1.7. Decision Tree

Decision Tree algorithm belongs to the family of supervised learning algorithms. Decision Trees are versatile Machine Learning algorithms that can perform both classification and regression tasks, but mostly it is preferred for solving Classification problems. A decision tree is an upside-down tree that makes decisions based on the conditions present in the data.

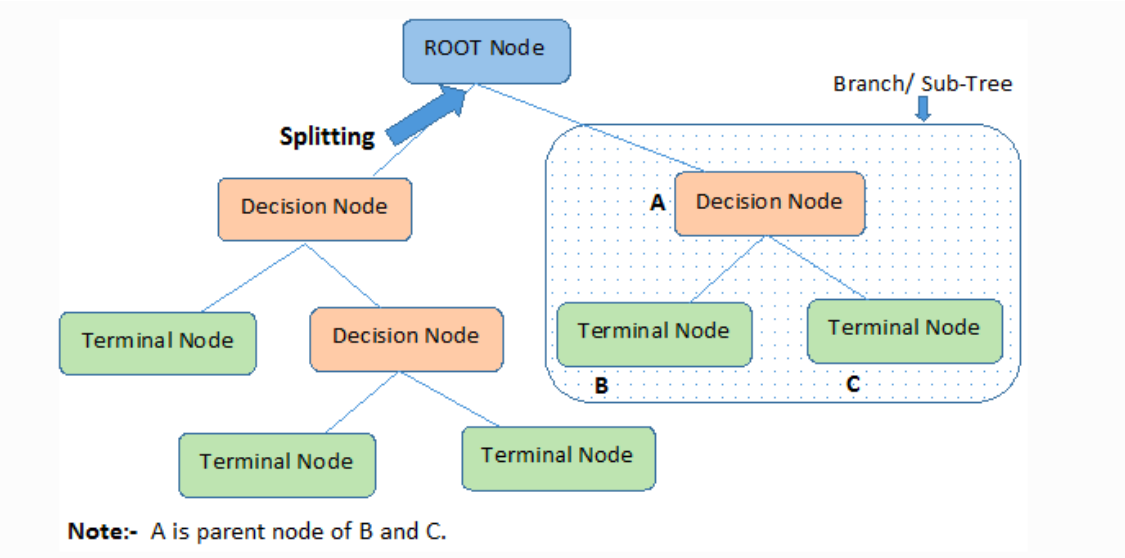
In general, decision trees are constructed via an algorithmic approach that identifies ways to split a data set based on different conditions. It is one of the most widely used and practical methods for supervised learning. Decision Trees are a non-parametric supervised learning method used for both classification and regression tasks.

Tree models where the target variable can take a discrete set of values are called classification trees. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.

The goal of using a Decision Tree is to create a training model that can be used to predict the class or value of the target variable by learning simple decision rules inferred from prior data (training data).

Important Terminology related to Decision Trees

- ***Root Node:*** It represents the entire population or sample and this further gets divided into two or more homogeneous sets.
- ***Splitting:*** It is a process of dividing a node into two or more sub-nodes.
- ***Decision Node:*** When a sub-node splits into further sub-nodes, then it is called the decision node.
- ***Leaf / Terminal Node:*** Nodes that do not split are called Leaf or Terminal nodes.
- ***Branch / Sub-Tree:*** A subsection of the entire tree is called branch or sub-tree.
- ***Parent and Child Node:*** A node, which is divided into sub-nodes is called a parent node of sub-nodes whereas sub-nodes are the child of a parent node.



Decision trees classify the examples by sorting them down the tree from the root to some leaf/terminal node, with the leaf/terminal node providing the classification of the example.

Each node in the tree acts as a test case for some attribute, and each edge descending from the node corresponds to the possible answers to the test case. This process is recursive in nature and is repeated for every subtree rooted at the new node.

Assumptions while creating Decision Tree

Below are some of the assumptions we make while using Decision tree:

1. In the beginning, the whole training set is considered as the root.
2. Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
3. Order to place attributes as the root or internal node of the tree is done by using some statistical approach.

The primary challenge in the decision tree implementation is to identify which attributes do we need to consider as the root node and each level. Handling this is to know the attributes selection. We have different attributes selection measures to identify the attribute which can be considered as the root note at each level.

This whole process generates a tree-like structure. The first splitting node is called the root node. The end nodes are called leaves and are associated to a class label. The paths from root to leaf produce the classification rules.

At each iteration, in order to decide which feature leads to the purest subset, we need to be able to measure the purity of a dataset. Different metrics and indices such as Information gain, Gini index have been proposed in the past.

During training, the selected quality measure is calculated for all candidate features to find out which one will produce the best split.

1. ID3 (Iterative Dicotomizer3)

It is named such because the algorithm iteratively (repeatedly) dichotomizes (divides features into two groups) at each step. The ID3 algorithm builds decision trees using a top-down, greedy approach. Means that we start building the tree from the top and the greedy approach means that at each iteration we select the best feature at the present moment to create a node.

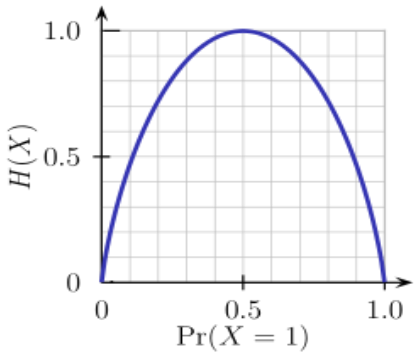
ID3 is only preferred for classification problems with nominal features only.

Now, the question is: *How does ID3 select the best feature?*

ID3 uses **Entropy** and **Information Gain** to find the best feature.

Entropy

Entropy is a measure of the randomness in the information being processed. The higher the entropy, the harder it is to draw any conclusions from that information. Flipping a coin is an example of an action that provides information that is random.



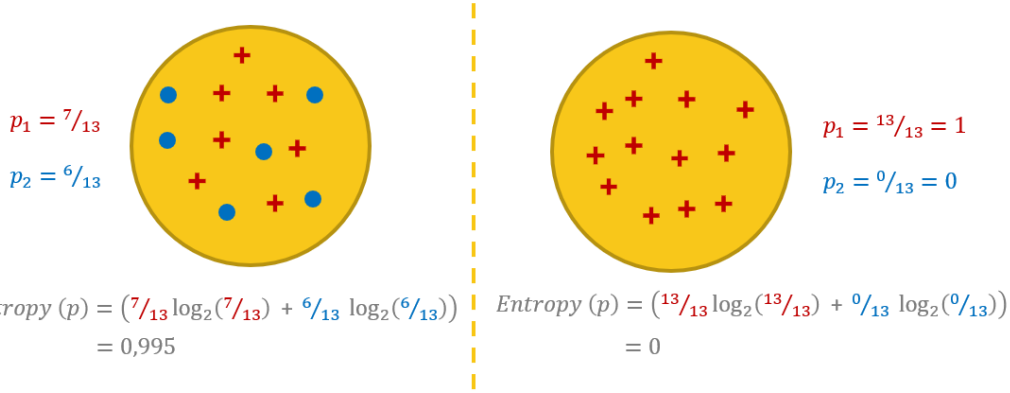
From the above graph, it is quite evident that the entropy $\mathrm{H(X)}$ is zero when the probability is either 0 or 1. The Entropy is maximum when the probability is 0.5 because it projects perfect randomness in the data and there is no chance of perfectly determining the outcome.

Mathematically Entropy for 1 attribute is represented as:

$$E(S) = \sum_{i=1}^{c} -p_i \, log_2 \, p_i$$

Where $S \rightarrow$ Current state, and $p_i \rightarrow$ Probability of an event i of state S or Percentage of class i in a node of state S.

To get a better understanding of entropy, let's work on two different example datasets, both with two classes, respectively represented as blue dots and red crosses.



In the example dataset on the left, we have a mixture of blue dots and red crosses. In the example of the dataset on the right, we have only red crosses.

For the example on the left, the probability is 7/13 for the class with red crosses and 6/13 for the class with blue dots. The formula above leads to an entropy value of 0.99.

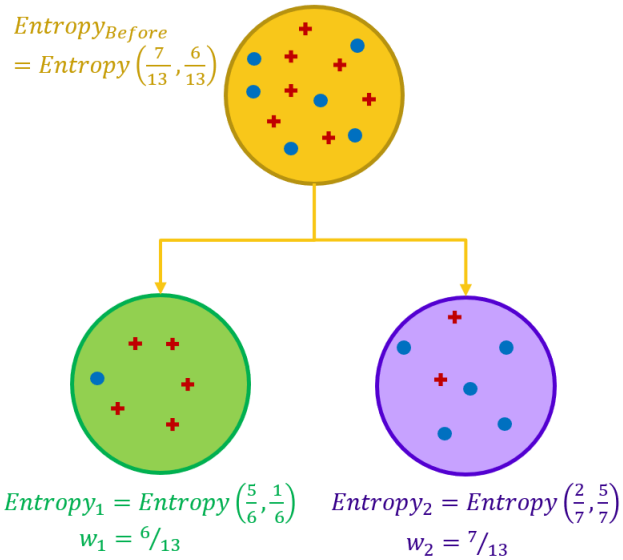
For the example on the right, the probability is 13/13 = 1.0 for the class with the red crosses and 0/13 = 0.0 for the class with the blue dots. In this case, the formula above leads to an entropy value of 0.0.

Entropy can be a measure of purity, disorder or information. Due to the mixed classes, the dataset on the left is less pure and more confused (more disorder, i.e., higher entropy). However, more disorder also means more information. Indeed, if the dataset has only points of one class, there is not much information you can extract from it no matter how long you try. In comparison, if the dataset has points from both classes, it also has a higher potential for information extraction. So, the higher entropy value of the dataset on the left can also be seen as a larger amount of potential information.

ID3 follows the rule — *A branch with an entropy of zero is a leaf node and A branch with entropy more than zero needs further splitting*. The goal of each split in a decision tree is to move from a confused dataset to two (or more) purer subsets.

Information Gain

As we discussed, in order to evaluate how good a feature is for splitting, the difference in entropy before and after the split is calculated.



That is, first we calculate the entropy of the dataset before the split, and then we calculate the entropy for each subset after the split. Finally, the sum of the output entropies weighted by the size of the subsets is subtracted from the entropy of the dataset before the split. This difference

measures the gain in information or the reduction in entropy. If the information gain is a positive number, this means that we move from a confused dataset to a number of purer subsets.

Information gain is a decrease in entropy. It computes the difference between entropy before split and average entropy after split of the dataset based on given attribute values. Mathematically,

$$Information\ Gain = Entropy(Before) - \sum_{j=1}^{k} Entropy(j, after)$$

Where "before" is the dataset before the split, k is the number of subsets generated by the split, and (j, after) is subset j after the split.

If the information gain is a positive number, this means that we move from a confused dataset to a number of purer subsets.

At each step, we would then choose to split the data on the feature with the highest value in information gain as this leads to the purest subsets. The algorithm that applies this measure is the ID3 algorithm. The ID3 algorithm has the disadvantage of favouring features with a larger number of values, generating larger decision trees.

Example: Playing game (Yes or No) based on weather condition

outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

ID3 generates a tree by considering the whole set S as the root node. It then iterates on every attribute and splits the data into fragments known as subsets to calculate the entropy or the information gain of that attribute.

We have four X values (*outlook*, *temp*, *humidity* and *windy*) being categorical and one y value (play Yes or No) also being categorical. This is a binary classification problem, let's build the tree using the ID3 algorithm.

To create a tree, we need to have a root node first and we know that nodes are features / attributes: outlook, temp, humidity and windy. But which one do we need to pick first??

Determine the attribute that best classifies the training data i.e., the attribute with the highest information gain in ID3. Use this attribute at the root of the tree. Repeat this process at for each branch.

Steps:

1. compute the entropy for data-set

2. for every attribute/feature:

    (i) calculate entropy for all categorical values

    (ii) take average information entropy for the current attribute

    (iii) calculate gain for the current attribute

3. pick the highest gain attribute.

4. repeat until we get the tree we desired.

Solution:

1. Compute the entropy for the weather data set:

$$E(S) = \sum_{i=1}^{2} -p_i \, log_2 \, p_i \qquad c = \{\text{yes}, \text{no}\}$$

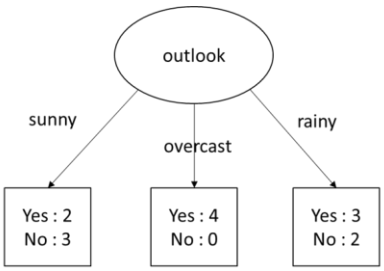
Out of 14 instances, 9 are classified as *Yes* and 5 as *No*.

$$p_{yes} = -\left(\frac{9}{14}\right) * log_2 \left(\frac{9}{14}\right) = 0.41$$

$$p_{no} = -\left(\frac{5}{14}\right) * log_2 \left(\frac{5}{14}\right) = 0.53$$

$$\therefore E(S) = p_{yes} + p_{no} = 0.94$$

2. For every feature calculate the entropy and information gain:



$$E(outlook = sunny) = -\frac{2}{5} log_2 \left(\frac{2}{5}\right) - \frac{3}{5} log_2 \left(\frac{3}{5}\right) = 0.971$$

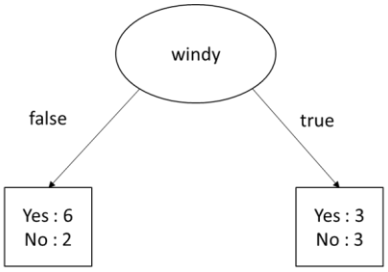
$$E(outlook = overcast) = -1 \, log_2 \, (1) \; - 0 \, log_2 \, (0) = 0$$

$$E(outlook = rainy) = -\frac{3}{5} \, log_2 \left(\frac{3}{5}\right) - \frac{2}{5} \, log_2 \left(\frac{2}{5}\right) = 0.971$$

$$Weighted \; Average \; Entropy \; for \; Outlook \; i.e. \; E(S, outlook)$$

$$E(S, outlook) = \frac{5}{14} * 0.971 + \frac{4}{14} * 0 + \frac{5}{14} * 0.971 = 0.693$$

$$Information \; Gain = E(S) - E(S, outlook) = 0.94 \; - 0.693$$



$$E(windy = false) = -\frac{6}{8}log_2\left(\frac{6}{8}\right) - \frac{2}{8}log_2\left(\frac{2}{8}\right) = 0.811$$

$$E(windy = true) = -\frac{3}{6}log_2\left(\frac{3}{6}\right) - \frac{3}{6}log_2\left(\frac{3}{6}\right) = 1$$

$$Weighted\ Average\ Entropy\ for\ windy\ i.e.\ E(S, windy)$$

$$E(S, windy) = \frac{8}{14} * 0.811 + \frac{6}{14} * 1 = 0.892$$

$$Information\ Gain = E(S) - E(S, windy) = 0.94 - 0.892$$

Similarity we can calculate for other two attributes (Humidity and Temp). Now pick the attribute with highest Information Gain,

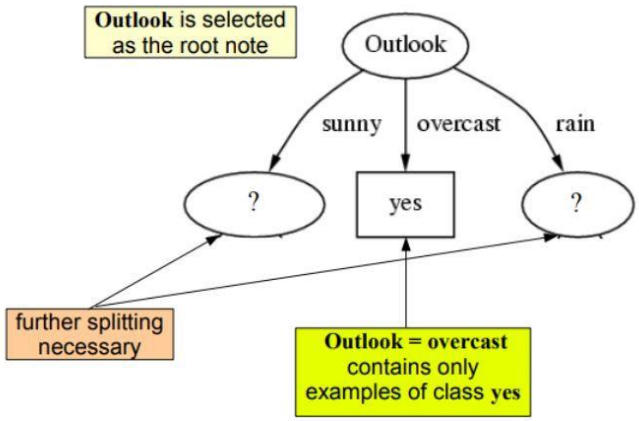
Information Gain (Outlook) : 0.247

Information Gain (temp.) : 0.029

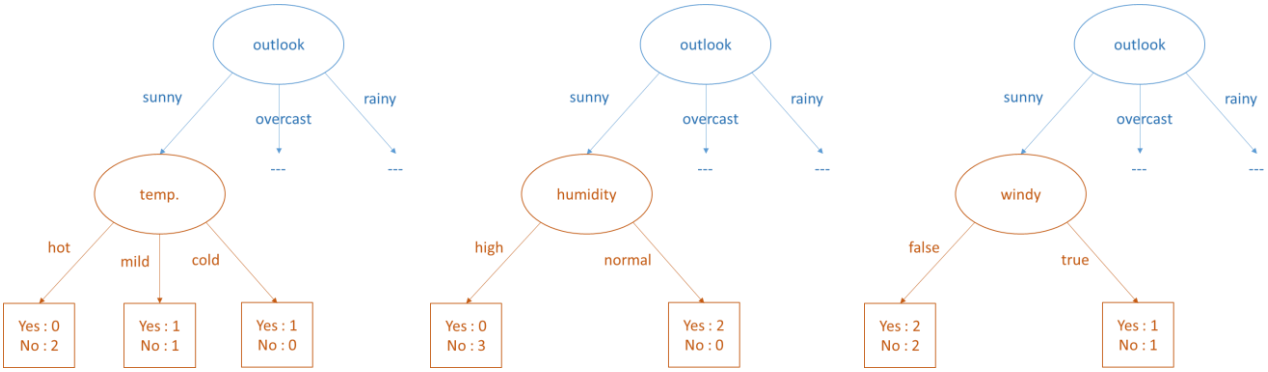
Information Gain (humidity) : 0.152

Information Gain (windy) : 0.048

So our root node is Outlook.



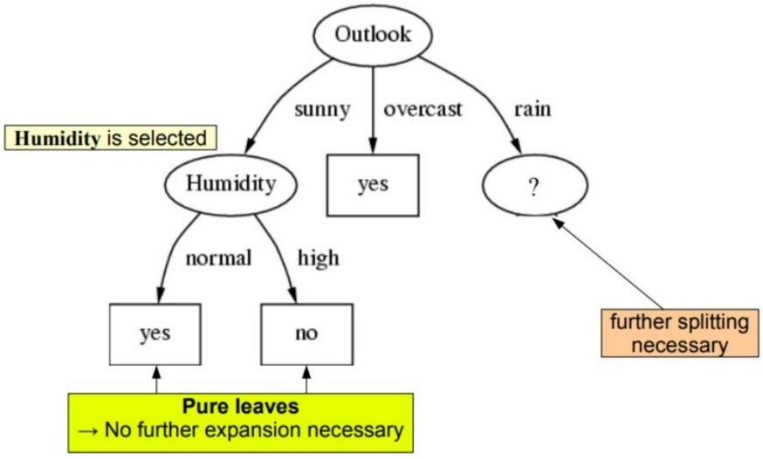
Repeat the same thing for sub-trees till we get the tree.



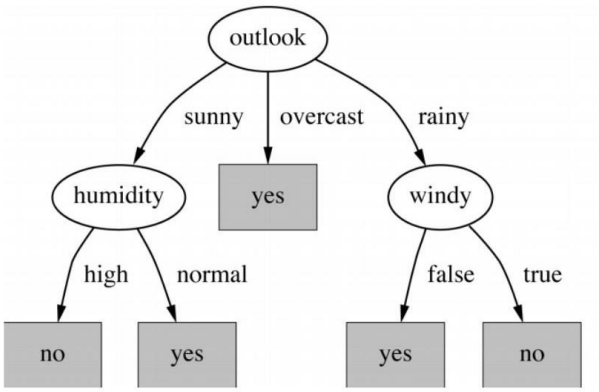
Information Gain (temp.) : 0.571

Information Gain (humidity) : 0.971

Information Gain (windy) : 0.020



Finally, we get the tree something like this.



2. C4.5 algorithm

Information gain is biased towards choosing attributes with a large number of values as root nodes. It means it prefers the attribute with a large number of distinct values.

C4.5, an improvement of ID3, uses *Gain Ratio* which is a modification of Information gain that reduces its bias and is usually the best option.

The C4.5 algorithm, introduces the ***SplitInfo*** concept.

SplitInfo is defined as the sum over the weights multiplied by the logarithm of the weights, where the weights are the ratio of the number of data points in the current subset with respect to the number of data points in the parent dataset.

The gain ratio is then calculated by dividing the information gain from the ID3 algorithm by the SplitInfo value.

$$Gain\ Ratio = \frac{Information\ Gain}{SplitInfo} = \frac{Entropy\ (before) - \sum_{j=1}^{K} Entropy(j, after)}{\sum_{j=1}^{K} w_j\ log_2\ w_j}$$

where,

$$w_j = \frac{\#\ samples\ in\ \ subset\ (j, after)}{\#\ samples\ in\ dataset\ (before)}$$

Where “before” is the dataset before the split, K is the number of subsets generated by the split, and (j, after) is subset j after the split.

3. CART (Classification And Regression Tree)

Another measure for purity — or actually impurity — used by the CART algorithm is the Gini index.

The Gini index is based on Gini impurity. Gini impurity is defined as:

$$Gini\ impurity\ (S) = 1 - \sum_{i=1}^{N} (p_i)^2$$

Where S is the whole dataset, N is the number of classes, and p_i is the probability of class i in the same dataset.

The Gini index is then defined as the weighted sum of the Gini impurity of the different subsets after a split, where each portion is weighted by the ratio of the size of the subset with respect to the size of the parent dataset.

$$Gini\ Index = \sum_{j=1}^{K} w_j\ Gini\ Impurity\ (j,\ after)$$

$$w_j = \frac{\#\ data\ in\ in\ subset\ (j, after)}{\#\ data\ in\ dataset\ (before)}$$

Where K is the number of subsets generated by the split and (j, after) is subset j after the split.

When training a decision tree, the best split is chosen by **maximizing the Gini Gain**,

Steps:

1.compute the Gini index for data-set

2.for every attribute/feature:

1.calculate Gini index for all categorical values

2.take average information entropy for the current attribute

3.calculate the Gini gain

3. pick the best Gini gain attribute.

4. Repeat until we get the tree we desired.

Optimizing Hyperparameters

Decision Trees make very few assumptions about the training data (as opposed to linear models, which obviously assume that the data is linear, for example). Such a model is often called a nonparametric model, not because it does not have any parameters (it often has a lot) but because the number of parameters is not determined prior to training, so the model structure is free to stick closely to the data.

Decision trees, like many other machine learning algorithms, are subject to potentially overfitting the training data. Trees that are too deep can lead to models that are too detailed and don't generalize on new data. On the other hand, trees that are too shallow might lead to overly simple models that can't fit the data. You see, the size of the decision tree is of crucial importance.

To avoid overfitting the training data, you need to restrict the Decision Tree's freedom during training, this is called regularization. The regularization hyperparameters depend on the algorithm used, but generally you can at least restrict the maximum depth of the Decision Tree.
