*University of Science*

*Viet Nam National University Ho Chi Minh City*

# Lab 02: Decision Tree

**Student's information:**

Name: Tô Quốc Thanh

ID: 22127388

Class: 22CLC10

## Contents

## 1. Introduction

A Decision Tree is a type of supervised machine learning algorithm. It works by splitting the data into subsets, selecting the feature that most significantly impacts the outcome at each node, and then creating new branches for other features if possible.

The data used to train and test the model is very important, as it determines the accuracy of the model.

In this project, we will use 529 data samples from the Breast Cancer Wisconsin (Diagnostic) dataset to train a model that predicts whether a tumor is benign or malignant.

## 2. Preparing the data sets

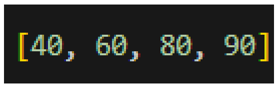
We use the dataset provided in Python as instructed in Lab 02, which comes from the ucimlrepo library.

From this dataset, we divide it into feature and label sets, where the features are the characteristic values that influence the label – the final result in each case.

To facilitate visualization in the next step, we convert the values in the label set to 0 and 1, corresponding to benign and malignant tumors, respectively.

As required by the task, we need to split the dataset into two parts for training and testing. To ensure the model's accuracy, we choose multiple ratios to split these two sets, specifically 40/60, 60/40, 80/20, and 90/10.

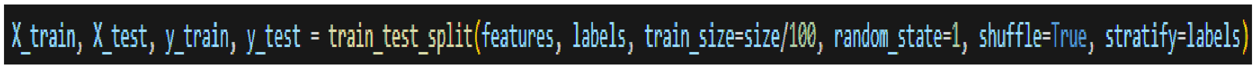
However, we only need to specify one value in each pair (e.g., train_size or test_size), as the train_test_split function will automatically calculate the remaining value. Therefore, we create the splitSize array with the following values:



```
[40, 60, 80, 90]
```

Here, splitSize will hold the proportions for the train set sizes, allowing the train_test_split function to handle the splitting appropriately based on these values.

For the actual splitting, we can use the train_test_split function available in the sklearn library. This function will split the dataset according to the parameter conditions we specify.



```
X_train, X_test, y_train, y_test = train_test_split(features, labels, train_size=size/100, random_state=1, shuffle=True, stratify=labels)
```

In this context, features are the input characteristics, and labels are the target set. The train_size parameter represents the size of the training set for each proportion, while the size of the test set does not need to be specified, as it will automatically be the remainder of the corresponding feature and label sets.

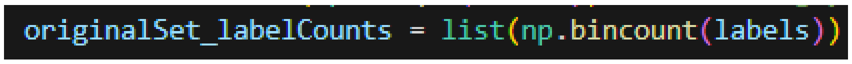
To shuffle the data, we set the shuffle parameter to True. However, to make the results easier to observe, we set the random_state parameter to a specific number, ensuring that the data remains consistent across different shuffling runs.

Additionally, to maintain the distribution of labels corresponding to their features, we use the stratify = labels parameter. This tells the function to ensure that each label value is properly paired with its corresponding features and that the label distribution is preserved during the shuffle.

After splitting, we have 4 subsets, we store the new training_feature, training_label, test_feature, test_label into an array and continue the process until all proportions have been handled, then we have 16 subsets. We have an array named "listData" after the loop.

**** Visualizing all data sets***

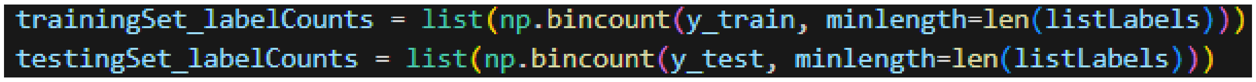
Since we have previously encoded the labels, they now only contain values of 0 (benign) or 1 (malignant). We count the number of occurrences of 0 and 1 in the original dataset using the bincount function from NumPy and save this information.



```
originalSet_labelCounts = list(np.bincount(labels))
```

We then iterate through the training and test sets stored in the listData array. For each set, we count the occurrences of 0 and 1 in the label column.

There might be cases where the train/test split results in one of the sets not containing all label values. To handle this, we use the minlength parameter in bincount to automatically pad with 0 for any missing label values.



```
trainingSet_labelCounts = list(np.bincount(y_train, minlength=len(listLabels)))
testingSet_labelCounts = list(np.bincount(y_test, minlength=len(listLabels)))
```

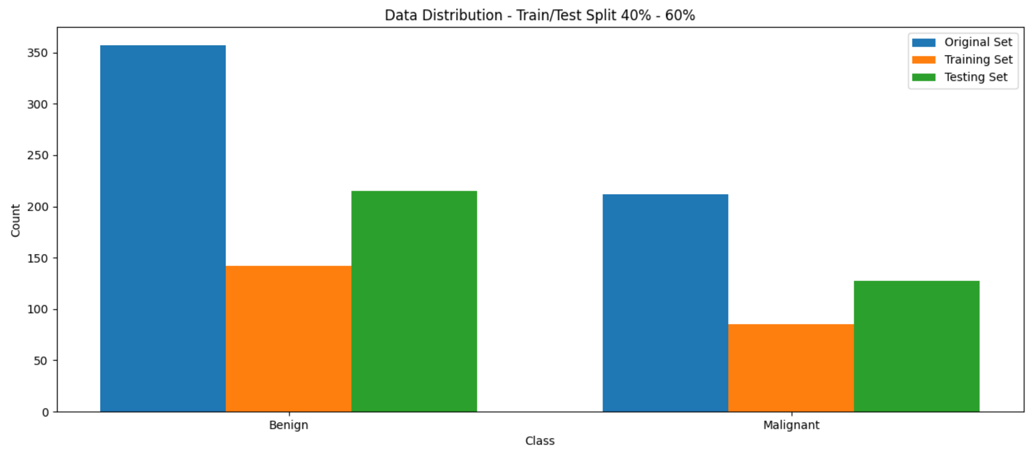
After obtaining the counts, we adjust the size of the plot and the position and size of each bar accordingly. We use Matplotlib to create bar charts to visualize the data after the split, ensuring that the data was split correctly.

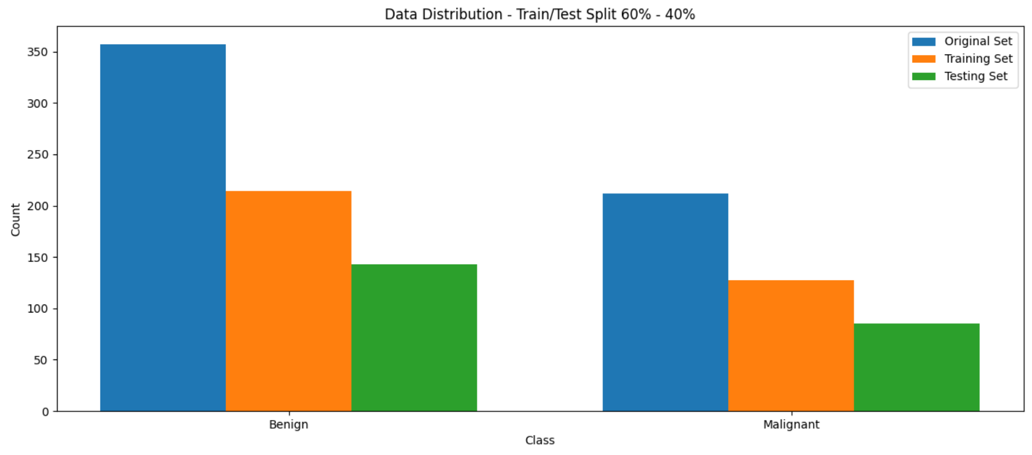
```
plt.bar(x - width, originalSet_labelCounts, width, label="Original Set")
plt.bar(x, trainingSet_labelCounts, width, label="Training Set")
plt.bar(x + width, testingSet_labelCounts, width, label="Testing Set")
```

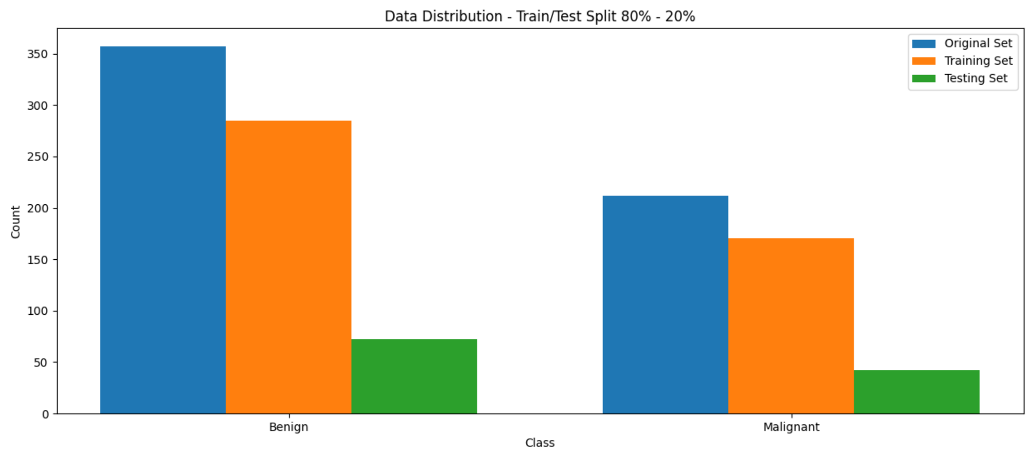
Because each list has two elements, is amount of 0 and 1 values, so we will have two group of bar chart, ‘0’ group for Benign and ‘1’ for Malignant.

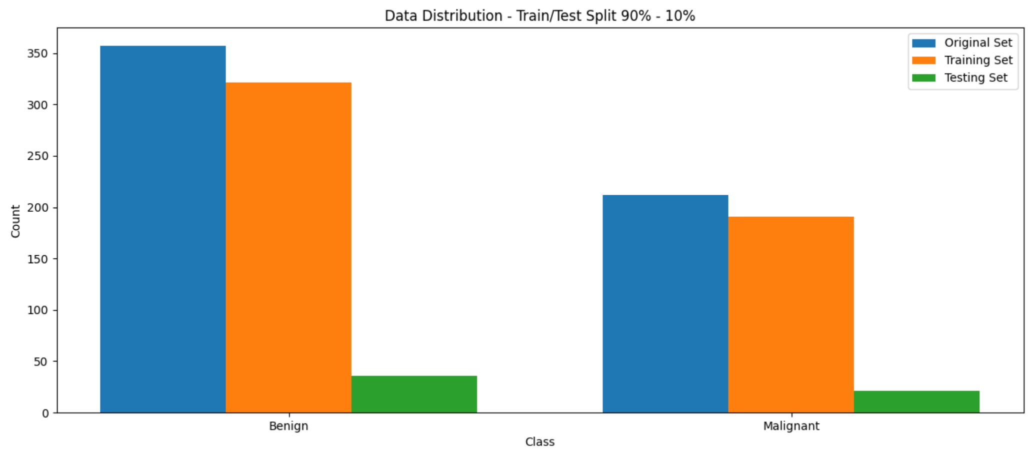
After creating the bar charts, save them to the "Output" folder for easy viewing and reporting.

These are the charts:





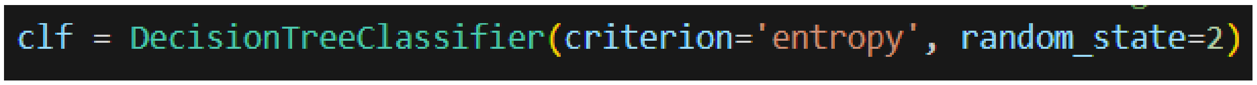




We can see that the splitting is quite good through these charts. The ratio between the train and test splits is as expected, and the number of elements in each group within the splits is also fairly balanced

## 3. Building the decision tree classifiers

To build a Decision Tree with the train and test sets after splitting according to the ratio, we use the DecisionTreeClassifier class and its functions from the sklearn library.

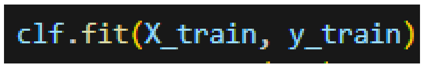


```
clf = DecisionTreeClassifier(criterion='entropy', random_state=2)
```

The criterion parameter specifies the function used to measure the quality of a split in the decision tree; in this case, we use the entropy index.

The random_state parameter can be any number. Setting this ensures that we get consistent results if we don't change the input values.

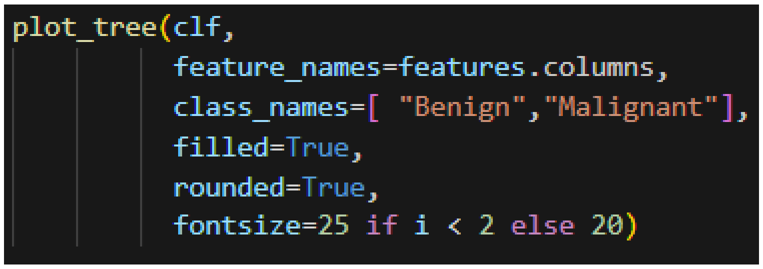
We can easily build a Decision Tree by using the fit function of the DecisionTreeClassifier class to find the optimal parameters based on the data, thus creating the best prediction model.



```
clf.fit(X_train, y_train)
```

To visualize the trees after building them, we use functions available in the Matplotlib library, specifically the plot_tree function.

This function takes as input an object of the DecisionTreeClassifier class, which contains the Decision Tree model after it has been built. Additionally, we need to provide information about the names of the feature sets, target sets, cell sizes, etc.



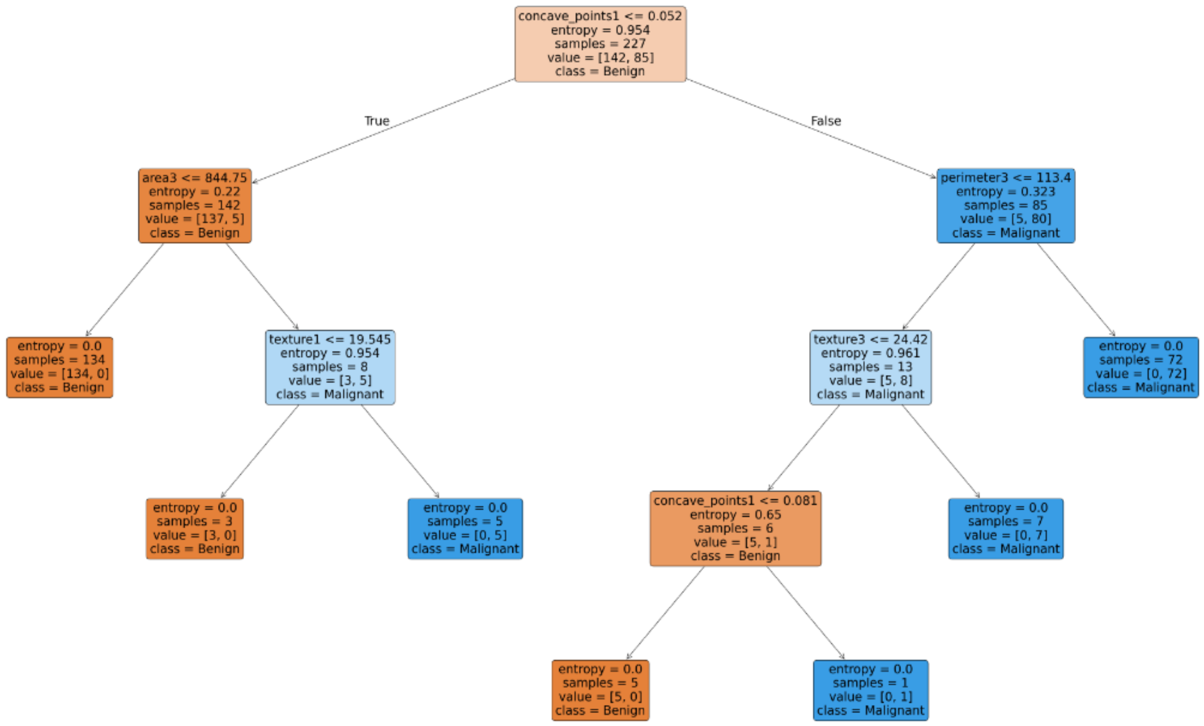
```
plot_tree(clf,
          feature_names=features.columns,
          class_names=[ "Benign","Malignant"],
          filled=True,
          rounded=True,
          fontsize=25 if i < 2 else 20)
```

To avoid overlapping branches in the decision tree visualizations, we should adjust the font size for each tree.

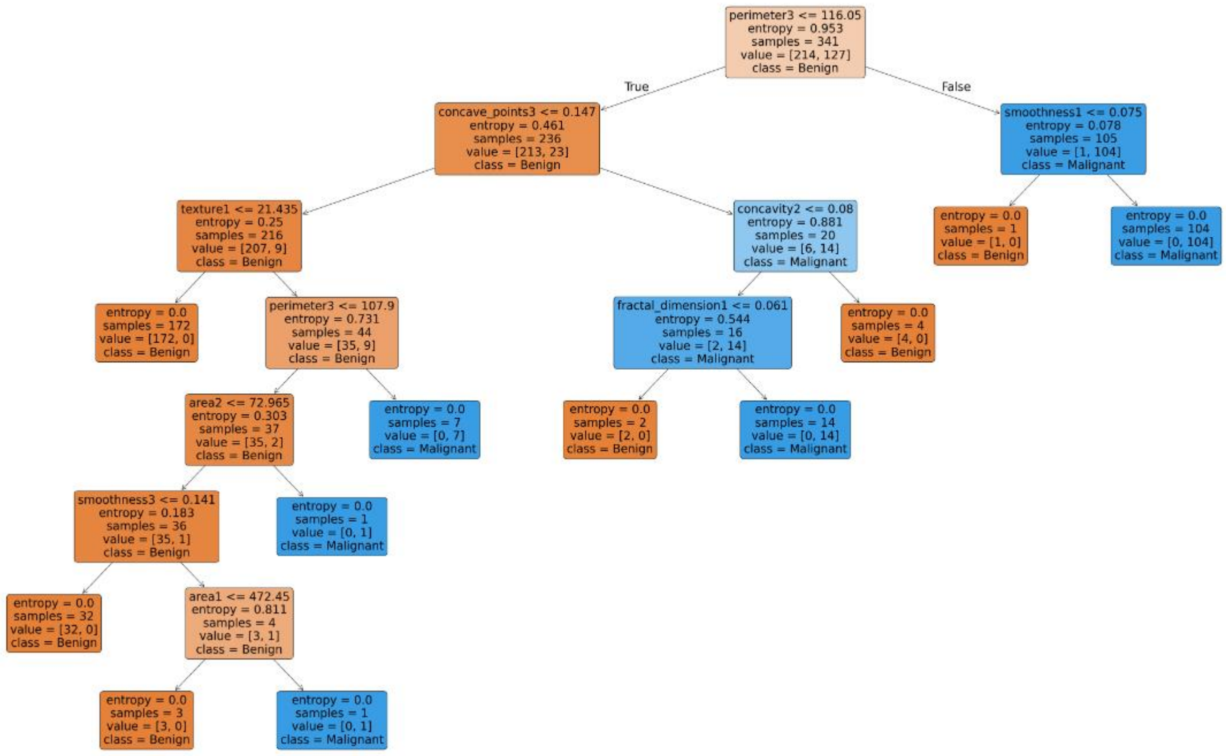
We should save these models to a list, so we don't need to train model again in other steps, we can named this is listTree

We save the results of each model into the "Output" folder for easy checking and reporting.

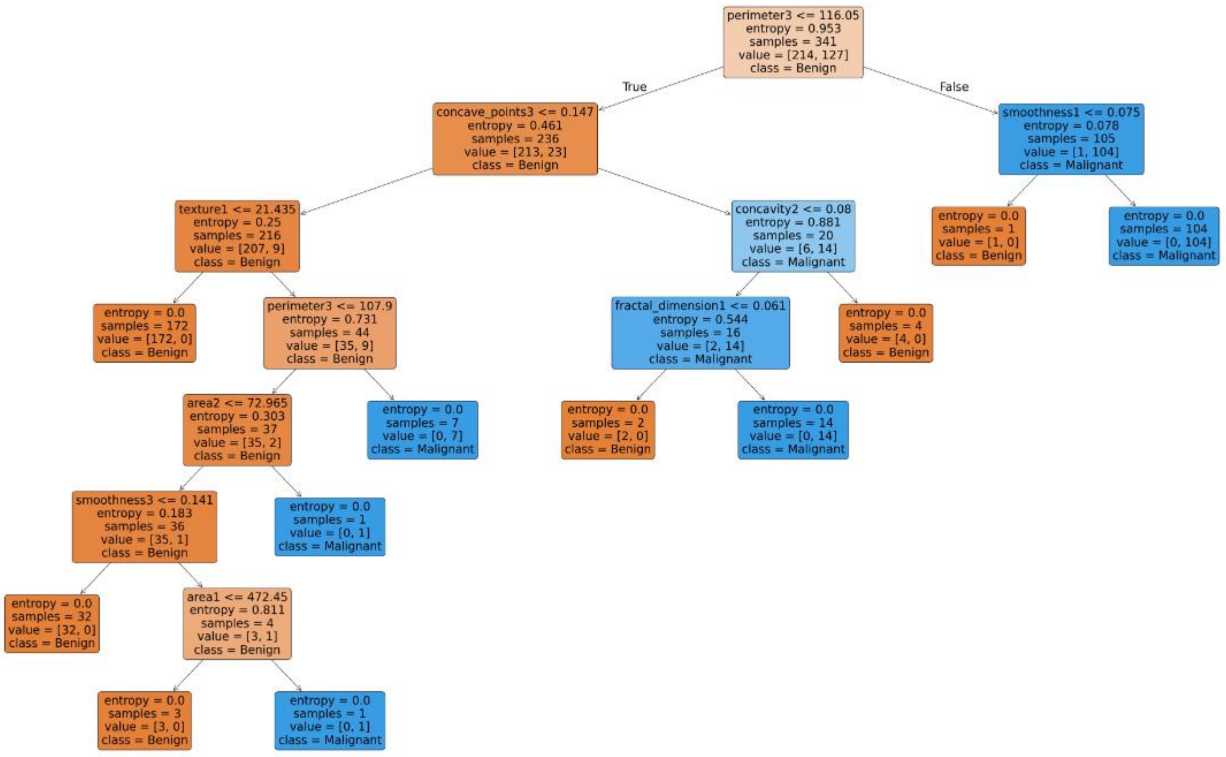
Decision Tree Visualization - Train/Test Split 40% - 60%



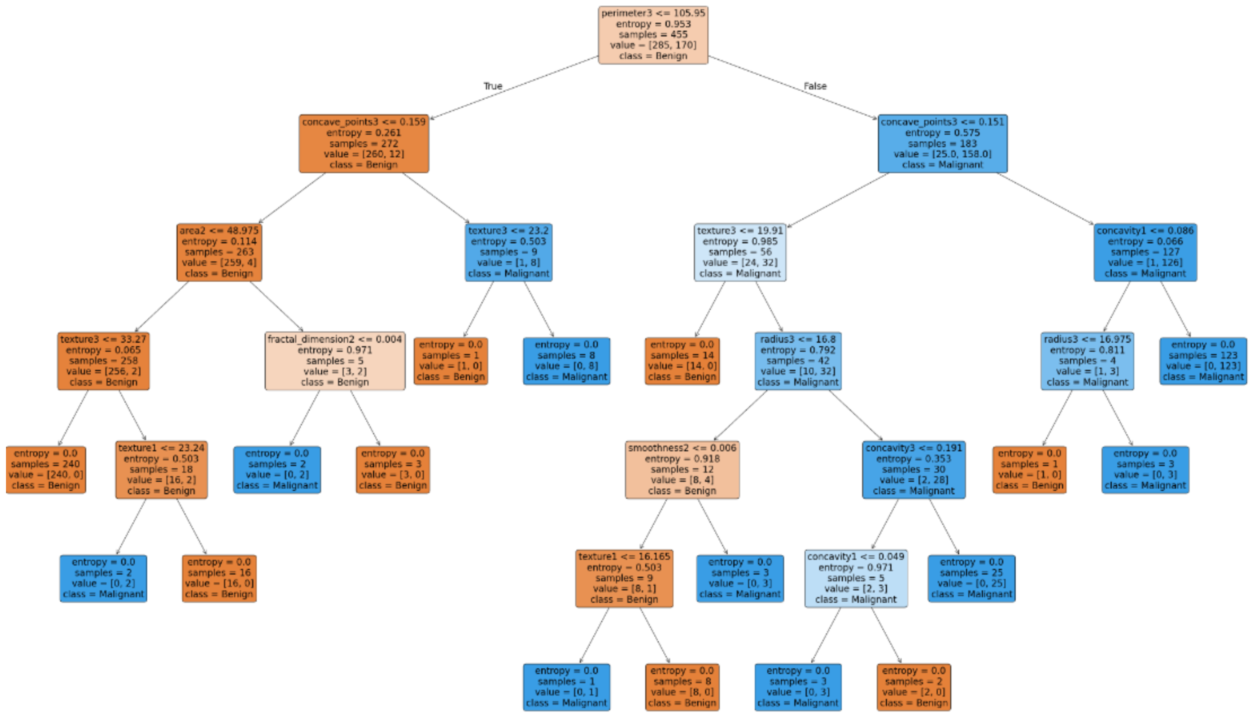
Decision Tree Visualization - Train/Test Split 60% - 40%

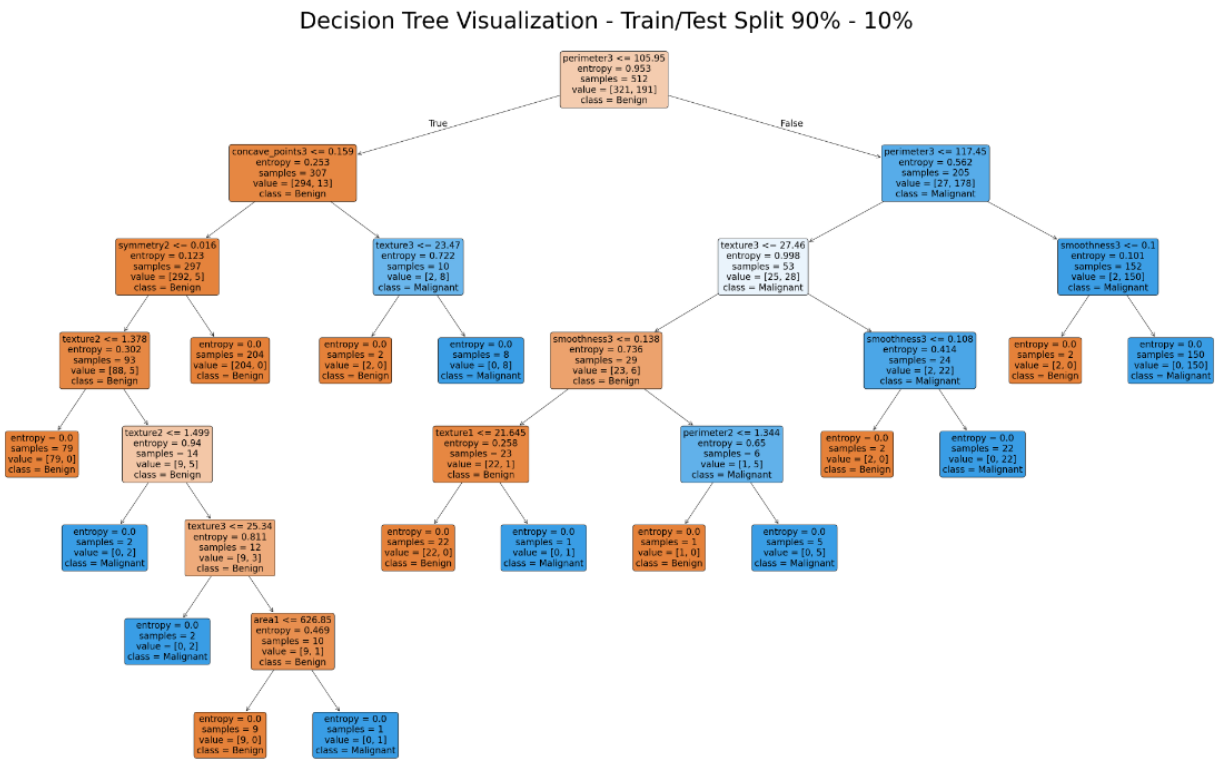


Decision Tree Visualization - Train/Test Split 60% - 40%



Decision Tree Visualization - Train/Test Split 80% - 20%




Decision Tree Visualization - Train/Test Split 90% - 10%
perimeter3 <= 105.95 entropy = 0.953 samples = 512 value = [321, 191] class = Benign
True
False
concave_points3 <= 0.159 entropy = 0.253 samples = 307 value = [294, 13] class = Benign
perimeter3 <= 117.45 entropy = 0.562 samples = 205 value = [27, 178] class = Malignant
symmetry2 <= 0.016 entropy = 0.123 samples = 297 value = [292, 5] class = Benign
texture3 <= 23.47 entropy = 0.722 samples = 10 value = [2, 8] class = Malignant
texture3 <= 27.46 entropy = 0.998 samples = 53 value = [25, 28] class = Malignant
smoothness3 <= 0.1 entropy = 0.101 samples = 152 value = [2, 150] class = Malignant
texture2 <= 1.378 entropy = 0.302 samples = 93 value = [88, 5] class = Benign
entropy = 0.0 samples = 204 value = [204, 0] class = Benign
entropy = 0.0 samples = 2 value = [2, 0] class = Benign
entropy = 0.0 samples = 8 value = [0, 8] class = Malignant
smoothness3 <= 0.138 entropy = 0.736 samples = 29 value = [23, 6] class = Benign
smoothness3 <= 0.108 entropy = 0.414 samples = 24 value = [2, 22] class = Malignant
entropy = 0.0 samples = 2 value = [2, 0] class = Benign
entropy = 0.0 samples = 150 value = [0, 150] class = Malignant
entropy = 0.0 samples = 79 value = [79, 0] class = Benign
texture2 <= 1.499 entropy = 0.94 samples = 14 value = [9, 5] class = Benign
texture1 <= 21.645 entropy = 0.258 samples = 23 value = [22, 1] class = Benign
perimeter2 <= 1.344 entropy = 0.65 samples = 6 value = [1, 5] class = Malignant
entropy = 0.0 samples = 2 value = [2, 0] class = Benign
entropy = 0.0 samples = 22 value = [0, 22] class = Malignant
entropy = 0.0 samples = 2 value = [0, 2] class = Malignant
texture3 <= 25.34 entropy = 0.811 samples = 12 value = [9, 3] class = Benign
entropy = 0.0 samples = 22 value = [22, 0] class = Benign
entropy = 0.0 samples = 1 value = [0, 1] class = Malignant
entropy = 0.0 samples = 1 value = [1, 0] class = Benign
entropy = 0.0 samples = 5 value = [0, 5] class = Malignant
entropy = 0.0 samples = 2 value = [0, 2] class = Malignant
area1 <= 626.85 entropy = 0.469 samples = 10 value = [9, 1] class = Benign
entropy = 0.0 samples = 9 value = [9, 0] class = Benign
entropy = 0.0 samples = 1 value = [0, 1] class = Malignant

## 4. Evaluating the decision tree classifiers

Iterating through listTree and listData, for each tree model and test data in data list (ignore train set, because we have the model before, so we don't need them), we replace the feature values in the test set to obtain the predicted y values. Combining these predicted values with the actual y values (taken from the label column of the test file), we pass them into the classification_report function to generate a classification report.

However, instead of saving the data as a CSV file, we want to save it as an image. To do this, we use functions available in Matplotlib, convert the data from CSV format into a table, and then save it as an image.

To create the confusion matrix, we pass the predicted y values obtained earlier, along with the actual y values from the label column in the test set, into the confusion_matrix() function. To visualize the confusion matrix, we use the Seaborn library and the heatmap function. We adjust information such as the chart title, column names, etc., to be appropriate before saving it into the "Output" folder.

**Differences Between Classification Report and Confusion Matrix:**

- **Confusion Matrix**: It is a table that represents the prediction results against the actual results based on True Positive and True Negative values. It simply shows the number of people with Benign and Malignant tumors in reality versus the model's predictions, giving an overview of the model's accuracy.
- **Classification Report**: It is also built on the differences between predicted and actual values, but it focuses on metrics calculated from those differences. The most important metrics are:
    - **Precision**: The proportion of people who actually have Malignant tumors compared to those predicted to have Malignant tumors.
    - **Recall**: The proportion of people who actually have Malignant tumors that are predicted correctly, and vice versa for Benign tumors.
    - **Other:** Support (number of sample), F1-score (is the harmonic mean of Precision and Recall, used to balance these two metrics), macro avg (mean without number of sample), weighted avg (mean with number of sample)

    User **accuracy,** which is derived from the factors mentioned above, to assess the overall accuracy of the model.

If not for the support from the built-in confusion_matrix() function in Python, calculating these metrics would have to be done based on the Confusion Matrix values. For details, see [1].

Based on these two metrics, we can assess the accuracy of a model. Additionally, the discrepancy between these two values can indicate issues the model is facing, providing a basis for further development and improvement.

**Classification Report Visualization:**

Classification Report - Train/Test Split 40% - 60%

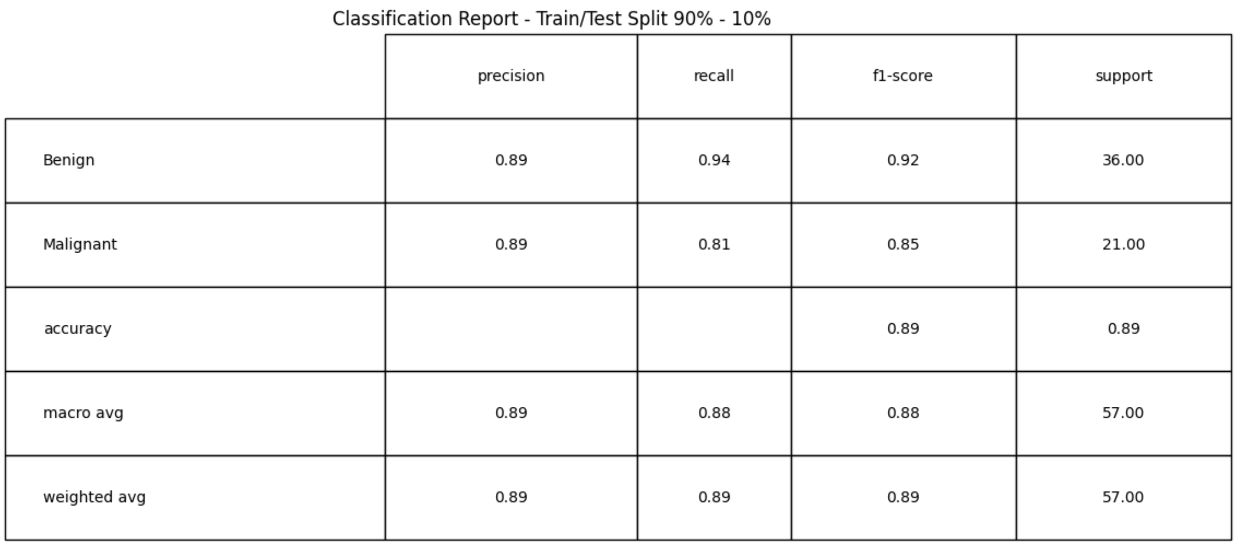
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Benign | 0.95 | 0.96 | 0.95 | 215.00 |
| Malignant | 0.93 | 0.91 | 0.92 | 127.00 |
| accuracy | | | 0.94 | 0.94 |
| macro avg | 0.94 | 0.93 | 0.94 | 342.00 |
| weighted avg | 0.94 | 0.94 | 0.94 | 342.00 |

Classification Report - Train/Test Split 60% - 40%

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Benign | 0.93 | 0.96 | 0.94 | 143.00 |
| Malignant | 0.92 | 0.87 | 0.90 | 85.00 |
| accuracy | | | 0.93 | 0.93 |
| macro avg | 0.93 | 0.91 | 0.92 | 228.00 |
| weighted avg | 0.93 | 0.93 | 0.92 | 228.00 |

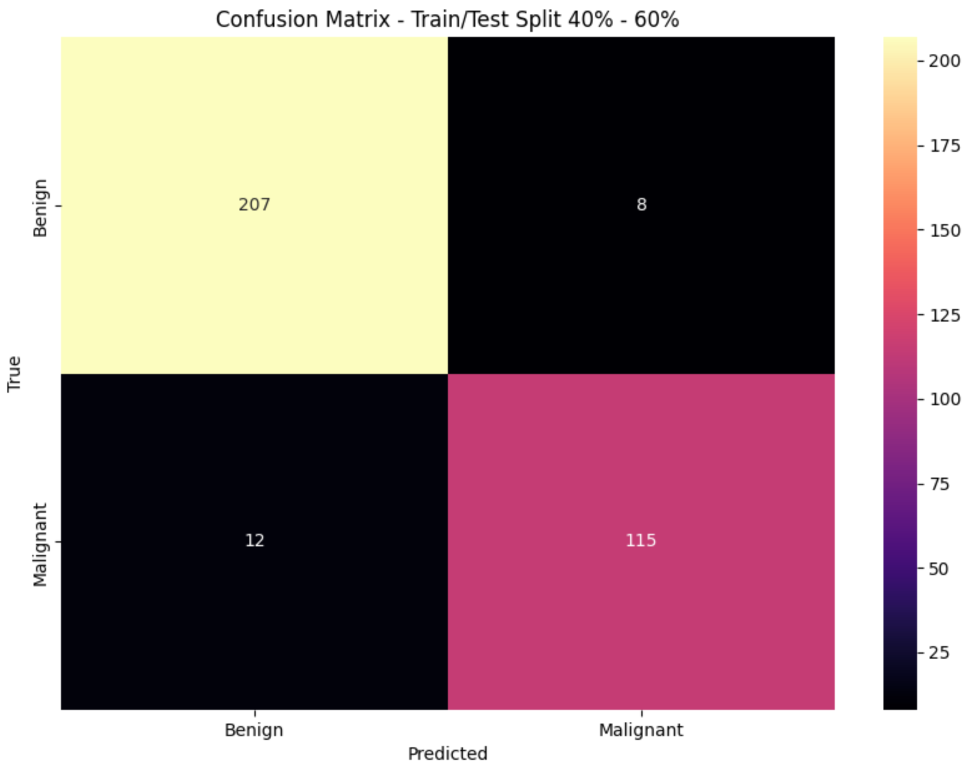
Classification Report - Train/Test Split 80% - 20%

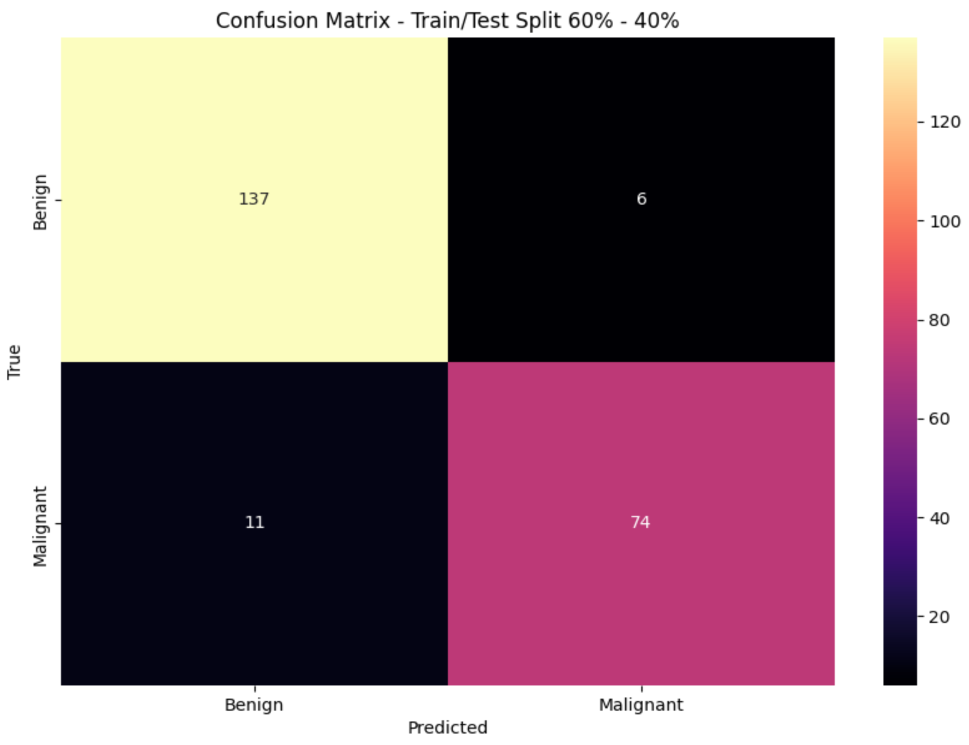
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Benign | 0.95 | 0.96 | 0.95 | 72.00 |
| Malignant | 0.93 | 0.90 | 0.92 | 42.00 |
| accuracy | | | 0.94 | 0.94 |
| macro avg | 0.94 | 0.93 | 0.93 | 114.00 |
| weighted avg | 0.94 | 0.94 | 0.94 | 114.00 |

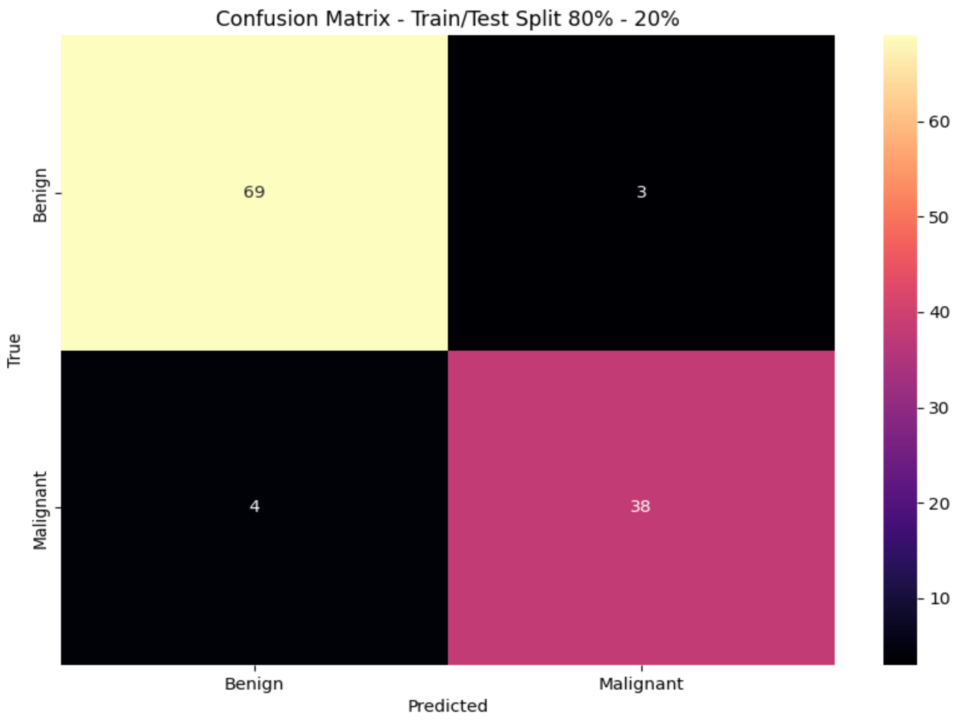

Classification Report - Train/Test Split 90% - 10%

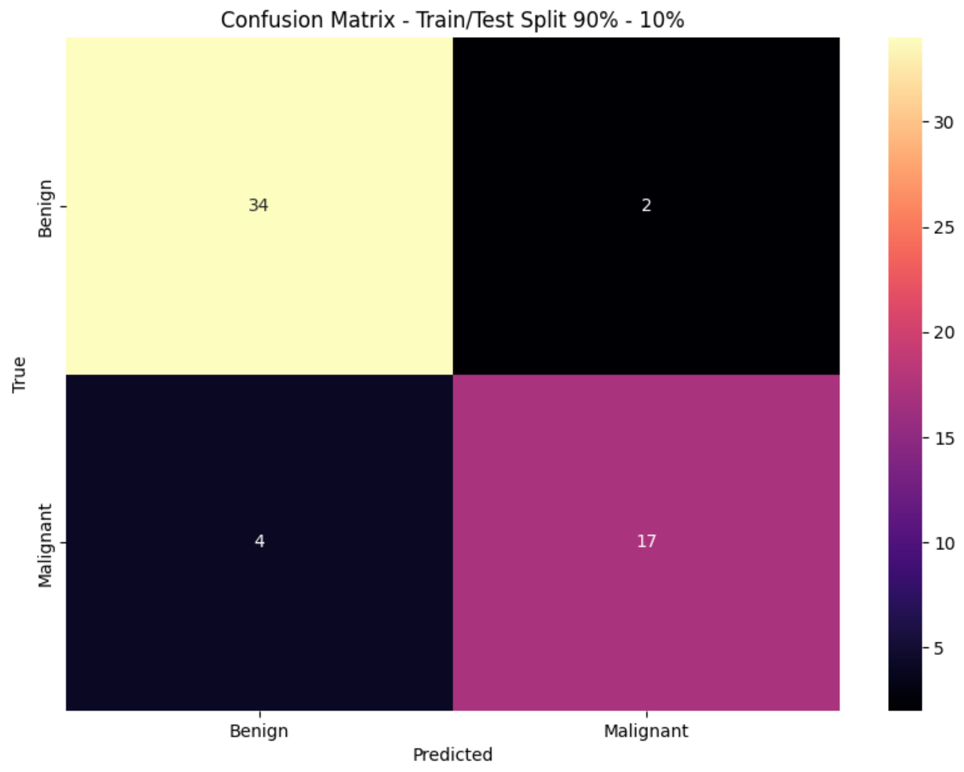
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Benign | 0.89 | 0.94 | 0.92 | 36.00 |
| Malignant | 0.89 | 0.81 | 0.85 | 21.00 |
| accuracy | | | 0.89 | 0.89 |
| macro avg | 0.89 | 0.88 | 0.88 | 57.00 |
| weighted avg | 0.89 | 0.89 | 0.89 | 57.00 |

## Confusion Matrix Visualization:




Confusion Matrix - Train/Test Split 60% - 40%
Benign
Malignant
True
137
6
11
74
Benign
Malignant
Predicted
120
100
80
60
40
20


Confusion Matrix - Train/Test Split 80% - 20%
Benign
Malignant
True
69
3
4
38
Benign
Malignant
Predicted
60
50
40
30
20
10



## 5. The depth and accuracy of a decision tree

Adjusting the maximum depth of a Decision Tree has a significant impact on the model's accuracy. Generally, having too many constraints (such as a very small or very large depth) can lead to inaccurate predictions.

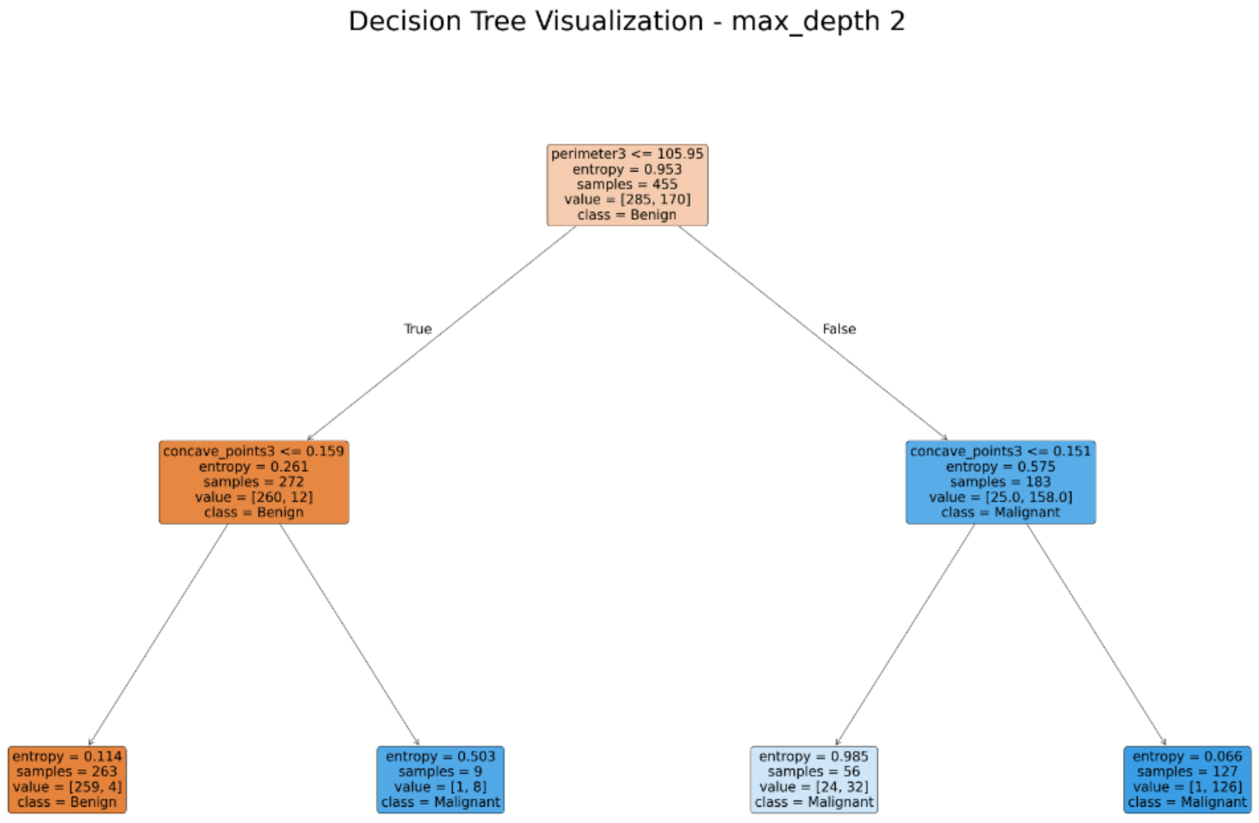
In this section, we will use the training and test data from the dataset split at an 80/20 ratio. We will observe the model's predictions compared to the actual results to draw conclusions about the model's accuracy at each specific maximum depth.

```
clf = DecisionTreeClassifier(max_depth=depth, criterion='entropy', random_state=42)
```

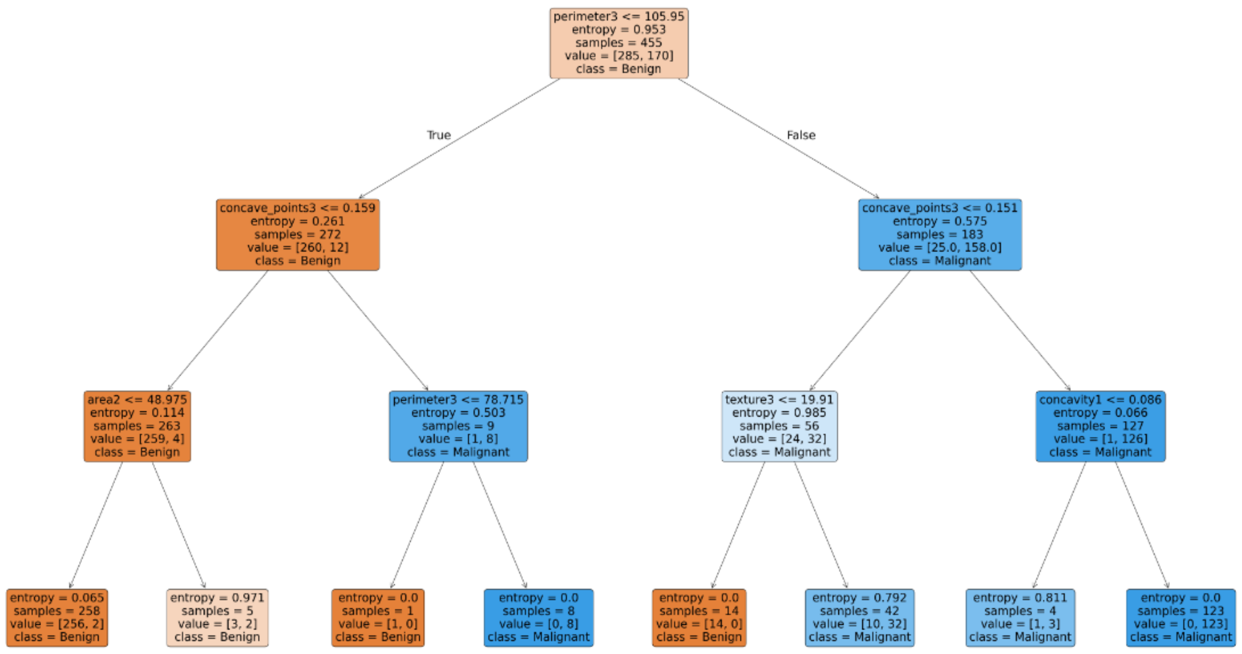
The steps to find optimal parameters and build the tree are similar to those previously described, that we use fit function in DecisionTreeClassifier class, but add parameter 'max_depth'. In this part of the report, I will also cover how to calculate accuracy.

After successfully building the model using the training data (features and target), we use the test data (features) to predict the target values. Then, we use the accuracy_score() function to calculate the accuracy by comparing the predicted values with the actual values in the test set.

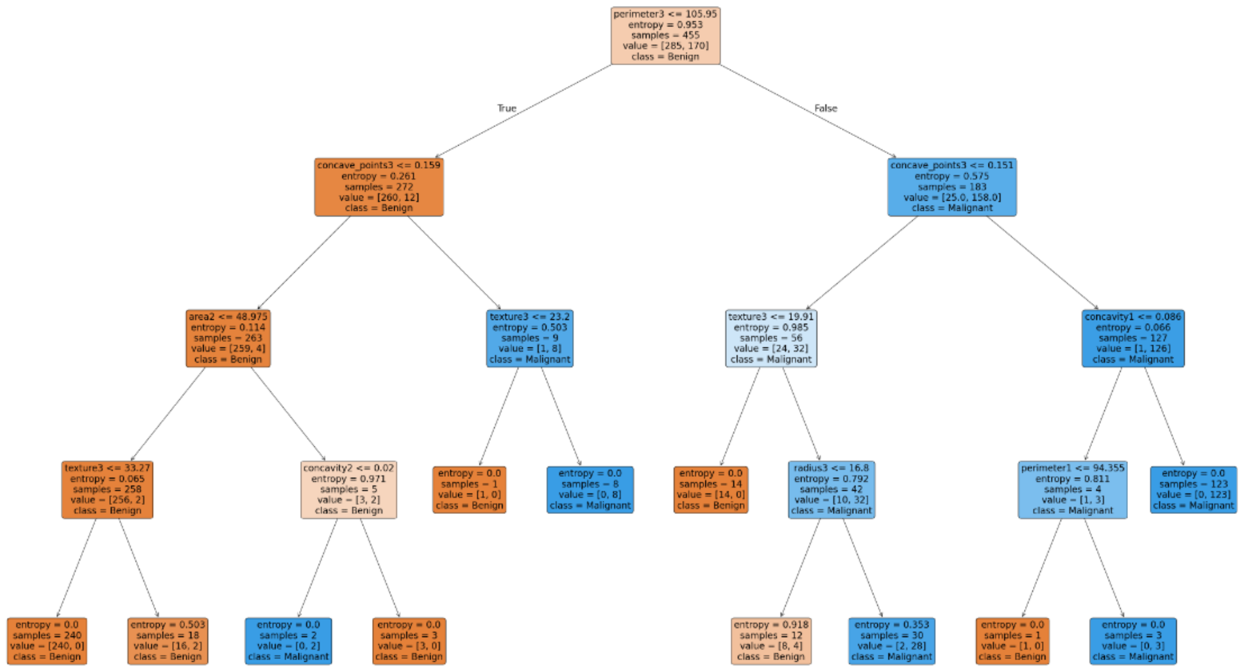
After building the tree and determining the accuracy, we save the results into the "Output" folder. If the Decision Trees are visualized as plots, the accuracy table will include the name of each column (maximum depth) and the corresponding accuracy values obtained earlier.



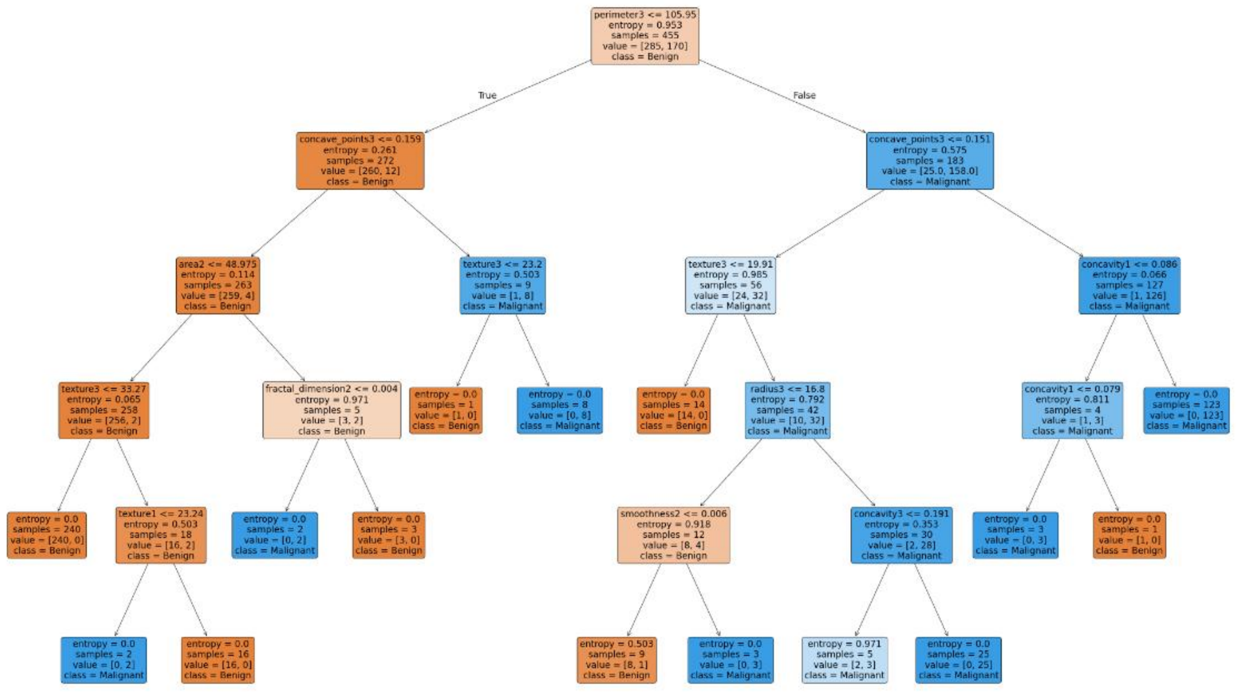
Decision Tree Visualization - max_depth 3



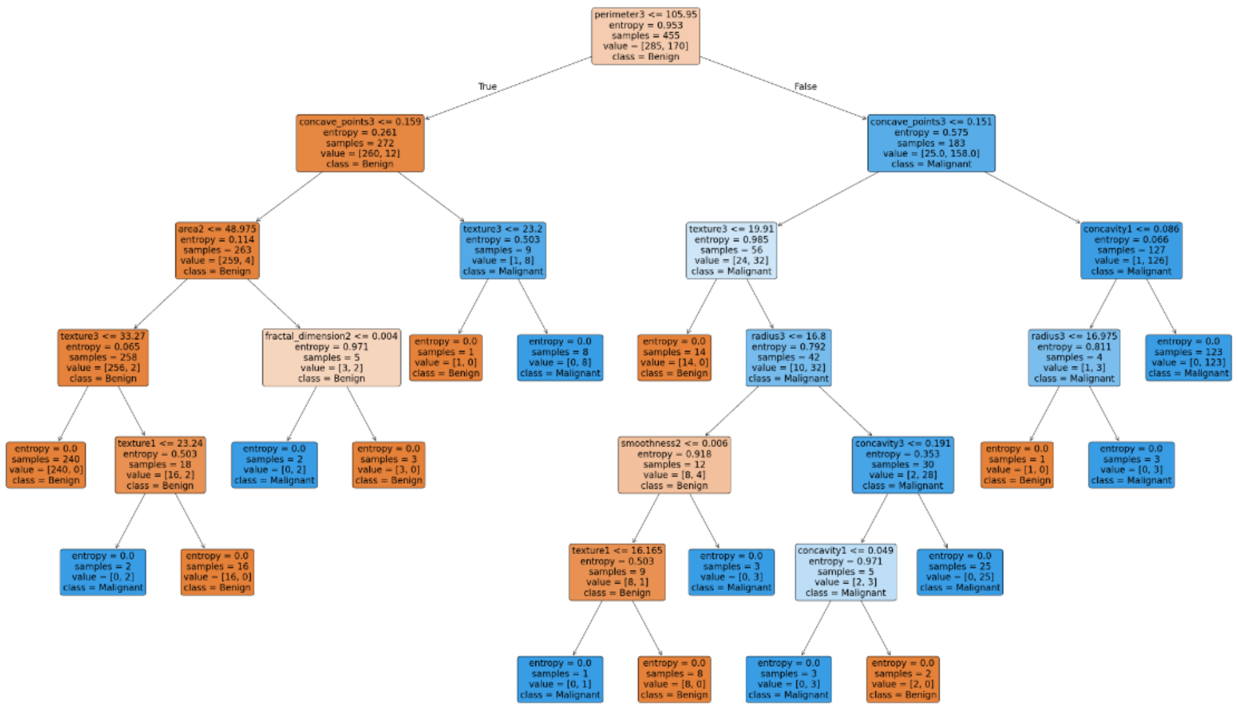
Decision Tree Visualization - max_depth 4

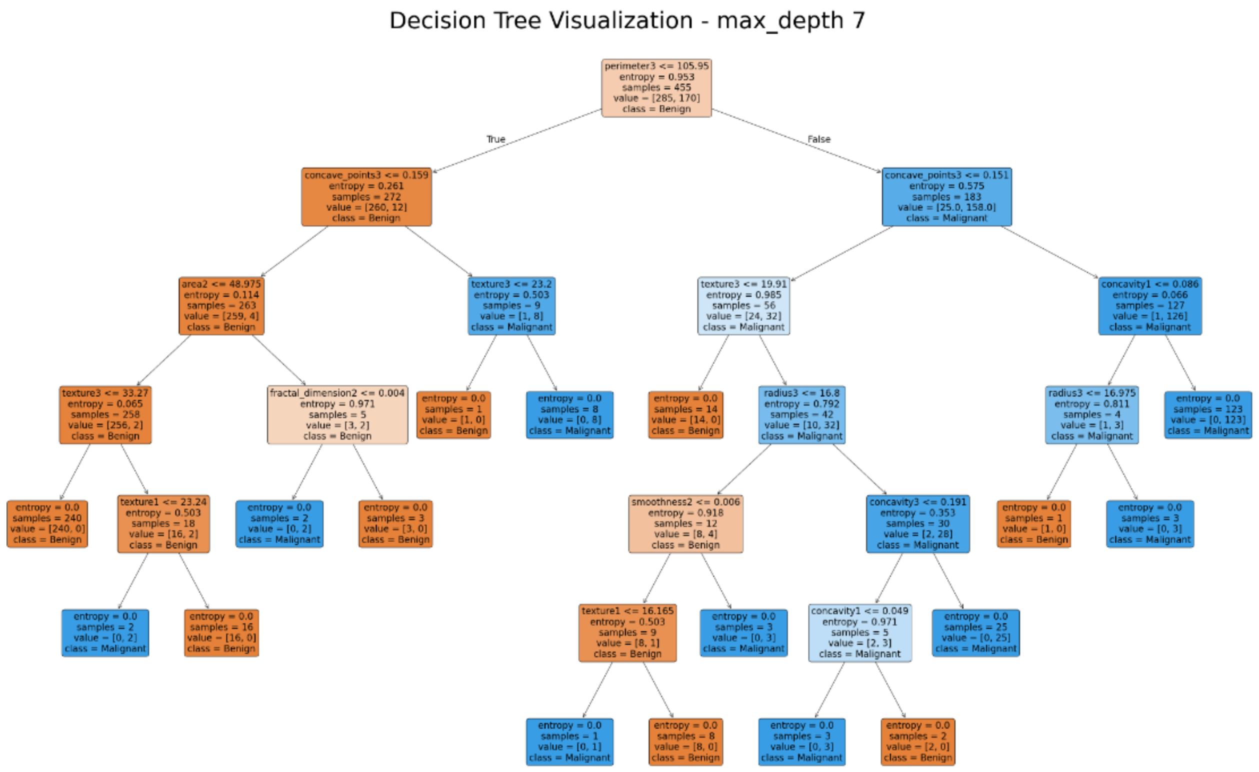


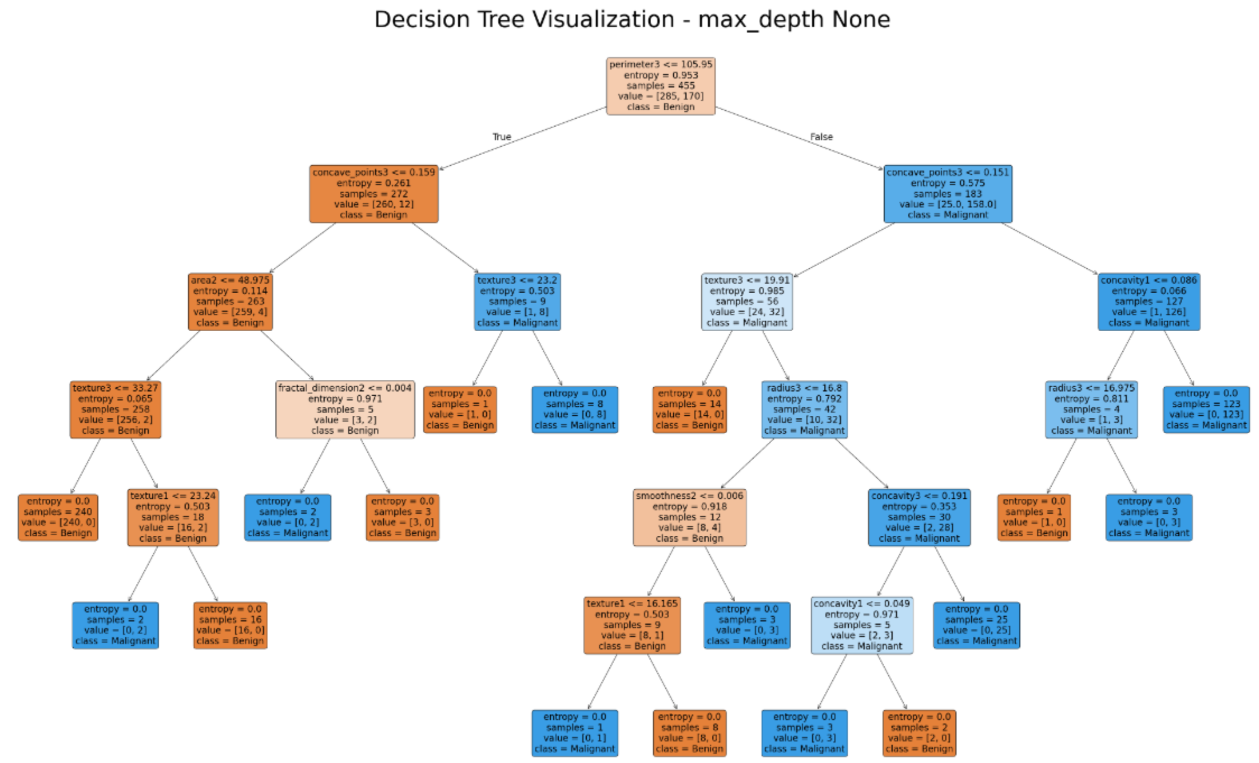
Decision Tree Visualization - max_depth 5


perimeter3 <= 105.95 entropy = 0.953 samples = 455 value = [285, 170] class = Benign
True
False
concave_points3 <= 0.159 entropy = 0.261 samples = 272 value = [260, 12] class = Benign
concave_points3 <= 0.151 entropy = 0.575 samples = 183 value = [25.0, 158.0] class = Malignant
area2 <= 48.975 entropy = 0.114 samples = 263 value = [259, 4] class = Benign
texture3 <= 23.2 entropy = 0.503 samples = 9 value = [1, 8] class = Malignant
texture3 <= 19.91 entropy = 0.985 samples = 56 value = [24, 32] class = Malignant
concavity1 <= 0.086 entropy = 0.066 samples = 127 value = [1, 126] class = Malignant
texture3 <= 33.27 entropy = 0.065 samples = 258 value = [256, 2] class = Benign
fractal_dimension2 <= 0.004 entropy = 0.971 samples = 5 value = [3, 2] class = Benign
entropy = 0.0 samples = 1 value = [1, 0] class = Benign
entropy = 0.0 samples = 8 value = [0, 8] class = Malignant
entropy = 0.0 samples = 14 value = [14, 0] class = Benign
radius3 <= 16.8 entropy = 0.792 samples = 42 value = [10, 32] class = Malignant
concavity1 <= 0.079 entropy = 0.811 samples = 4 value = [1, 3] class = Malignant
entropy = 0.0 samples = 123 value = [0, 123] class = Malignant
entropy = 0.0 samples = 240 value = [240, 0] class = Benign
texture1 <= 23.24 entropy = 0.503 samples = 18 value = [16, 2] class = Benign
entropy = 0.0 samples = 2 value = [0, 2] class = Malignant
entropy = 0.0 samples = 3 value = [3, 0] class = Benign
smoothness2 <= 0.006 entropy = 0.918 samples = 12 value = [8, 4] class = Benign
concavity3 <= 0.191 entropy = 0.353 samples = 30 value = [2, 28] class = Malignant
entropy = 0.0 samples = 3 value = [0, 3] class = Malignant
entropy = 0.0 samples = 1 value = [1, 0] class = Benign
entropy = 0.0 samples = 2 value = [0, 2] class = Malignant
entropy = 0.0 samples = 16 value = [16, 0] class = Benign
entropy = 0.503 samples = 9 value = [8, 1] class = Benign
entropy = 0.0 samples = 3 value = [0, 3] class = Malignant
entropy = 0.971 samples = 5 value = [2, 3] class = Malignant
entropy = 0.0 samples = 25 value = [0, 25] class = Malignant

Decision Tree Visualization - max_depth 6

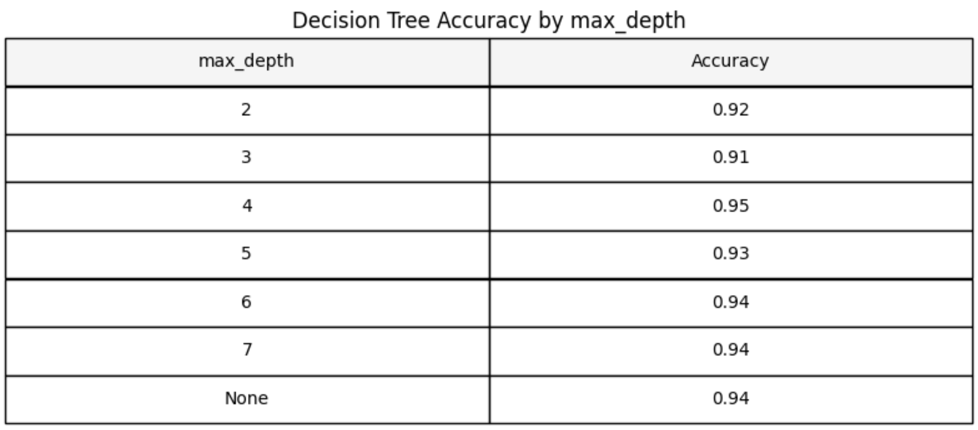

perimeter3 <= 105.95 entropy = 0.953 samples = 455 value = [285, 170] class = Benign
True
False
concave_points3 <= 0.159 entropy = 0.261 samples = 272 value = [260, 12] class = Benign
concave_points3 <= 0.151 entropy = 0.575 samples = 183 value = [25.0, 158.0] class = Malignant
area2 <= 48.975 entropy = 0.114 samples = 263 value = [259, 4] class = Benign
texture3 <= 23.2 entropy = 0.503 samples = 9 value = [1, 8] class = Malignant
texture3 <= 19.91 entropy = 0.985 samples = 56 value = [24, 32] class = Malignant
concavity1 <= 0.086 entropy = 0.066 samples = 127 value = [1, 126] class = Malignant
texture3 <= 33.27 entropy = 0.065 samples = 258 value = [256, 2] class = Benign
fractal_dimension2 <= 0.004 entropy = 0.971 samples = 5 value = [3, 2] class = Benign
entropy = 0.0 samples = 1 value = [1, 0] class = Benign
entropy = 0.0 samples = 8 value = [0, 8] class = Malignant
entropy = 0.0 samples = 14 value = [14, 0] class = Benign
radius3 <= 16.8 entropy = 0.792 samples = 42 value = [10, 32] class = Malignant
radius3 <= 16.975 entropy = 0.811 samples = 4 value = [1, 3] class = Malignant
entropy = 0.0 samples = 123 value = [0, 123] class = Malignant
entropy = 0.0 samples = 240 value = [240, 0] class = Benign
texture1 <= 23.24 entropy = 0.503 samples = 18 value = [16, 2] class = Benign
entropy = 0.0 samples = 2 value = [0, 2] class = Malignant
entropy = 0.0 samples = 3 value = [3, 0] class = Benign
smoothness2 <= 0.006 entropy = 0.918 samples = 12 value = [8, 4] class = Benign
concavity3 <= 0.191 entropy = 0.353 samples = 30 value = [2, 28] class = Malignant
entropy = 0.0 samples = 1 value = [1, 0] class = Benign
entropy = 0.0 samples = 3 value = [0, 3] class = Malignant
entropy = 0.0 samples = 2 value = [0, 2] class = Malignant
entropy = 0.0 samples = 16 value = [16, 0] class = Benign
texture1 <= 16.165 entropy = 0.503 samples = 9 value = [8, 1] class = Benign
entropy = 0.0 samples = 3 value = [0, 3] class = Malignant
concavity1 <= 0.049 entropy = 0.971 samples = 5 value = [2, 3] class = Malignant
entropy = 0.0 samples = 25 value = [0, 25] class = Malignant
entropy = 0.0 samples = 1 value = [0, 1] class = Malignant
entropy = 0.0 samples = 8 value = [8, 0] class = Benign
entropy = 0.0 samples = 3 value = [0, 3] class = Malignant
entropy = 0.0 samples = 2 value = [2, 0] class = Benign





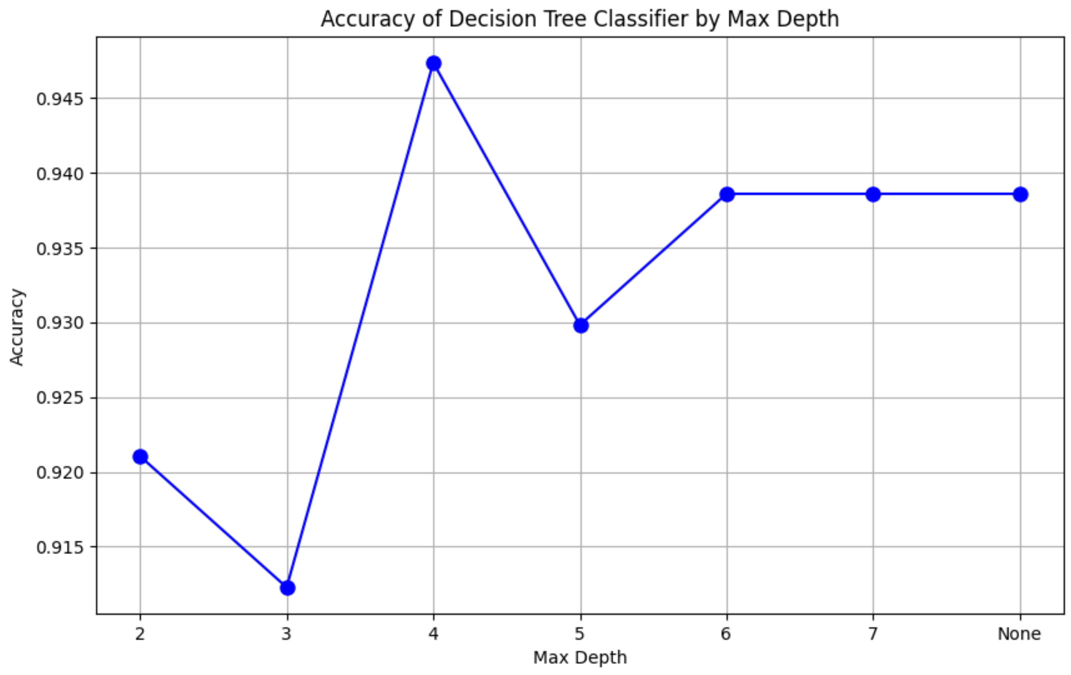
**Observation**: Since the dataset is relatively small (529 samples), the training data isn't very diverse. As a result, the maximum depth of the tree doesn't change significantly across the levels 6, 7, and None

We calculate the accuracy of each model at different depths, and summarize the results in the following table.


Decision Tree Accuracy by max_depth

| max_depth | Accuracy |
|---|---|
| 2 | 0.92 |
| 3 | 0.91 |
| 4 | 0.95 |
| 5 | 0.93 |
| 6 | 0.94 |
| 7 | 0.94 |
| None | 0.94 |

Or visualize it with a line chart for easier comparison.



Max Depth = 4: The model achieves the highest accuracy. So this is the best model for the data set which split into 80/20 (train/test).

Max Depth = 6, 7, None: The accuracy is the same because the models are similar, as previously noted.

Max Depth = 2, 3: The accuracy is lower due to many constraints leading to early termination of the models.

Max Depth = 5: Although there are fewer constraints, resulting in a decrease in accuracy, it is still higher than at depths 2 and 3.

## 6. Self Grading

| No. | Specifications | Percent |
|---|---|---|
| 1 | Preparing the data sets | 100% |
| 2 | Building the decision tree classifiers | 100% |
| 3 | Classification report and confusion matrix | 100% |
|  | Comments | 100% |
| 4 | Trees, tables, and charts | 100% |
|  | Comments | 100% |

## 7. Reference

[1]: Understanding the Confusion Matrix in Machine Learning - GeeksforGeeks (View at 23/08/2024)

[2]: Matplotlib Tutorial - GeeksforGeeks (View at 22/08/2024)

[3]: Building and Implementing Decision Tree Classifiers with Scikit-Learn: A Comprehensive Guide - GeeksforGeeks (View at 22/08/2024)