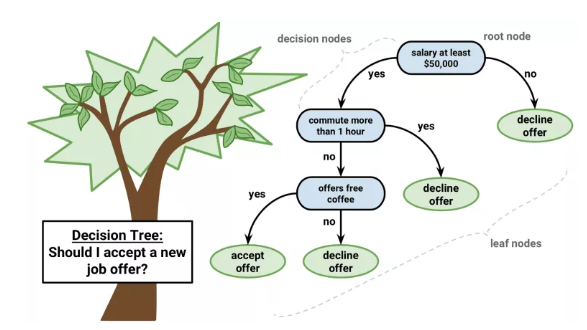
**Decision Tree**

Decision Tree algorithm is a supervised learning algorithm. Unlike other supervised learning algorithms, decision tree algorithm can be used for solving regression and classification problems too. The general motive of using Decision Tree is to create a training model which can use to predict class or value of target variables by learning decision rules inferred from prior data (training data).

The understanding level of Decision Trees algorithm is so easy compared with other classification algorithms. The decision tree algorithm tries to solve the problem, by using tree representation. **Each internal node of the tree corresponds to an attribute, and each leaf node corresponds to a class label**.



**Pseudo Code for Decision Tree**:

* Place the best attribute of the dataset at the root of the tree.
* Split the training set into subsets. Subsets should be made in such a way that each subset contains data with the same value for an attribute.
* Repeat step 1 and step 2 on each subset until you find leaf nodes in all the branches of the tree.

**Decision Tree assumptions**:

* At the beginning, the whole training set is considered as the root.
* Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
* Records are distributed recursively based on attribute values.
* Order to placing attributes as root or internal node of the tree is done by using some statistical approach.

**Methods to calculate root node and other ones**:

1. Information Gain
2. Gini index

Information Gain:

By using information gain as a criterion, we try to estimate the information contained by each attribute. To measure the randomness or uncertainty of a random variable X is defined by Entropy.

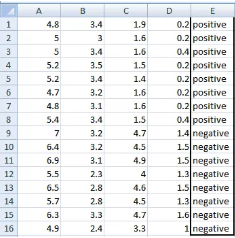
For a binary classification problem with only two classes, positive and negative class.

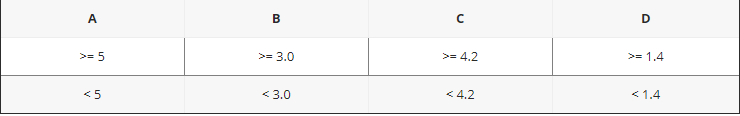
* If all examples are positive or all are negative then entropy will be zero i.e, low.
* If half of the records are of positive class and half are of negative class then entropy is one i.e., high.



By calculating entropy measure of each attribute, we can calculate their information gain. Information Gain calculates the expected reduction in entropy due to sorting on the attribute.

Let’s take following example to understand information gain and entropy.





There are 2 steps for calculating information gain for each attribute:

* Calculate entropy of Target.
* Entropy for every attribute A, B, C, D needs to be calculated. Using information gain formula, we will subtract this entropy from the entropy of target.

**Entropy of the target variable:**

E(8,8) = -1\*( (p(+ve)\*log2( p(+ve)) + (p(-ve)\*log2( p(-ve)) )

= -1\*( (8/16)\*log2(8/16)) + (8/16) \* log2(8/16) )

= 1

**Entropy of the variable A:**

A has value >=5 for 12 records out of 16 and 4 records with value <5 value.

For Var A >= 5 & class == positive: 5/12

For Var A >= 5 & class == negative: 7/12

Entropy(5,7) = -1 \* ( (5/12)\*log2(5/12) + (7/12)\*log2(7/12)) = 0.9799

For Var A <5 & class == positive: 3/4

For Var A <5 & class == negative: 1/4

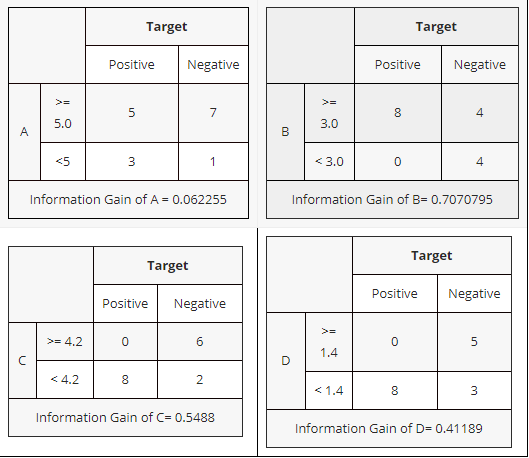
Entropy(3,1) = -1 \* ( (3/4)\*log2(3/4) + (1/4)\*log2(1/4)) = 0.81128

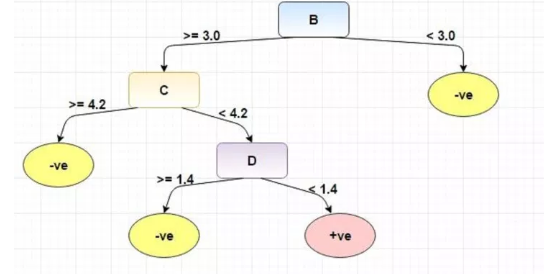
Entropy(Target, A) = P(>=5) \* E(5,7) + P(<5) \* E(3,1)

= (12/16) \* 0.9799 + (4/16) \* 0.81128 = 0.937745

Information Gain (A) = 1 – 0.94 = 0.06

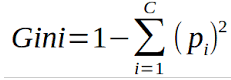
On similar lines IG for other variables can be calculated.





**Gini index**:

Gini Index is a metric to measure how often a randomly chosen element would be incorrectly identified. It means an attribute with lower gini index should be preferred.



Gini index for variable A:

For Var A >= 5 & class == positive: 5/12

For Var A >= 5 & class == negative: 7/12

gini(5,7) = 1- ( (5/12)2 + (7/12)2 ) = 0.4860

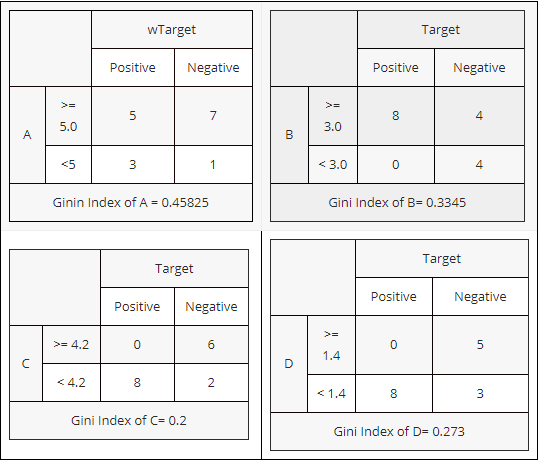
For Var A <5 & class == positive: 3/4

For Var A <5 & class == negative: 1/4

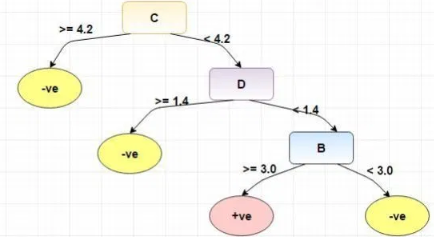
gini(3,1) = 1- ( (3/4)2 + (1/4)2 ) = 0.375

gini(Target,A) = 1- (12/16\*0.486 + 4/16\*0.375) = 0.458

Similarly, the same can be calculated for others.



From the above gini index is lowest for C followed by D, B, A.



**Overfitting**

Overfitting is a practical problem while building a decision tree model. The model is having an issue of overfitting is considered when the algorithm continues to go deeper and deeper in the to reduce the training set error but results with an increased test set error i.e, Accuracy of prediction for our model goes down. It generally happens when it builds many branches due to outliers and irregularities in data.

Two approaches which we can use to avoid overfitting are:

**Pre-Pruning**

In pre-pruning, it stops the tree construction bit early. It is preferred not to split a node if its goodness measure is below a threshold value. But it’s difficult to choose an appropriate stopping point.

**Post-Pruning**

In post-pruning first, it goes deeper and deeper in the tree to build a complete tree. We use a cross-validation data to check the effect of our pruning. It tests whether expanding a node will make an improvement or not. We can continue by expanding that node if there is an improvement. But if it shows a reduction in accuracy then it should not be expanded i.e., the node should be converted to a leaf node.

**Advantages:**

* Decision Trees are easy to explain. It results in a set of rules.
* It follows the same approach as humans generally follow while making decisions.
* Interpretation of a complex Decision Tree model can be simplified by its visualizations. Even a naive person can understand logic.

**Disadvantages:**

* There is a high probability of overfitting in Decision Tree.
* Need to be careful with parameter tuning.
* Can create biased learned trees if some classes dominate.
* Information gain in a decision tree with categorical variables gives a biased response for attributes with greater no. of categories.

**Hyperparameter tuning:**

Parameters which define the model architecture are referred to as hyperparameters and thus this process of searching for the ideal model architecture is referred to as hyperparameter tuning.

These hyperparameters might address model design questions such as:

What should be the maximum depth allowed for my decision tree?

What should be the minimum number of samples required at a leaf node in my decision tree?

Some of them are mentioned below.

**max\_depth**: int or None, optional (default=None)

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples.

**min\_samples\_split**: int, float, optional (default=2)

The minimum number of samples required to split an internal node.

**min\_samples\_leaf**: int, float, optional (default=1)

The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min\_samples\_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.

**max\_features**: int, float, string or None, optional (default=None)

The number of features to consider when looking for the best split:

If int, then consider max\_features features at each split.

If float, then max\_features is a fraction and int(max\_features \* n\_features) features are considered at each split.

If “auto”, then max\_features=sqrt(n\_features).

If “sqrt”, then max\_features=sqrt(n\_features).

If “log2”, then max\_features=log2(n\_features).

If None, then max\_features=n\_features.

**class\_weight**: dict, list of dicts, “balanced” or None, default=None

Weights associated with classes in the form {class\_label: weight}. If not given, all classes are supposed to have weight one. For multi-output problems, a list of dicts can be provided in the same order as the columns of y.

**How to perform hyperparameter tuning?**

* Define a model
* Define the range of possible values for all hyperparameters
* Define a method for sampling hyperparameter values
* Define an evaluation criterion to judge the model
* Define a cross-validation method

**Methods used for parameter tuning**

**Grid Search**

Grid search is arguably the most basic hyperparameter tuning method. With this technique, we simply build a model for each possible combination of all the hyperparameter values provided, evaluating each model, and selecting the architecture which produces the best results.

For example, we would define a list of values to try for both n\_estimators and max\_depth and a grid search would build a model for each possible combination.

Performing grid search over the defined hyperparameter space.

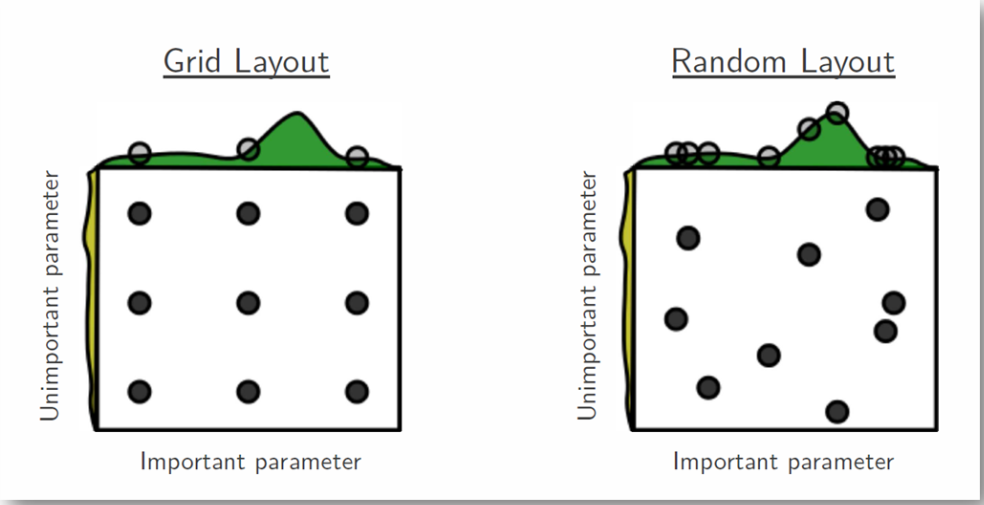


Each model would be fit to the training data and evaluated on the validation data. As you can see, this is an exhaustive sampling of the hyperparameter space and can be quite inefficient.

**Random search**

Random search differs from grid search in that we longer provide a discrete set of values to explore for each hyperparameter; rather, we provide a statistical distribution for each hyperparameter from which values may be randomly sampled.

We can also define how many iterations we'd like to build when searching for the optimal model. For each iteration, the hyperparameter values of the model will be set by sampling the defined distributions above. One of the main theoretical backings to motivate the use of random search in place of grid search is the fact that for most cases, hyperparameters are not equally important.



As you can see, this search method works best under the assumption that not all hyperparameters are equally important. While this isn't always the case, the assumption holds true for most datasets.