DECISION TREE

Assignment I

Anindya Sikdar (19CS10010)
Akshat Singh Rathore (19CS30003)
Indian Institute of Technology, Kharagpur
CS60050, Machine Learning

INDEX

Abstract

The report has been made as an answer to Assignment-I for CS60050 which encompasses the following questions:

Question

1. Build a decision-tree classifier by randomly splitting the dataset as 80/20 split. Use the impurity measures- 1) Gini index and 2) information gain. Analyze the impact of using individual impurity measures on the prediction.
2. Provide the accuracy by averaging over 10 random 80/20 splits. Consider that particular tree that provides the best test accuracy as the desired one.
3. What is the best possible depth limit to be used for your dataset? Provide a plot explaining the same. Also, provide a plot of the test accuracy vs. the total number of nodes in the trees.
4. Perform the pruning operation over the tree with the highest test accuracy in question 2 using a valid statistical test for comparison.
5. Print the final decision tree obtained from question 2 following the hierarchical levels of data attributes as nodes of the tree.
6. A brief report explaining the procedure and the results.

Brief description of Methods Used

1. Decision Tree Classifier

- ***General***

  Decision tree learning or **induction of decision trees** is one of the predictive modeling approaches used in statistics, data mining, and machine learning. It uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). Tree models where the target variable can take a discrete set of values are called **classification trees**; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees are among the most popular machine learning algorithms given their intelligibility and simplicity.

- ***Implementation***

  1. ID3 Algorithm (Iterative Dichotomiser 3)
  2. Similar(to ID3) Hill-Climbing algorithm using Gini Impurity as a measure
  3. Data Splitting: 80/20 (Training/ Test) - Defined by the Assignment

- ***Metrics Used***

  Information Gain

  In the context of decision trees, the term is sometimes used synonymously with mutual information, which is the conditional expected value of the Kullback–Leibler divergence of the univariate probability distribution of one variable from the conditional distribution of this variable given the other one.

  Used by the ID3, C4.5, and C5.0 tree-generation algorithms. Information gain is based on the concept of entropy and information content from information theory.

  Entropy is defined as below

$$\mathrm{H}(T) = \mathrm{I}_E(p_1, p_2, \ldots, p_J) = -\sum_{i=1}^{J} p_i \log_2 p_i$$

where, $p_1, p_2, p_3 \ldots$ are fractions that add up to 1 and represent the percentage of each class present in the node that results from a split in the tree.

$$\overbrace{E_A(\mathrm{IG}(T,a))}^{\text{expected information gain}} = \overbrace{I(T;A)}^{\text{mutual information between } T \text{ and } A} = \overbrace{\mathrm{H}(T)}^{\text{entropy (parent)}} - \overbrace{\mathrm{H}(T \mid A)}^{\text{weighted sum of entropies (children)}}$$

$$= -\sum_{i=1}^{J} p_i \log_2 p_i - \sum_{a} p(a) \sum_{i=1}^{J} -\Pr(i \mid a) \log_2 \Pr(i \mid a)$$

That is, the expected information gain is the mutual information, meaning that on average, the reduction in the entropy of *T* is the mutual information.

Used by the CART (classification and regression tree) algorithm for classification trees, Gini impurity (named after Italian mathematician Corrado Gini) is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset. The Gini impurity can be computed by summing the probability p_i of an item with label *i* being chosen times the probability of a mistake in categorizing that item. It reaches its minimum (zero) when all cases in the node fall into a single target category.

2. **<u>Accuracy Measurement</u>**

   The accuracy of a particular decision classifier is measured by parsing (while considering the branch conditions) through the child nodes of the tree until a leaf node is reached for each case (row) of the test set. The label of the leaf node is now compared to the given target value for the corresponding data. Then the accuracy of the classifier is taken as - No of matched rows / Total number of rows.

3. **Pruning**

- ***General***

  Pruning is a data compression technique in machine learning and search algorithms that reduces the size of decision trees by removing sections of the tree that are non-critical and redundant to classify instances. Pruning reduces the complexity of the final classifier, and hence improves predictive accuracy by the reduction of overfitting.

  One of the questions that arise in a decision tree algorithm is the optimal size of the final tree. A tree that is too large risks overfitting the training data and poorly generalizing to new samples. A small tree might not capture important structural information about the sample space. However, it is hard to tell when a tree algorithm should stop because it is impossible to tell if the addition of a single extra node will dramatically decrease error. This problem is known as the horizon effect. A common strategy is to grow the tree until each node contains a small number of instances then use pruning to remove nodes that do not provide additional information

- ***Types***
  1. ***Pre-pruning*** procedures prevent a complete induction of the training set by replacing a stop() criterion in the induction algorithm (e.g. max. Tree depth or information gain (Attr) > threshold Gain). Pre-pruning methods are considered to be more efficient because they do not induce an entire set, but rather trees remain small from the start. Pre Pruning methods share a common problem, the horizon effect. This is to be understood as the undesired premature termination of the induction by the stop() criterion.
  2. ***Post-pruning*** (or just pruning) is the most common way of simplifying trees. Here, nodes and subtrees are replaced with leaves to reduce complexity. Pruning can not only significantly reduce the size but also improve the classification accuracy of unseen objects. It may be the case that the accuracy of the

assignment on the train set deteriorates, but the accuracy of the classification properties of the tree increases overall.

- ***Method Applied: Bottom-Up Pruning with Chi-Square as a Statistical Test***

  This approach to pruning is to apply a statistical test to the data to determine whether a split on some feature X_k is statistically significant, in terms of the effect of the split on the distribution of classes in the partition on data induced by the split. Here the null hypothesis is considered, that the data is independently distributed according to a distribution on data consistent with that at the current node. If this null hypothesis cannot be rejected with high probability, then the split is not adopted and ID3 is terminated at this node. It is based only on the distribution of classes induced by the single decision of splitting at the node and not by the decisions made as a result of growing a full subtree below this node as in the case of post pruning. So here the null hypothesis is stated as: feature X_k is unrelated to the classification of data given features already branched on before this node. This is the hypothesis that we form to determine whether or not to reject the split. The split is only accepted if this null hypothesis can be rejected with high probability. We can perform chi-squared test as:

$$x^2 = \sum \frac{(Observed\ value - expected\ value)^2}{(Expected\ value)}$$

  The Pearson's chi-square test shall be used here as an independence test.

Procedure

Packages Used

- *collections.deque*: Deque data structure for level order traversal of DT
- *matplotlib.pyplot*: plot comparisons in Data Analysis
- *math*: calculate logs in entropy calculations
- *Graphviz*: create dot file for graphical representation

Data Classes:

I. Attributes (Information about the attributes of the dataset)
   A. States:
      1. *name(str)*: Name of Attribute (e.g. Thal, Age, etc.)
      2. *id(int)*: Uniquely identifies an Attribute
      3. *type(bool)*: Classifies attribute type as discrete or continuous
      4. *ntype(int)*: Number of different values an attribute holds
      5. *values*: Distinct values the attribute holds

II. Nodes (Building nodes for Decision Tree)
   A. States
      1. *id(int)*: Uniquely identifies the nodes
      2. *depth(int)* : depth of node (root at depth 0)
      3. *attribute(Attribute)*: attribute used to classify data of node into children (None for leaf node)
      4. *pnode(Node)*: parent node (None for root)
      5. *childnodes(dictionary)*: {branch, childnode}
      6. *data(list)*: entire data represented by this node
      7. *bestsplitc(float)*: None for discrete, best splitting point for continuous
      8. *gini(float)*: gini impurity of data at the node
      9. *ginigain(float)*: gain in gini after further classification of data in the node

10. *entropy(float)*: entropy impurity of data at the node
11. *infogain(float)*: Information gain after further classification of data in the node.
12. label(bool): 1 or 2 for leaf nodes, None for others

B. Methods:

1. *get_most_frequent_label()* : returns most frequent label in the data represented by the node
2. *count_pos_samples()*: returns no. of positive records in the data represented by the node.
3. *gini_calculator()*: returns gini value for a node
4. *entropy_calculator()*: returns entropy value for a node
5. *bestSplitGini()*: returns best splitting attribute, maximum ginigain, split point (None if the attribute is discrete)
6. *bestSplitEntropy()*: returns best splitting attribute, maximum infogain, split point (None if the attribute is discrete)

III. Decision Tree (represents a decision tree)

A. States:

1. *maxdepth(int, Optional)*: maximum depth of tree
2. *count(int)*: total number of nodes
3. *depth(int)*: Depth of the tree
4. *root(Node)*: root node of a tree
5. *start_id(int)*: utility id to assign unique ids to individual nodes

B. Methods:

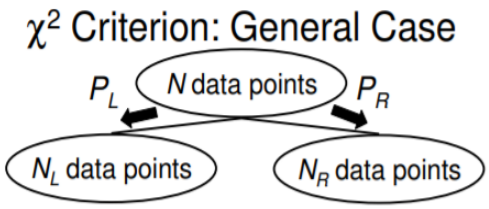
1. *generateDT(attribute_list, training data, method)*:
   (method should be either "GINI" or, "ENTROPY")
   Generates Decision Tree recursively given the method and the root node to start with.
2. *post_pruning()*: Prunes decision tree iteratively removing unnecessary nodes using the statistical test of Pearson Chi Square
3. *evaluate(test_instance)*: Utility function to evaluate a single instance, returns either true or, false
4. *print_Tree()*: utility function to print the decision tree in a hierarchical manner

5. *createIMG()*: creates a dot file for a given decision tree to later create a graphical representation of the tree in png or svg format

Special Methods and their Description

- *generateDT(attribute_list, training data, method)*:
  Recursive function to generate Decision Tree, given data, attribute list, method
  Base conditions:
  1. *Pure Node Reached*: In case of pure node simply return
  2. *All attributes assigned*: Most frequent label of the data in the node is assigned and returned
  3. *Maximum permissible depth reached*: most frequent label assigned and returned

  Individual Iteration:
  1. If some Base Condition satisfies return
  2. Best Splitting attribute is found for the current node based on the method provided and removed from the attribute list.
  3. Branches and childnodes created for each value of attribute. For each branch:
     a. Count of nodes increased
     b. Data is partitioned based on branch value
     c. If partition is empty, most frequent label of the parent is assigned to the child node
     d. Else, generateDT() is recursively called for child node
- *postPruning()*
  Prunes decision tree iteratively removing unnecessary nodes using the statistical test of Pearson Chi Square. Description of algorithm:



$$K = \sum_{\substack{\text{all classes } i \\ \text{children } j}} \frac{(N_{ij} - N'_{ij})^2}{N'_{ij}}$$

where,

N_{ij}= Number of data points from class i in child j

N'_{ij}= Number of data points from class i in child j assuming a random selection

$N'_{ij} = N_i \times P_j$, where $P_j = \frac{Size\ of\ child\ data}{Size\ of\ parent\ data}$

NOTE:

- Small (Chi-square) values indicate low statistical significance Remove the splits that are lower than a threshold K < t.
- Lower t, bigger trees (more overfitting).
- Larger t, smaller trees (less overfitting, but worse classification error).
- Threshold Values for this assignment is taken to be at **0.05 significance level.**

- *random_splitting(data)*:
  Compares the accuracy of 10 gini-based DTs and entropy-based DTs generated from randomly splitted data and prints the accuracy. It also computes the average of these accuracies and reports it. To compute the randomly generated datasets, it uses a utility function create_training_test_data() which simply splits the dataset into 80/20 splits randomly.

- *generate_attribute_list(data)*
  Generates Attribute objects from the dataset provided.
- *depth_node_analysis(data)*
  Uses the package matplotlib to plot accuracy vs. max depth and accuracy vs. number of nodes for both Gini based DTs and Information Gain based DTs.

Guidelines for code execution

I. **Setup the environment (virtualenv must be installed before make)**
To setup the environment, add main.py, requirements.txt and the Makefile(or the MakeinUnix) in the same directory and execute make in the terminal. For **Windows:**

```
> make
> make run
> make clean
```

For Unix-based OS,

```
$ make -f MakeinUnix
$ make -f MakeinUnix run
$ make -f MakeinUnix clean
```

II. **Execute the code**
[*Note: the following snippets should be used in the main function to properly utilize the code***]**

1. Creating a Decision Tree

```
D = DecisionTree() # creates a tree without restrictions
D1 = DecisionTree(maxdepth = 3) # creates a tree with max depth = 3
```

2. Training a Decision Tree

```
best_split = random_splitting(data) # generate data with maximum accuracy
# best_split <- (training data, test data, training set permutation)
attrib_list = generate_attribute_list(best_split[0]) #generate attributes
D.generateDT(attrib_list , best_split[0],  type)
# type = "GINI" or "ENTROPY" to generate a tree with the required measure.
# Default value of type is "GINI"
```

3. Testing a Decision Tree

```
acctest = D.accuracy_test(best_split[1]) # stores accuracy for testset
```

```
acctrain = D.accuracy_test(best_split[0])# stores accuracy for trainingset
print(acctest , acctrain , D.depth, D.count)
# print the accuracies with depth and and count of node of tree
```

4. Printing a Decision Tree

```
D.print_tree()
```

5. Creating a PNG or SVG format image for the tree

```
D.createIMG(name)
# name(str): prefix of filename to store the image in DOT format
```

6. Pruning a Decision Tree

```
D.post_pruning()
```

Results

Here, we list the results obtained in individual parts of the problem statement:

I: Comparison of Impurities: Gini vs Entropy for 10 random samples with 80/20 split:

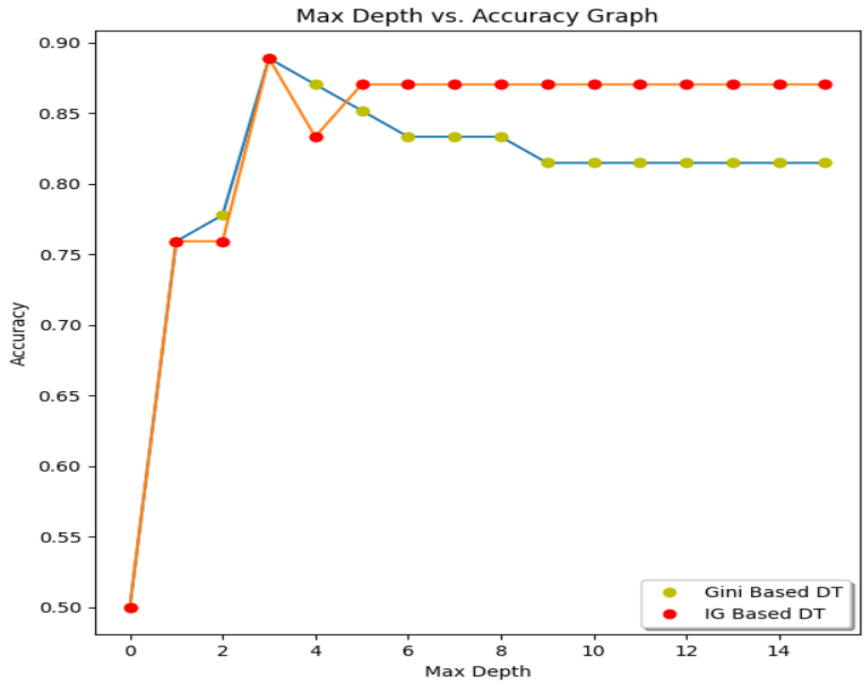
```
Gini based DT accuracy     vs.   Information Gain based DT accuracy:
=================================================================
===========
0.8518518518518519,          0.8333333333333334
0.8518518518518519,          0.7222222222222222
0.7407407407407407,          0.7962962962962963
0.7222222222222222,          0.7407407407407407
0.7222222222222222,          0.7777777777777778
0.8148148148148148,          0.8703703703703703
0.8148148148148148,          0.7777777777777778
0.7592592592592593,          0.6851851851851852
0.8148148148148148,          0.8518518518518519
0.7407407407407407,          0.7777777777777778

<Avg. accuracy Gini based DT - 0.7833333333333334>,

<Avg. accuracy Information Gain based DT - 0.7833333333333333>
```

Although the values for the Gini-based accuracy and entropy-based accuracy differ in different random splits, equal average value represents that both of them have evaluation power of the same order. Lower accuracy in the majority of the cases suggests that the decision tree is most likely overfitting the training data in most of the cases.

The range of values for the accuracy for gini is | 0.852 - 0.722 | = 0.13. The corresponding value for entropy is | 0.87 - 0.69 | = 0.18. This variation suggests that choosing the optimal split for training and test data is crucial as it can affect the accuracy a lot. The cases where the accuracy is low is mostly because the

corresponding decision tree is overfitting. To solve this problem, we used the Pearson Chi-Square Test with a 0.05 confidence level as the threshold value for performing post-pruning in the decision tree.

II: Plot for Maximum Depth vs Accuracy

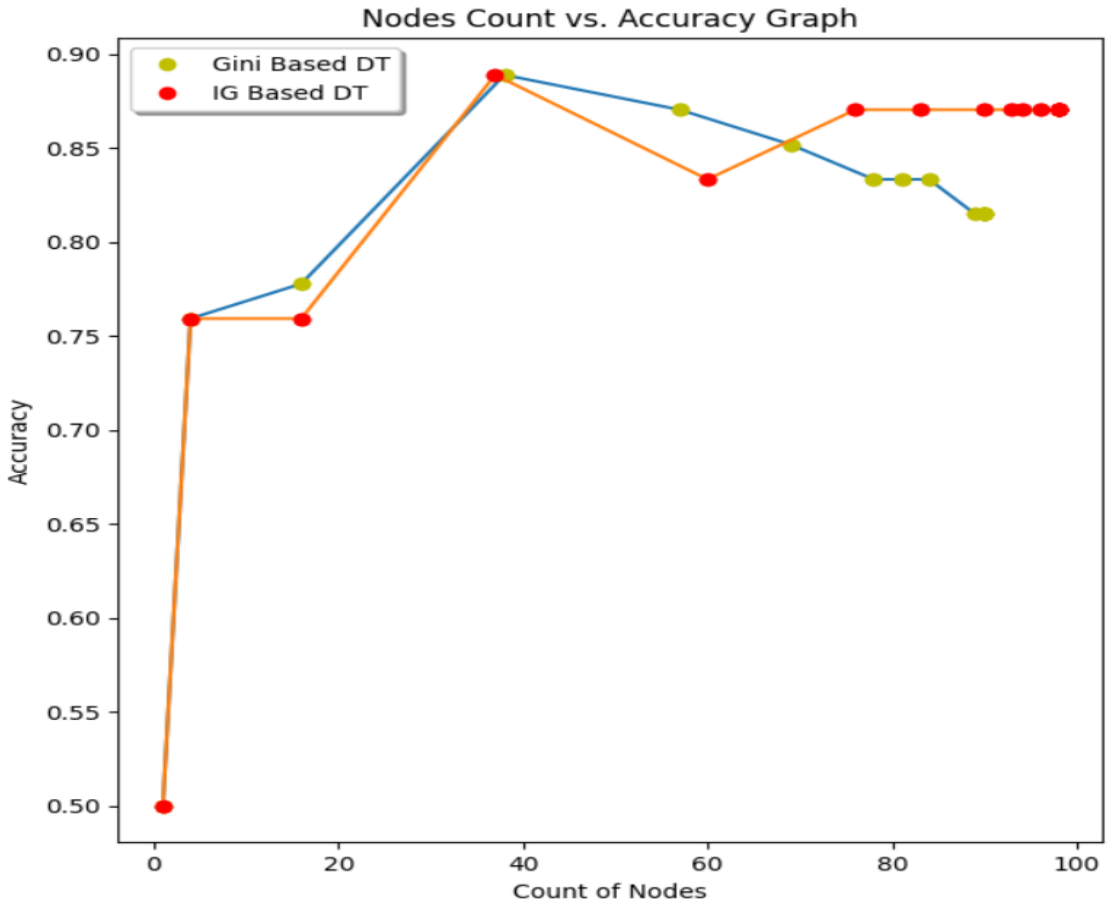


PLOT 1: Max-Depth vs Accuracy (Based on Test Data)

The plot between the Maximum depth of the decision tree and the Accuracy is highly satisfactory and the maximum accuracy obtained at a maximum depth value of 3 can be used with confidence since it is backed by both the gini gain-based plot and the information gain-based plot. Initially the accuracy of the DT greatly increases, however, after a height of 3, the change decreases and becomes almost constant. This happens due to the action of "overfitting" of the decision tree which increases the generalization error for the test set. One must note that the slight increase in the accuracy after height 5 is highly specific to the training data and is not to be assumed as the general property of the decision tree. The following plot of Cardinality of Nodes vs the Accuracy also explains this phenomenon of overfitting in an equivalent sense to the previous plot since the number of nodes increases with increase in depth and subsequently, the error through initially decreases it starts

increasing after a critical value which is observed to be around 40-60. As discussed later the optimal depth here is only specific to the test data, if we consider both training data and test data the actual optimal depth will lie in a slightly higher interval ([5-8]).

Plot for Number of Nodes vs Accuracy

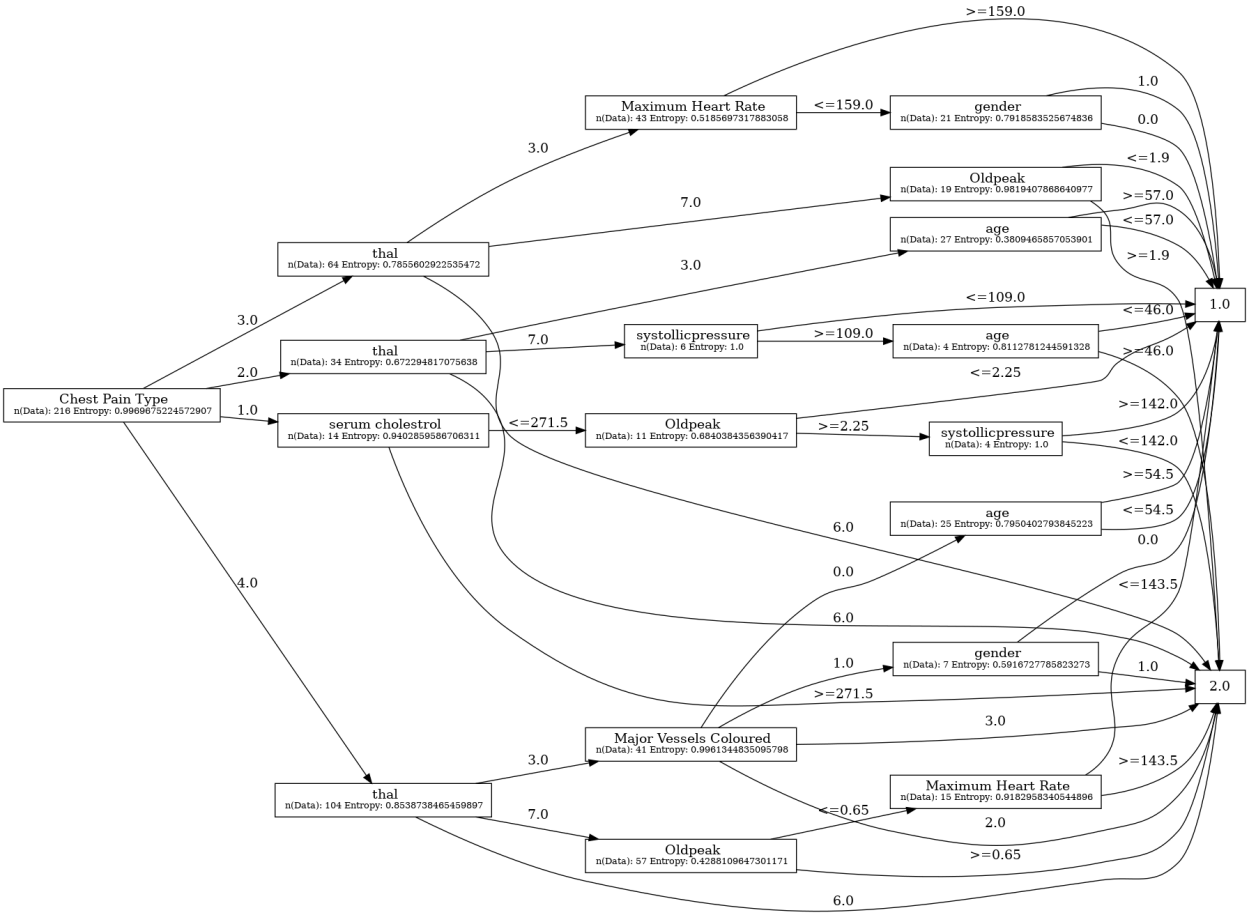


PLOT 2: Number of nodes vs Accuracy(Based on test data)

III: Post Pruning:

Pruning Status	Accuracy	Depth	Number of Nodes

Before Pruning	0.87	8	98
After Pruning	0.89	4	44



Decision Tree After Pruning

(PNG formatted diagrams for the decision tree before and after pruning is submitted alongwith)

Although not illustrated due to limitation of width, one can easily imagine the extent and dense nature of the decision tree before pruning is performed with its 98 nodes and a depth of 8. By removing the insignificant nodes and their contribution to the decision tree, we manage to increase the accuracy of the tree by minimizing the generalization error and at the same time making the tree more representable. The

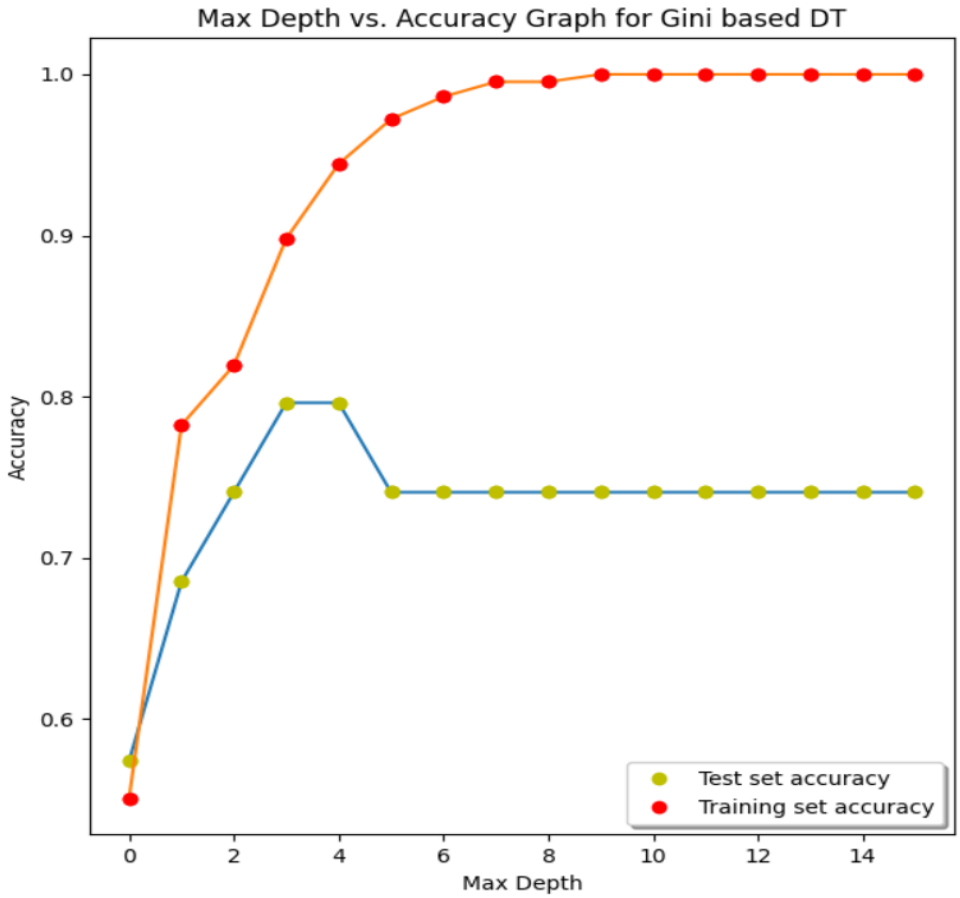
decrease in the complexity of the tree is really worth noting as it reduces both the height and nodes count by about 50% and still manages to outperform the previous test-set accuracy! A point worth noting here is that the statistical test is independent of any test set or validation set as the Chi-square method for post-pruning doesn't prune the tree targeted to improve a specific test set accuracy. This might result in the decrease in the accuracy of the predictions in the test set in a few cases but, that's totally specific to that particular split as the post-pruned DT in general should perform better over a large test set. It's also worth noting that the post-pruning improves the performance of the tree while maintaining a reasonably high training set accuracy. The following table illustrates the discussion above:

Random Split Id	Test Set Accuracy		Training Set Accuracy		Depth		Number of nodes	
	Before	After	Before	After	Before	After	Before	After
1	0.81	0.85	1	0.95	10	8	90	64
2	0.79	0.87	1	0.92	8	6	100	48
3	0.81	0.81	1	0.94	12	5	100	64
4	0.85	0.87	1	0.94	9	6	102	66
5	0.79	0.91	1	0.92	11	7	108	49

TABLE: VALUES AFTER PRUNING OF DECISION TREE

One more thing to note here is that the depth of the DT after post-pruning lies in the range [5-8] which is not what we have seen in the depth vs. accuracy analysis (The optimal depth was 3-4). The reason behind this is that the post-pruning method tries to improve performance irrespective of any data set. That's why the performance of

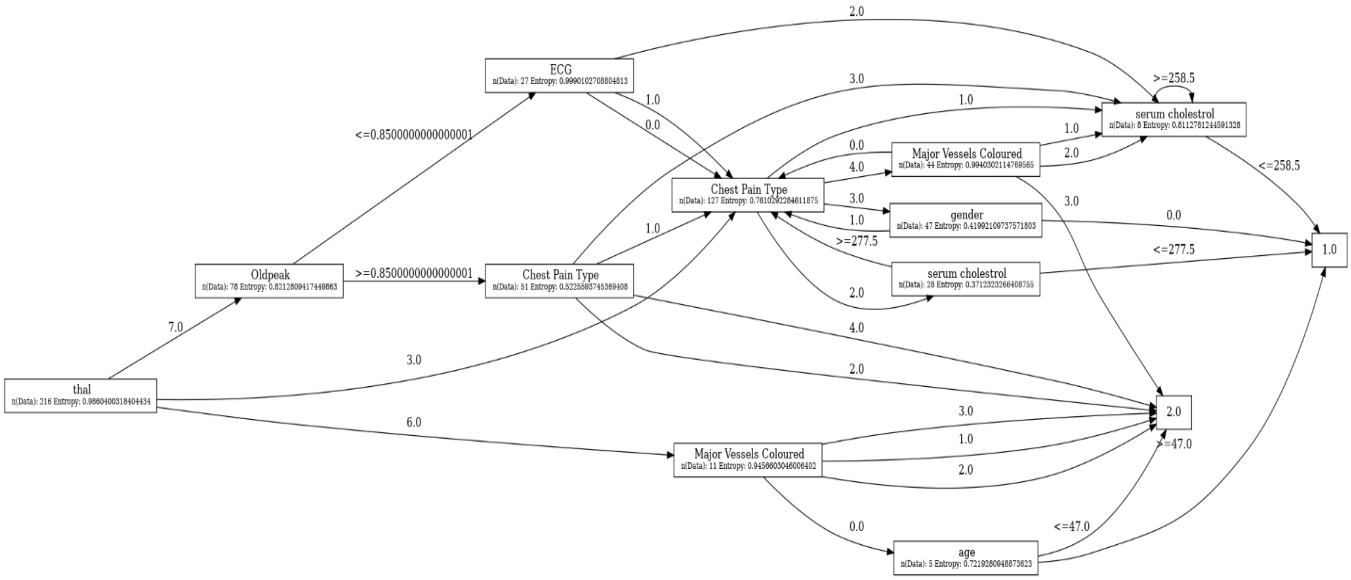
the DT is reasonably good in both training set and test set. The optimal depth of 3-4 is specific to the test set. The corresponding performance of the tree in the training set won't be any better than what we have achieved. This can be depicted by the following plot -



Plot 3: Plot of max depth vs. accuracy for an unpruned DT (For test set and training set)

As it can be seen a depth of 2-4 is only optimal for the test set and hence the post-pruning algorithm chooses a depth limit of [5-8] which increases overall performance for both training data and test data.

IV: Decision Tree with best Max-Depth:



Nodes: 33, Depth: 3, Accuracy: 0.852

Agreeing with Plot 1 and Plot 2, when limiting the maximum depth of the decision tree to 3, we get a huge increase in the accuracy on test data from the average value obtained in the result I. Also, the decrease in the depth of the decision tree directly implies a reduction in the generalization error which results in the increase in the accuracy in unforeseen error. On a note, one should not compare the numeric value of this accuracy with the post pruning accuracy of 0.87 and infer that depth reduction gives a smaller decrease in the error, since the marginal difference is due to evaluation on different data sets randomly allocated.

Hierarchical Printing of pruned tree

The DT is printed in the following format -

<Branch value, <Parent-ID, ID, Data amount, attr, Entropy/Gini, Label>>

This means that the node with node-ID 'ID' is branched from the parent node having ID 'Parent-ID' using the attribute 'attr' with value = Branch value. Entropy/Gini denotes the impurity of data. Label is either '1' or '2' for leaf nodes and 'None' for non-leaf nodes.

Each level of the DT is separated by '====..==='.

<branch-None, <Root, Node-ID-0, data-216, attr-Chest Pain Type, entropy-0.9969675224572907, label-None>>
==========
<branch-1.0, <parent-0, Node-ID-1, data-14, attr-serum cholestrol, entropy-0.9402859586706311, label-None>>
<branch-2.0, <parent-0, Node-ID-8, data-34, attr-thal, entropy-0.672294817075638, label-None>>
<branch-3.0, <parent-0, Node-ID-23, data-64, attr-thal, entropy-0.7855602922535472, label-None>>
<branch-4.0, <parent-0, Node-ID-52, data-104, attr-thal, entropy-0.8538738465459897, label-None>>
==========
<branch-<= 271.5, <parent-1, Node-ID-2, data-11, attr-Oldpeak, entropy-0.6840384356390417, label-None>>
<branch-> 271.5, <parent-1, Node-ID-3, data-3, attr-None, entropy-0, label-2.0>>
<branch-3.0, <parent-8, Node-ID-9, data-27, attr-age, entropy-0.3809465857053901, label-None>>
<branch-6.0, <parent-8, Node-ID-17, data-1, attr-None, entropy-0, label-2.0>>
<branch-7.0, <parent-8, Node-ID-18, data-6, attr-systollicpressure, entropy-1.0, label-None>>
<branch-3.0, <parent-23, Node-ID-24, data-43, attr-Maximum Heart Rate, entropy-0.5185697317883058, label-None>>
<branch-6.0, <parent-23, Node-ID-38, data-2, attr-None, entropy-0, label-2.0>>
<branch-7.0, <parent-23, Node-ID-39, data-19, attr-Oldpeak, entropy-0.9819407868640977, label-None>>
<branch-3.0, <parent-52, Node-ID-53, data-41, attr-Major Vessels Coloured, entropy-0.9961344835095798, label-None>>
<branch-6.0, <parent-52, Node-ID-80, data-6, attr-None, entropy-0.9182958340544896, label-2.0>>
<branch-7.0, <parent-52, Node-ID-87, data-57, attr-Oldpeak, entropy-0.4288109647301171, label-None>>
==========
<branch-<= 2.25, <parent-2, Node-ID-4, data-7, attr-None, entropy-0, label-1.0>>
<branch-> 2.25, <parent-2, Node-ID-5, data-4, attr-systollicpressure, entropy-1.0, label-None>>

<branch-<= 57.0, <parent-9, Node-ID-10, data-21, attr-None, entropy-0, label-1.0>>
<branch-> 57.0, <parent-9, Node-ID-11, data-6, attr-None, entropy-0.9182958340544896, label-1.0>>
<branch-<= 109.0, <parent-18, Node-ID-19, data-2, attr-None, entropy-0, label-1.0>>
<branch-> 109.0, <parent-18, Node-ID-20, data-4, attr-age, entropy-0.8112781244591328, label-None>>
<branch-<= 159.0, <parent-24, Node-ID-25, data-21, attr-gender, entropy-0.7918583525674836, label-None>>
<branch-> 159.0, <parent-24, Node-ID-26, data-22, attr-None, entropy-0, label-1.0>>
<branch-<= 1.9, <parent-39, Node-ID-40, data-15, attr-None, entropy-0.8366407419411673, label-1.0>>
<branch-> 1.9, <parent-39, Node-ID-41, data-4, attr-None, entropy-0, label-2.0>>
<branch-0.0, <parent-53, Node-ID-54, data-25, attr-age, entropy-0.7950402793845223, label-None>>
<branch-1.0, <parent-53, Node-ID-70, data-7, attr-gender, entropy-0.5916727785823273, label-None>>
<branch-2.0, <parent-53, Node-ID-73, data-6, attr-None, entropy-0.9182958340544896, label-2.0>>
<branch-3.0, <parent-53, Node-ID-79, data-3, attr-None, entropy-0, label-2.0>>
<branch-<= 0.65, <parent-87, Node-ID-88, data-15, attr-Maximum Heart Rate, entropy-0.9182958340544896, label-None>>
<branch-> 0.65, <parent-87, Node-ID-89, data-42, attr-None, entropy-0, label-2.0>>
==========
<branch-<= 142.0, <parent-5, Node-ID-6, data-2, attr-None, entropy-0, label-2.0>>
<branch-> 142.0, <parent-5, Node-ID-7, data-2, attr-None, entropy-0, label-1.0>>
<branch-<= 46.0, <parent-20, Node-ID-21, data-1, attr-None, entropy-0, label-1.0>>
<branch-> 46.0, <parent-20, Node-ID-22, data-3, attr-None, entropy-0, label-2.0>>
<branch-0.0, <parent-25, Node-ID-27, data-10, attr-None, entropy-0, label-1.0>>
<branch-1.0, <parent-25, Node-ID-28, data-11, attr-None, entropy-0.9940302114769565, label-1.0>>
<branch-<= 54.5, <parent-54, Node-ID-55, data-12, attr-None, entropy-0, label-1.0>>
<branch-> 54.5, <parent-54, Node-ID-56, data-13, attr-None, entropy-0.9957274520849256, label-1.0>>
<branch-0.0, <parent-70, Node-ID-71, data-1, attr-None, entropy-0, label-1.0>>
<branch-1.0, <parent-70, Node-ID-72, data-6, attr-None, entropy-0, label-2.0>>
<branch-<= 143.5, <parent-88, Node-ID-90, data-3, attr-None, entropy-0, label-1.0>>

<branch-> 143.5, <parent-88, Node-ID-91, data-12, attr-None, entropy-0.6500224216483541, label-2.0>>

==========

Discussion

Handling Attributes with Continuous Values:

Although, above mentioned algorithms can swiftly handle the discrete attributes, attributes with continuous values require some extra evaluation since the tree cannot have infinite branches from a node. We solve this problem with careful partition of data:

In case we encounter a node in the decision tree where the best attribute turns out to be continuous, we partition the data further by finding the best split point for the values in the continuous distribution, i.e., the point which when chosen to be the partitioning point gives the largest information gain (or ginigain, as the case may be). This is done by the following algorithm:

- Sort the continuous values
- For every pair of successive distinct values, take the point(say, j) midway to these as the partitioning point.Thus, we get two branches: <= j or > j. Let S denote the total amount of data , $S_{<=j}$ denote data having the attribute value <= j and $S_{>j}$ denote data having attribute value > j.

  So, the information gain or, gini gain for this split is given by -

  $$Gain = Imp(S) - \frac{|S_{<=j}|}{|S|} Imp(S_{<=j}) - \frac{|S_{>j}|}{|S|} Imp(S_{>j})$$

  Imp(X) denote the entropy or, gini impurity of data X
- Find the gain for all such points and take the point giving maximum gain to be the split point and treat the continuous attribute with 2 branches, <= j and > j.

Comparison between Pruning with Chi-square and Limiting the Maximum-Depth of the DT

Random Split	Accuracy			Depth		
	Normal	Post-pruning	Fixed and reduced Max-depth	Normal	Post-pruning	Fixed and reduced Max-depth
1	0.74	0.87	0.78	11	5	3
2	0.76	0.80	0.80	8	7	3
3	0.77	0.72	0.76	8	6	4
4	0.83	0.81	0.81	8	6	4
Average	0.775	0.80	0.79			

TABLE: COMPARISON OF PRE AND POST PRUNING FOR TEST SET

Random Split	Accuracy			Depth		
	Normal	Post-pruning	Fixed and reduced Max-depth	Normal	Post-pruning	Fixed and reduced Max-depth
1	1.0	0.94	0.90	11	5	3
2	1.0	0.95	0.88	8	7	3
3	1.0	0.95	0.90	8	6	4

4	1.0	0.96	0.92	8	6	4
Average	1.0	0.95	0.90			

TABLE: COMPARISON OF PRE AND POST PRUNING FOR TRAINING SET

The above data clearly shows that the post-pruned DT outperforms both the unpruned DT and height limited DT when both the training data and test data are considered. Though in some cases the height limited DT may perform slightly better in the test data but, when considered the training data as well the post-pruned tree clearly outperforms. This also shows that the optimal depth limit lies in the range [5-8] when both the training data and the test data is considered while the optimal depth limit for the test data lies in the range [3-4] as shown in PLOTS 1 and 2.

Brief Subjective analysis of Data

Finally, we make a subjective analysis of the data recovered from the decision trees. The database contains 13 attributes

Attribute Information:

-- 1. age
-- 2. sex
-- 3. chest pain type (4 values)
-- 4. resting blood pressure
-- 5. serum cholesterol in mg/dl
-- 6. fasting blood sugar > 120 mg/dl
-- 7. resting electrocardiographic results (values 0,1,2)
-- 8. maximum heart rate achieved
-- 9. exercise induced angina
-- 10. oldpeak = ST depression induced by exercise relative to rest
-- 11. the slope of the peak exercise ST segment
-- 12. number of major vessels (0-3) colored by fluoroscopy
-- 13. thal: 3 = normal; 6 = fixed defect; 7 = reversible defect

Attributes types

Real: 1,4,5,8,10,12
Ordered:11,
Binary: 2,6,9
Nominal:7,3,13

Variable to be predicted

Absence (1) or presence (2) of heart disease

From the Hierarchal Printing of the data we can conclude that the most important factors in determining if an unknown person has a heart disease is the state of the attribute thal or, Chest Pain Type in most of the cases which is followed in terms of importance in our deduction by Major Vessels Coloured and oldpeak. During the best-depth estimation from the plots, we found that the maximum depth of the decision tree for maximum accuracy in the test set lies in [3-4] while for overall better performance the optimal height should be in the range [5-8]. The optimal number of nodes in the decision tree should be around 40-70.

References

The following sources were used in making this report:

http://www.csroc.org.tw/journal/JOC29_6/JOC-2906-01.pdf

https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.303.6431&rep=rep1&type=pdf

https://en.wikipedia.org/wiki/Decision_tree_pruning

https://datahub.io/machine-learning/heart-statlog/r/1.html