Introduction to Decision Trees

With high interpretability and intuitive nature, decision trees mimic the human decision-making process and also excel in dealing with categorical as well as continuous data. Using decision trees, you can easily explain all the factors or rules leading to a particular decision/prediction. Hence, decision trees are easily understood by people.

So far, you have learnt about linear and logistic regression for classification and regression problems and are able to make predictions using these models. So, what is the need for tree models? Aren't linear models enough for this purpose?

There are certain cases where you cannot directly apply linear regression to solve a regression problem. Linear regression fits a linear relationship between the target and feature variables; however, decision trees internally make multiple subsets based on rules to handle such cases of non-linearity where linear regression cannot operate well.

The advantages of tree models over linear models are as follows:

- Predictions made by a decision tree are easily interpretable.
- A decision tree is versatile in nature. It does not assume anything specific about the nature of the attributes in a data set. It can seamlessly handle all kinds of data such as numeric, categorical, strings, Boolean, etc.
- A decision tree is scale-invariant. It does not require normalisation, as it only has to compare the values within an attribute, and it handles multicollinearity better.
- Decision trees often give us an idea of the relative importance of the explanatory attributes that are used for prediction.
- They are highly efficient and fast algorithms.
- They can identify complex relationships and work well in certain cases where you cannot fit a single linear relationship between the target and feature variables.

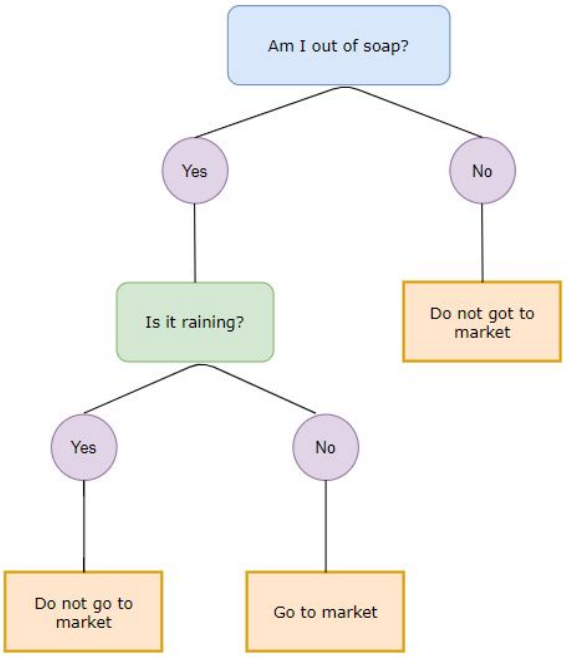
Decision Trees: Introduction

Decision Trees naturally represent the way we make decisions. Think of a machine learning model as a decision-making engine that takes a decision on any given input object (data point). Imagine a doctor making a decision (the diagnosis) on whether a patient is suffering from a particular condition given the patient data, an insurance company making a decision on how much claims on a particular insurance policy needs to be paid out given the policy and the claim data, a company deciding on which role an applicant seeking a position in the company is eligible to apply for, based on the past track record and other details of the applicant, a real estate company aiming to optimise the selling price of the properties, based on important factors such as area, bedrooms, parking, etc. Solutions to each of these can be thought of as machine learning models trying to mimic the human decision making.

Solutions to each of these can be thought of as machine learning models trying to mimic the human decision making.

A decision tree is a flowchart-like structure that helps in making decisions/predictions. It is a predictive model that resembles an upside-down tree. It is a supervised learning method, i.e., it has a fixed target variable, but unlike logistic Regression, it is not parametric. They are free to learn any functional form from the training data and do not have a fixed set of parameters to define the model.

Decision trees can be considered a set of if-then-else statements. In other words, the process of making decisions involves asking questions with two or more possible outcomes. Let's understand this decision-making process with the help of a simple decision tree example shown in the image given below.



As you already know, variables are of two types: categorical and continuous. One of the major advantages of using a decision tree is that it can handle both categorical and continuous variables. Another advantage of decision trees is that they can be used for both classification and regression tasks. The two types of decision trees are as follows:

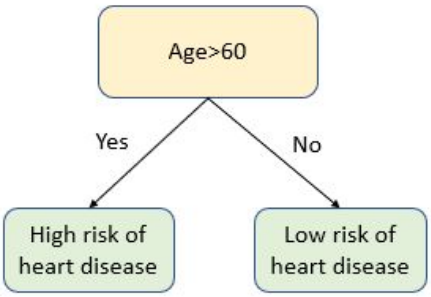
1. Classification decision trees
2. Regression decision trees

Components of a Decision Tree

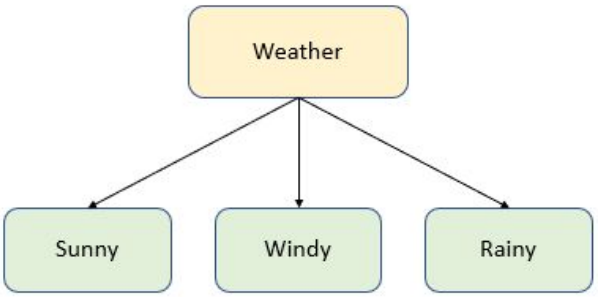
The other two types of decision trees based on the split-criteria are as follows:

1. **Binary decision tree:** The test splits variables into exactly two partitions.
2. **Multiway decision tree:** The test splits variables into more than two partitions.

Take a look at the image given below to understand these decision trees better.

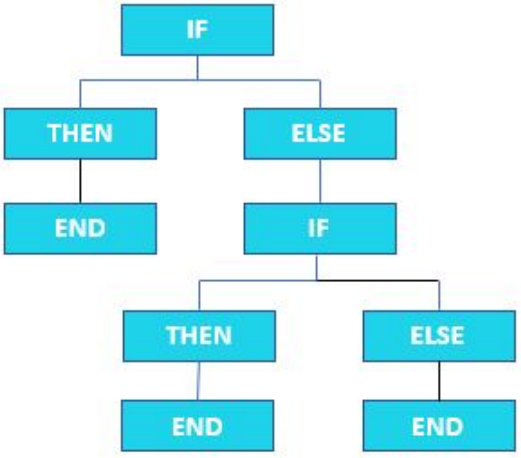


Binary Decision Tree

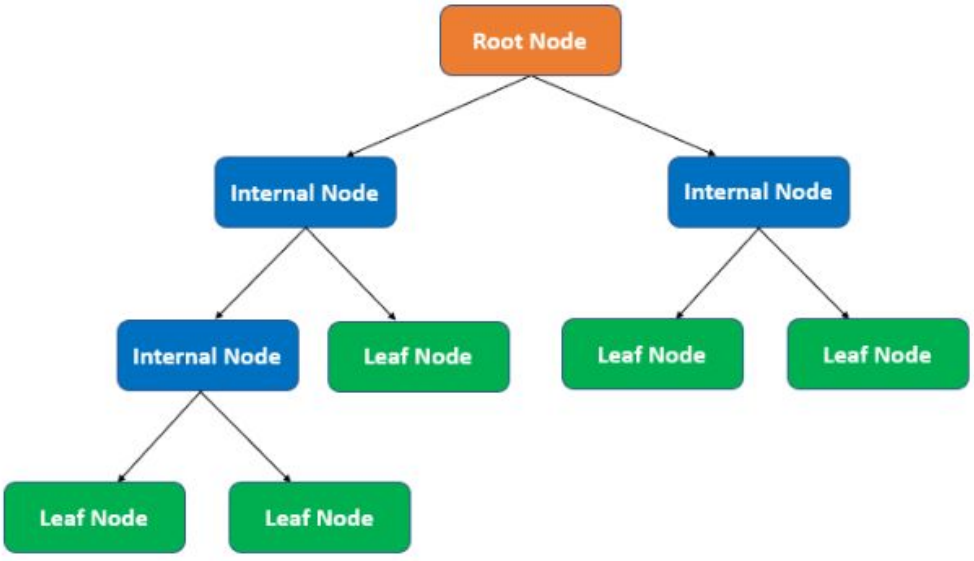


Multi-way Decision Tree

In order to construct a decision tree, you need to be aware of the basic terminologies related to decision trees. Now that you have been introduced to the problem statement and have visualised the decision tree, try to compare the decision tree structure shown in the image given above with the if-then-else statement as shown in the figure below.



The image given below shows the various components of a decision tree.



The various components of a decision tree are as follows:

1. **Root node:** This is the starting/first node of a decision tree. It is further split into different nodes to form a tree. The arrows in a decision tree always point away from this node.

2. **Leaf/Terminal node:** The node that does not split into more nodes is known as a leaf node or a terminal node. The arrows in a decision tree always point towards this node.

3. **Internal node**: The intermediate nodes between the root and the leaf nodes are called the internal nodes.

4. **Branch/Subtree:** A subsection of the tree is called a branch or a subtree.

5. **Parent and child nodes:** A node that is divided into subnodes is called a parent node, and these subnodes can be referred to as the child nodes as they are children of the parent node.

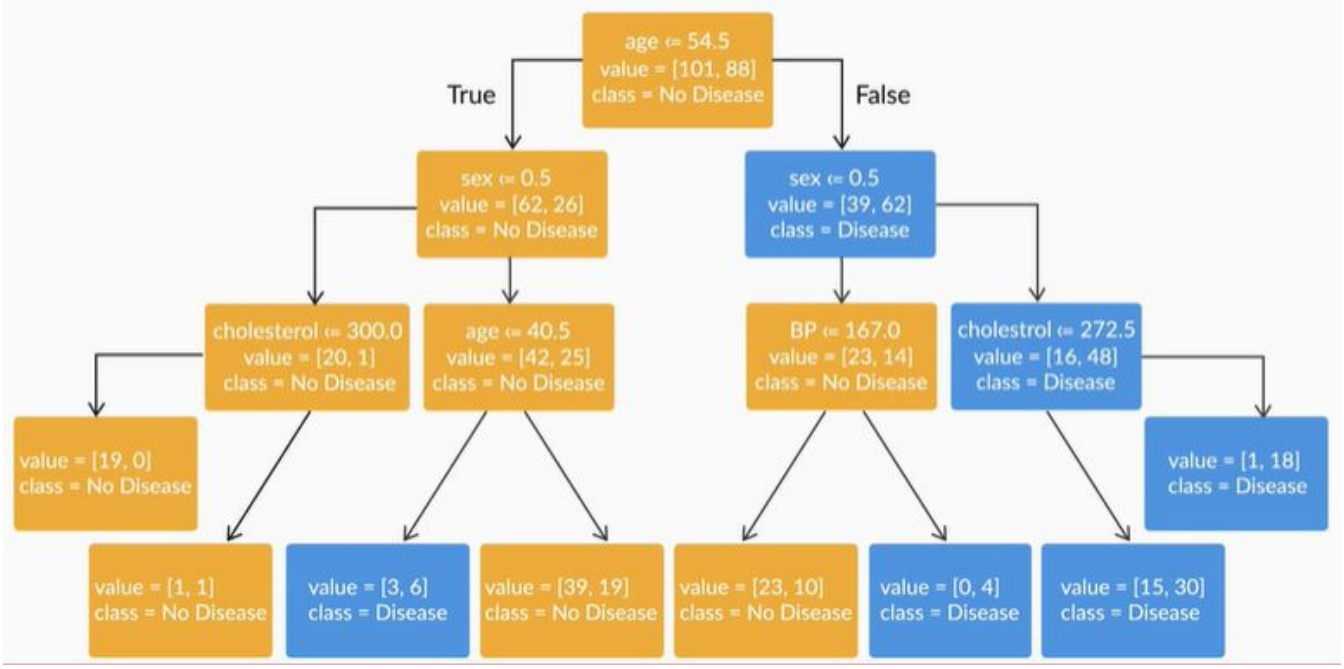
Interpreting Decision Trees

In a decision tree, you start from the top (root node) and traverse left/right according to the result of the condition. Each new condition adds to the previous condition with a logical 'and', and you may continue to traverse further until you reach the final condition's leaf node. A decision is the value (class/quantity) that is assigned to the leaf node.

A decision tree is nothing but a tree asking a series of questions to arrive at a prediction. Suppose you want to predict whether a person has heart disease or not. Based on the values of various attributes such as gender, age, cholesterol, the decision trees try to make a prediction and output a flowchart-like diagram.

Now, if you were a doctor, you could ask these series of questions, and depending on the answers, you can probably make an educated prediction of whether the patient has the disease or not. This prediction here will obviously be based on your past learnings and experiences of treating such patients. A decision tree does the same thing - on the training data, it checks how the patients are doing based on their different attributes (this acts as the algorithm's experience) and based on that experience, it asks the users a series of questions to predict whether a person has heart disease or not.

It is easy to interpret a decision tree, and you can almost always identify the various factors that lead to a particular decision. In fact, trees are often underestimated in their ability to relate the predictor variables to their predictions. As a rule of thumb, if interpretability by layman is what you are looking for in a model, then decision trees should be at the top of your list.



As depicted in the heart disease example in the image above, the leaf nodes (bottom) are labelled 'Disease' (indicating that the person has heart disease) or 'No Disease' (which means the person does not have heart disease).

Note that the splits are effectively partitioning the data into different groups with similar chances of heart disease.

So, in decision trees, you can traverse the attributes backwards and identify the factors that lead to a particular decision.

In the heart disease example, the decision tree predicts that if the 'age' of a person is less than or equal to 54.5, the person is female, and her cholesterol level is less than or equal to 300, then the person will not have heart disease, i.e., young females with a cholesterol level <= 300 have a low chance of being diagnosed with heart disease.

Similarly, there are other paths that lead to a leaf being labelled Disease/No Disease. In other words, each decision is reached via a path that can be expressed as a series of 'if' and logical 'and' conditions that are satisfied together. The final decisions are stored in the form of class labels in leaves.

Building Decision Trees

Constructing a decision tree involves the following steps:

- Recursive binary splitting/partitioning the data into smaller subsets
- Selecting the best rule from a variable/ attribute for the split
- Applying the split based on the rules obtained from the attributes
- Repeating the process for the subsets obtained
- Continuing the process until the stopping criterion is reached
- Assigning the majority class/average value as the prediction

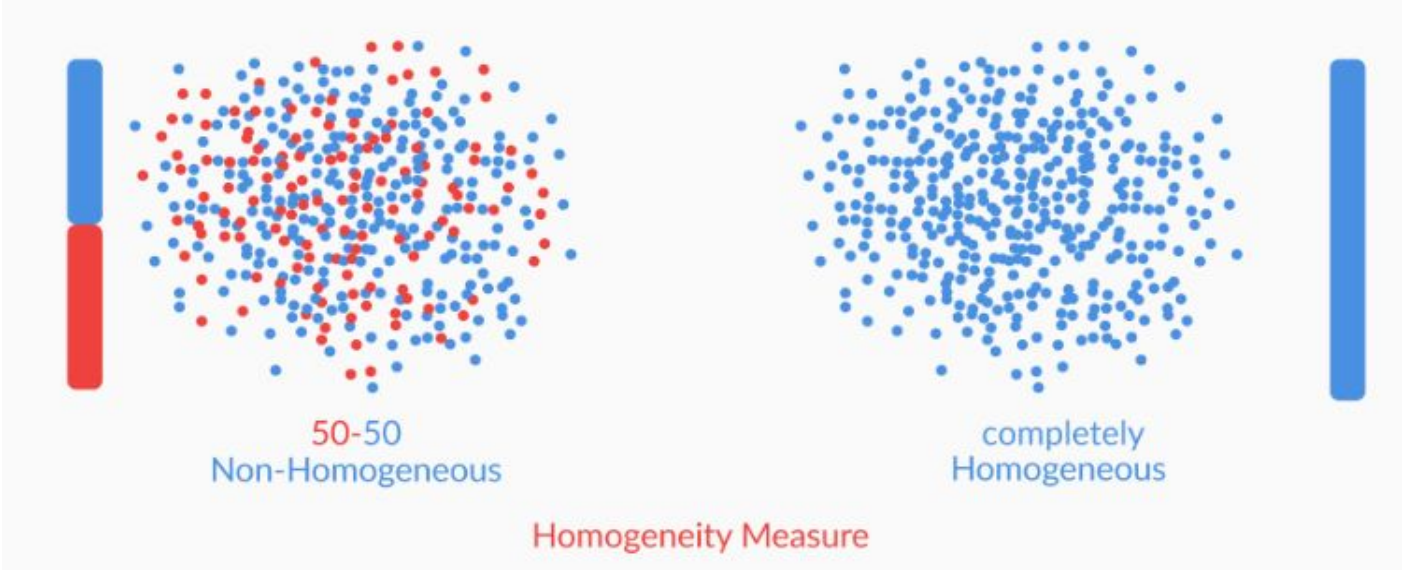
The decision tree building process is a top-down approach. The top-down approach refers to the process of starting from the top with the whole data and gradually splitting the data into smaller subsets.

The reason we call the process greedy is because it does not take into account what will happen in the next two or three steps. The entire structure of the tree changes with small variations in the input data. This, in turn, changes the way you split and the final decisions altogether. This means that the process is not holistic in nature, as it only aims to gain an immediate result that is derived after splitting the data at a particular node based on a certain rule of the attribute.

Splitting and Homogeneity

In order to construct a decision tree, you must know how to select the node that will lead to the best possible solution. Homogeneity/Purity is one of the major factors while constructing a decision tree.

For classification tasks, a data set is considered completely homogeneous if it contains only a single class label, which is extremely difficult to achieve in a real-world data set. So, try to do the best you can, i.e., try to split the nodes such that the resulting nodes are as homogenous as possible. Take a look at the image given below for a better understanding of the concept of homogeneity.



While creating a decision tree, you should follow a step-by-step approach by picking an attribute first and then splitting the data such that the homogeneity of the child nodes increase after every split. You should stop splitting when the resulting leaves are sufficiently homogenous. For this, you need to define the degree of homogeneity, and when the tree achieves this degree of homogeneity, you should stop splitting the data further. A split that would give you a homogenous subset is much more desirable than one that would result in a 50-50 distribution (as in the case of two labels). Note that in a completely homogeneous data set, all the data points belong to one label.

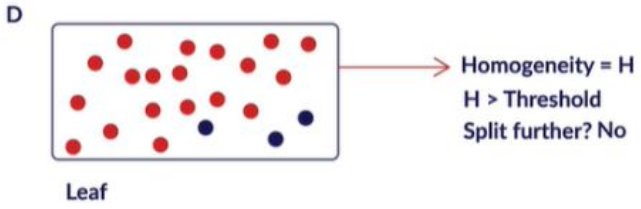
If a partition contains data points with identical labels (for example, label 1), then you can classify the entire partition as that particular label (label 1). However, this is an oversimplified example. In real-world data sets, you will almost never have completely homogenous data sets (or even nodes) after splitting the data. Hence, it is important that you

try to split the nodes such that the resulting nodes are as homogenous as possible. One important thing to remember is that homogeneity here is always referred to in terms of the response (target) variable's homogeneity.

Let's take a look at the illustration given below to further understand homogeneity.

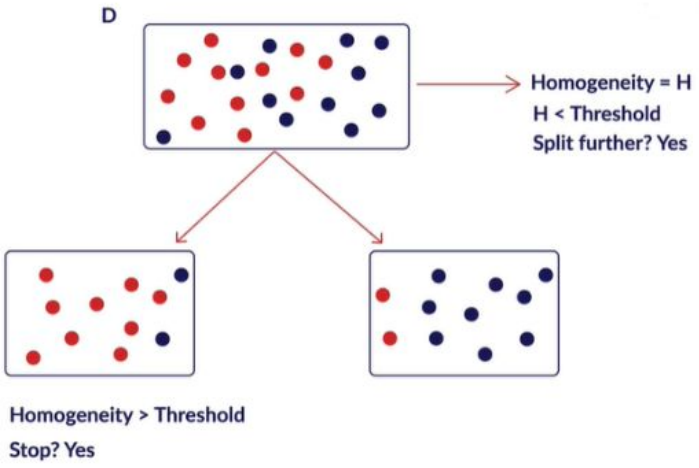
Consider a data set 'D' with homogeneity 'H' and a defined threshold value. When homogeneity exceeds the threshold value, you need to stop splitting the node and assign the prediction to it. As this node does not need further splitting, it becomes the leaf node.

Suppose that you keep the threshold value as 70%. The homogeneity of the node in the illustration given below is clearly above 70% (~86%). The homogeneity value, in this case, falls above the threshold limit. Hence no splitting is required and this node becomes a leaf node.



But what do you do when the homogeneity 'H' is less than the threshold value?

Now keeping the same threshold value as 70%, you can see from the below illustration that homogeneity of the first node is exactly 50%. The node contains an equal number of data points from both the class labels. Hence, you cannot give a prediction to this node as it does not meet the passing criterion which has been set. There is still some ambiguity existing in this node as there is no clarity on the label that can be assigned as the final prediction. This brings in the need for further splitting and we compare the values of homogeneity and threshold again to arrive at a decision.



Till the homogeneity 'H' is less than the threshold, you need to continue splitting the node. The process of splitting needs to be continued until homogeneity exceeds the threshold value and the majority data points in the node are of the same class.

Now, how do you handle best split for different attributes in a decision tree using the CART algorithm?
CART stands for classification and regression decision trees. They are quite similar to the human decision-making process and are, hence, easy to understand. However, the most attractive feature of this algorithm is the flexibility of input variables, i.e., it handles both categorical and continuous variables.

A tree can be split based on different rules of an attribute and these attributes can be categorical or continuous in nature. If an attribute is nominal categorical, then there are $2^{k-1}-1$ possible splits for this attribute, where k is the number of classes. In this case, each possible subset of categories is examined to determine the best split.

If an attribute is ordinal categorical or continuous in nature with n different values, there are n - 1 different possible splits for it. Each value of the attribute is sorted from the smallest to the largest and candidate splits based on the individual values are examined to determine the best split point which maximizes the homogeneity at a node.

There are various other techniques like calculating percentiles and midpoints of the sorted values for handling and getting rules from continuous features for splitting in different algorithms and this process is known as discretization.

Building a Decision Tree: Classificaion

There are various methods used to quantify homogeneity, such as the classification error, Gini index and entropy.

The classification error is calculated as follows:

$$E = 1 - max(p_i)$$

In practice, classification error does not perform well. So, we generally prefer using either the Gini index or entropy over it.

While using CART, the measure used for calculating the information gain for categorical target variables is the **Gini Index**. The Gini index measures the number of times a random variable is incorrectly identified. It is calculated using the following formula:

$$G = \sum_{i=1}^{k} p_i(1-p_i)$$

where pi is the probability of finding a point with the label i, and k is the number of classes.

Gini index of 0 indicates that all the data points belong to a single class. Gini index of 0.5 indicates that the data points are equally distributed among the different classes.

Suppose you have a data set with two class labels. If the data set is completely homogeneous, i.e., all the data points belong to label 1, then the probability of finding a data point corresponding to label 2 will be 0 and that of label 1 will

be 1. So, p1 = 1 and p2 = 0. The Gini index, which is equal to 0, will be the lowest in such a case. Hence, the higher the homogeneity, the lower the Gini index.

CART is an algorithm for building decision trees with the Gini index as a splitting criterion. It is a binary tree that can be built by splitting nodes into two child nodes repeatedly. You can refer to the following steps for building a decision tree using the CART algorithm:

1. Calculate the Gini index before splitting on the whole data set.
2. Consider any one of the available attributes.
3. Calculate the Gini index after splitting on that particular attribute for each of the levels of the attribute.
4. Combine the Gini index of all the levels to obtain the Gini index of the overall attribute.
5. Repeat steps 2–5 with another attribute until you have exhausted all of them.
6. Compare the Gini index across all the attributes and select the one that has the minimum Gini index.

Let's go ahead and see how Gini index can be used to decide where to make the split on the data point using an example.

Suppose you have a dataset for 100 patients and the target variable consists of two classes: class 0 having 60 people with no heart disease and class 1 having 40 people with a heart disease with two attributes i.e. 'sex' and 'cholesterol'. While making your first split, you need to choose an attribute such that the purity gain is maximum. You can calculate the Gini index of the split on 'sex' (gender) and compare that with the Gini index of the split based on 'cholesterol'.

No disease : 60 (Class 0)
Disease : 40 (Class 1)

Expressing this in terms of probabilities you get:

$$p_0 = \frac{60}{60+40} = 0.6 \qquad p_1 = \frac{40}{60+40} = 0.4$$

Now, you can calculate the gini index for the data before making any splits as follows:

Gini Impurity before split:

$$p_0(1-p_0) + p_1(1-p_1) = 0.6(1-0.6) + 0.4(1-0.4) = 0.48$$

Let's now evaluate which split gives the maximum reduction in impurity among the possible choices. You have the following information about the target variable and the two attributes.

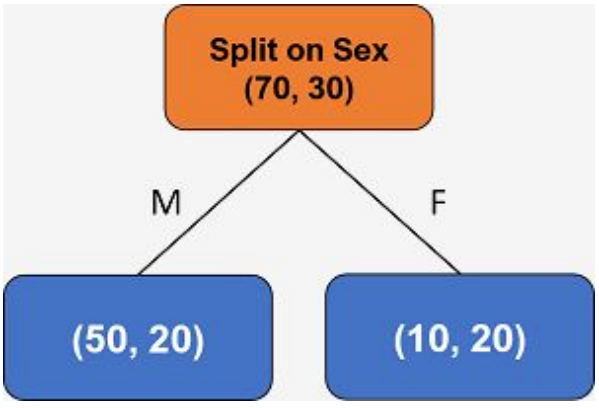
Features / Classes	Sex		Total
	M	F	
No Disease	50	10	60
Disease	20	20	40
Total	70	30	100

Features / Classes	Cholesterol		Total
	< 250	> 250	
No Disease	50	10	60
Disease	10	30	40
Total	60	40	100

As you can see, the table above shows the number of diseased/non-diseased people w.r.t. the levels in the two attributes - 'Sex' and 'Cholesterol'. Let's calculate the homogeneity reduction on each attribute individually, starting with 'Sex'.

Split based on Sex

Let's consider the first candidate split based on sex/gender. As you can see from the first table, of the 100 people, you have 70 males and 30 females. Among the 70 males i.e. the child node containing males, **50 belong to class 0** i.e, they do not have a heart disease and the rest **20 males belong to class 1** having a heart disease. So basically for the split on "Sex", you have something like this —



[Note that (x, y) on any node means (# Label 0, # Label 1)]

Now the probabilities of the two classes within the male subset comes out to be:

$$p0=50/70=0.714 \quad \text{and} \quad p1=20/70=0.286$$

Now using the same formula, Gini impurity for males becomes:

$$0.714(1-0.714)+0.286(1-0.286)=0.41$$

Let's now take the other case i.e. the child node containing females, where there are 30 females out of which 10 belong to class 0 having no heart disease and 20 belong to class 1 having a heart disease. The probabilities of the two classes within the female subset comes out to be:

$$p0=10/30=0.333 \quad \text{and} \quad p1=20/30=0.667$$

Now using the formula, Gini impurity for females becomes:

$$0.333(1-0.333)+0.667(1-0.667)=0.44$$

Now how do you get the overall impurity for the attribute 'sex' after the split? You can aggregate the Gini impurity of these two nodes by taking a weighted average of the impurities of the male and female nodes. So, you have -

$$pmale=70/100=0.7 \quad \text{and} \quad pfemale=30/100=0.3$$

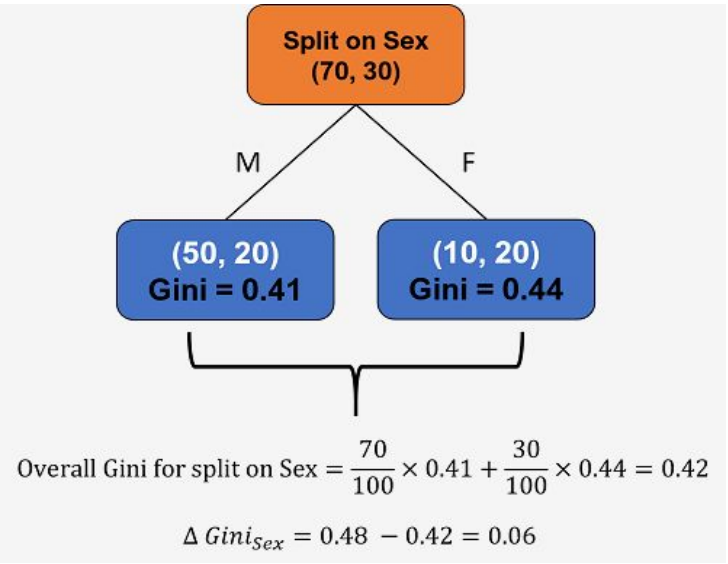
This gives the Gini impurity after the split based on gender as:

$$0.7\times0.41+0.3\times0.44=0.42$$

Thus, the split based on gender gives the following insights:

- Gini impurity before split = 0.48
- Gini impurity after split = 0.42
- **Reduction in Gini impurity = 0.48 - 0.42 = 0.06**

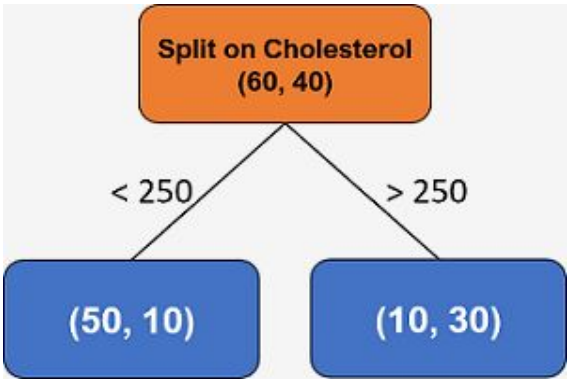
Hence, you get the following tree after splitting on 'Sex' —



Split based on Cholesterol

Let's now take another candidate split based on cholesterol. You divide the dataset into two subsets: Low Cholesterol (Cholesterol < 250) and High Cholesterol (Cholesterol > 250). There are 60 people belonging to the low cholesterol group and 40 people belonging to the high cholesterol group.

If you see the second table given above, you will notice that among the 60 low cholesterol people, 50 belong to class 0, i.e, they do not have a heart disease and the rest 10 belong to class 1 having a heart disease. So basically for the split on "Cholesterol", you have something like this —



Now the probabilities of the two classes within the low cholesterol subset comes out to be:

$$p0=50/60=0.833 \quad \text{and} \quad p1=10/60=0.167$$

Now using the formula, Gini impurity for low cholesterol subset becomes:

$$0.833(1-0.833)+0.167(1-0.167)\approx 0.27$$

Let's now take the other case where there are 40 high cholesterol (Cholesterol > 250) people out of which 10 belong to class 0 having no heart disease and 30 belong to class 1 having a heart disease. The probabilities of the two classes within the high cholesterol subset comes out to be:

$$p0=10/40=0.25 \quad \text{and} \quad p1=30/40=0.75$$

Now using the formula, Gini impurity for high cholesterol subset becomes:

$$0.25(1-0.25)+0.75(1-0.75)\approx 0.37$$

The overall impurity for the data after the split based on cholesterol can be computed by taking a weighted average of the impurities of the high and low cholesterol nodes. So, you have -

$$p_{low-cholesterol}=60/100=0.6 \quad \text{and} \quad p_{high-cholesterol}=40/100=0.4$$

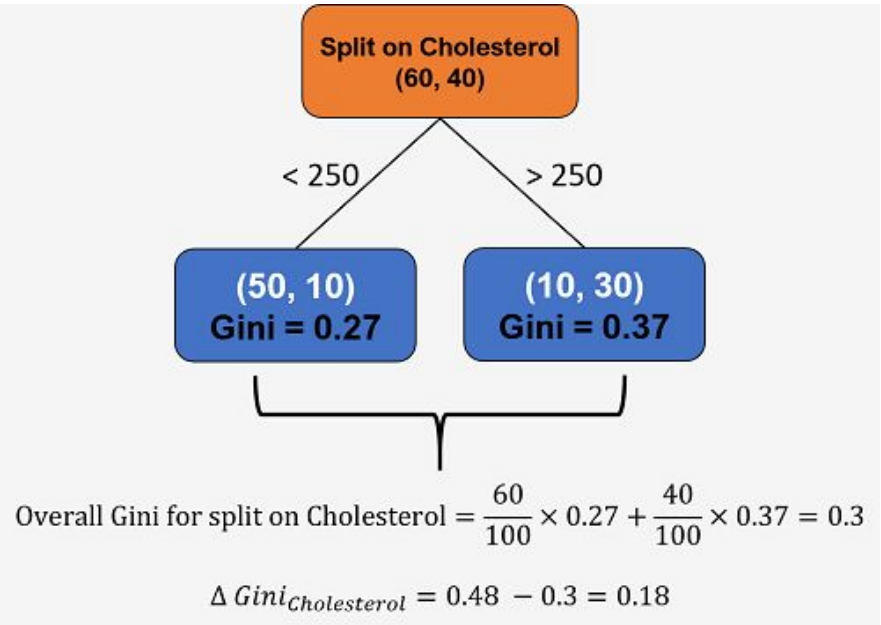
This gives the Gini impurity after the split based on cholesterol as:

$$0.6\times 0.27+0.4\times 0.37\approx 0.3$$

Thus, the split based on cholesterol gives the following insights:

- Gini impurity before split = 0.48
- Gini impurity after split = 0.3
- Reduction in Gini impurity = 0.48 - 0.3 = 0.18

Hence, you get the following tree after splitting on 'Cholesterol' —



Hence, from the above example, it is evident that we get a significantly higher reduction in Gini impurity when you split the dataset on cholesterol as compared to when you split on gender.

While dealing with CART, the Gini index is the splitting criterion for classification trees or categorical variables. However, in the case of continuous variables, the splitting criterion is the MSE (mean squared error). The MSE is not different from variance, which is nothing but the square of standard deviation.

Some additional points and summary of key learnings so far have been included below:

1. A categorical tree is one that has class labels as its leaf nodes, whereas a continuous tree has numerical values as its leaf nodes.

2. CART can only deal with binary trees, whereas ID3 can deal with non-binary trees (trees that have more than two splits) as well.

3. ID3 uses entropy/information gain to measure the impurity of the variable, whereas CART uses the Gini index for categorical target variables.

4. Both the entropy and the Gini index measure the impurity of the variable or the number of times a variable is incorrectly classified. Therefore, your main aim while building the tree is to achieve a reduction in these

values.

5. CART uses MSE/variance for continuous variables.

6. Standard deviation is nothing but the square root of the MSE value or the variance of a variable.

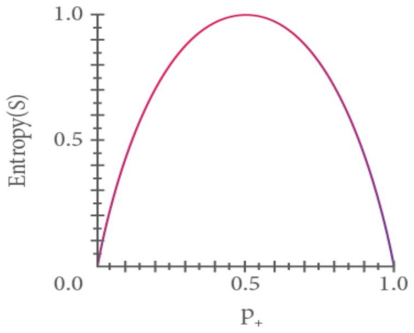
Entropy: Entropy is the measure of the randomness of a variable. Thus, a higher entropy means more randomness, which leads to less purity. Entropy quantifies the degree of disorder in the given data. Its value varies from 0 to 1.

In simple words, a pure data set means that all the data belongs to the same class and the entropy is zero, whereas an impure one means data is divided over different classes and the entropy is greater than zero but less than one. If a data set contains an equal distribution of both the classes, then the entropy of that data set will be 1, i.e., there is complete disorder in the data. Hence, like the Gini index, the higher the homogeneity, the lower the entropy. The entropy of a variable in a given class can be calculated using the following formula:

$$D = -\sum_{i=1}^{k} p_i . log_2(p_i)$$

where pi is the probability of finding a point with the label i, and k is the number of classes.

The graphical representation given below will help you understand the outcomes of the formula:



Some important facts about entropy that can be derived from the formula and the image given above are as follows:

1. If all the variables belong to the same class, the probability is one or zero, the entropy is zero and the purity is maximum.

2. If the variables are equally divided over two classes (as in the case of binary classification), the probability is 0.5 for both the classes, the entropy is 1 and the purity is minimum.

Information gain: This is the main deciding parameter in determining the feature that is to be split at each node. While splitting, the main factor to be kept in mind is purity. The end solution should have maximum purity as compared with other trees that can be created in its place. The measure that you use for purity is information gain.

As the name suggests, information gain calculates the measure of information that you can obtain from a variable. You need to continue the splitting process until the value of information gain is zero or very small. However, in order to do so, you should keep in mind the following assumptions while constructing a tree:

1. When you begin creating the tree, the whole set of training data is considered a root node.

2. A recursive approach is followed while creating the tree.

3. You will use a statistical approach while finding the root node and the internal nodes.

The change in impurity or the purity gain is given by the difference of impurity post-split from impurity pre-split, i.e.,
Δ Impurity = Impurity (pre-split) – Impurity (post-split)

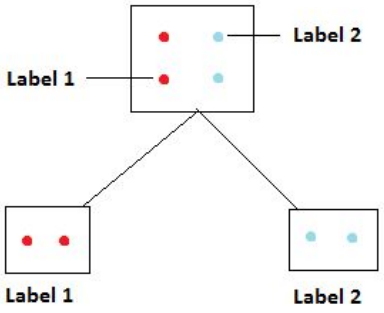
The post-split impurity is calculated by finding the weighted average of two child nodes. The split that results in maximum gain is chosen as the best split.

To summarise, the information gain is calculated by:

$$Gain = D - D_A$$

Where D is the entropy of the parent set (data before splitting),DA is the entropy of the partitions obtained after splitting on attribute A. Note that reduction in entropy implies information gain.

Let's understand how we compute information gain with an example. Suppose you have four data points out of which two belong to the class label '1', and the other two belong to the class label '2'. You split the points such that the left partition has two data points belonging to label '1', and the right partition has the other two data points that belong to label '2'. Now let's assume that you split on some attribute called 'A'.



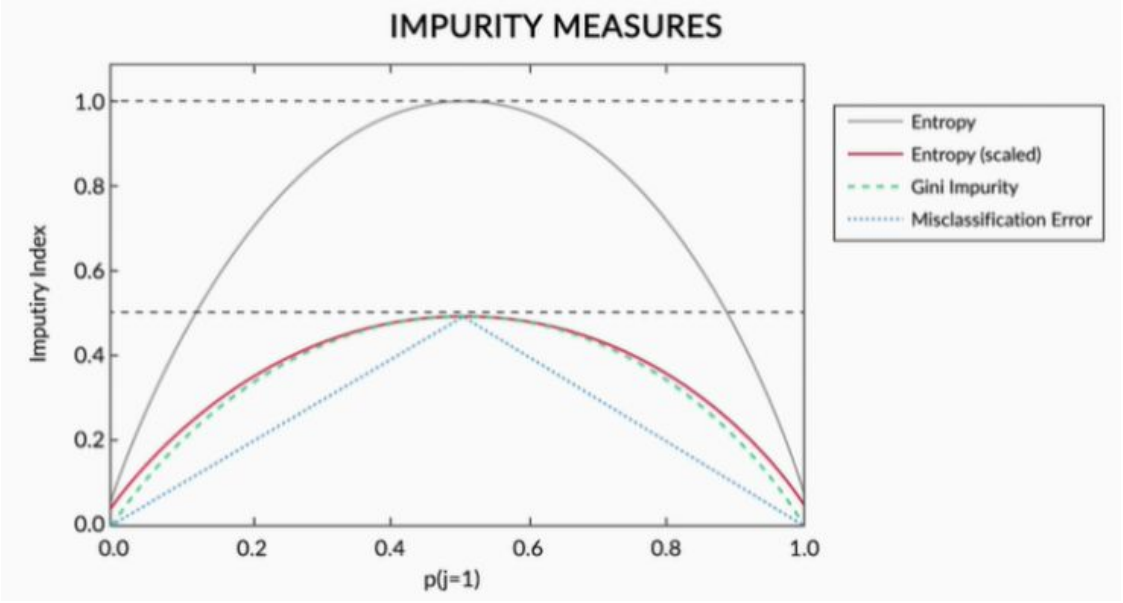
1. Entropy of original/parent data set is $D = -[(\frac{2}{4})log_2(\frac{2}{4}) + (\frac{2}{4})log_2(\frac{2}{4})] = 1.0.$
2. Entropy of the partitions after splitting is $D_A = -0.5 * log_2(\frac{2}{2}) - 0.5 * log_2(\frac{2}{2}) = 0.$
3. Information gain after splitting is $Gain = D - D_A = 1.0.$

So, the information gain after splitting the original data set on attribute 'A' is 1.0. You always try to maximise information gain by achieving maximum homogeneity and this is possible only when the value of entropy decreases from the parent set after splitting.

In case of a classification problem, you always try to maximise purity gain or reduce the impurity at a node after

every split and this process is repeated till you reach the leaf node for the final prediction.

Look at the figure given below to understand the range of values each of the impurity measures can take.



The scaled version of the entropy in the figure is nothing but entropy/2. It has been used to emphasize that the Gini index is an intermediate measure between entropy and the classification error.

Building a Decision Tree: Regression

In regression problems, a decision tree splits the data into multiple subsets. The difference between decision tree classification and decision tree regression is that in regression, each leaf represents the average of all the values as the prediction as opposed to a class label in classification trees. For classification problems, the prediction is assigned to a leaf node using majority voting but for regression, it is done by taking the average value. This average is calculated using the following formula:

$$\hat{y}_t = \frac{1}{N_t} \sum_{i \epsilon D_t} y^{(i)}$$

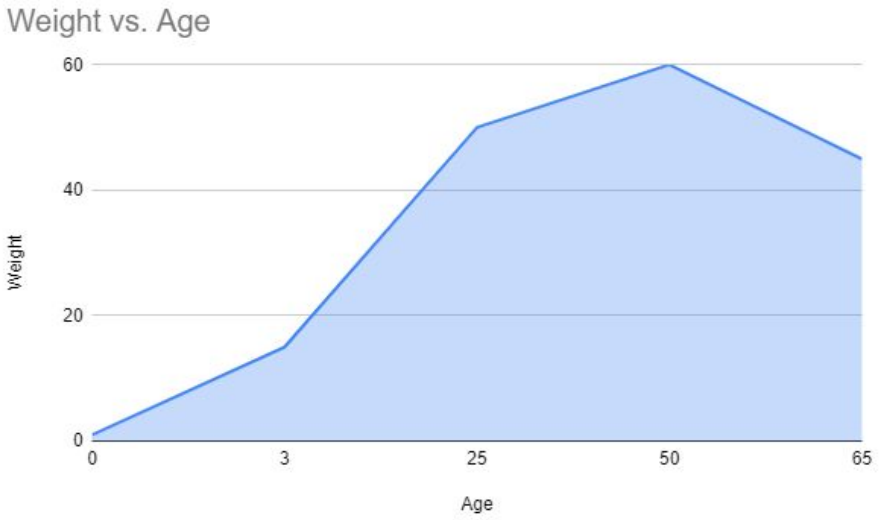
Where yi's represent the observations in a node, Nt refers to the total number of observations and Dt refers to the dataset.

For example, suppose you are predicting the sales your company will have based on various factors such as marketing, no. of products, etc. Now, if you use a decision tree to solve this problem (the process of actually building a regression tree is covered in the next session), and if one of the leaf nodes has, say, 5 data points, 1 Cr, 1.3 Cr, 0.97 Cr, 1.22 Cr, 0.79 Cr. Now, you will just take the average of these five values which comes out to be 1.07 Cr, and that becomes your final prediction.

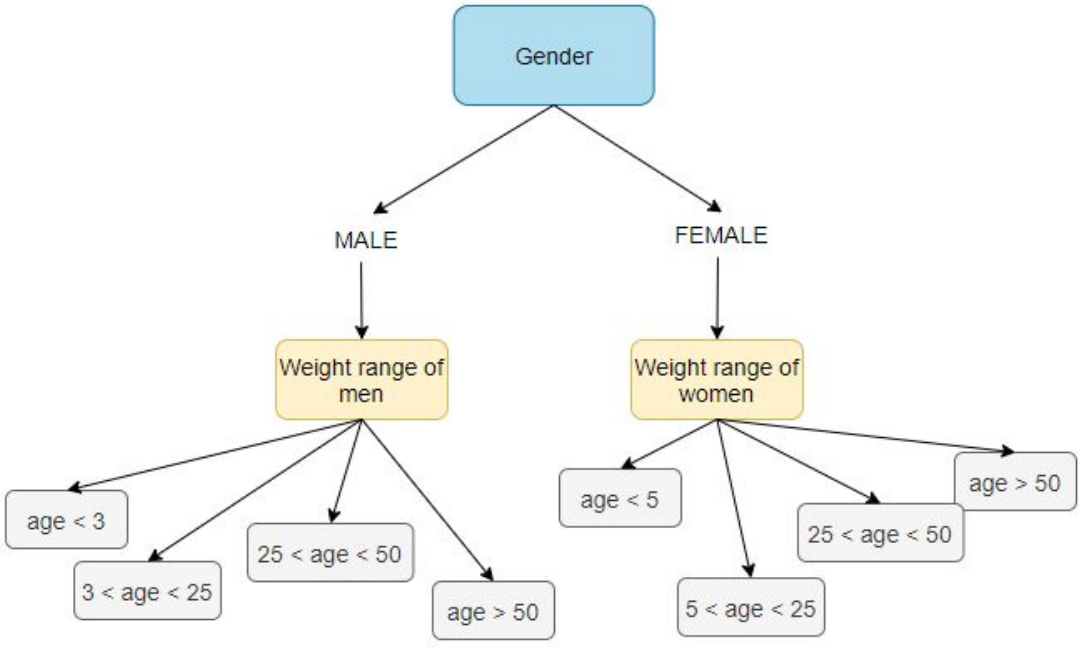
Decision tree classification is what you'll most commonly work on. However, remember that if you get a data set where you want to perform regression, decision tree regression is also a good idea.

Let's take another example to understand regression trees better.

Suppose you want to predict the weight of a person, given their age and height. If you try to plot the age on the x-axis and the height on the y-axis in the form of a linear model, you will get a graph similar to the one shown in the image given below.



For infants (age < 3 years) and people aged above 25 years (age > 25), the growth will not be very rapid, and hence, the slope is not very steep. However, 3–25 years (3 < age < 25) can be considered to be the growing age of a person. So, the steep slope will signify the rapid increase in the weight of an individual. However, a slowdown in the growth rate is seen after the age of 25 (age > 25), and the decrease grows rapidly during old age. As a result, not a single linear model will be able to predict this completely. Hence, you can divide them into different buckets and use a decision tree model as shown in the image given below.



In regression problems, a decision tree splits the data into multiple subsets. In this case, the leaf node consists of continuous values as opposed to the categorical labels in the case of classification problems. While working with a categorical target variable, you used entropy/gini index to calculate the information gain of a feature, but this method does not work for a continuous target variable. In the case of a continuous target variable, the impurity measure for a given node is measured by the weighted mean square error (WMSE), also known as variance, which is calculated by the following formula:

$$MSE(t) = \frac{1}{N_t} \sum_{i \epsilon D_t} (y^{(i)} - \hat{y}_t)^2$$

This is nothing but the variance of all data points.

A higher value of MSE means that the data values are dispersed widely around mean, and a lower value of MSE means that the data values are dispersed closely around mean and this is usually the preferred case while building a regression tree.

The regression tree building process can be summarised as follows:

- Calculate the MSE of the target variable.
- Split the data set based on different rules obtained from the attributes and calculate the MSE for each of these nodes.
- The resulting MSE is subtracted from the MSE before the split. This result is called the MSE reduction.
- The attribute with the largest MSE reduction is chosen for the decision node.
- The dataset is divided based on the values of the selected attribute. This process is run recursively on the non-leaf branches, until you get significantly low MSE and the node becomes as homogeneous as possible.
- Finally, when no further splitting is required, assign this as the leaf node and calculate the average as the final prediction when the number of instances is more than one at a leaf node.

So, you need to split the data such that the weighted MSE of the partitions obtained after splitting is lower than that obtained with the original or parent data set. In other words, the fit of the model should be as 'good' as possible after splitting. As you can see, the process is surprisingly similar to what you did for classification using trees.

Feature Importance in Decision Trees

Feature importance plays a key role in contributing towards effective prediction, decision-making and model performance. It eliminates the less important variables from a large data set and helps in identifying the key features that can lead to better prediction results.

Decision trees help in quantifying the importance of each feature by calculating the reduction in the impurity for each feature at a node. The feature that results in a significant reduction in the impurity is the important variable, and the one that results in less impurity reduction is the less important variable.

Advantages and Disadvantages of Decision Trees

The advantages of decision trees can be summarised as follows:

1. Predictions made by decision trees are easily interpretable.
2. Decision trees work well with both categorical and continuous variables.
3. They can handle both linearly separable and non-separable data.
4. They do not assume anything specific about the nature of the attributes in a data set. They can seamlessly handle all kinds of data, including numeric, categorical, strings and boolean.
5. They do not require the data to be normalised, as they have to compare only the values for splitting within an attribute. Therefore, little or no preprocessing of data is required.
6. Decision trees often give you an idea of the relative importance of the explanatory attributes that are used for prediction. Intuitively, the closer the variable that is used for splitting is to the root node, the higher is the importance.
7. The rules are easily explainable, as decision trees follow a similar approach that human beings generally do while making decisions.
8. The tree-like visualisations of decision trees can be used to simplify a complex model, and even a layman can understand the logic behind the decisions/predictions.

The disadvantages of decision trees can be summarised as follows:

1. Decision trees tend to overfit the data. If allowed to grow with no check on its complexity, a decision tree will keep splitting until it has correctly classified all the data points in the training data set.
2. Decision trees tend to be extremely unstable, which is an implication of overfitting. A few changes in the data can change a tree considerably.
3. The mathematical calculations for entropy and information gain (IG) and for all the features require a lot of time and memory, as you need to perform the split for every feature at every splitting point.
4. Greedy algorithms do not always give a globally optimal model. Decision trees are called greedy because you have to optimise at every split and not overall. The process is not holistic in nature, as it only aims to gain an immediate result that is derived after splitting the data at a particular node based on a certain rule of the attribute. This can be handled well using random forests.

Truncation and Pruning

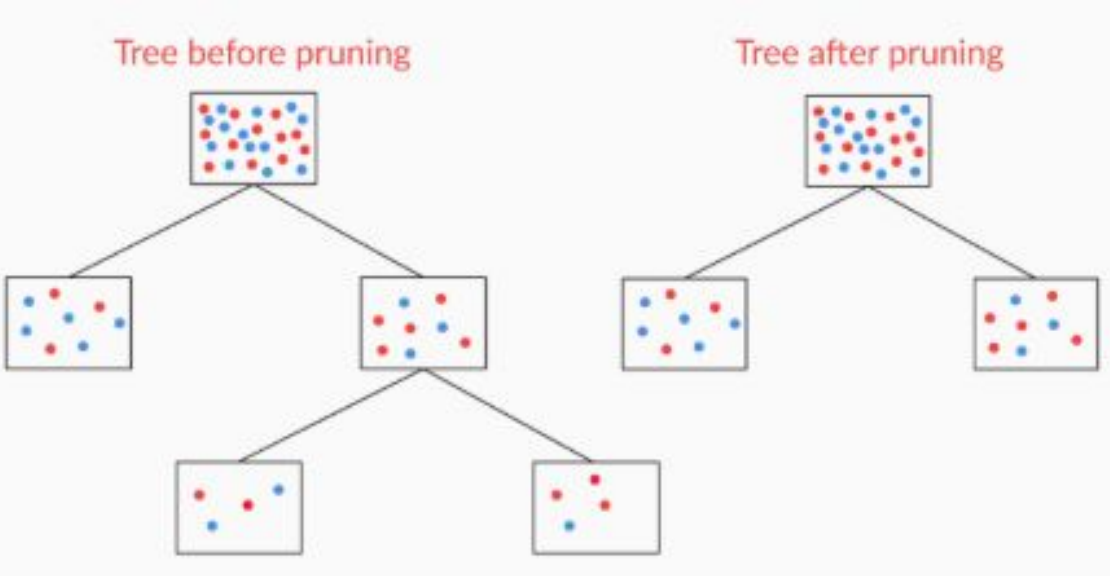
Earlier, you observed that decision trees have a strong tendency to overfit data, which is a major problem. So, you have to pay attention to the size of the tree. A very large and overfitted tree will eventually have leaf nodes with a single data point. Overfitting in decision trees can be controlled using the following two broad strategies:

1. **Truncation**: This process involves stopping the tree while it is still growing so that it does not end up with leaves containing only a few data points. Truncation is also referred to as pre-pruning.

2. **Pruning:** This process involves letting the tree grow to any level of complexity and then cutting the branches of the tree in a bottom-up fashion, starting from the leaves. It is recommended that you use pruning strategies to avoid overfitting in practical implementations.

Pre-pruning or early-stopping is a method of stopping the growth of a decision tree before it overfits. In the case of pre-pruning, the tree is pruned back to the point where the cross-validation error is minimum. On the other hand, post-pruning involves chopping off the branches of a decision tree created from the entire data available. In this case, the tree is pruned slightly farther than the minimum error point. These techniques increase the accuracy of decision trees by reducing overfitting.

Pre-pruning is preferred to post-pruning because it is less rigorous and saves time. Also, it is more practical to stop a tree at a feasible point than creating the whole tree first and then chopping it off. Consider the tree shown in the image given below.



This image shows the versions of a tree 'before' and 'after' pruning. The red dots belong to the class 'Label 1', and the blue dots belong to the class 'Label 2'.

The DecisionTreeClassifier function in the sklearn library provides the following hyperparameters, which you can control in order to prune your trees:

- **criterion (Gini/IG or entropy):** This defines the function to measure the quality of a split. The sklearn library supports the 'gini' criterion for the Gini Index and the 'entropy' criterion for the information gain. By default, it takes the 'gini' value.

- **max_features:** This defines the number of features to be considered while finding the best split. You can input integer, float, string and None values.
    1. If an integer is an input type, then it considers that value as max features at each split.
    2. If a float value is taken, then max_features is a fraction and int(max_features * n_features) features are considered at each split.
    3. If 'auto' or 'sqrt' is taken, then max_features=sqrt(n_features) is considered at each split.
    4. If 'log2' is taken, then max_features= log2(n_features) is considered at each split.
    5. If the 'None' value is taken, then max_features=n_features is considered at each split. By default, it takes the 'None' value.

- **max_depth:** This denotes the maximum depth of a tree. It can take any integer or the ''None'' value. If it takes the ''None'' value, then its nodes are expanded until all the leaves are pure or contain less than min_samples_split samples. By default, it takes the 'None' value.

- **min_samples_split:** This denotes the minimum number of samples that are required to split an internal node. If an integer value is taken, then min_samples_split is considered to be the minimum number. If a float value is taken, then it shows the percentage. By default, it takes the '2' value.

- **min_samples_leaf:** This denotes the minimum number of samples that are required to be at a leaf node. If an integer value is taken, then min_samples_leaf is considered to be the minimum number. If a float value is taken, then it shows the percentage value. By default, it takes the '1' value.

Apart from these, there are other hyperparameters in the DecisionTreeClassifier function. You can read the documentation in Python using the following:

```
help(DecisionTreeClassifier)
```

Disclaimer: All content and material on the upGrad website is copyrighted material, either belonging to upGrad or its bonafide contributors and is purely for the dissemination of education. You are permitted to access, print and download extracts from this site purely for your own education only and on the following basis:

- You can download this document from the website for self-use only.
- Any copy of this document, in part or full, saved to disk or to any other storage medium, may only be used for subsequent, self-viewing purposes or to print an individual extract or copy for non-commercial personal use only.
- Any further dissemination, distribution, reproduction, copying of the content of the document herein or the uploading thereof on other websites or use of the content for any other commercial/unauthorised purposes in any way which could infringe the intellectual property rights of upGrad or its contributors, is strictly prohibited.
- No graphics, images or photographs from any accompanying text in this document will be used separately for unauthorised purposes.
- No material in this document will be modified, adapted or altered in any way.
- No part of this document or upGrad content may be reproduced or stored in any other website or included in any public or private electronic retrieval system or service without upGrad's prior written permission.
- Any right not expressly granted in these terms are reserved.