set and Testing

Training set and Testing set 90/10 split

Figure 50: Class distribution of the 90/10 split training set and testing set

6.12 Building the Decision Tree Classifiers

This task involved conducting experiments on the train/test proportions prepared earlier (40/60, 60/40, 80/20, and 90/10). For each training set, an instance of `sklearn.tree.DecisionTreeClassifier` *was fitted using information gain.*

The resulting decision trees were visualized using Graphviz for each experiment. These visualizations provide a clear understanding of the tree structure and the decision rules derived from the data.

6.13 Evaluating the Decision Tree Classifiers

The classification report and confusion matrix for the split are as followed:

40/60 Split:

Classification Report for {proportion} Split :

	precision	recall	f1-score	support
Bad Performance	0.37	0.39	0.38	346
Good performance	0.05	0.05	0.05	73
Normal Performance	0.62	0.58	0.60	611
accuracy			0.48	1030
macro avg	0.34	0.34	0.34	1030
weighted avg	0.49	0.48	0.49	1030

Figure 51: Classification report for the 40/60 Dataset

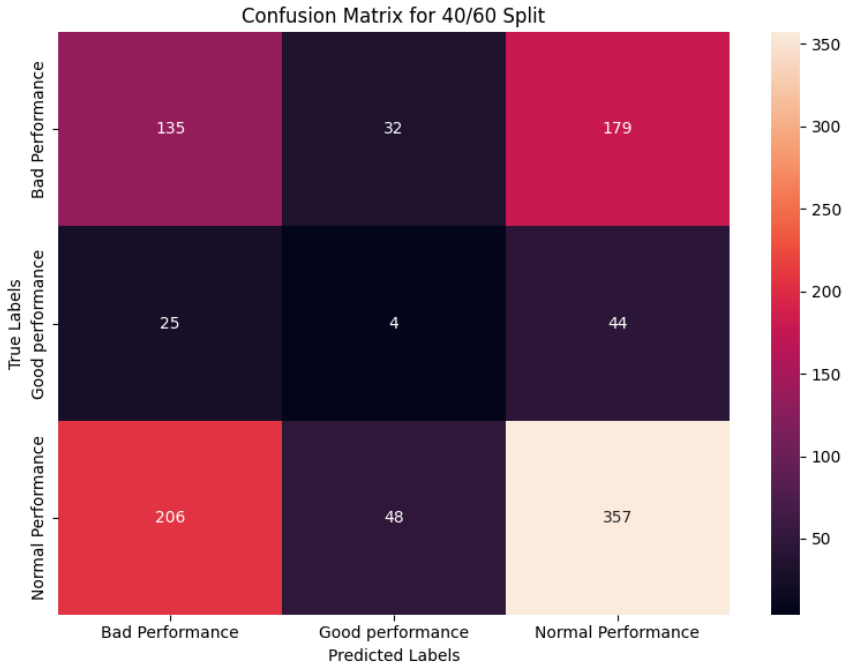

Figure 52: Confusion matrix for the 40/60 Dataset

60/40 Split:

Classification Report for {proportion} Split :

	precision	recall	f1-score	support
Bad Performance	0.34	0.34	0.34	231
Good performance	0.00	0.00	0.00	48
Normal Performance	0.60	0.59	0.59	408
accuracy			0.47	687
macro avg	0.31	0.31	0.31	687
weighted avg	0.47	0.47	0.47	687

Figure 53: Classification report for the 60/40 Dataset

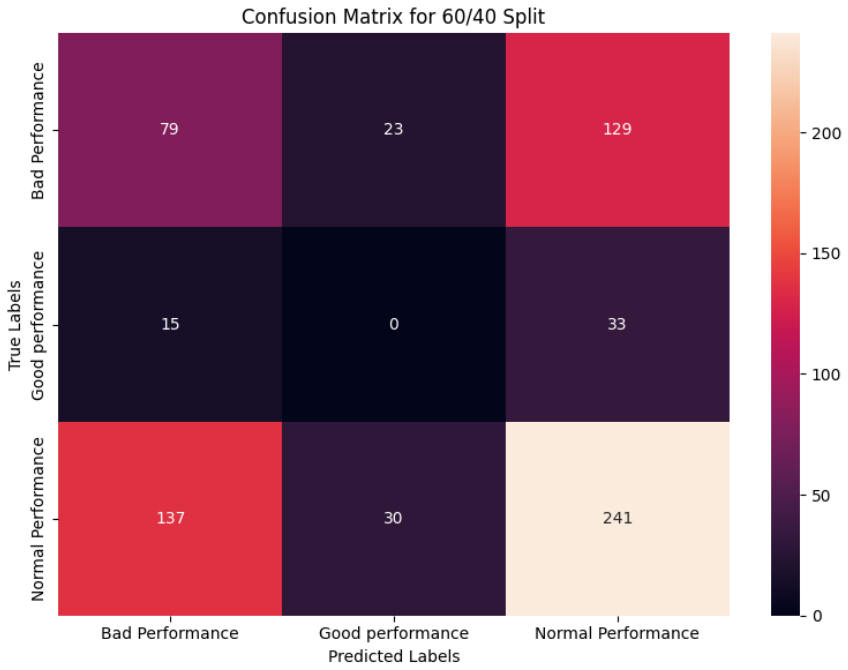

Figure 54: Confusion matrix for the 60/40 Dataset

80/20 Split:

Classification Report for {proportion} Split :

	precision	recall	f1-score	support
Bad Performance	0.37	0.35	0.36	116
Good performance	0.05	0.04	0.04	24
Normal Performance	0.60	0.62	0.61	204
accuracy			0.49	344
macro avg	0.34	0.34	0.34	344
weighted avg	0.48	0.49	0.49	344

Figure 55: Classification report for the 80/20 Dataset

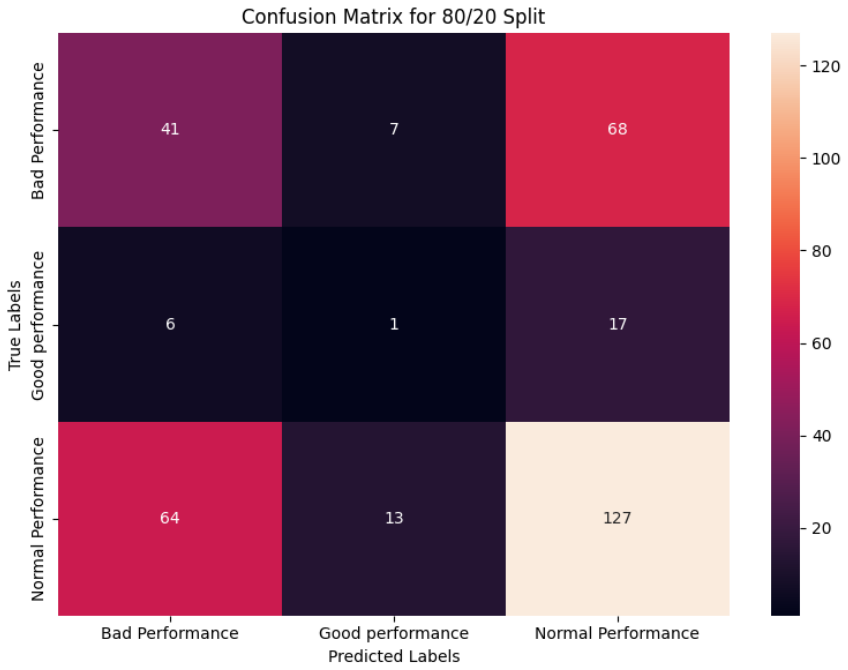

Figure 56: Confusion matrix for the 80/20 Dataset

90/10 Split:

Classification Report for {proportion} Split :

	precision	recall	f1-score	support
Bad Performance	0.39	0.40	0.39	58
Good performance	0.00	0.00	0.00	12
Normal Performance	0.59	0.62	0.61	102
accuracy			0.50	172
macro avg	0.33	0.34	0.33	172
weighted avg	0.48	0.50	0.49	172

Figure 57: Classification report for the 90/10 Dataset

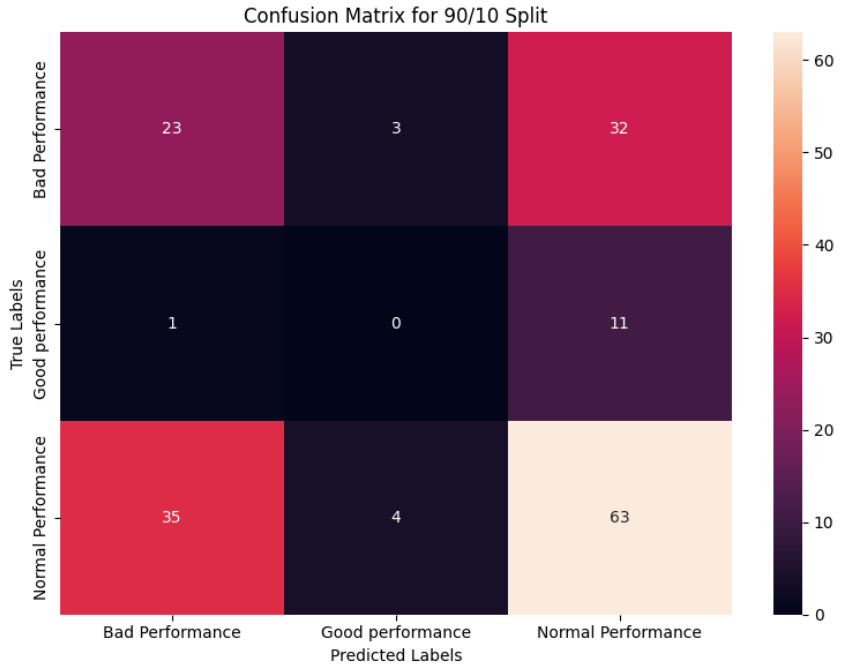

Figure 58: Confusion matrix for the 90/10 Dataset

6.13.1 Classifications report interpretation + insight

Overall, this confusion report show that the decision tree has high precision when predicting both classes. More than 90% *for both, this show that the model has minimized the number of false positive.*

However, the model has a lower recall, especially for malignant tumor where it drop to below 90%

in split 40/60. While still is a high number, this infer that the model may occasionally misses malignant tumors (false negatives).

About F-1 score, for B: 0.97, and for M: 0.94, reflecting a strong balance between precision and recall for both classes. The slightly lower F1 score for M is due to its lower recall.

Overall accuracy is 96%*, indicating that the model correctly classified* 96% *of all test samples. This high accuracy suggests the decision tree is well-suited for this dataset.*

Averages precision, recall, and F1 scores equally across both classes. At 0.95, it confirms balanced performance. Weighted Average (weighted by support for each class): 0.96, further emphasizing good overall performance.

6.13.2 Confusion matrix interpretation + insight

From the confusion matrix, it seem that the model is slightly better at predicting benign tumor than malignant one. However, this is likely due to the fact that their is a different between the two classes distribution. Hence, with more benign tumor to learn from, it is likely the model will find it easier to identify benign tumor. Still, with more malignant tumor, this may pose a problem when using this model in practice as for malignant cases are of greater concern in a medical context since missing a malignant tumor could have serious consequences.

6.14 Depth and accuracy for decision tree of 80/20 split dataset

6.14.1 Decision tree

The result of the depth is as below:

Table 8: Accuracy of Decision Tree Classifier for Different max_depth Values for 80/20 Dataset

max_depth	None	2	3	4	5	6	7
Accuracy	0.9561	0.8859	0.9386	0.9298	0.9561	0.9298	0.9561

This can be visualize in this chart:

Figure 59: Accuracy of different decision tree depth on the 80/20 dataset

6.14.2 Insights on the Statistics Reported

The result give us some insight about our model, as the features is chosen almost based on the author intentions, the model is not surprisingly performed poorly. However, this also show us that there may exist a correlation between these factor, which may be interesting to support other research in the future, maybe about E-sports.

6.15 For other dataset:

For simplicity, we will show only the accuracies of the decision tree:

40/60 Split

Table 9: Accuracy of Decision Tree Classifier for Different max_depth Values for 40/60 Dataset

max_depth	None	2	3	4	5	6	7
Accuracy	0.9123	0.9327	0.9327	0.9123	0.9123	0.9123	0.9123

Figure 60: Accuracy of different decision tree depth on the 40/60 dataset

60/40 Split

begintable[H] Accuracy of Decision Tree Classifier for Different max_depth Values for 60/40

Dataset

max_depth	*None*	*2*	*3*	*4*	*5*	*6*	*7*
Accuracy	*0.9386*	*0.9298*	*0.9386*	*0.9254*	*0.9605*	*0.9430*	*0.9430*

Figure 61: Accuracy of different decision tree depths on the 60/40 dataset

6.15.1 Interpreation + Ínsights

For most splits, the accuracy of the classifier is stable, indicating that further increases in the depth of the decision tree may not provide significant improvement in performance.

For larger training datasets, it seems to provide better accuracy, especially for the 60/40 split. However, the performance tends to stagnate when $max_depthisaround3to5, reinforcingouranalysisthatdeepertrees$

The accuracy of the decision tree for the 40/60 split remained steady across all depths, ranging from ***91.23%*** *to* ***93.27%****. For the 60/40 split, the accuracy improved slightly with deeper trees, with*

96.05% *observed for* $max_depth = 5 and$