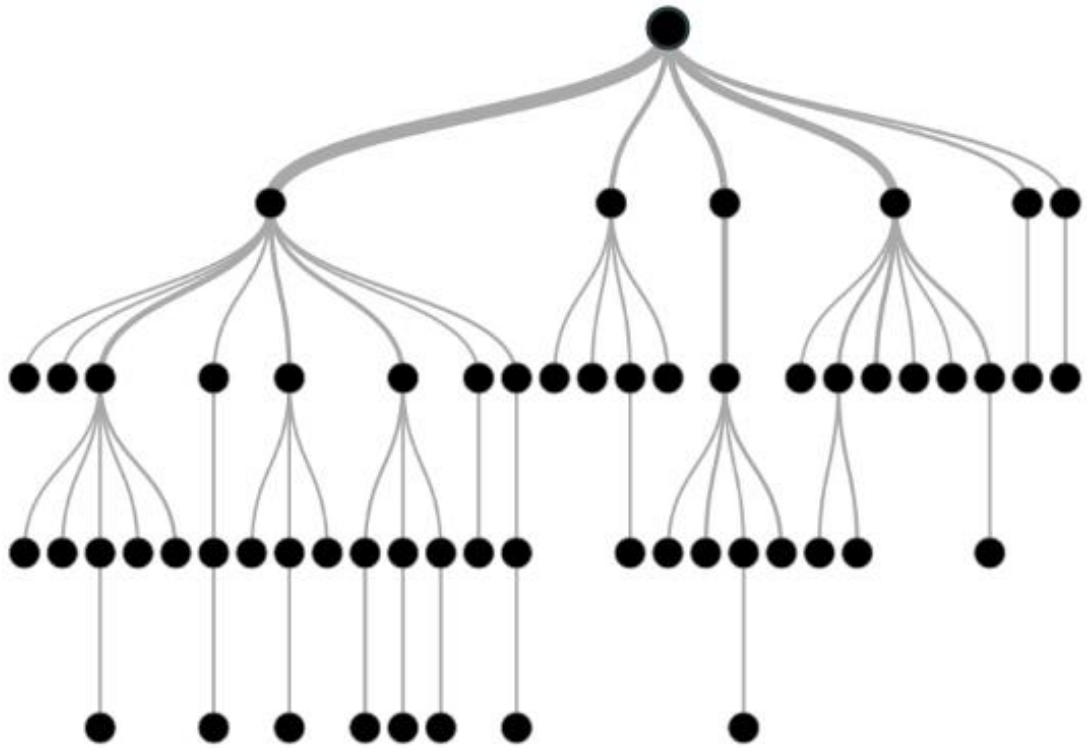
What is a Decision Tree?

A decision tree is a predictive model that uses a flowchart-like structure to make decisions based on input data. It divides data into branches and assigns outcomes to leaf nodes. Decision trees are used for classification and regression tasks, providing easy-to-understand models.

A decision tree is a hierarchical model used in decision support that depicts decisions and their potential outcomes, incorporating chance events, resource expenses, and utility. The tree structure is comprised of a root node, branches, internal nodes, and leaf nodes, forming a hierarchical, tree-like structure.

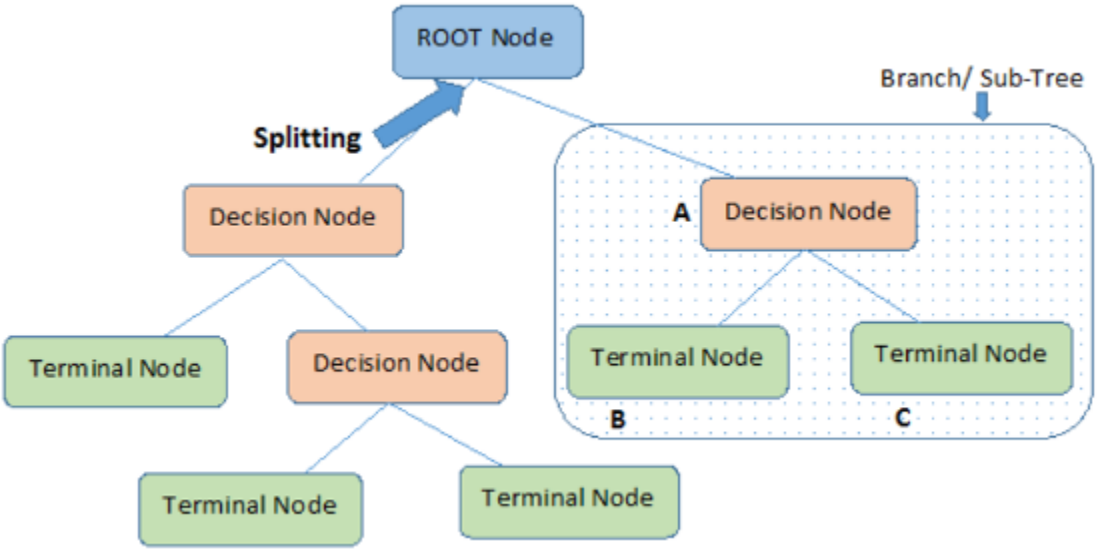
It is a tool that has applications spanning several different areas. The name itself suggests that it uses a flowchart like a tree structure to show the predictions that result from a series of feature-based splits. It starts with a root node and ends with a decision made by leaves.



Decision Tree Terminologies

Before learning more about decision trees let's get familiar with some of the terminologies:

- *Root Nodes* – It is the node present at the beginning of a decision tree from this node the population starts dividing according to various features.
- *Decision Nodes* – the nodes we get after splitting the root nodes are called Decision Node
- *Leaf Nodes* – the nodes where further splitting is not possible are called leaf nodes or terminal nodes.
- *Sub-tree* – just like a small portion of a graph is called sub-graph similarly a sub-section of this decision tree is called sub-tree.
- *Pruning* – is nothing but cutting down some nodes to stop overfitting.



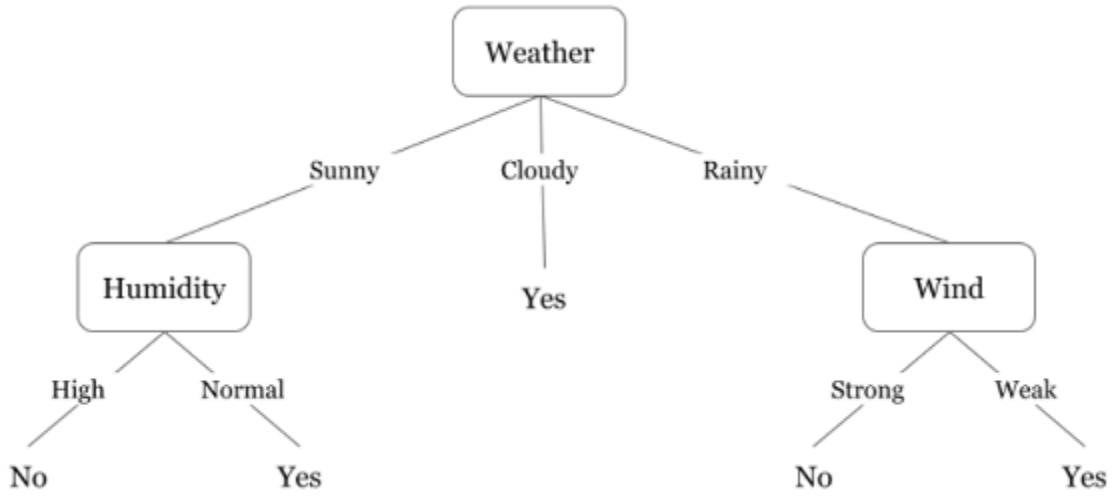
Example of Decision Tree

Let's understand decision trees with the help of an example:

Day	Weather	Temperature	Humidity	Wind	Play?
1	Sunny	Hot	High	Weak	No
2	Cloudy	Hot	High	Weak	Yes
3	Sunny	Mild	Normal	Strong	Yes
4	Cloudy	Mild	High	Strong	Yes
5	Rainy	Mild	High	Strong	No
6	Rainy	Cool	Normal	Strong	No
7	Rainy	Mild	High	Weak	Yes
8	Sunny	Hot	High	Strong	No
9	Cloudy	Hot	Normal	Weak	Yes
10	Rainy	Mild	High	Strong	No

Decision trees are upside down which means the root is at the top and then this root is split into various several nodes. Decision trees are nothing but a bunch of if-else statements in layman terms. It checks if the condition is true and if it is then it goes to the next node attached to that decision.

In the below diagram the tree will first ask what is the weather? Is it sunny, cloudy, or rainy? If yes, then it will go to the next feature which is humidity and wind. It will again check if there is a strong wind or weak, if it's a weak wind and it's rainy then the person may go and play.



Did you notice anything in the above flowchart? We see that if the *weather is cloudy* then we must go to play. Why didn't it split more? Why did it stop there?

To answer this question, we need to know about few more concepts like entropy, information gain, and Gini index. But in simple terms, I can say here that the output for the training dataset is always yes for cloudy weather, since there is no disorderliness here we don't need to split the node further.

The goal of machine learning is to decrease uncertainty or disorders from the dataset and for this, we use decision trees.

Now you must be thinking how do I know what should be the root node? what should be the decision node? when should I stop splitting? To decide this, there is a metric called "Entropy" which is the amount of uncertainty in the dataset.

Entropy

Entropy is nothing but the uncertainty in our dataset or measure of disorder. Let me try to explain this with the help of an example.

Suppose you have a group of friends who decides which movie they can watch together on Sunday. There are 2 choices for movies, one is *"Lucy"* and the second is *"Titanic"* and now everyone has to tell their choice. After everyone gives their answer we see that *"Lucy" gets 4 votes* and *"Titanic" gets 5 votes.*

Which movie do we watch now? Isn't it hard to choose 1 movie now because the votes for both the movies are somewhat equal.

This is exactly what we call disorderness, there is an equal number of votes for both the movies, and we can't really decide which movie we should watch. It would have been much easier if the votes for "Lucy" were 8 and for "Titanic" it was 2. Here we could easily say that the majority of votes are for "Lucy" hence everyone will be watching this movie.

In a decision tree, the output is mostly "yes" or "no"

The formula for Entropy is shown below:

$$E(S) = -p_{(+)}\log p_{(+)} - p_{(-)}\log p_{(-)}$$

Here p_+ is the probability of positive class.

p_- is the probability of negative class.

S is the subset of the training example

How do Decision Trees use Entropy?

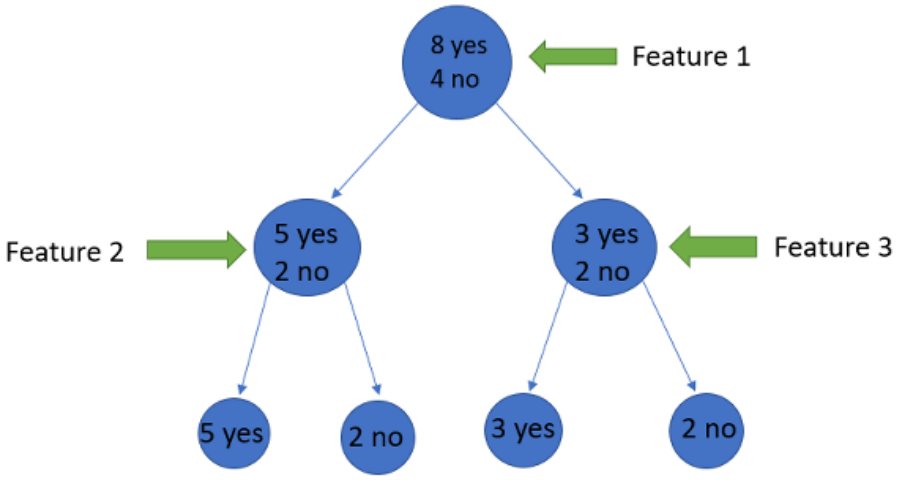
Now we know what entropy is and what is its formula, Next, we need to know that how exactly does it work in this algorithm.

Entropy basically measures the impurity of a node. *Impurity is the degree of randomness; it tells how random our data is*. A pure sub-split means that either you should be getting "yes", or you should be getting "no".

Suppose a feature has 8 "yes" and 4 "no" initially, after the first split the left node *gets 5 'yes' and 2 'no'* whereas right node *gets 3 'yes' and 2 'no'.*

We see here the split is not pure, why? Because we can still see some negative classes in both the nodes. To make a decision tree, we need to calculate the impurity of each split, and when the purity is 100%, we make it as a leaf node.

To check the impurity of feature 2 and feature 3 we will take the help for Entropy formula.



For Feature 2,

$$\Rightarrow -\left(\frac{5}{7}\right)log_2\left(\frac{5}{7}\right) - \left(\frac{2}{7}\right)log_2\left(\frac{2}{7}\right)$$

$$\Rightarrow -(0.71 * -0.49) - (0.28 * -1.83)$$

$$\Rightarrow -(-0.34) - (-0.51)$$

$$\Rightarrow 0.34 + 0.51$$

$$\Rightarrow 0.85$$

For feature 3,

$$\Rightarrow -\left(\frac{3}{5}\right)log_2\left(\frac{3}{5}\right) - \left(\frac{2}{5}\right)log_2\left(\frac{2}{5}\right)$$

$$\Rightarrow -(0.6 * -0.73) - (0.4 * -1.32)$$

$$\Rightarrow -(-0.438) - (-0.528)$$

$$\Rightarrow 0.438 + 0.528$$

$$\Rightarrow 0.966$$

We can clearly see from the tree itself that left node has low entropy or more purity than right node since left node has a greater number of "yes" and it is easy to decide here.

Always remember that the higher the Entropy, the lower will be the purity.

As mentioned earlier the goal of machine learning is to decrease the uncertainty or impurity in the dataset, here by using the entropy we are getting the impurity of a particular node, we don't know if the parent entropy or the entropy of a particular node has decreased or not.

For this, we bring a new metric called "Information gain" which tells us how much the parent entropy has decreased after splitting it with some feature.

Information Gain

Information gain measures the reduction of uncertainty given some feature and it is also a deciding factor for which attribute should be selected as a decision node or root node.

$$Information\ Gain\ =\ E(Y)\ -\ E(Y|X)$$

It is just entropy of the full dataset – entropy of the dataset given some feature.

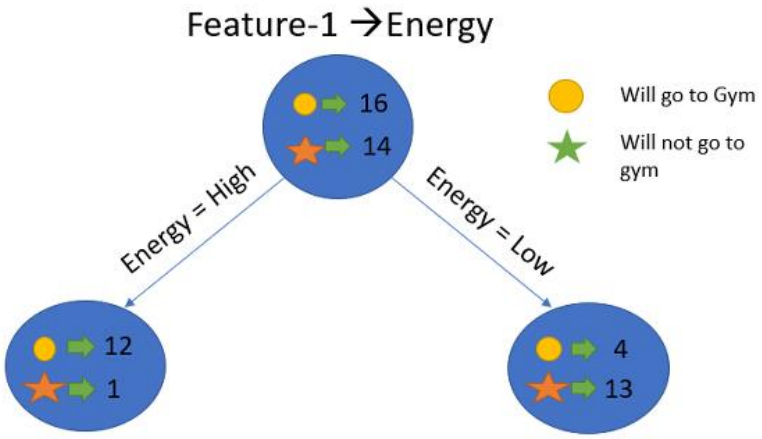
To understand this better let's consider an example: Suppose our entire population has a total of 30 instances. The dataset is to predict whether the person will go to the gym or not. Let's say 16 people go to the gym and 14 people don't

Now we have two features to predict whether he/she will go to the gym or not.

Feature 1 is "Energy" which takes two values *"high" and "low"*

Feature 2 is "Motivation" which takes 3 values *"No motivation", "Neutral" and "Highly motivated".*

Let's see how our decision tree will be made using these 2 features. We'll use information gain to decide which feature should be the root node and which feature should be placed after the split.



Let's calculate the entropy

$$E(Parent) = -\left(\frac{16}{30}\right)\log_2\left(\frac{16}{30}\right) - \left(\frac{14}{30}\right)\log_2\left(\frac{14}{30}\right) \approx 0.99$$

$$E(Parent|Energy = "high") = -\left(\frac{12}{13}\right)\log_2\left(\frac{12}{13}\right) - \left(\frac{1}{13}\right)\log_2\left(\frac{1}{13}\right) \approx 0.39$$

$$E(Parent|Energy = "low") = -\left(\frac{4}{17}\right)\log_2\left(\frac{4}{17}\right) - \left(\frac{13}{17}\right)\log_2\left(\frac{13}{17}\right) \approx 0.79$$

To see the weighted average of entropy of each node we will do as follows:

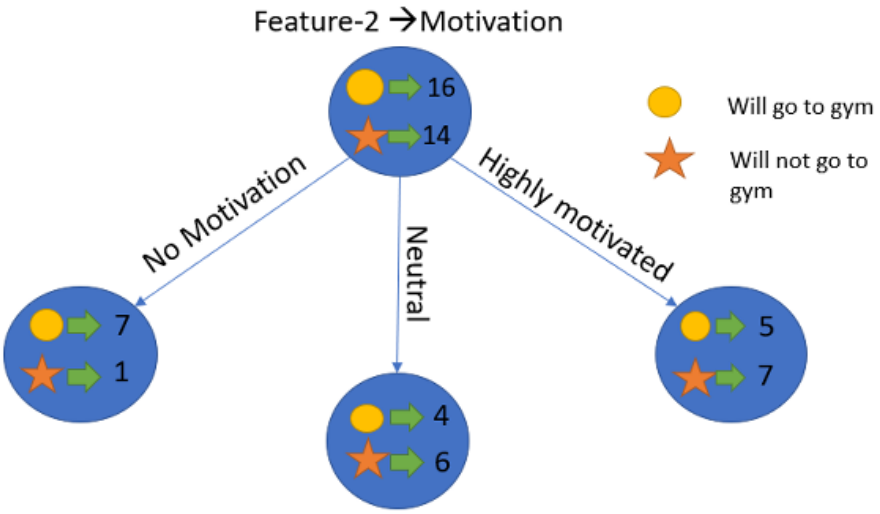
$$E(Parent|Energy) = \frac{13}{30} * 0.39 + \frac{17}{30} * 0.79 = 0.62$$

Now we have the value of E(Parent) and E(Parent|Energy), information gain will be:

$$\begin{aligned} Information\ Gain &= E(parent) - E(parent|energy) \\ &= 0.99 - 0.62 \\ &= 0.37 \end{aligned}$$

Our parent entropy was near 0.99 and after looking at this value of information gain, we can say that the entropy of the dataset will decrease by 0.37 if we make "Energy" as our root node.

Similarly, we will do this with the other feature "Motivation" and calculate its information gain.



Let's calculate the entropy here:

$$E(Parent) = 0.99$$

$$E\Big(Parent|Motivation = "No\ motivation"\Big) = -\left(\frac{7}{8}\right)log_2\left(\frac{7}{8}\right) - \frac{1}{8}\log_2\left(\frac{1}{8}\right) = 0.54$$

$$E(Parent|Motivation = "Neutral") = -\left(\frac{4}{10}\right)log_2\left(\frac{4}{10}\right) - (\frac{6}{10})\log_2\left(\frac{6}{10}\right) = 0.97$$

$$E(Parent|Motivation = "Highly\ motivated") = -\left(\frac{5}{12}\right)log_2\left(\frac{5}{12}\right) - (\frac{7}{12})\log_2\left(\frac{7}{12}\right) = 0.98$$

To see the weighted average of entropy of each node we will do as follows:

$$E(Parent|Motivation) = \frac{8}{30} * 0.54 + \frac{10}{30} * 0.97 + \frac{12}{30} * 0.98 = 0.86$$

Now we have the value of E(Parent) and E(Parent|Motivation), information gain will be:

$$\begin{aligned} Information\ Gain &= E(Parent) - E(Parent|Motivation) \\ &= 0.99 - 0.86 \\ &= 0.13 \end{aligned}$$

We now see that the "Energy" feature gives more reduction which is 0.37 than the "Motivation" feature. Hence we will select the feature which has the highest information gain and then split the node based on that feature.

In this example "Energy" will be our root node and we'll do the same for sub-nodes. Here we can see that when the energy is "high" the entropy is low

and hence we can say a person will definitely go to the gym if he has high energy, but what if the energy is low? We will again split the node based on the new feature which is "Motivation".

When to Stop Splitting?

You must be asking this question to yourself that when do we stop growing our tree? Usually, real-world datasets have a large number of features, which will result in a large number of splits, which in turn gives a huge tree. Such trees take time to build and can lead to overfitting. *That means the tree will give very good accuracy on the training dataset but will give bad accuracy in test data.*

There are many ways to tackle this problem through hyperparameter tuning. We can set the maximum depth of our decision tree using the max_depth parameter. The more the value of max_depth, the more complex your tree will be. The training error will off-course decrease if we increase the max_depth value but when our test data comes into the picture, we will get a very bad accuracy. Hence you need a value that will not overfit as well as underfit our data and for this, you can use GridSearchCV.

Another way is to set the minimum number of samples for each spilt. It is denoted by $min\_samples\_split$. Here we specify the minimum number of samples required to do a spilt. For example, we can use a minimum of 10 samples to reach a decision. That means if a node has less than 10 samples then using this parameter, we can stop the further splitting of this node and make it a leaf node.

There are more hyperparameters such as :

- $min\_samples\_leaf$ – represents the minimum number of samples required to be in the leaf node. The more you increase the number, the more is the possibility of overfitting.
- $max\_features$ – it helps us decide what number of features to consider when looking for the best split.

To read more about these hyperparameters you can read it here.

Pruning

Pruning is another method that can help us avoid overfitting. It helps in improving the performance of the tree by cutting the nodes or sub-nodes which are not significant. Additionally, it removes the branches which have very low importance.

There are mainly 2 ways for pruning:

- Pre-pruning – we can stop growing the tree earlier, which means we can prune/remove/cut a node if it has low importance while growing the tree.
- Post-pruning – once our tree is built to its depth, we can start pruning the nodes based on their significance.

Random Forest algorithm

The *Random Forest* algorithm is a popular machine learning technique that is used for both classification and regression tasks. Popular decision tree algorithms include *Random Forest*, ID3, C4.5 and CART. *Random Forest* is considered one of the best algorithms as it combines multiple decision trees to improve accuracy and reduce overfitting. It is an ensemble learning method that combines multiple decision trees to make predictions.

Here's a simple explanation of how the Random Forest algorithm works:

1. Data Preparation: The algorithm requires a labeled dataset, where the input features (attributes) are used to predict a target variable. The dataset is divided into a training set and a testing set.

2. Building Decision Trees: Random Forest creates a collection of decision trees. Each decision tree is built using a random subset of the training data. This random subset is known as a bootstrap sample. Additionally, at each node of the decision tree, only a random subset of the features is considered for splitting the data.

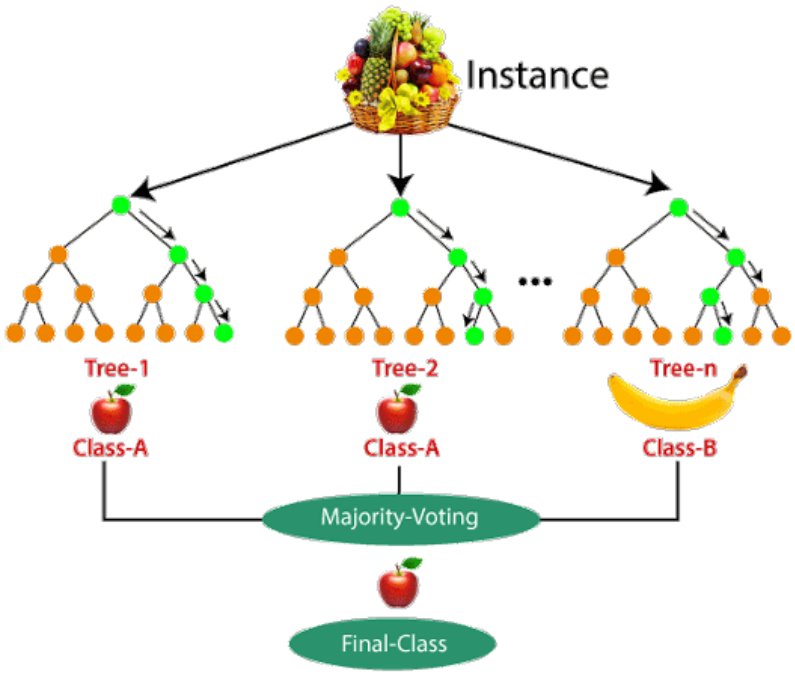
3. Voting Mechanism: To make a prediction, the Random Forest algorithm combines the predictions of all the decision trees. For classification tasks, the most common class predicted by the trees is chosen as the final prediction. For regression tasks, the average of all the predictions is taken.

4. Bagging and Randomness: Random Forest introduces two key concepts - bagging and randomness. Bagging refers to the process of creating multiple decision trees on different subsets of the data. Randomness is introduced by considering random subsets of features at each node, which helps to reduce overfitting and improve the model's generalization.

5. Predictions and Evaluation: Once the *Random Forest* model is trained, it can be used to make predictions on the testing data. The accuracy of the predictions can be evaluated using various performance metrics such as accuracy, precision, recall, or mean squared error, depending on the problem type.

Some key advantages of the Random Forest algorithm include its ability to handle large datasets with high dimensionality, robustness against overfitting, and good performance in both classification and regression tasks. However, it may not be as interpretable as individual decision trees.

For example: consider the fruit basket as the data as shown in the figure below. Now n number of samples are taken from the fruit basket, and an individual decision tree is constructed for each sample. Each decision tree will generate an output, as shown in the figure. The final output is considered based on majority voting. In the below figure, you can see that the majority decision tree gives output as an apple when compared to a banana, so the final output is taken as an apple.



Important Features of Random Forest

- Diversity: Not all attributes/variables/features are considered while making an individual tree; each tree is different.
- Immune to the curse of dimensionality: Since each tree does not consider all the features, the feature space is reduced.
- Parallelization: Each tree is created independently out of different data and attributes. This means we can fully use the CPU to build random forests.
- Stability: Stability arises because the result is based on majority voting/ averaging.