Machine Learning (CS60050) – Assignment 1

Rajat Bachhawat (19CS10073)

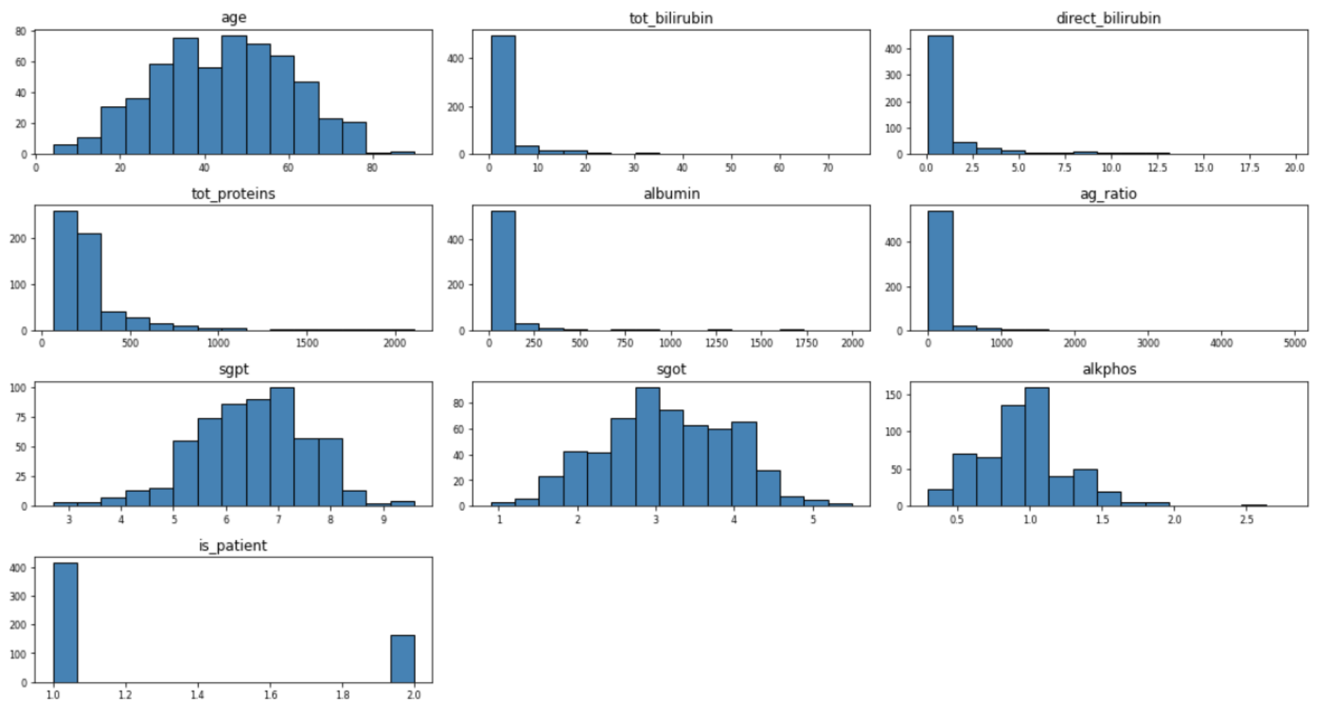
Kaushal Banthia (19CS10039)

REPORT

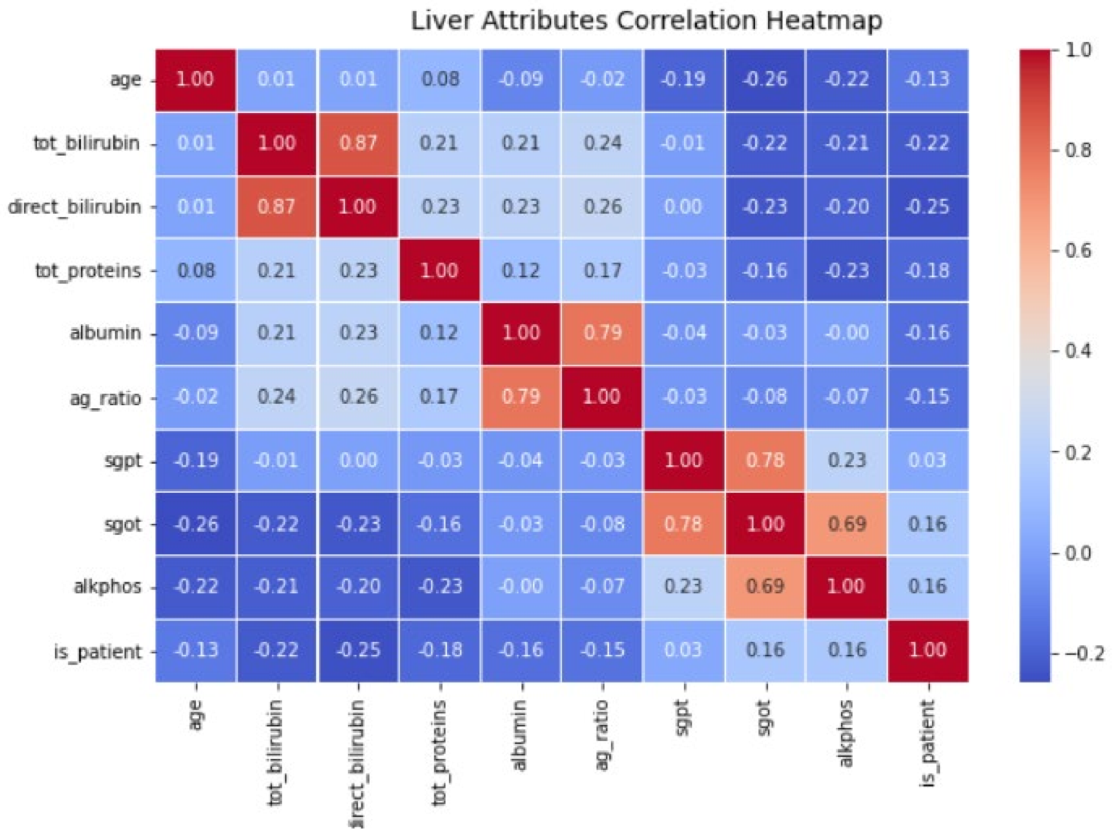
Question 1 Part 1

- The code is available on Google Colab using this link.
- The code is also added in the zip file as a .ipynb file.
- The creation of the decision tree was done using the **ID3 Algorithm**, coupled with **Reduced Error Pruning (post-pruning)**.
- We begin with importing the Indian Liver Patient Dataset (ILPD), and dropping those rows that have NA values in any of their columns and doing some **exploratory analysis on the dataset** using inbuilt `pandas` functions and some graphs from `matplotlib` and `seaborn`.

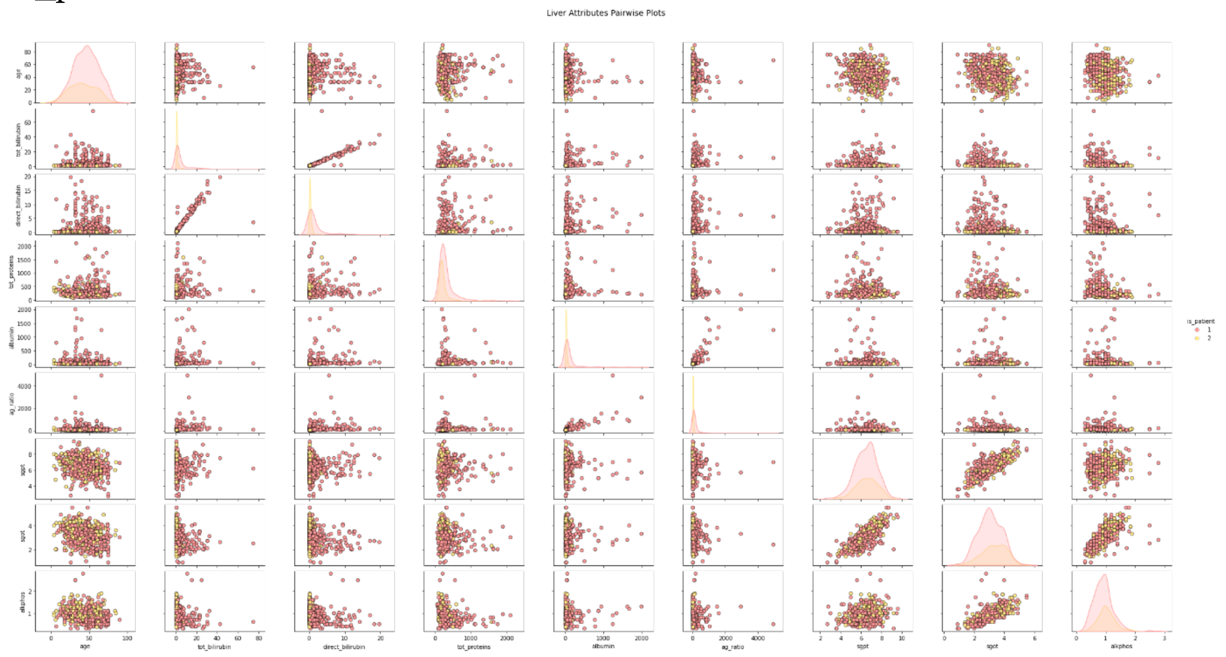
The first graph we make is the **histograms** of the various attributes, with their value on the x axis and their frequency on the y axis. It helps us to know how the attributes are distributed.



The next graph that we make is a **heatmap**. This lets us know the correlation between any two attributes. This knowledge enables us to drop any columns that have high correlation with another column, so that the model doesn't have any redundancy in it.



We then move on to a **pairplot**, marked with the hue of the 'is_patient' column (as it is the target variable). It gives us the relation between 2 attributes with respect to the 'is_patient' column.



Question 1 Part 1

- We have created a few functions that help us in building the Decision Tree Classifier.

  1. `accuracy_score()`: Calculates the accuracy of the predicted data, by comparing it with the actual data.
  2. `findEntropy()`: Calculates the entropy for the data in the current node.
  3. `findGini()`: Calculates the gini impurity for the data in the current node.
  4. `findGain()`: Calculate the information gain / gini gain given current node's impurity and the data at the two child nodes
  5. `split()`: Split the examples in the current node based on the passed attribute-value pair. If the values are continuous, classify >=value as TRUE, else FALSE. Else, classify == value as TRUE, else FALSE.
  6. `findBestSplit()`: Find the best splitting of the passed examples by checking for all the available attributes and their values. For continuous valued attributes, we do a binary split on each of the candidate thresholds (after sorting by values), and use the one which gives best gain. We then return the attribute-value pair with the max gain.

- A few classes were also made, for the process of creating our Decision Tree Classifier.

  1. `Node`: This class is used for a Decision Tree Node. Contains information about the **attribute-value pair** that it uses for splitting, **right child and left child**, the **gain**, the **vote** value, the **count of each label** and whether the node **is a leaf** node or not. It has the following functions:

     - `__init__()`: Assigns all the information to the current Node.

  2. `Decision Tree`: This is the main class for building our Decision Tree Classifier and it contains all the major functions that are necessary for building the tree, predicting the output and pruning the tree. It allows user to choose the **impurity measure (gini/entropy)** and **max depth of the tree,** by constructing with appropriate arguments. It has the following functions:

     - `__init__()`: Assigns the information to the Decision Tree.
     - `buildTree()`: Builds the decision tree recursively given the training examples. It is ensured that no attribute is repeated in any

path of the decision tree (as specified in ID3). It builds the tree by creating a Node and then calling itself to create the left and the right child of the current Node. The base condition for the recursive function is, when we create a Node, that also happens to be a Leaf Node.

- `classify()` : Classifies the given example and returns the output (the vote of the current node).
- `predict()` : Returns the list of predictions made by the Decision Tree on the input Dataset.
- `prune()` : Prunes the tree according to the best candidate Node (Which causes the highest increase in the accuracy). Makes use of the BFS traversal, to go through the entire tree and make each node as the leaf node once (temporarily) and checks the accuracy. Once the best Node is found (that gives the highest accuracy increase), it is made into a leaf node and its right and left child are changed to None. These steps are repeated over and over again until, we do not get a best Node which increases the old accuracy.

We first make the impurity function equal to **Gini index (gini gain)**.

Question 1 Part 2

- After making provisions to build and use the classifier, we start with the training and the testing of the model.
- **PROCEDURE:** We set the impurity function to be **gini** and then we create 10 variants of the dataset (by splitting it randomly into train, test and validation sets. The split is done so as to make 80% of the data as training data and 20% of the data as test data. The training data is then further split into 80% actual training data and 20% validation data, as we need the validation set for pruning) and build the Decision Tree over each variant and make predictions. We then choose that variant, which provides us with the best test accuracy. We then build our model over that dataset.
- **RESULT:**

The 10 trees that we have trained give the following accuracy, depth and number of nodes.

```
Tree 1 Accuracy = 58.620689655172406 % Depth = 10 No. Of Nodes = 119

Tree 2 Accuracy = 64.65517241379311 % Depth = 10 No. Of Nodes = 103
Tree 3 Accuracy = 69.82758620689656 % Depth = 10 No. Of Nodes = 87

Tree 4 Accuracy = 66.37931034482759 % Depth = 10 No. Of Nodes = 117

Tree 5 Accuracy = 66.37931034482759 % Depth = 10 No. Of Nodes = 123

Tree 6 Accuracy = 61.206896551724135 % Depth = 10 No. Of Nodes = 155

Tree 7 Accuracy = 65.51724137931035 % Depth = 10 No. Of Nodes = 113

Tree 8 Accuracy = 60.3448275862069 % Depth = 10 No. Of Nodes = 99

Tree 9 Accuracy = 67.24137931034483 % Depth = 10 No. Of Nodes = 89

Tree 10 Accuracy = 70.6896551724138 % Depth = 10 No. Of Nodes = 133

Average Accuracy over 10 random 80/20 splits = 65.08620689655172 %

Best Accuracy over 10 random 80/20 splits = 70.6896551724138 %
```

Here we choose the split that gives us the best accuracy: **70.69%**.

Also, the average accuracy over the 10 splits is **65.08%**.

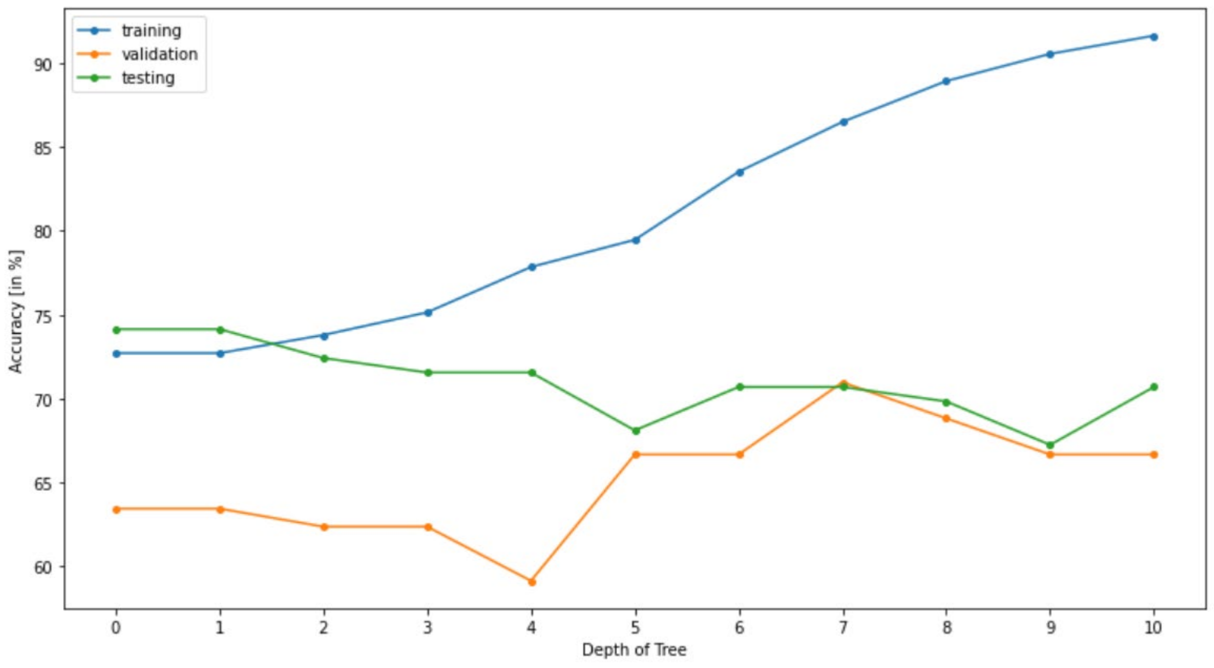
- **CONCLUSION:** We then go over testing the best tree (`dTree1`) with its accuracy vs the depth of the tree and its accuracy vs the number of nodes in the tree and get the following graphs for the train, test and validation sets.

Question 1 Part 3

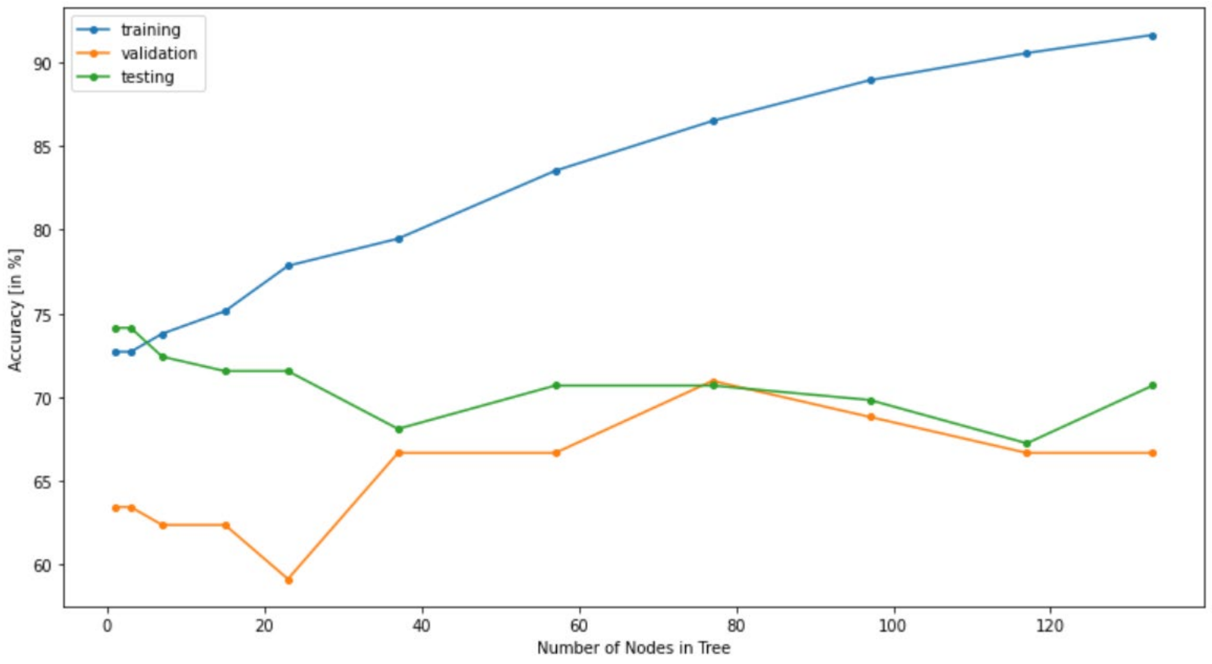
- **PROCEDURE:**

```
for depth between 0 and actual_depth_of_tree:
  retrain decision tree on training data (dTree)
with constraint: depth is the max_depth
  numNodes <- number of nodes in dTree
  predictions <- dTree.predict() on training,
testing, validation sets
  obtain accuracy of each of the predictions
  plot (depth, accuracy_train)
  plot (depth, accuracy_test)
  plot (depth, accuracy_valid)
  plot (numNodes, accuracy_train)
  plot (numNodes, accuracy_test)
  plot (numNodes, accuracy_valid)
```

See observed graphs below



Accuracy [in %] vs Depth of Tree



Accuracy [in %] vs Number of Nodes in Tree

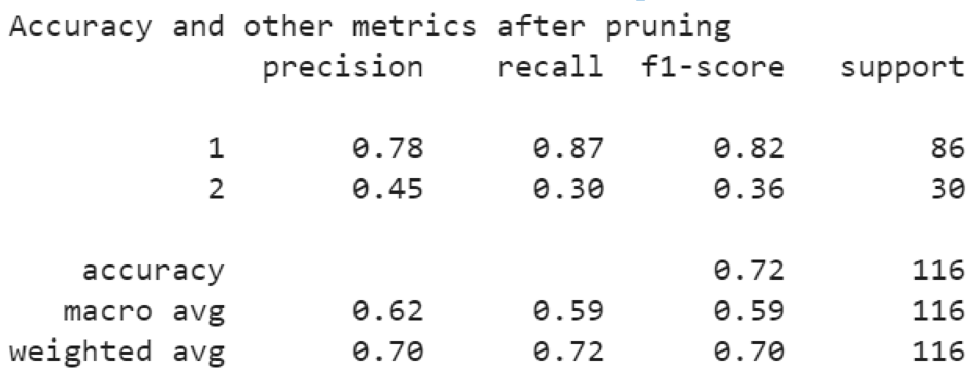
CONCLUSION: From these graphs, the **best depth** of the tree can be estimated to be **7**, as at that depth, the validation accuracy is the highest and the testing accuracy is also reasonably good. We can see that the training accuracy could improve with an increase in the depth, but that would lead to the corresponding decrease in the test and the validation accuracy (overfitting). Although the testing accuracy is greater when the depth is 0 or 1, but that would be a case of massive underfitting and hence we are not considering that case.

Question 1 Part 4

- **PROCEDURE:**

```
old_accuracy <- accuracy of dTree over validation set
do:
    for each non-leaf node in dTree:
            temporarily make it a leaf
            new_accuracy = accuracy of dTree over validation set
            if new_accuracy > best_new_accuracy
                    best_new_accuracy = new_accuracy
                    best_candidate = curr_node
            make it non leaf again
    if best_new_accuracy > old_accuracy
            update old_accuracy
            make best_candidate a leaf
    else
            break
```

- **RESULT:** After these observations, we prune the tree using Reduced Error Pruning (a post-pruning method) and see that the depth of the tree reduces to 6 and the accuracy increases from **70.69%** before pruning to **72.41%** after pruning. We also print the classification report after importing it from sklearn. It gives us the Precision, Recall and the F1 Score of the predictions too.

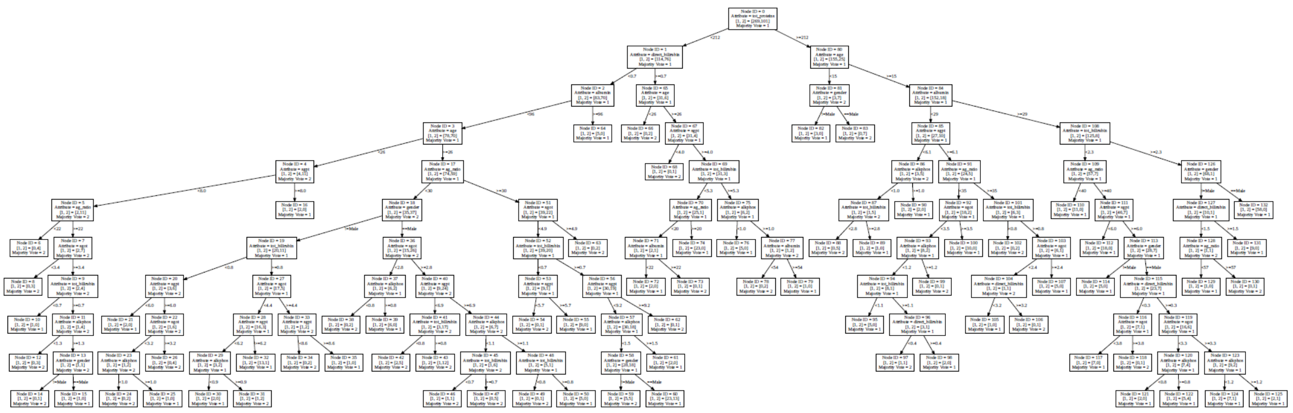


```
Accuracy and other metrics after pruning
              precision    recall  f1-score   support

           1       0.78      0.87      0.82        86
           2       0.45      0.30      0.36        30

    accuracy                           0.72       116
   macro avg       0.62      0.59      0.59       116
weighted avg       0.70      0.72      0.70       116
```

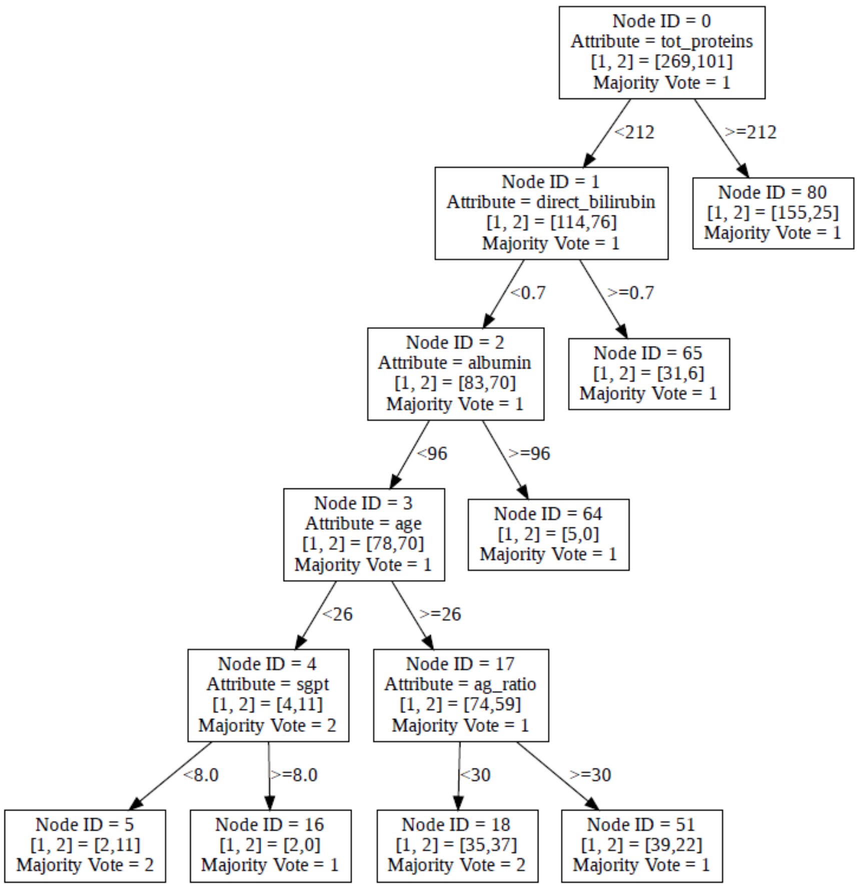
- **CONCLUSION:**
    - The model complexity reduces as the tree becomes concise (number of nodes significantly decreased).
    - Thus, we avoided the overfitting of the training data, as the tree becomes more general.
    - We also saw that post-pruning helps increase the test accuracy.

Question 1 Part 5 (for Gini)

- Finally, we print the complete decision tree using the `graphviz` library. Here are the complete trees before and after pruning. (The tree before pruning is large and hence would not be clearly visible in this report. It can be viewed clearly in the accompanying PDF file in the zip file).



Decision Tree before Pruning



Decision Tree after Pruning

We now change the impurity function to **Entropy (information gain)** instead of gini and repeat from Question 1 Part 2.

Question 1 Part 2

- **PROCEDURE:** Now we set the impurity function to **entropy (information gain)** instead of **gini** and then create 10 variants of the dataset (by splitting it randomly into train, test and validation sets. The split is done so as to make 80% of the data as training data and 20% of the data as test data. The training data is then further split into 80% actual training data and 20% validation data, as we need the validation set for pruning) and build the Decision Tree over each variant and make predictions. We then choose that variant, which provides us with the best test accuracy. We then build our model over that dataset.
- **RESULT:**
  The 10 trees that we have trained give the following accuracy, depth and number of nodes.

```
Tree 1 Accuracy = 58.620689655172406 % Depth = 10 No. Of Nodes = 107
Tree 2 Accuracy = 68.10344827586206 % Depth = 10 No. Of Nodes = 77
Tree 3 Accuracy = 68.10344827586206 % Depth = 10 No. Of Nodes = 75
Tree 4 Accuracy = 60.3448275862069 % Depth = 10 No. Of Nodes = 71
Tree 5 Accuracy = 69.82758620689656 % Depth = 10 No. Of Nodes = 83
Tree 6 Accuracy = 68.10344827586206 % Depth = 10 No. Of Nodes = 57
Tree 7 Accuracy = 63.793103448275865 % Depth = 10 No. Of Nodes = 61
Tree 8 Accuracy = 64.65517241379311 % Depth = 10 No. Of Nodes = 57
Tree 9 Accuracy = 68.96551724137932 % Depth = 10 No. Of Nodes = 87
Tree 10 Accuracy = 70.6896551724138 % Depth = 10 No. Of Nodes = 83
Average Accuracy over 10 random 80/20 splits = 66.12068965517241 %
Best Accuracy over 10 random 80/20 splits = 70.6896551724138 %
```

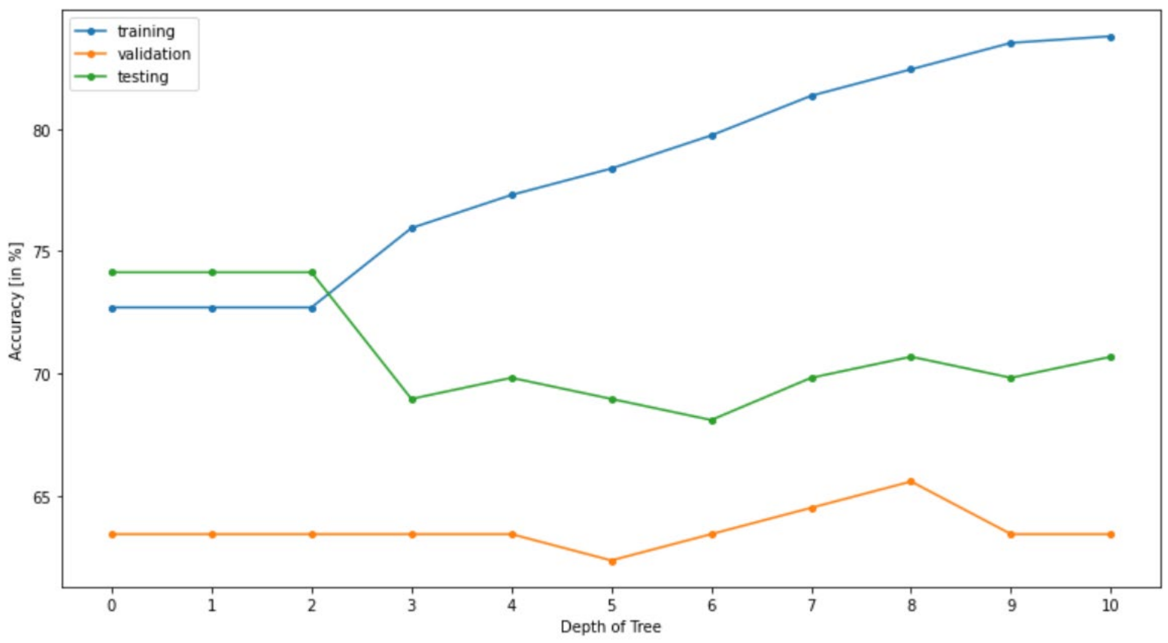
Here we choose the split that gives us the best accuracy: **70.69%**.

Also, the average accuracy over the 10 splits is **66.12%**.

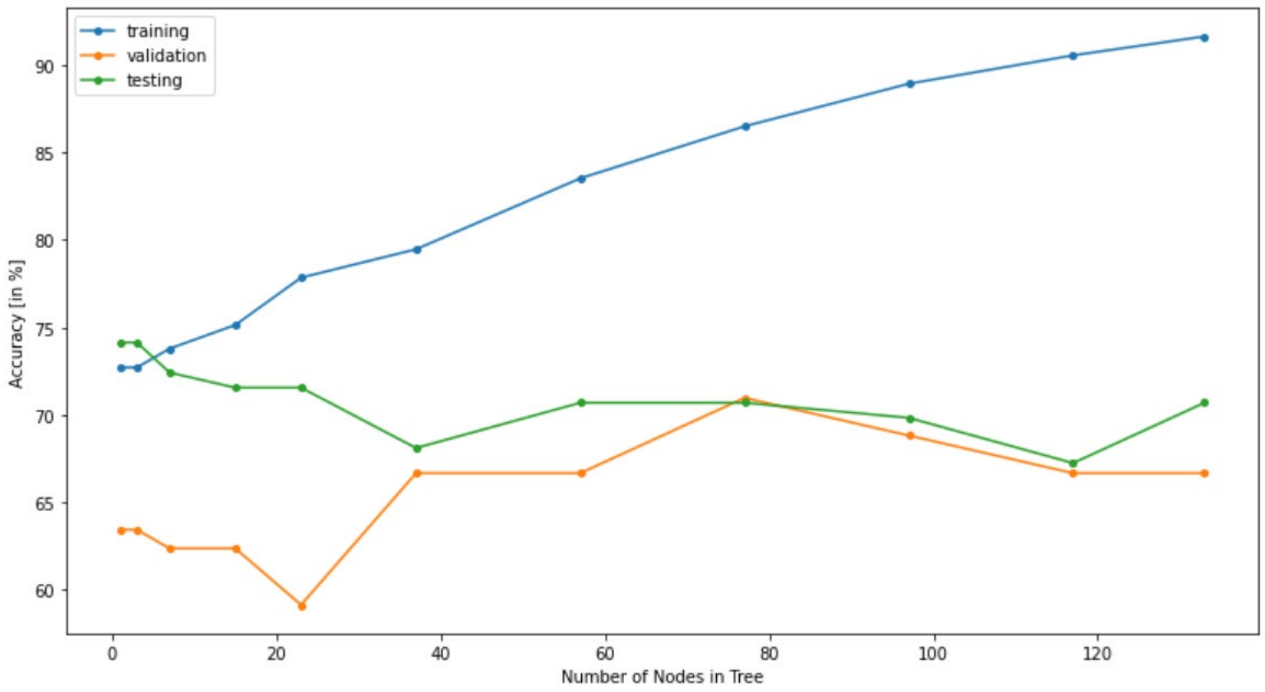
- **CONCLUSION:** We then go over testing the tree with its accuracy vs the depth of the tree and its accuracy vs the number of nodes in the tree and get the following graphs for the train, test and validation sets.

Question 1 Part 3

➢ **PROCEDURE (Same as gini gain section)**



Accuracy [in %] vs Depth of Tree

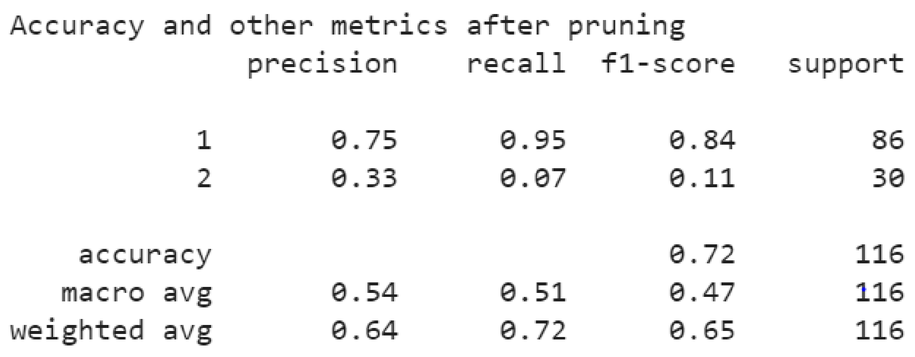


Accuracy [in %] vs Number of Nodes in Tree

CONCLUSION: From these graphs, the best depth of the tree can be estimated to be 8, as at that depth, the validation accuracy is the highest and the testing accuracy is also reasonably good. We can see that the training accuracy could improve with an increase in the depth, but that would lead to the corresponding decrease in the test and the validation accuracy (overfitting). Although the testing accuracy is greater when the depth is 0, 1 or 2, but that would be a case of underfitting and hence we are not considering that case.

Question 1 Part 4

- **PROCEDURE (Same as gini gain section)**
- **RESULT:** After these observations, we prune the tree using Reduced Error Pruning (a post-pruning method) and see that the depth of the tree reduces to 4 and the accuracy increases from **70.69%** before pruning to **72.41%** after pruning. We also print the classification report after importing it from sklearn. It gives us the Precision, Recall and the F1 Score of the predictions too.



```
Accuracy and other metrics after pruning
              precision    recall  f1-score   support

           1       0.75      0.95      0.84        86
           2       0.33      0.07      0.11        30

    accuracy                           0.72       116
   macro avg       0.54      0.51      0.47       116
weighted avg       0.64      0.72      0.65       116
```

- **CONCLUSION:**
    - The model complexity reduces as the tree becomes concise (number of nodes significantly decreased).
    - Thus, we avoided the overfitting of the training data, as the tree becomes more general.
    - We also saw that post-pruning helps increase the test accuracy.

Comparison between Gini Gain and Info Gain [Continuation of Question 1 Part 1]

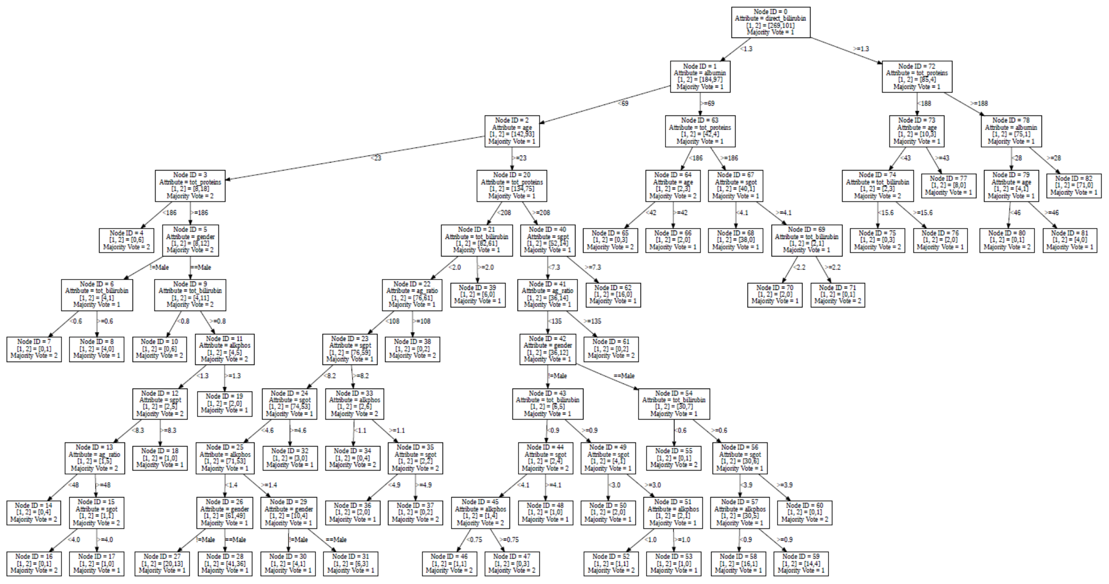
As we have now observed decision trees using both gini gain and information gain as impurity measures, we saw the following interesting results:

- Both gini gain and information gain gave decision trees with **equal best accuracy.**
- Information gain (entropy) gave **better average accuracy** over 10 random data-splits than gain gain.
- Information gain (entropy) gave trees with **lesser number of nodes** both before and after pruning compared to gini gain.
- Both gini gain and information gain gave decision trees with **equal accuracy** after pruning.

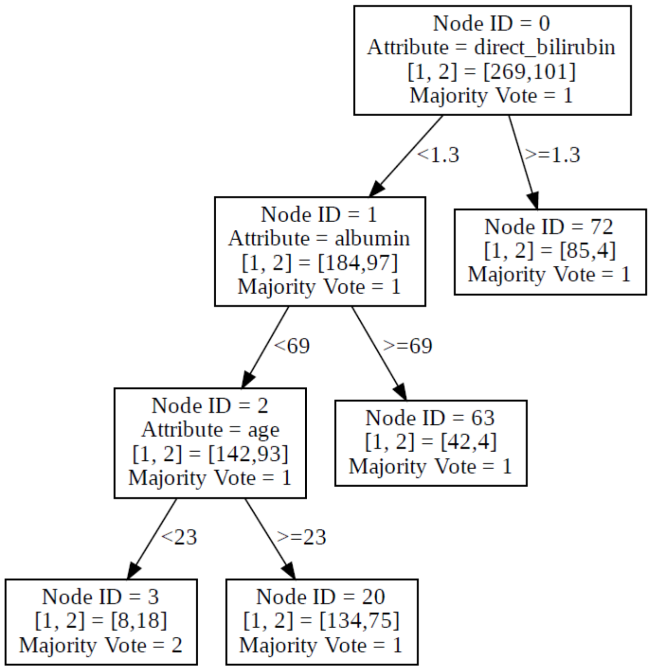
We can conclude that for our dataset, information gain (entropy) was the slightly more fruitful impurity measure.

Question 1 Part 5 (for Entropy)

➢ Finally, we print the complete decision tree using the `graphviz` library. Here are the complete trees before and after pruning. (The tree before pruning is large and hence would not be clearly visible in this report. It can be viewed clearly in the accompanying PDF file in the zip file).



Decision Tree before Pruning



Decision Tree after Pruning