**Decision Tree:**

This is a tree-based approach and A Decision Tree is a Supervised learning technique that can be used for classification and Regression problems, however, it is mostly preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome



**Decision Tree Classifier**

The decision tree splits the data and forms a tree-like structure consisting of

1. Root nodes.
2. Child nodes.
3. Leaf nodes.

**Root nodes**: The top-level nodes in a tree structure from which all other nodes descend.

**Child nodes**: Nodes that are directly connected and descend from a parent node.

**Leaf nodes**: Nodes that have no children and are located at the end of a tree branch.

When constructing the tree, the model must choose one feature among multiple features as the parent node. This selection is based on the concept of **information gain**, where the feature with the highest information gain becomes the parent node, while the remaining features become the child nodes.

Information gain is calculated based on below two concepts. ( i.e. Entropy or Gini Impurity)

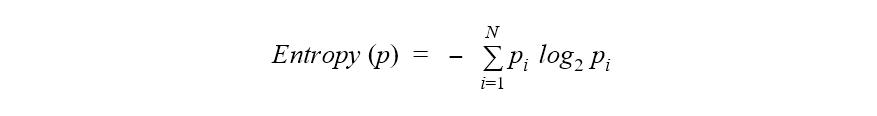
What are impurity measures (e.g., Gini index, entropy)

Impurity measures, such as the Gini index and entropy, are used in decision trees and other classification algorithms to quantify the "impurity" or "purity" of a dataset. These measures help determine how well a particular split separates the data into different classes or categories. The impurity measures are commonly used to select the best feature and threshold for splitting the data in a decision tree.

1. **Entropy:**

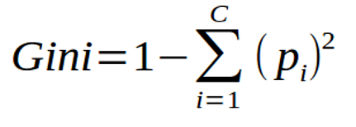
Entropy is an additional impurity measure used in decision trees for classification tasks. It originates from information theory and gauges the uncertainty or disorder within a dataset. The entropy formula calculates the sum of class probabilities multiplied by their logarithm base 2 for a given node. The entropy value varies between 0 and log2(k), where k represents the number of classes in the dataset.

Like the Gini index, when deciding on a split for a node, the preferred split is the one that yields the lowest entropy (highest information gain). This choice ensures the best separation of classes, leading to more pure subsets after the split. By utilizing entropy, decision trees can effectively partition data and make accurate classifications based on the purity of the resulting subsets.



1. **Gini Impurity**

The Gini index is a widely used impurity measure in decision trees for classification tasks. It evaluates the disorder or purity of a dataset at a particular node based on class probabilities. The index ranges from 0 to 1, where 0 represents a perfectly pure node with all items belonging to the same class, and 1 indicates a completely impure node with items evenly distributed across all classes. In decision trees, the preferred split for a node is the one that results in the lowest Gini index (highest reduction in impurity) as it leads to more homogeneous subsets after the split, enhancing the tree's ability to classify data accurately.



The formula for the information Gain for Entropy

H(S) = Entropy or Gini of the root Node.  
Sv = total values in a particular node.  
S = Total Number of Values of Output Feature  
H(Sv) = Entropy or Gini of the Child Node

The determination of the "root node," as well as the subsequent "child nodes" and "leaf nodes," is decided by Information Gain, and Information Gain is either calculated by subtracting Entropy or Gini Impurity. These measures help identify the level of impurity or randomness associated with a specific feature.

There are three types of Techniques used in decision trees the same as below.

1. ID3 (Entropy)
2. CART (Gini Impurity, Gini Coefficient)

**ID3 (Iterative Dicotomizer):**

Here “Information Gain” is calculated based on Entropy, basically, if we are using the ID3 algorithm it means we are calculating nothing but “Information Gain” using Entropy. In the context of Decision Trees, entropy is a measure of disorder or impurity in a node.

**CART (Classification and Regression Tree):**

In the context of the CART algorithm, “Information Gain” is calculated based on Gini Impurity, the criterion used to evaluate and split nodes. Information gain is calculated based on “Gini Impurity or Gini Coefficient”. Gini Impurity is a measure of impurity or randomness in a specific feature or attribute.

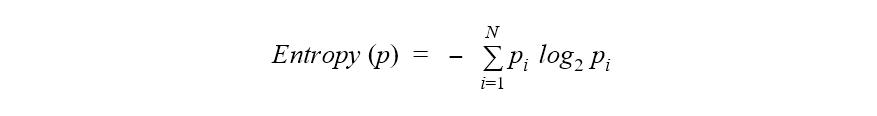
**Example: Please calculate the information gained for the below dataset using the below dataset**

|  |  |  |
| --- | --- | --- |
| Feature 01 | Feature 02 | Yes/No |
| C1 | B1 | Y |
| C1 | B1 | Y |
| C1 | B1 | Y |
| C1 | B1 | Y |
| C1 | B1 | Y |
| C1 | B2 | Y |
| C2 | B2 | Y |
| C2 | B2 | Y |
| C2 | B2 | Y |
| C1 | B1 | N |
| C1 | B2 | N |
| C2 | B2 | N |
| C2 | B2 | N |
| C2 | B2 | N |

|  |  |
| --- | --- |
| **Split Feature One** | **Split Feature Two** |
|  |  |

**Steps: -**

1. Calculate the entropy of all features.
2. Calculate the Gini Impurity of all features.
3. Utilize the obtained entropy values in the Information Gain Formula to identify the feature with the highest information gain.
4. Utilize the resulting Gini Impurity values in the Information Gain Formula to determine the feature with the highest information gain.
5. Compare the information gain values obtained from Gini and Entropy and consider the equation with the highest value to determine the root node.

**Calculate the entropy of all features(here we will calculate only for two since there are only two features).  
Entropy Formula**

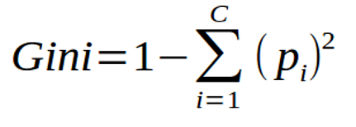
**Calculate the Entropy of Feature One:1**

|  |  |  |  |
| --- | --- | --- | --- |
| ..   |  | | --- | | **Entropy of Root Node(9y , 5n)** | |  |   .. | |
| ..   |  | | --- | | **Entropy of Node (6y , 2n)** | |  |   .. | ..   |  | | --- | | **Entropy of a Node(3y , 3n)** | |  |   .. |

**Calculate the Entropy of Feature One:2**

|  |  |  |  |
| --- | --- | --- | --- |
| ..   |  | | --- | | **Entropy of Root Node(9y , 5n)** | |  |   .. | |
| |  | | --- | | **Entropy of a Node (5y , 1n)** | |  |   ….. | |  | | --- | | **Entropy of a Node(4y , 4n)** | |  |   …. |

Calculate the Gini Impurity



**Calculate the Gini Impurity of Feature One:1**

|  |  |  |  |
| --- | --- | --- | --- |
| ..   |  | | --- | | **Entropy of Root Node(9y , 5n)** | |  |   .. | |
| ..   |  | | --- | | **Entropy of (6y , 2n)** | |  |   .. | ..   |  | | --- | | **Entropy of Node(3y , 3n)** | |  |   .. |

**Calculate the Gini Impurity of Feature Two**

|  |  |  |  |
| --- | --- | --- | --- |
| ..   |  | | --- | | **Entropy of Root Node(9y , 5n)** | |  |   .. | |
| ..   |  | | --- | | **Entropy of (5y , 1n)** | |  |   .. | ..   |  | | --- | | **Entropy of Root Node(4y , 4n)** | |  |   .. |

H(S) = Entropy or Gini of the root Node.  
Sv = total values in a particular node.  
S = Total Number of Values of Output Feature  
H(Sv) = Entropy or Gini of the Child Node

Information Gain is calculated using **Entropy**

|  |  |
| --- | --- |
| Information Gain of “Feature1“ | Information Gain of Feature-2 |
|  |  |

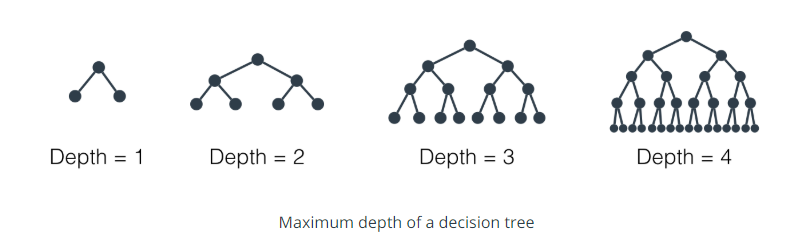
Information Gain calculated using **Gini Impurity**

|  |  |
| --- | --- |
| Information Gain of Feature-1 | Information Gain of Feature-2 |
|  |  |

If we compare the information gain across all the features with each other we can understand that **Feature-2** has the **highest information gain** which is which is calculated using **Gini Impurity**, hence We select "Feature-2" (0.074) as the root node.

**Hyper parameter tuning.**

**Max Depth:-** The height of a decision tree, also known as its depth, represents the number of levels or layers it contains.



**Min Sample Leaf:**

we are talking about the minimum number of samples required to be left at the leaf node. A split will only be considered if there are at least min\_samples\_leaf samples on the left and right branches.

**Min Sample Split:**

Minimum sample split decides or holds the value for the minimum number of samples necessary to split a nonterminal node. By default, the decision tree tries to split every node with two or more data rows inside it. This can again cause memorization of training data, resulting in a lesser generalized mode.

**Decision tree Regressor**

A Decision Tree Regressor is a supervised machine learning algorithm used for solving regression problems. Unlike its classification counterpart, the Decision Tree Classifier, which predicts discrete class labels, the Decision Tree Regressor predicts continuous numerical values.

Here's a step-by-step explanation of how the Decision Tree Regressor works:

1. \*\*Input Data:\*\* Take a dataset with input features (X) and continuous target values (Y).

2. \*\*Tree Construction:\*\* Build a binary tree structure starting with the root node to divide the input space into regions.

3. \*\*Node Splitting:\*\* Identify the best feature and split point at each node using criteria like MSE or MAE.

4. \*\*Finding the Best Split:\*\* Evaluate all possible splits for each feature and choose the one that minimizes the selected criterion.

5. \*\*Recursive Splitting:\*\* Continue recursively splitting nodes into child nodes until a stopping condition is met.

6. \*\*Stopping Criteria:\*\* Stop when reaching the maximum tree depth, having a minimum number of data points in a node, or achieving a minimum decrease in the criterion during a split.

7. \*\*Leaf Nodes (Terminal Nodes):\*\* Assign a constant value (prediction) to each leaf node, usually the mean or median of the target values within that node.

8. \*\*Prediction:\*\* To predict a new data point, traverse the tree from the root node, following feature splits until reaching a leaf node. The prediction is the value assigned to that leaf node.

9. \*\*Model Evaluation:\*\* Use regression metrics like MSE, MAE, or R-squared to evaluate the model's performance.

10. \*\*Overfitting Mitigation:\*\* To reduce overfitting, apply techniques like pruning, setting a maximum tree depth, or using ensemble methods like Random Forest.

**How to choose the best feature as the root node?**

Below are the steps to perform Decision Tree Regressor.

1. Sort the values.
2. Calculate the mean of the output column.
3. Calculate the MSE, MAE, or RMSE on Output Feature.
4. Create nodes from independent features.
5. Split the data based on the created nodes.
6. Calculate the mean of the dependent feature for the left and right nodes.
7. Perform MSE, MAE, or RMSE on each node only for the dependent feature.
8. Calculate the reduction in variance on each Node.
9. Select the node with the lower reduction in variance as the root node.

Example: Build DTR for the below example

|  |  |
| --- | --- |
| Height | Weight  (Output Column) |
| 165 | 65 |
| 175 | 70 |
| 160 | 50 |
| 170 | 85 |
| 180 | 90 |

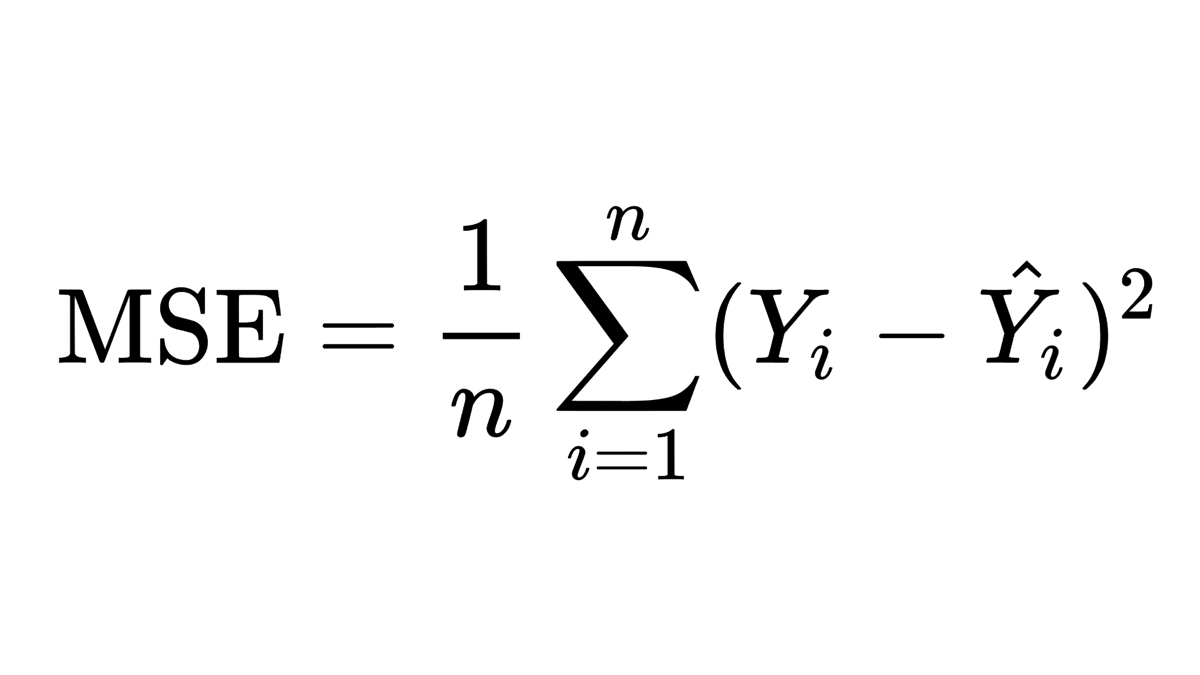
1. **Sort the values.**

|  |  |
| --- | --- |
| Height | Weight  (Output Column) |
| 160 | 50 |
| 165 | 65 |
| 170 | 85 |
| 175 | 70 |
| 180 | 90 |

1. **Calculate the mean of the output column.**

The Mean of the “weight” is 72.

1. **Calculate the MSE, MAE, or RMSE on Output Feature. For the above example, we will use MSE.**



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Variance  [Weight] = | (72-50)2+(72-65)2+(72-85)2+(72-70)2+(72-90)2 | | | | |
| 5 | | | | |
| Variance  [Weight] = | 206 |  |  |  |  |

Create nodes from independent features as shown below.

|  |  |  |  |
| --- | --- | --- | --- |
| Nodes | Calculate Node | Height | Weight |
|  |  | 160 | 50 |
| 162.5 | (160+165)/2 |
| 165 | 65 |
| 167.5 | (165+170)/2 |
| 170 | 85 |
| 172.5 | (170+175)/2 |
| 175 | 70 |
| 177.5 | (175+180)/2 |
| 180 | 90 |
|  |  |

1. Split the data based on the nodes created & Calculate the mean of the dependent feature for the left and right nodes.



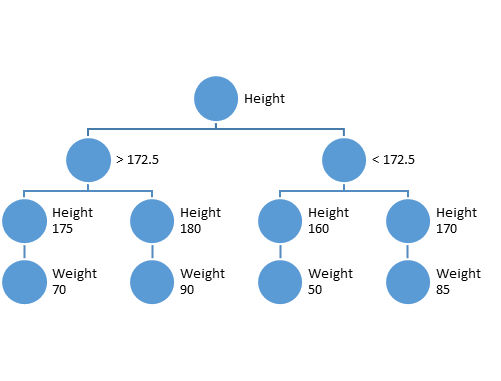
Perform MSE, MAE, or RMSE on each node only for the dependent feature. For this example, we will take **MSE**.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| LEFT NODE | | | | | | | RIGHT NODE | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| Varience[162.5] Left Node = | (77.5-65)2+(77.5-85)2+ (77.5-70)2+ (77.5-90)2 | | | | |  | Varience[162.5] Right Node= | (50-50)2 | | | | |
| 4 | | | | |  | 1 | | | | |
| Varience[162.5] Left Node = | 106.3 | |  |  |  |  | Varience[162.5] Right Node = | 0 |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| Varience[167.5] Left Node = | (81.7-85)2+(81.7-70)2+(81.7-90)2 | | | | |  | Varience[167.5] Right Node = | (57.5-50)2+(57.5-65)2 | | | | |
| 3 | | | | |  | 2 | | | | |
| Varience[167.5] Left Node = | 72.2 |  |  |  |  |  | Varience[167.5] Right Node = | 56.3 |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| Varience[172.5] Left Node = | (80-70)2+(80-90)2 | | | | |  | Varience[172.5] Right Node = | (66.7-50)2+(66.7-65)2+(66.7-85)2 | | | | |
| 2 | | | | |  | 3 | | | | |
| Varience[172.5] Left Node = | 100.0 |  |  |  |  |  | Varience[172.5] Right Node = | 205.6 |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| Varience[177.5] Left Node = | (90-90)2 | | | | | | Varience[177.5] Right Node = | (67.5-50)2+(67.5-65)2+(67.5-85)2+(67.5-70)2+(67.5-90)2 | | | | |
| 1 | | | | | | 4 | | | | |
| Varience[177.5] Left Node = | 156.3 |  |  |  |  |  | Varience[177.5] Right Node = | 0 |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

Calculate the reduction in variance on each Node & Select the node with the lower reduction in variance as the root node.

|  |  |  |  |
| --- | --- | --- | --- |
| Average Value of Height | Reduction in Variance | Height | Weight |
|  |  | 160 | 50 |
| 162.5 | 121.00 |
| 165 | 65 |
| 167.5 | 136.97 |
| 170 | 85 |
| 172.5 | 84.89 |
| 175 | 70 |
| 177.5 | 174.75 |
| 180 | 90 |
|  |  |

Since the “reduction in Variance” of 172.5 is the lowest(84.89), Hence 172.5 is to be used as the root node. And hence below is the Decision tree



|  |  |  |
| --- | --- | --- |
| Height | Weight | BMI |
| 6.4 | 50 | 15.6 |
| 6.6 | 60 | 18.2 |
| 5.6 | 70 | 25.0 |
| 5.8 | 80 | 27.6 |
| 6 | 140 | 46.7 |
| 6.2 | 100 | 32.3 |
| 5.2 | 110 | 42.3 |
| 5.4 | 120 | 44.4 |

**Pruning:**

Pruning in decision trees is a technique used to reduce the size and complexity of a tree by removing certain branches or nodes. It involves cutting back the tree to its essential parts, often to prevent **overfitting** and improve the tree's **generalization ability** on unseen data.

Overfitting occurs when a decision tree becomes too complex, capturing noise or specific patterns present only in the training data but not in the overall population. As a result, the overfitted tree may perform poorly on new, unseen data, leading to reduced predictive accuracy.

**What is Pre-Prunning?**

Pre-pruning, also known as **early stopping or top-down induction** of decision trees, is a pruning strategy used in the construction of decision trees to stop the tree-building process early before it becomes fully grown. In pre-pruning, the tree expansion process is controlled by setting certain conditions or constraints on **when to stop adding new nodes and branches**.

The key idea behind pre-pruning is to **limit the tree's growth during the learning process**, preventing it from becoming overly complex and overfitting the training data. By stopping the tree-building process early, pre-pruning aims to create a more generalized model that can better generalize to unseen data.

There are several common pre-pruning techniques:

Maximum Depth Limit: Set a maximum depth for the tree, such that the expansion process stops when the tree reaches the specified depth. This ensures that the tree doesn't grow too deep, limiting its complexity.

Minimum Samples per Leaf: Define the minimum number of samples required in a leaf node. If a node contains fewer samples than the specified threshold, further splitting of that node is halted, preventing the creation of small, noise-driven branches.

Minimum Samples per Split: Specify a minimum number of samples required to perform a node split. If a node has fewer samples than the threshold, it is not split, effectively controlling the growth of the tree.

Maximum Leaf Nodes: Set a limit on the total number of leaf nodes allowed in the tree. Once the tree reaches this limit, the growth process stops.

**What is post Pruning?**

Post-pruning, also known as **backward pruning** or error-based pruning, is a pruning technique used in decision trees after the tree has been fully grown. Unlike pre-pruning, which stops the tree-building process early, post-pruning involves removing some of the **already-grown branches** and nodes from the tree based on certain criteria.

The primary objective of post-pruning is to simplify the decision tree, reduce overfitting, and improve the model's generalization ability. By removing unnecessary branches that might have captured noise or specific patterns only present in the training data, post-pruning helps the tree perform better on unseen data.

The post-pruning process typically involves the following steps:

Validation Set: The original decision tree is first grown using the training data.

Validation Data: A separate validation dataset, different from the training data, is used to evaluate the performance of the fully grown tree.

Pruning Criterion: A pruning criterion is used to evaluate the impact of removing branches/nodes from the tree. Common pruning criteria include measures like cross-validation error, classification error rate, Gini impurity, or Mean Squared Error (MSE) for regression tasks.

Pruning Iteration: Starting from the leaf nodes, the tree is iteratively pruned upwards. At each step, a node is replaced with its parent node, creating a simpler subtree.

Validation Performance: The performance of the pruned tree is evaluated on the validation dataset using the chosen pruning criterion.

Best Pruned Tree: The iteration continues until the performance on the validation data starts to deteriorate. The tree at this point is considered the best-pruned version.

**Interview Question:**

1. Is scaling an impact on a Decision tree?

Ans: Decision trees and ensemble methods do not require feature scaling as they are not sensitive to the variance in the data.

1. Is the threshold impacted by an outlier?

Ans: In the “Decision tree”, the threshold does not leave any impact on an outlier. I guess one big reason is that they do a slice of the data, and then after that slice, it doesn't matter how big of a value you have. If you had five data points and one of their features looked like {1,2,3,4,1000000}