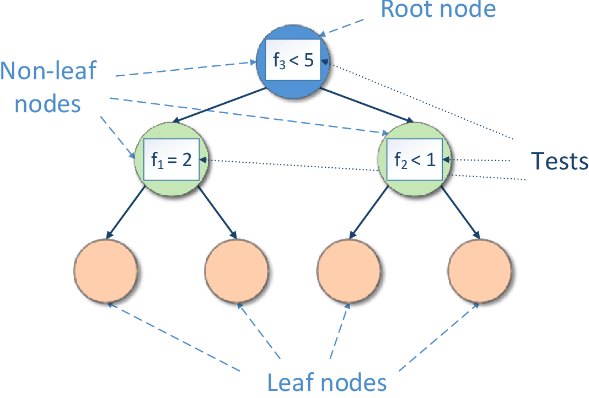
Decision Trees: is a method to apply if-else conditions to classify data.

Decision Trees are a [non-parametric](https://machinelearningmastery.com/parametric-and-nonparametric-machine-learning-algorithms/) supervised learning method used for both classification and regression tasks of machine learning.

Decision trees uses a branching method to illustrate every possible output for a specific input. They are constructed via an algorithmic approach that identifies ways to split a data set based on different conditions and decide about process. A decision tree is a flowchart-like structure that is composed of leaf, internal nodes, and branches. Each internal node represents a test on a feature (e.g., whether a coin flip comes up heads or tails), each leaf node represents a class label (decision taken after computing all features) and branches represent conjunctions of features that lead to those class labels. The paths from root to leaf represent classification rules. The flowchart is as follows



**Problem of Decision trees:**

Decision trees are prone to **overfitting**, especially when a tree is particularly deep. This causes A small change in the data can cause a large change in the structure of the decision tree causing instability is overfitting in decision tree. The supported criterion of decision tree is “gini” and “entropy” that are criterion for information gain. DecisionTreeClassifier(criterion=”gini” or “entropy”).

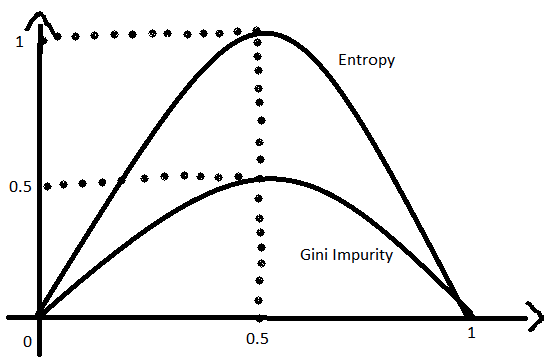
**Entropy:** Entropy helps us to build an appropriate decision tree for selecting the best splitter. Entropy can be defined as a measure of the purity of the sub split. Entropy always lies between 0 to 1. The entropy of any split can be calculated by this formula. The algorithm calculates the entropy of each feature after every split and as the splitting continues on, it selects the best feature and starts splitting according to it.

is the probability of class ci in a node.

**Gini Impurity:** Gini impurity is also somewhat similar to the working of entropy in the Decision Tree. In the Decision Tree algorithm, both are used for building the tree by splitting as per the appropriate features but there is quite a difference in the computation of both the methods. Gini Impurity of features after splitting can be calculated by using this formula.

The algorithm calculates the entropy of each feature after every split and as the splitting continues on, it selects the best feature and starts splitting according to it.

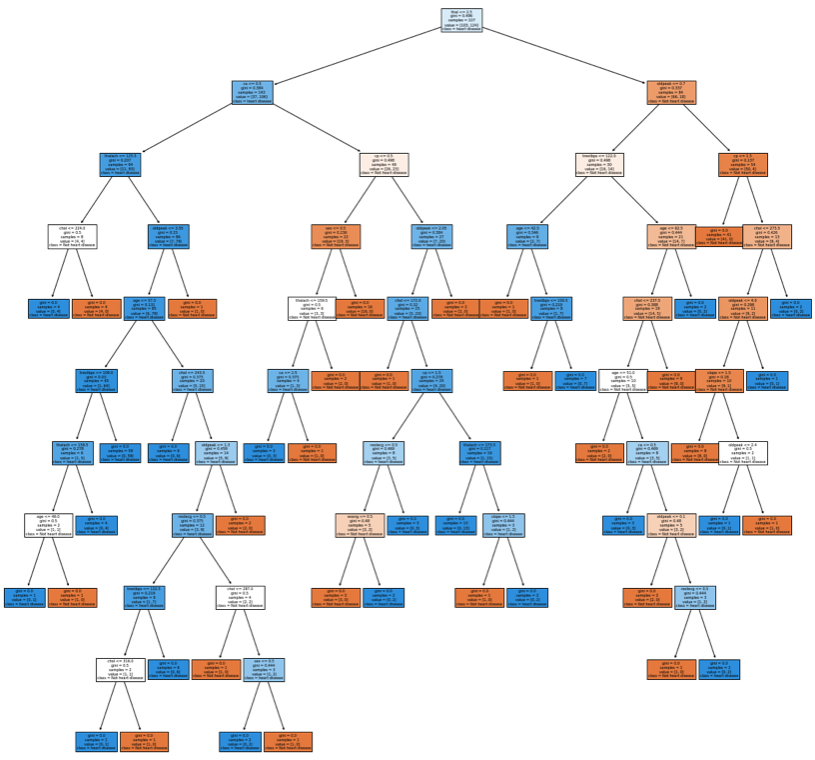
**Entropy Vesus Gini Impurity:**  
Now we have learned about Gini Impurity and Entropy and how it works. Also, we have seen how we can calculate Gini Impurity/Entropy for a split/feature. But the major question arises here is why do we need to have both the methods for computation, and which is better?



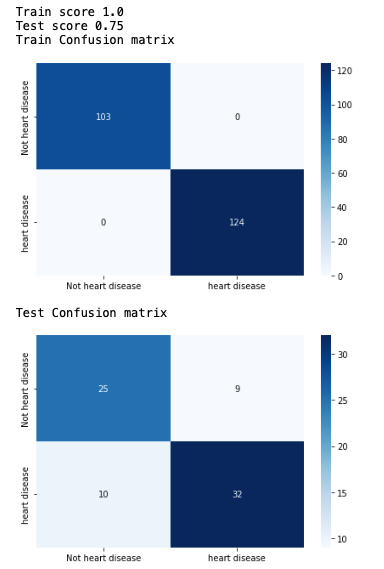
The internal working of both Gini and Entropy methods is very similar, and both are used for computing the feature/split after every new splitting. But if we compare both the methods then Gini Impurity is more efficient than entropy in terms of computing power. As you can see in the graph for entropy, it first increases up to 1 and then starts decreasing, but in the case of Gini impurity it only goes up to 0.5 and then it starts decreasing, hence it requires less computational power. The range of Entropy lies in between 0 to 1 and the range of Gini Impurity lies in between 0 to 0.5. Hence, we can conclude that **Gini Impurity** is **better** as compared to entropy for **selecting the best features**. That is the reason the default criterion of DecisionTreeClassifier is gini.

**Example:** The example of overfitting in decision tree is decision tree in heart disease. The code is attached to this paper.

# Apply Decision Tree with Default Parameter and Overfitting



# Confusion Metrics for Train and Test with Decision tree Default Parameters



As you can see above, the size of tree is grown and also the training accuracy is 99% and the test accuracy is 75% which is much lower than the training accuracy. Therefore, it is very clear that our decision tree model is overfitting.

**Avoid Overfitting:**

Two approaches to avoiding overfitting are distinguished: pre-pruning and post-pruning.

**1)Pre-pruning:** Early stopping or pre-pruning which stops the growth of a decision tree at an early stage. It generates a tree with fewer branches. For that we can limit the growth of trees by setting constrains. We can limit parameters like max\_depth , min\_samples etc.

An effective way to do is that we can **grid search** those parameters and choose the optimum values that gives better performance on test data.

As of now we will control these parameters

* **max\_depth:** maximum depth of decision tree. If None (default), then nodes are expanded until all leaves are pure (i.e. fitting the model with 100% accuracy). Decreasing this value prevents overfitting.
* **min\_sample\_split:** The minimum number of samples required to split an internal node. The default is 1. Increasing this value prevents overfitting.
* **min\_samples\_leaf:** The minimum number of samples required to be at a leaf node. The default is 2. If we set its value to 5, no further splits are permitted for nodes with five samples or fewer.

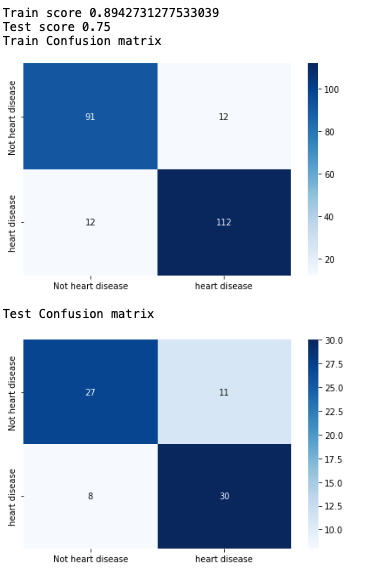
**Among these hyperparameters, max\_depth is the most important one. We can easily apply pre-pruning to decision trees by tuning the max\_depth hyperparameter alone.**

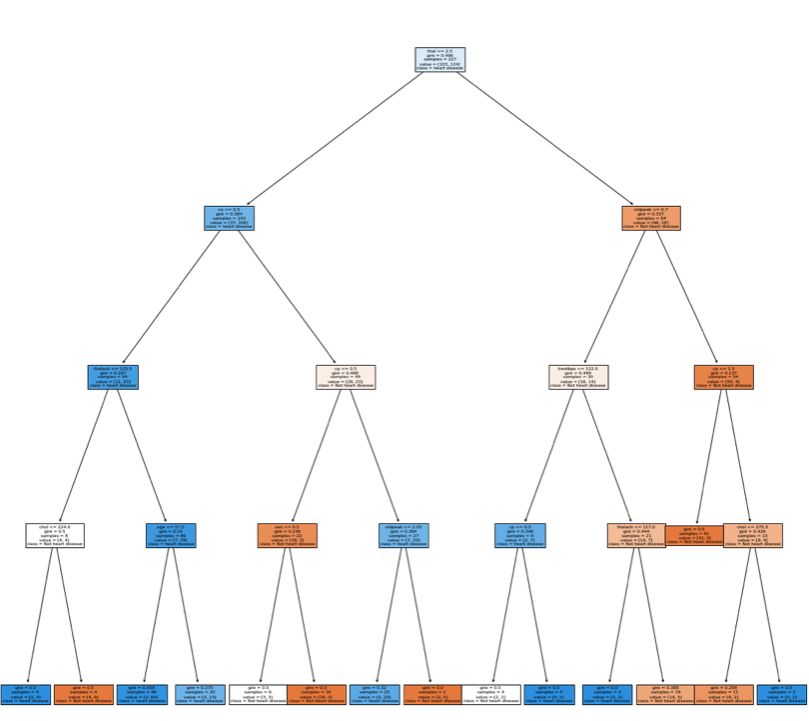
At each stage of splitting the tree, we check the cross-validation error. If the error does not decrease significantly enough, then we stop. Early stopping may underfit by stopping too early.

At each stage of splitting the tree, we check the cross-validation error. If the error does not decrease significantly enough, then we stop. Early stopping may underfit by stopping too early.

**Pre-prunning that is optimization after the Grid Search Cross validation:**

After Gid Search CV (gridsearchcv) and choosing the best parameters for max\_depth, min\_sample\_split and min\_samples\_leaf, we see the following performance metrics and decision tree.



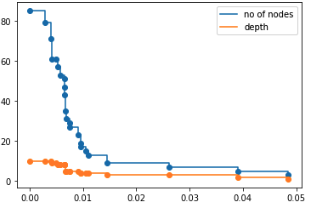


**Result after Preprunning:** As you can see in the above output, the size of tree is reduced compared to the size before training and training accuracy is 89% and the test accuracy is 75%. The training score is lower than the previous training accuracy. This shows the improvement when tree is pruned, considering overfitting. But still there is still scope of improvement. So, we do post pruning.

**2) Post pruning techniques:** post-pruning that is generating a tree in full and then removing parts of it. This is most effective way to avoid overfitting is to use post pruning methods like cost complexity pruning. This helps to improve test accuracy and get a better model.

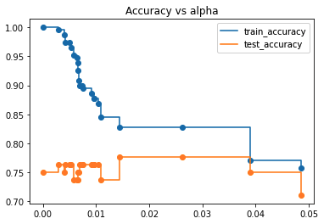
Cost complexity pruning is all about finding the right parameter for alpha. Greater values of alpha increase the number of nodes pruned.

We will get the alpha values for this tree and will check the accuracy with the pruned trees.



Output above shows as alpha increases no of nodes and depth decreases.

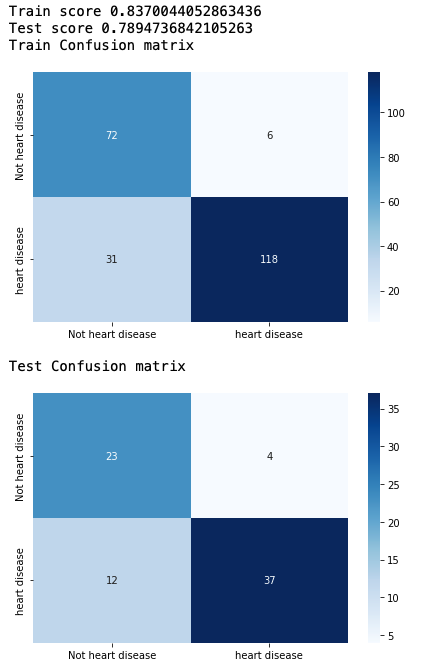
**What is the Best Alpha in Post Prunning?**

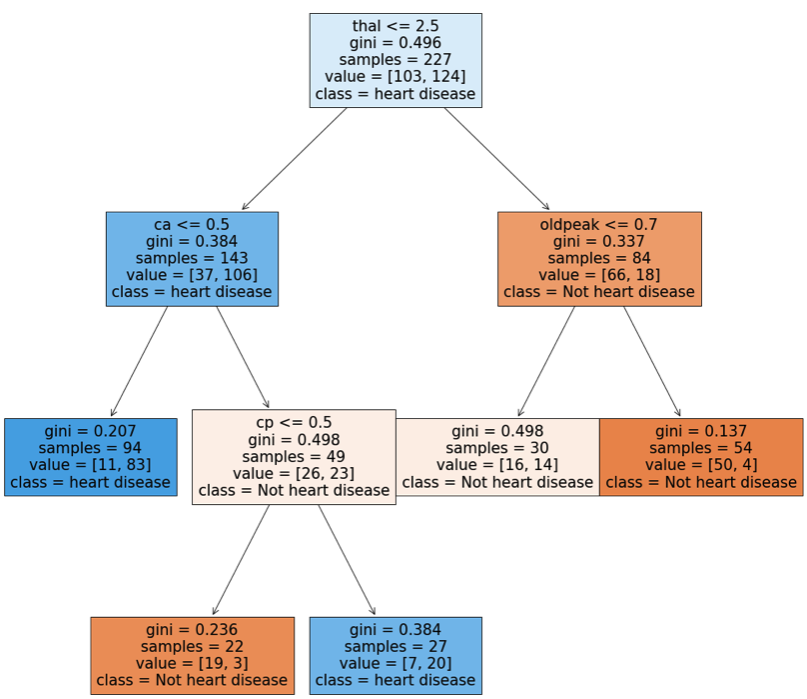


By looking at the plot, we can decide the optimal alpha value that mitigates overfitting in our decision tree model is alpha = 0.020

**Decision Tree and Performance Metrics After Setting the Best Alpha of Post Prunning:**

This is after finding the best alpha, we have the following results:





**Result After Post Pruning:**

We can see that the train score is getting lower and the accuracy of test score improves. This shows that there is not thet much gap between train and test scores. Also, size of decision tree significantly got reduced. Also postpruning is much efficient than prepruning. In this kernel we have used accuracy as metricas data was balanced.