Machine Learning Project

Suryam Arnav Kalra

Procedure :

We have used the ID3 algorithm to build the decision tree using the two impurity measures gini index and information gain.

1) Importing Dataset

a) Imported the dataset from the heart.dat file provided
b) It contained 270 data samples, the data contained 13 features and 1 column for target values.
c) There were all real valued features and the target value can have only two values (1 and 2)

2) Splitting Dataset for training and validation

a) splitted the dataset by averaging over 10 random 80/20 splits
b) 80 percent of the data is used for training and the rest 20 percent is used for testing.
c) Every Time before splitting the data we did a random shuffle with a different seed
.

3) Building Decision tree

a) After the splitting of the dataset we built decision trees having different depths.
b) Used both the impurities gini index and information gain and got the accuracies for different depths.
c) The above found information is used in plotting the depth vs accuracy graph.

4) Building Decision tree using gini index

a) Wrote a recursive function that builds the tree top to down.
b) At any particular node we used gini index to calculate which feature should be used to divide the tree from that node.
c) To get the best split at a node we first looped through every feature and computed it's gini index.
d) In the above mentioned process we considered every value given in dataset for that feature to be a threshold once
e) After calculating and storing the gini index for every feature along with the best threshold, we then sorted the computed values to get the best feature and the best threshold.
f) Using this feature and threshold we splitted the data into two parts and then recursively called the function to build the left and the right subtree of this particular node.

5) Building Decision tree using Information Gain

a) Wrote a recursive function that builds the tree top to down.
b) At any particular node we used information gain to calculate which feature should be used to divide the tree from that node.
c) To get the best split at a node we first looped through every feature and computed it's gini index.
d) In the above mentioned process we considered every value given in the dataset for that feature to be a threshold once.
e) In order to get the information gain for a particular feature we divided the dataset into two parts on the basis of a threshold and calculated the difference in entropy before and after the division.
f) After calculating and storing the information gain for every feature along with the best threshold, we then sorted the computed values to get the best feature and the best threshold.

g) Using this feature and threshold we splitted the data into two parts and then recursively called the function to build the left and the right subtree of this particular node.

6) Taking the tree with best accuracy

a) To get the tree with best accuracy we splitted the data 10 times using random shuffle and random seed.
b) In splitting we made an 80/30 split, 80% for training and 20 % for testing.
c) After splitting we created two trees for every iteration one using gini index and other using information gain
d) Then calculated the accuracy for these trees
e) We stored the tree with maximum accuracy and returned it's head.

7) Pruning the tree

a) For pruning we used the tree with the best accuracy.
b) We then pruned the tree details of which are given in a later section.

8) Printing the Decision tree

a) For printing the decision tree we used the pruned tree.
b) We made a digraph of the tree with some data (Predicted Class , Feature Index , Threshold) to be printed at each node
c) After this printed the tree using the library Sklearn and graphviz.

Functions:

1) **import_data**: Imports the data from the given 'heart.dat' file and converts it to a numpy array with the elements being in float data type.

2) **get_validation_and_train_data**: This function randomly shuffles the data imported from the function above and does a 80:20 split to get the training and testing data. The data is further divided into train_X, train_Y, test_X, test_Y sets which denote respectively the training attributes, target values for training, testing attributes and target values for testing.

3) **get_entropy:** This function is used to calculate the entropy of the data present at a given node as shown below:

$$E(S) = \Sigma - p \log_2 (p)$$

4) **calculate_gini_index**: This function is used to calculate the gini index to choose the best attribute that should be used for splitting at a particular node. Lower the gini index for an attribute the better it is.

$$\text{gini impurity} = 1 - \Sigma p^2$$

$$\text{gini index} = \Sigma \frac{|S_V|}{|S|} \text{gini\_impurity}(\mathrm{S}_V)$$

5) **calculate_information_gain:** This function is used to calculate the information gain to choose the best attribute that should be used for splitting at a particular node. Higher the information gain for an attribute the better it is.

$$I.G. = E(S) - \Sigma \frac{|S_V|}{|S|} E(S_V)$$

6) **split_data:** This function is used to split the data present at a node into the left subpart and right subpart based on the attribute and threshold chosen at this node using gini index or information gain.

7) **get_best_split_gini_index**: This function is used to get the best split of data based on the impurity measure gini index at a particular node.

8) **get_best_split_information_gain:** This function is used to get the best split of data based on the measured information gain at a particular node.

9) **construct_tree_using_gini_index**: This function is used to construct the tree based on the ID3 algorithm. It uses the measure gini index to choose the best attribute for splitting at a particular node.

10) **construct_tree_using_information_gain**: This function is used to construct the tree based on the ID3 algorithm. It uses the measure information gain to choose the best attribute for splitting at a particular node.

11) **predict**: This function is used to predict the value of one single data set. It follows the path in the tree based on the attribute used at that node for splitting.

12) **predict_target_values**: This function calls the **predict()** function for predicting the target values of each and every data present in the test set.

13) **check_accuracy**: This function calculates the accuracy of our prediction by checking against the target test values.

14) **get_best_root**: This function calls the **get_validation_and_train_data()** 10 times to get the randomly split data and constructs the tree for this data using both the gini index and information gain. It then returns the tree with the best test accuracy obtained till now.

15) **get_plot**: This function is used to get the plot using the matplotlib library and save it as a jpeg file.

16) **count_nodes:** This function is used to count the number of nodes present in the tree.

17) **get_depth_and_number_of_nodes_plot**: This function calls the **get_plot()** function to get the plots of accuracy vs depth and accuracy vs number of nodes.

18) **is_leaf**: This function is used to check whether the node we are currently seeing is a leaf node or not.

19) **prune**: This function is used for pruning the tree using Reduced Error Pruning method.

20) **get**: This function is used to get the characteristics of a particular node such as the predicted class of that node, the best attribute chosen to split at that node and the best threshold on which the split was done. These characteristics are used while printing the tree.

21) **print_tree**: This function is used to print the tree using the graphviz module and uses the **get()** function to get the characteristics of each node.

Pruning:

There are two methods for pruning the tree: Pre-pruning and Post-pruning. We have kept the availability of both in the code.

1) Pre-pruning the tree

a) It is the process in which we stop growing the tree further from a particular node if enough data is not available for splitting at that node.
b) We have provided a provision for this in the code by passing the size of the minimum data required as a parameter to the construct_tree_using_gini_index() and construct_tree_using_information_gain() function.
c) When the size of the data falls below this threshold we simply return the node from that level and stop any further splitting.
d) By default, we have set it to zero as no pre-pruning was to be done in the assignment.

2) Post-pruning the tree

a) It is the process in which we first grow the tree to its full capacity and then remove the nodes to achieve a better test accuracy.
b) For this we use a depth-first based search strategy (DFS) to first start pruning the trees from the leaf level and then approaching towards the root.
c) Let us consider we are at a particular node (N). First, we try pruning the left child of N and then the right child of N by recursively calling the same function.
d) After the pruning of the children is done, we try pruning the current node N.
e) We compare the accuracies on the test data obtained from the tree obtained till now and the tree obtained after setting the left and right child of N to Null so as to prune the node N.
f) If the accuracy obtained after pruning is higher than the accuracy obtained before pruning then we do not restore the children of the node N and return to its parent.
g) Otherwise, we restore the children of the node N since pruning didn't give us any better accuracy.

Results:

1) Analyze impact of gini index and information gain

a) The trees constructed using gini index and information gain have similar average accuracy as can be seen below.

```
Accuracy_Gini_Tree: 0.7962962962962963
Accuracy_Info_Tree: 0.7592592592592593
```

b) Since both are measuring the impurity of the data present at the given node, the result of similar accuracy was expected.

c) Underlying complexity of calculating gini index is slightly less than the complexity of calculating information gain due to repeated calls for getting the entropy of the nodes.

d) The attributes taken by gini index for splitting and those taken by information gain for splitting are **almost** the same with **almost** the same threshold values as can be seen from the trees given below

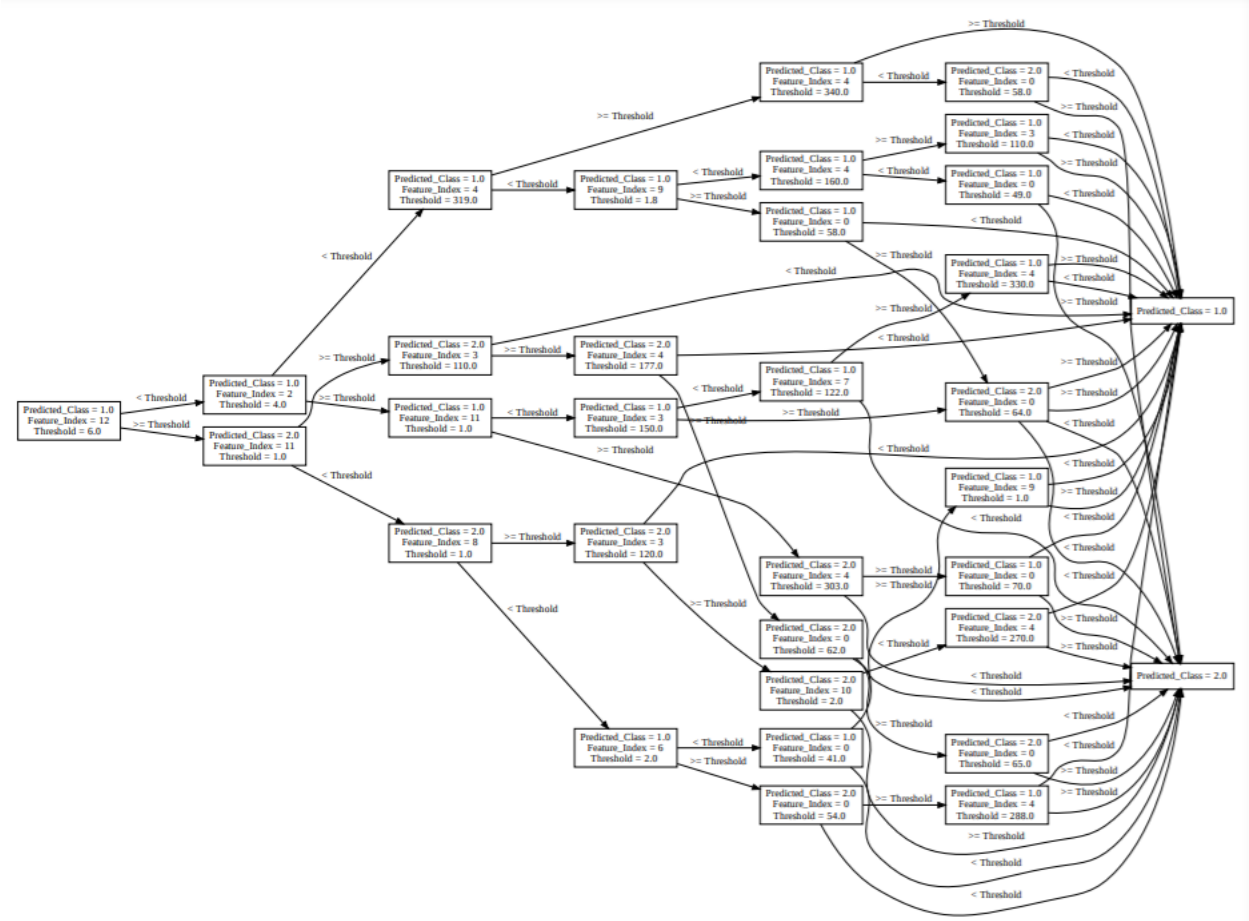

Fig a) Tree obtained from gini index

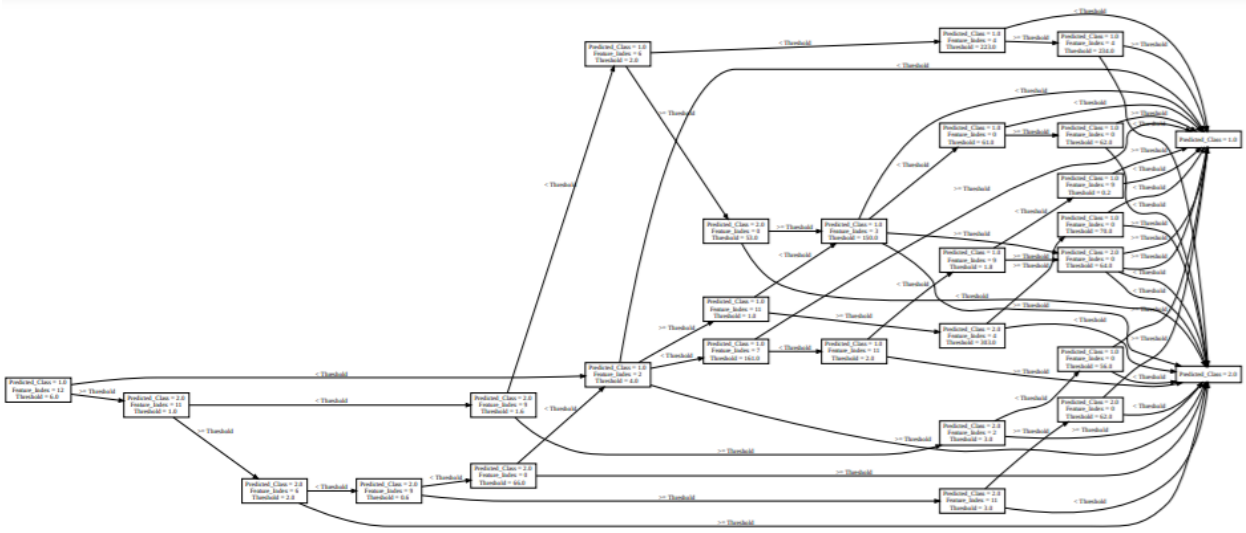

Fig b) Tree obtained from information gain

e) Also, the complexity of the tree is similar with both the trees having a **similar** number of nodes in their structure.

```
Nodes_Gini_Tree:   65
Nodes_Info_Tree:   59
```

2) Accuracy by averaging 80/20 split , best accuracy

a) The average accuracy obtained after randomly splitting the data 10 times and constructing the tree (full/maximum height trees) is given below:

```
Average accuracy using gini index:  0.7314814814814816
Average accuracy using information gain: 0.7296296296296296
```

b) The accuracy of the best tree (the desired one) obtained from the above random splits is given below:

```
Accuracy_best_root: 0.9444444444444444
```

3) Depth vs Accuracy plot

a) With increasing depth we can observe that the accuracy reaches a certain maxima and then drops for a while and then after a particular depth, the accuracy becomes stagnant for the test data.
b) The behaviour given above was expected from the trees since with increasing depth we are overfitting the tree to the training data which results in higher generalization error which can also be seen with the decrease in accuracy in the given plots.
c) It can also be observed that the accuracy follows the same pattern for both the impurity measures as was expected.
d) Therefore, for this data the **depth of around 6 to 8 was optimal** for a low training error as well as a low generalization error (test error).

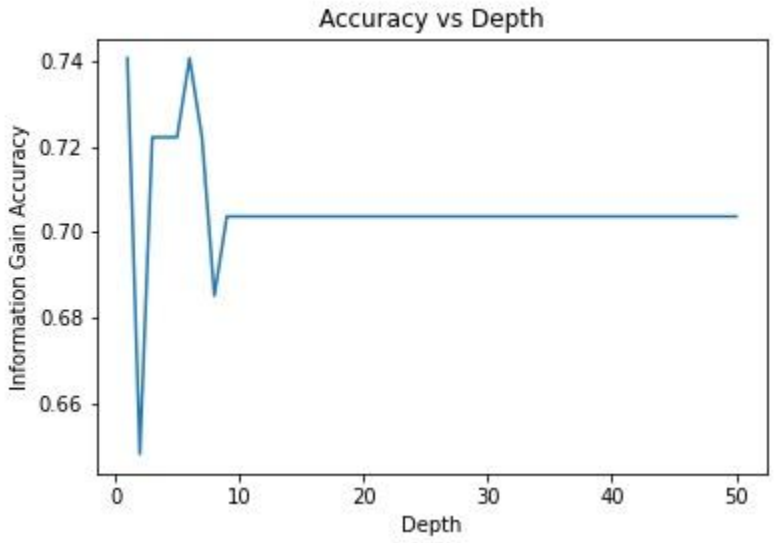

Fig c) Accuracy vs Depth for Tree constructed using Information gain

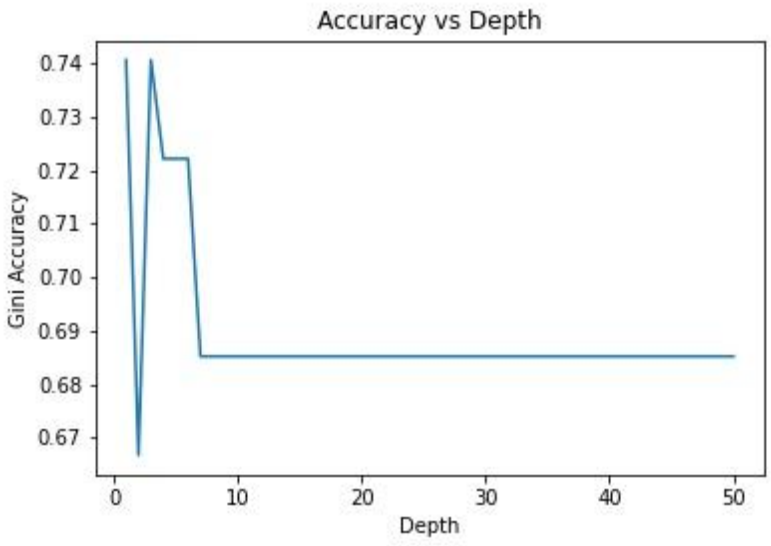

Fig d) Accuracy vs Depth for Tree obtained from Gini Index

4) Number of Nodes vs Accuracy plot

a) We can observe that with the increasing number of nodes, that is increasing depth as well as increasing complexity the accuracy decreases as was expected.

b) With the increasing number of nodes, we are overfitting the tree to the training data which results in a lower training error but a very high generalization error which in turn gives us lower accuracy.

c) It is in harmony with **Ocam's Razor** that a simpler tree gives a better result than a more complex tree.

d) Therefore, **nodes around 30 to 40 are optimal** for this dataset which are obtained at a height of around 6 to 8.

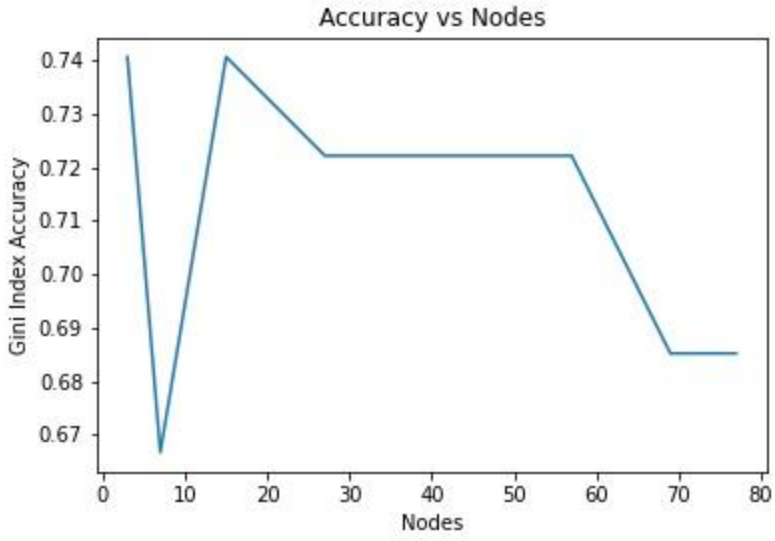

Fig e) Accuracy vs Nodes for Tree obtained with Gini Index

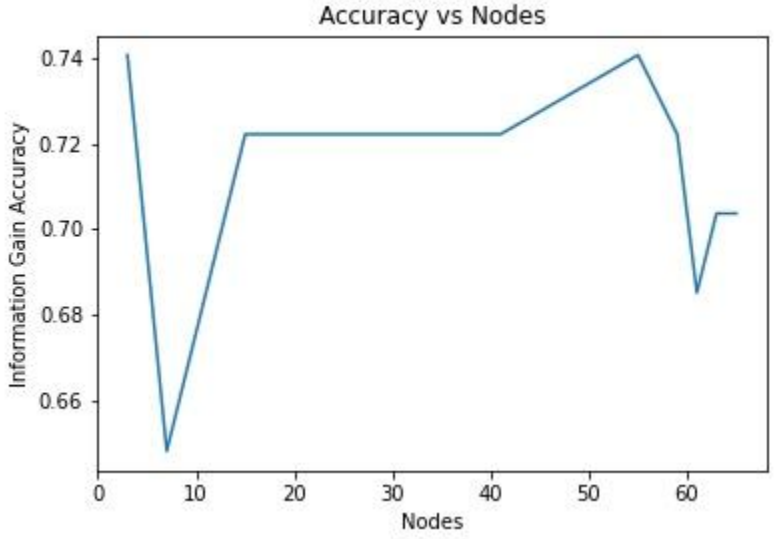

Fig f) Accuracy vs Nodes for Tree obtained with Information Gain

5) Results after Pruning

a) The accuracy of the best tree before pruning was as shown below:

```
Accuracy_before_pruning:  0.9444444444444444
```

b) The accuracy of the best tree after pruning was as shown below:

```
Accuracy_after_pruning: 0.9629629629629629
```

c) The number of nodes of the best tree before pruning were as shown below:

```
Nodes_before_pruning:  87
```

d) The number of nodes of the best tree after pruning were as shown below:

```
Nodes_after_pruning:  37
```

e) Since we are doing post pruning which prunes the nodes after construction we can see that the number of nodes after pruning have decreased significantly as expected.

f) Since the original tree is overfitting the data, it gives a lower accuracy on the test data whereas since pruning only removes those subtrees whose removal result in a higher test accuracy the new pruned tree has a higher accuracy on the test data.

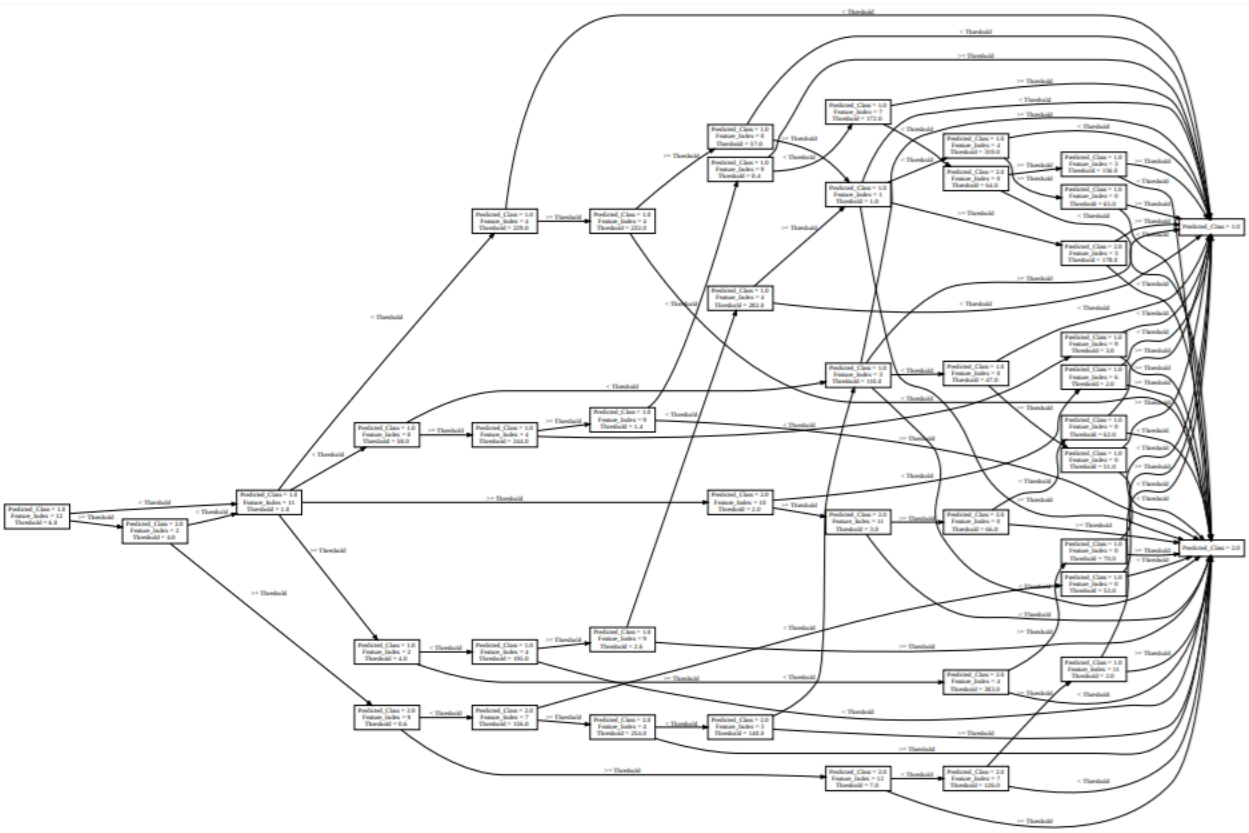

Fig g) Tree before pruning

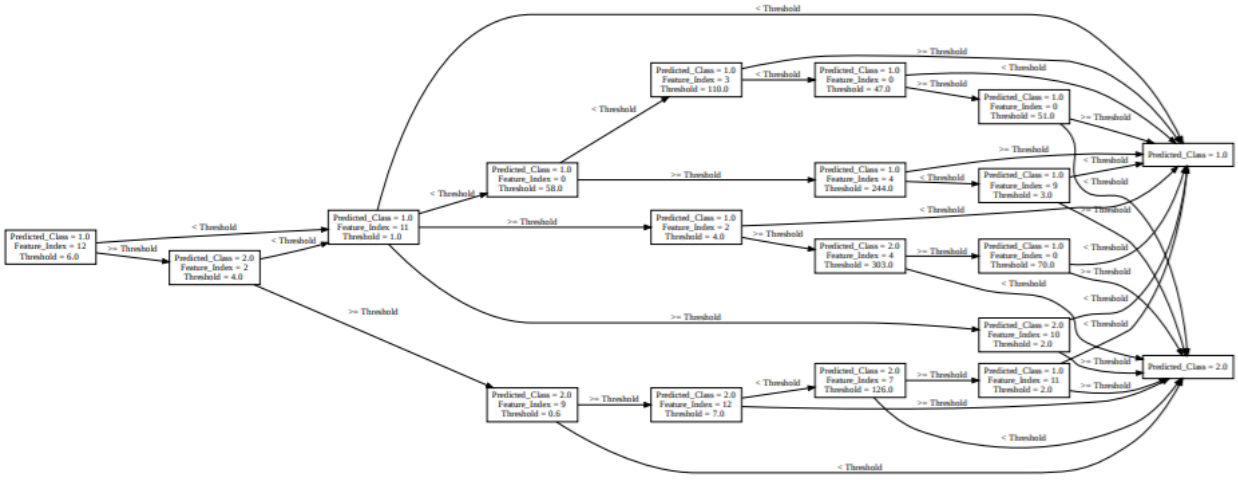

Fig i) Tree after pruning

Folder hierarchy:

1. main.py - This is the main file that call all the operations
2. train.py - This file contains all the functions for constructing the decision tree and it's pruning
3. predict.py - This file contains all the functions for predicting and checking the accuracy of the decision tree built.
4. plot.py - This file contains all the functions used for plotting the graph required in the assignment and it also contains the functions to print the decision tree, the graphs and the trees will be saved in the same folder with suitable file naming.
5. requirements.txt - This contains all the packages required for this code to run
6. Report.pdf - This contains a report for this assignment.
7. decision_tree_after_pruning.gv.pdf - This contains decision tree after pruning
8. decision_tree_before_pruning.gv.pdf - This contains decision tree after pruning
9. decision_tree_using_gini_index.gv.pdf - This contains decision tree constructed using gini index
10. decision_tree_using_info_gain.gv.pdf - This contains decision tree constructed using information gain
11. gini_index_accuracy.jpeg - This contains plot of Accuracy vs depth of tree constructed using gini index
12. gini_index_nodes.jpeg - This contains plot of Accuracy vs node of tree constructed using gini index
13. information_gain_accuracy.jpeg - This contains plot of Accuracy vs depth of tree constructed using information gain
14. information_gain_nodes.jpeg - This contains a plot of Accuracy vs node of tree constructed using information gain.

How to run the code:

1. Download this directory into your local machine
2. Copy the ‘heart.dat’ file in the Source Code Directory
3. Ensure all the necessary dependencies with required version and latest version of Python3 are available (verify with requirements.txt)

   ```
   pip3 install -r requirements.txt
   ```
4. Run the source code with the command

   ```
   python3 main.py
   ```
5. The image of the final decision tree and the four plots gini accuracy vs depth , info gain accuracy vs depth , gini index accuracy vs nodes and information gain vs nodes will be created in the same directory as the Source Code.

- Make sure graphviz is installed the command to install graphviz is
  - `pip3 install graphviz`
- **If you face any difficulty in installing packages, you can run the code on google collab.**