Classification Using Decision Tree

Module 1 : Introduction to Decision Trees

Introduction:-

Welcome to the module 'Tree Models'. In this module, you will learn about two important machine learning models: **decision trees** and **random forests**.

You will first learn about decision trees and then proceed to learn about random forests, which are a collection of multiple decision trees. A collection of multiple models is called an **ensemble**.

With high interpretability and an intuitive algorithm, decision trees mimic the human decision-making process and are efficient in dealing with categorical data. Unlike other algorithms, such as logistic regression and support vector machines (SVMs), decision trees do not help in finding a linear relationship between the independent variable and the target variable. However, they can be used to **model highly non-linear data**.

You can use decision trees to explain all the factors that lead to a particular decision/prediction. And so can be used in explaining certain business decisions to entrepreneurs. Decision trees form the building blocks for random forests, which are commonly used among the Kaggle community.

Random forests are **collections of multiple trees** and are considered to be one of the most efficient machine learning models. By the end of this module, you should be able to use decision trees and random forests to solve both classification and regression problems.

In this session:

- Introduction to decision trees
- Interpretation of decision trees
- Building decision trees
- Tree models over linear models
- Decision trees for regression problems

Introduction to Decision Trees:-

You have learnt some classical machine learning algorithms like linear regression, logistic regression etc. for solving both regression and classification problems. Then what do you think is the need of going ahead with these linear models. Linear models cannot handle collinearity and non linear relationships in the data well. Now here comes the role of decison trees which leverages these properties. You will learn about each of these in detail as you go ahead.

A **decision tree**, as the term suggests, uses a tree-like model to make predictions. It resembles an upside-down tree and uses a similar process that you do to make decisions in real life, i.e., by asking a series of questions to arrive at a decision.

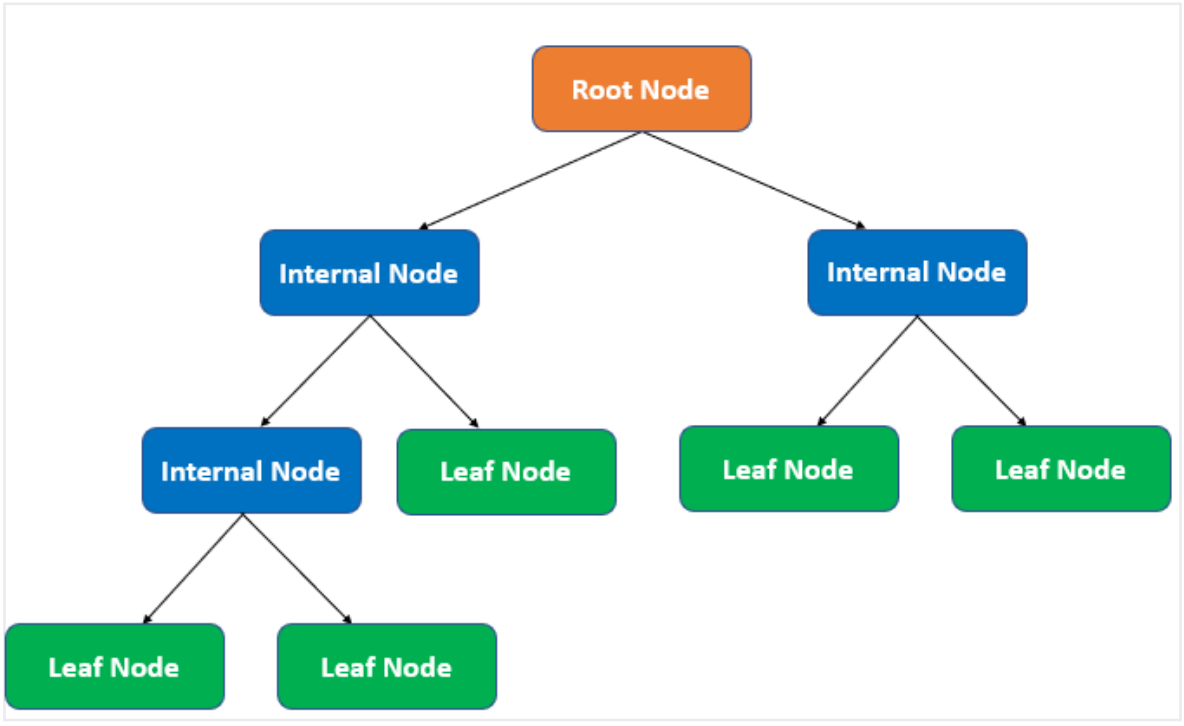
A decision tree splits data into multiple sets of data. Each of these sets is then further split into subsets to arrive at a decision. Let's hear from Prof. Raghavan as he explains this process in detail.

As you saw in this video, a decision tree uses a natural decision-making process, i.e., it asks a series of questions in a nested if-then-else structure. On each node, you ask a question to further split the data that is held by the node. If the test passes, you move to the left; otherwise, you move to the right.

The first and top node of a decision tree is called the **root node**. The arrows in a decision tree always point away from this node.

The node that cannot be further classified or split is called the **leaf node**. The arrows in a decision tree always point towards this node.

Any node that contains descendant nodes and is not a leaf node is called the **internal node**.



In the next segment, you will take a look at some real-life examples to understand decision trees better.

Additional Reading:
Root Node vs Internal Node

Interpreting a Decision Tree:-

Before even jumping to the concepts of decision trees and constructing one on your own in Python, it is important that you first understand how a decision tree is interpreted for you to have a better appreciation of the model building process.

Like we have mentioned earlier as well, a decision tree is nothing but a tree asking a series of questions to arrive at a prediction. The problem at hand is to predict whether a person has heart disease or not. Based on the values that various attributes such as gender, age, cholesterol, the decision trees try to make a prediction and output a flowchart-like diagram. Let's hear Rahim describe this output and how to interpret it.

Please find below the data set used for the construction of decision trees around heart disease.

As you saw, interpreting a decision tree is nothing but asking a series of questions - something similar to what a doctor would do when they are diagnosing their patients. The first question we asked was, "Is the age less than 54.5?". Depending on the answer, we moved to the next step where we asked whether the person is a male or a female, and so on. At the end of this line of question, lies an answer - whether the person has heart disease or not.

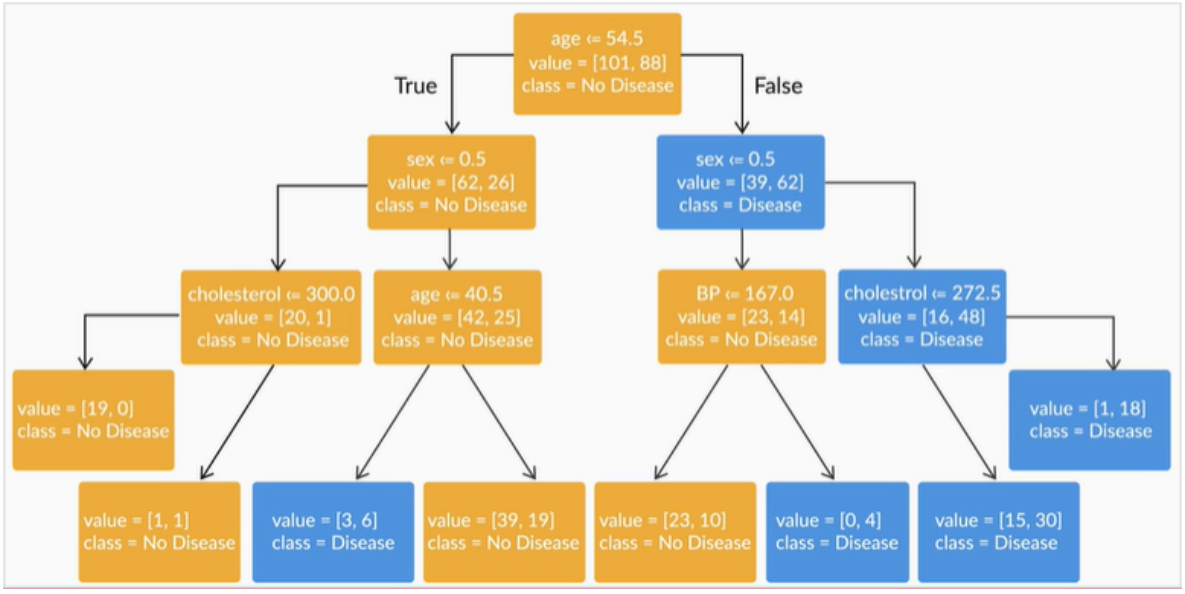
Now, if you were a doctor, you could ask these series of questions, and depending on the answers, you can probably make an educated prediction of whether the patient has the disease or not. This prediction here will obviously be based on your past learnings and experiences of treating such patients. 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.

Let's now look at the other side of the tree and understand the factors leading to a heart disease.

As you saw in this video, 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.

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.



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.

Comprehension - Interpreting a Decision Tree
Mithali is an incredible cricketer. She plays cricket only when the pitch is dry, and the field is well lit. Based on past observations, you also know that she doesn't play cricket when either the pitch is wet, or the light is dim.

Now that you have undetstood how to interpret the end result of a decision tree, let's now learn how to actually construct this tree in the first place.

Building Decision Trees:-

Now that you have learnt how to interpret decision trees, you must be wondering how does one exactly build decision trees? What is the tree building process? As expected, it involves performing a series of splits. Let's understand this process in detail in the following video.

So, as you saw in the video, constructing a decision tree involves the following steps:

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

Now, you might have many questions lingering on your mind like what is the best variable for the split, how do I know what is the stopping criterion, and so on. Don't worry, as the steps you learnt just now is just a high-level view of the tree building process. In the coming sessions, you will learn each of these processes in detail.

Now, 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.

In the next segment, you will learn to build decision trees in Python

Comprehension - Decision Tree Classification in Python:-

Let's consider the heart disease data set that we discussed in the earlier segment. The data lists various tests that were conducted on patients along with some other details of the patients. Now, given the test results and other attributes, suppose you want to predict whether a person has a heart disease or not.

Please download the dataset from below.

Please note that:

Heart disease = 0 means that the person does not have heart disease.
Heart disease = 1 means that the person has heart disease.

sex = 0 means that the person is female.
sex = 1 means that the person is male.

Please download the Python code from below to practice along.

To keep this simple and focus on building a decision tree only, we are skipping any data preparation or feature manipulation techniques.

You also need to install Graphviz to visualize the decision tree. The steps to be followed are provided towards the end of the page.

Note: In the video above at timestamp [7:07], Rahim meant 81 records in the test set instead of 189.

So you've imported the required libraries, read and inspected the data. Also, the entire data set has been split into train and test sets. Let's now move on to building the decision tree using the default parameters of the **DecisionTreeClassifier()** function except for the tree depth.

Now that you have built the decision tree and visualised it using the graphviz library, let's now evaluate how the model that we built is performing on the unseen data.

You can see that the model that we have now is not performing well on the test set. This is because we built our model on the default parameters except for the depth and didn't change any other hyperparameters. Hyperparameter tuning can improve the performance of decision trees to a great extent. So in the upcoming sessions, we will go ahead and exploit these parameters to improve the model and give better prediction results.

What are hyperparameters?
Hyperparameters are simply the parameters that we pass on to the learning algorithm to control the training of the model. Hyperparameters are choices that the algorithm designer makes to 'tune' the behaviour of the learning algorithm. The choice of hyperparameters, therefore, has a lot of bearing on the final model produced by the learning algorithm.

So basically anything that is passed on to the algorithm before it begins its training or learning process is a hyperparameter, i.e., these are the parameters that the user provides and not something that the algorithm learns on its own during the training process. Here, one of the hyperparameters you input was "max_depth" which essentially determines how many levels of nodes will you have from root to leaf. This is something that the algorithm is incapable of determining on its own and has to be provided by the user. Hence, it is a

hyperparameter.

Now, obviously, since hyperparameters can take many values, it is essential for us to determine the optimal values where the model will perform the best. This process of optimising hyperparameters is called hyperparameter tuning. You will learn to do that in the next session. First, let's answer some questions based on your learnings so far.

You need to use the resultant decision tree structure to answer the following questions.

Installing Graphviz
Python requires the library 'pydotplus' and the external software Graphviz to visualise the decision tree. If you are using Windows, then you will need to specify the path to the pydotplus library in order to access the dot file from Graphviz.

Please refer the user guide and the steps below to install Graphviz.

Steps for Windows users:

- Download Graphviz from here (ZIP file)
- Unzip the file and copy-paste it in **C:\Program Files (x86)**
- Make sure your file is unzipped and placed in Program Files (x86)
- Environment Variable: Add **C:\Program Files (x86)\graphviz-2.38\release\bin** to the user path
- Environment Variable: Add **C:\Program Files (x86)\graphviz-2.38\release\bin\dot.exe** to the system path
- Install the Python Graphviz package - pip install graphviz
- Install pydotplus - pip install pydotplus

Instructions to add the environment variable: click here

anaconda users
conda install pydotplus

pip users
pip install pydotplus

Steps for Mac Users:

- To install the Graphviz on your Mac, you can use Homebrew:
  - Install homebrew from here
  - Run this in the terminal

brew install graphviz

- Install pydotplus, pip install pydotplus

- Install the python graphviz module, pip install graphviz

Alternative method for Graphviz

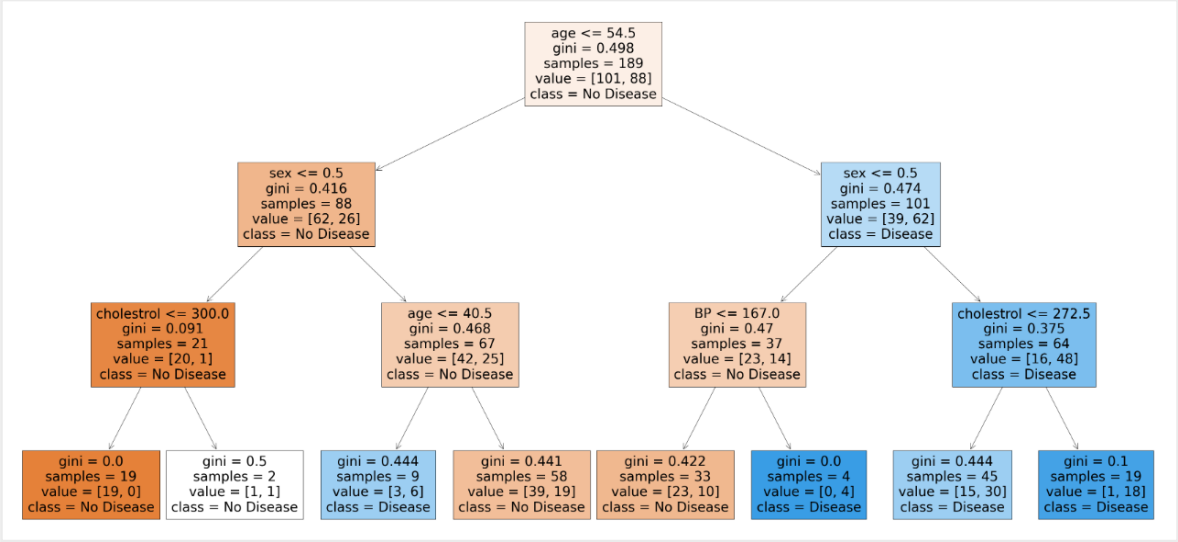
For those who are facing issues with visualizing a decision tree using the Graphviz software can use another function called plot_tree from sklearn.tree. You can read more about the function from the documentation provided here.

The code for visualizing a decision tree using plot_tree along with the output has been provided below:

Code:

```
from sklearn.tree import plot_tree
plt.figure(figsize=(60,30))
plot_tree(dt, feature_names = X.columns,class_names=['No Disease',
"Disease"],filled=True);
```

Output:



Additional Readings:

Parameters and Hyperparameters

Tree Models Over Linear Models:-

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 only one model to the entire data set; however, you may want to **divide the data set into multiple subsets** and apply decision tree algorithm in such cases to handle non-

linearity.

Let's understand why exactly we need tree models and what are their advantages in the following video.

Let's summarise the advantages of tree models one by one in the following order:

- 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. This is where regression with decision trees comes into the picture.

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:

$$\hat{y}_t = \frac{1}{N_t} \sum_{i \epsilon D_t} y^{(i)}, \text{where } y_i\text{'s represent the observations in a node.}$$

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.

This module includes an optional demonstration on regression trees for those who want to explore and understand the process in detail.

Additional Readings:

- Decision Trees scikit-learn documentation

Summary:-

Let's summarise some of the concepts that you learnt in this session.

Decision trees are easy to interpret, as you can always traverse backwards and identify the various factors that lead to a particular decision. A decision tree requires you to perform certain tests on attributes in order to split the data into multiple partitions.

In classification, each data point in a leaf has a class label associated with it.

You cannot use the linear regression model to make predictions when you need to divide the data set into multiple subsets, as each subset has an independent trend corresponding to it. In such cases, you can use the decision tree model to make predictions because it can split the data into multiple subsets and assign average values as the prediction to each set independently.

Module 2 : Algorithms for Decision Tree Construction

Introduction:-

Welcome to the session 'Algorithms for Decision Tree Construction'. In the previous session, you learnt about the underlying concepts of decision trees and their interpretation. You also learnt how to construct a decision tree. You learnt about the advantages of decision trees and also understood how they help in solving the regression problems that cannot be handled by linear regression. In this session, you will learn about the methods for decision tree construction.

In this session:

- Splitting and homogeneity
- Impurity measures
- Best split
- Regression trees

Splitting and Homogeneity:-

Recall the scenario from session one where you were a doctor asking a series of questions to determine whether a person has heart disease or not. Based on your past experience with other patients, the answers to these questions would finally lead to a prediction depicting whether the person has heart disease or not. Now, if you're a doctor, you would know which questions to ask first, right? When a person might be at the risk of heart disease, the doctor would probably first ask/measure the cholesterol of the person. If it's higher than 300, the doctor can be pretty sure that the person is at risk. Then he may ask some more questions to confirm his predictions even further.

Now suppose instead of asking/measuring the cholesterol, the doctor first asks the age of the person. If the person says, say, 55, the doctor may not be completely sure whether the person has heart disease or not. He/She would need to ask further questions.

From both these scenarios, it is easy to infer that to predict the target variable (in this case, heart disease), there are obviously some questions/attributes that are more important for its prediction than others. In our case, you saw that the cholesterol level is a more significant attribute than age in the prediction of heart disease. Why is that? It's so because, for the doctor, it might be evident from past records that people will higher cholesterol levels have a high chance of having heart disease. Say, in his/her experience that doctor has seen that out of every 100 patients who have cholesterol higher than 300, 90 had a heart disease while 10 did not. Also, say, out of 100 patients that the doctor has consulted over the age of 54, fifty of them had a heart disease. Now, it is evident that 'cholesterol' would be better to determine heart disease than 'age' since 90% of the patients having cholesterol greater than 300 were diagnosed with heart disease.

You can clearly see that one of the classes (disease) is significantly dominant over the other (non-disease) after splitting on the basis of cholesterol > 300 and this dominance is helping the doctor make a more confident prediction unlike 'age' which gives us a 50-50 split of both classes leaving us in a dilemma again.

So we basically arrive at these questions; Given many attributes, how do you decide which rules obtained from the attributes to choose in order to split the data set? From a single feature, you can get many rules and you may use any of these to make the split. Do you randomly select these and split the data set or should there be a selection criterion for choosing one over the other? What are you trying to achieve with the split?

All these questions will be covered in the following session and you will learn about the various algorithms and criteria that are involved in constructing a decision tree.

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 response (target) variable's homogeneity.

For example, let's suppose we consider the same heart disease example in which you wanted to classify whether a person has a heart disease or not. If one of the nodes is labelled 'Blood Pressure', try to split it with a rule such that all the data points that pass the rule have one label and those that do not pass the rule have a different label. Thus, you need to ensure that the response variable's homogeneity in the resultant splits is as high as possible.

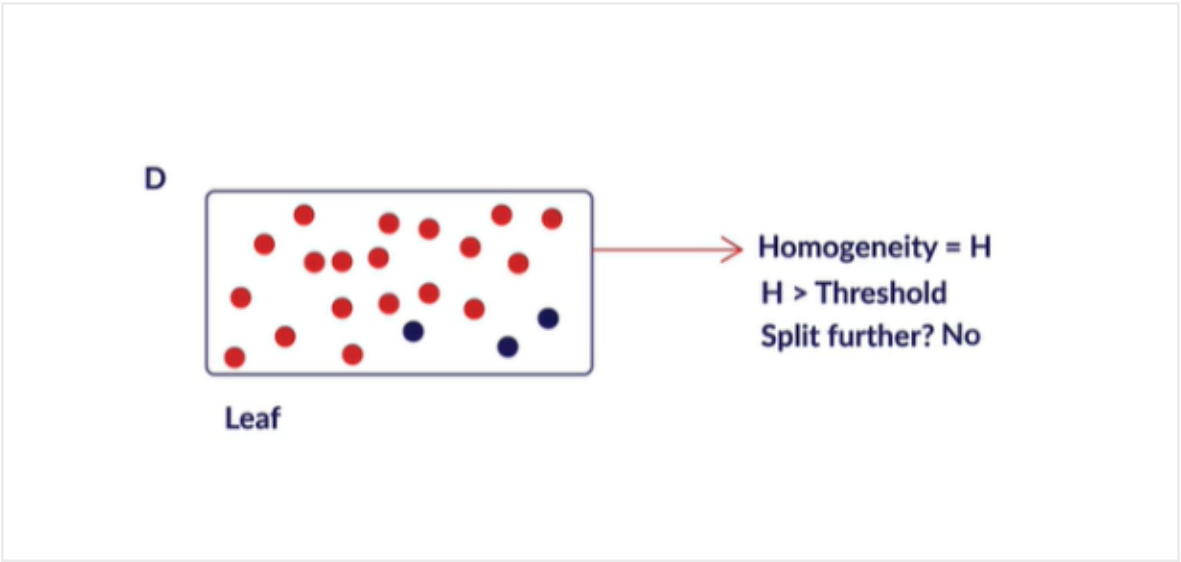
A split that results in a homogenous subset is much more desirable than the one that results in a 50-50 distribution (in the case of two labels). In a completely homogenous set, all the data points belong to one particular label. Hence, you must try to generate partitions that result in such sets.

For classification purposes, a data set is completely homogeneous if it contains only a single class label. For regression purposes, a data set is completely homogeneous if its variance is as small as possible. You will understand regression trees better in the upcoming segments.

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

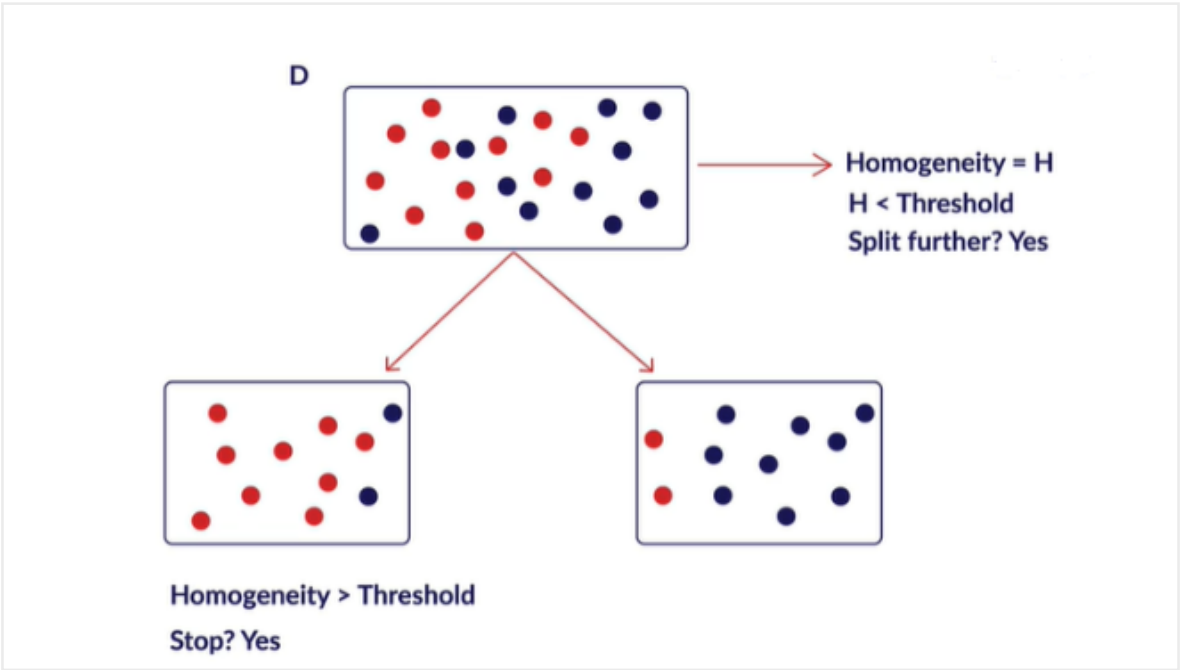
Consider a data set 'D' with **homogeneity 'H'** and a defined **threshold** value. When homogeneity exceeds the threshold value, you need to stop splitting the node and assign the prediction to it. As this node does not need further splitting, it becomes the leaf node.

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. This is continued until the homogeneity value exceeds the threshold value.



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.

This is an abstract example to give you an intuition on homogeneity and splitting. You will get better clarity in the next segment when you learn how to quantify and measure homogeneity to arrive at a prediction through continuous splitting. You will learn how to use specific methods to measure homogeneity, namely the Gini index, entropy, classification error (for classification), and MSE (for regression).

Now, how do you handle best split for different attributes in a decision tree using CART algorithm.

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 is 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 continuous features in different algorithms and this process is known as discretization. Although the exact technicalities are out of the scope of this module, curious students can read more about this in detail from the additional resources given below.

Now let's answer some questions based on your learnings so far.

Impurity Measures:-

Now, you have narrowed down the decision tree construction problem to this: you want to split the data set such that the homogeneity of the resultant partitions is maximum. But how do you measure this homogeneity?

Various methods, such as the **classification error**, **Gini index** and **entropy**, can be used to quantify homogeneity. You will learn about each of these methods in the upcoming video.

Note: At the timestamp [5:11] in the video, the formula for entropy includes a

negative sign along with the expression. Also, at [5:42], the 2nd term will be -0.8 log(0.8) and not -0.8 log(-0.8).

The classification error is calculated as follows:

- $E = 1 - max(p_i)$

The Gini index is calculated as follows:

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

Entropy is calculated as follows:

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

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

Let's now tweak the above example and try to understand how these impurity measures change with the class distribution.

From the above example, you understood how to calculate different impurity measures and how do they change with different class distributions.

Impurity Measures	**Case I** Class 0: 20 Class 1: 80	**Case II** Class 0: 50 Class 1: 50	**Case III** Class 0: 80 Class 1: 20
Classification Error	0.2	0.5	0.2
Gini Impurity	0.32	0.5	0.32
Entropy	0.72	1	0.72

You can see that for a completely non-homogeneous data with equal class distribution, the value of Classification Error and Gini Impurity are the same i.e. 0.5 and that of Entropy is 1.

The **scaled version of the entropy** in the illustration shown in the video 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.

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

Gini Index
Gini index is the degree of a randomly chosen datapoint being classified incorrectly. The formula for Gini index can also be written as follows:

$$G = \sum_{i=1}^{k} p_i(1 - p_i) = \sum_{i=1}^{k} (p_i - p_i^2) = \sum_{i=1}^{k} p_i - \sum_{i=1}^{k} p_i^2 = 1 - \sum_{i=1}^{k} p_i^2$$

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

(***Think why was*** $\sum_{i=1}^{k} p_i$ ***equal to 1?***)

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.**

Entropy
Entropy quantifies the degree of disorder in the given data, its value varies from 0 to 1. Entropy and the Gini index are similar numerically. If a data set is completely homogenous, then the entropy of such a data set will be 0, i.e., there is no disorder in the data. 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.**

Now that you have understood the different methods to quantify the purity/ impurity of a node, how do you identify the attribute that results in the best split? Let's learn more about this in the upcoming video.

The **change in impurity** or the **purity gain** is given by the difference of impurity post-split from impurity pre-split, i.e.,

$$\textbf{Δ Impurity} = \textbf{Impurity (pre-split)} - \textbf{Impurity (post-split)}$$

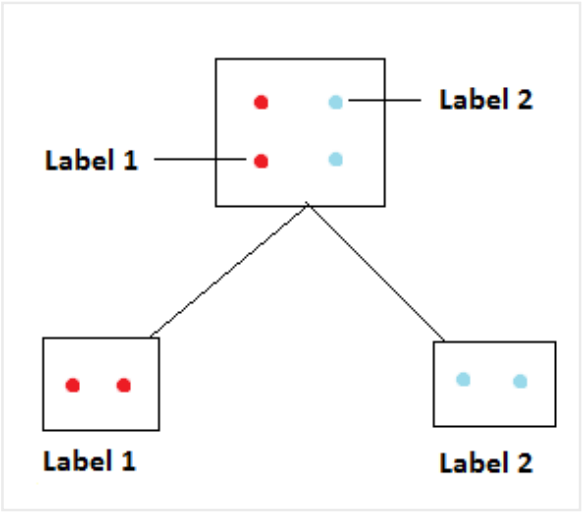
The **post-split impurity** is calculated by finding the **weighted average of two child nodes**. The split that results in **maximum gain** is chosen as the **best split**.

To summarise, the information gain is calculated by:

- $Gain = D - D_A$

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

Let's understand how do 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 $D_A = -1 * log_2(\frac{2}{2}) - 1 * log_2(\frac{2}{2}) = 0.$
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.

Additional Readings:
If you are curious to know about these topics, you may go through the following link.

- CHAID algorithm for decision trees (Chi squared criteria for splitting and multiway decision trees)
- Impurity Measures
- What happens when we use a base other than 2 for log in entropy?

Comprehension: The Gini Index:-

Let's consider the heart disease example that was introduced in the earlier segments to understand decision trees. Now, you will calculate the homogeneity measure for some of the features on some numbers using the Gini index to determine the attribute that you should split on first.

Recall that the Gini index is calculated as follows:

- $G = \sum_{i=1}^{k} p_i(1 - p_i) = 1 - \sum_{i=1}^{k} p_i^2$

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

The data set is not homogeneous, and you need to split the data such that the resulting partitions are as homogenous as possible. This is a classification problem, and there are two output classes or labels - having a heart disease or not. Here, you use the Gini index as the homogeneity measure. Let's go ahead and see how Gini index can be used to decide where to make the split on the data point. 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 on ‘cholesterol’.

Suppose you gave the data 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.

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:

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

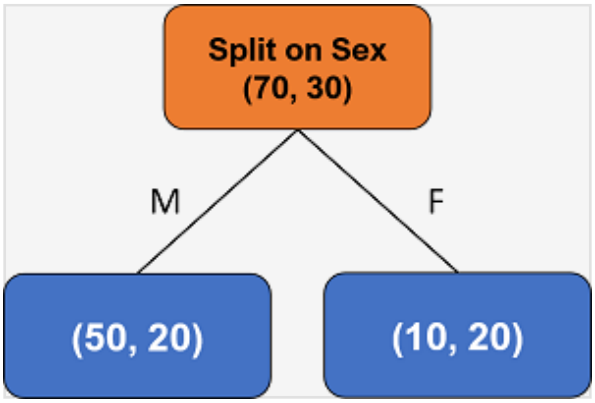
Let's now evaluate which split gives the maximum reduction in impurity among the possible choices. You have the following information about the target variable and the two attributes.

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 person 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:

$$p_0 = \frac{50}{70} = 0.714 \quad \text{and} \quad p_1 = \frac{20}{70} = 0.286$$

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:

$$p_0 = \frac{10}{30} = 0.333 \quad \text{and} \quad p_1 = \frac{20}{30} = 0.667$$

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 -

$$p_{male} = \frac{70}{100} = 0.7 \quad \text{and} \quad p_{female} = \frac{30}{100} = 0.3$$

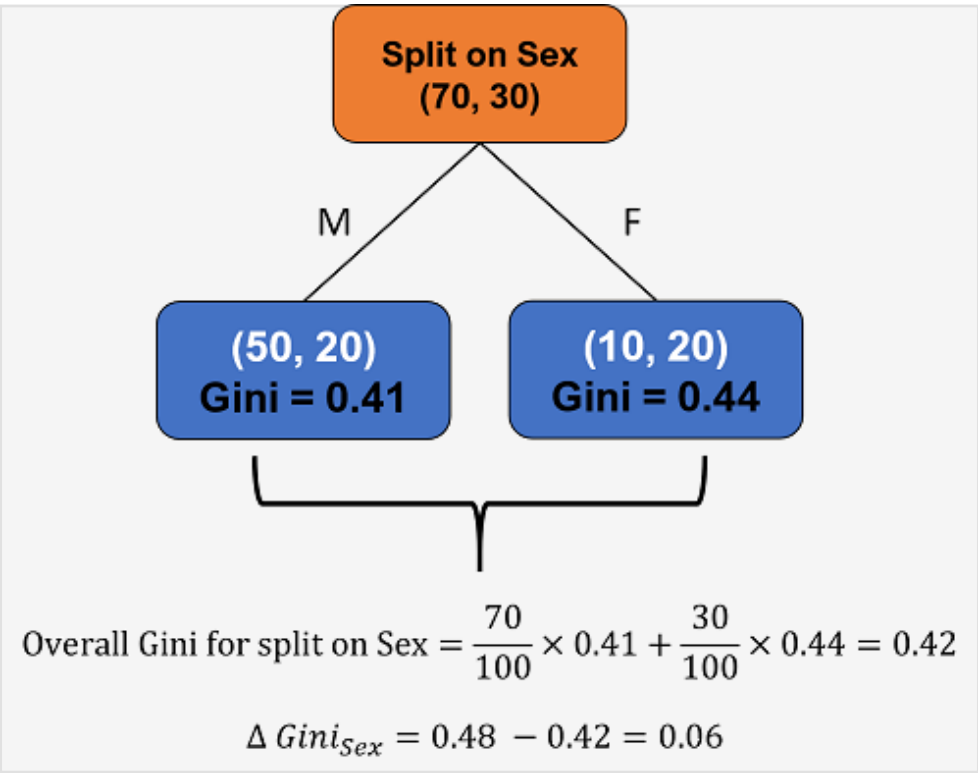
This gives the Gini impurity after the split based on gender as:

$$0.7 \times 0.41 + 0.3 \times 0.44 = 0.42$$

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' —

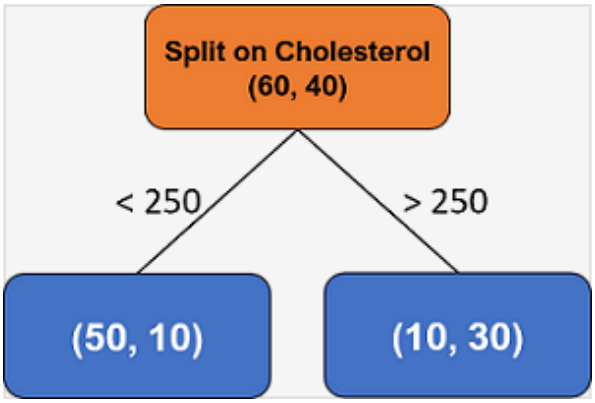


Split based on Cholesterol

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:

$$p_0 = \frac{50}{60} = 0.833 \quad \text{and} \quad p_1 = \frac{10}{60} = 0.167$$

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:

$$p_0 = \frac{10}{40} = 0.25 \quad \text{and} \quad p_1 = \frac{30}{40} = 0.75$$

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 -

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

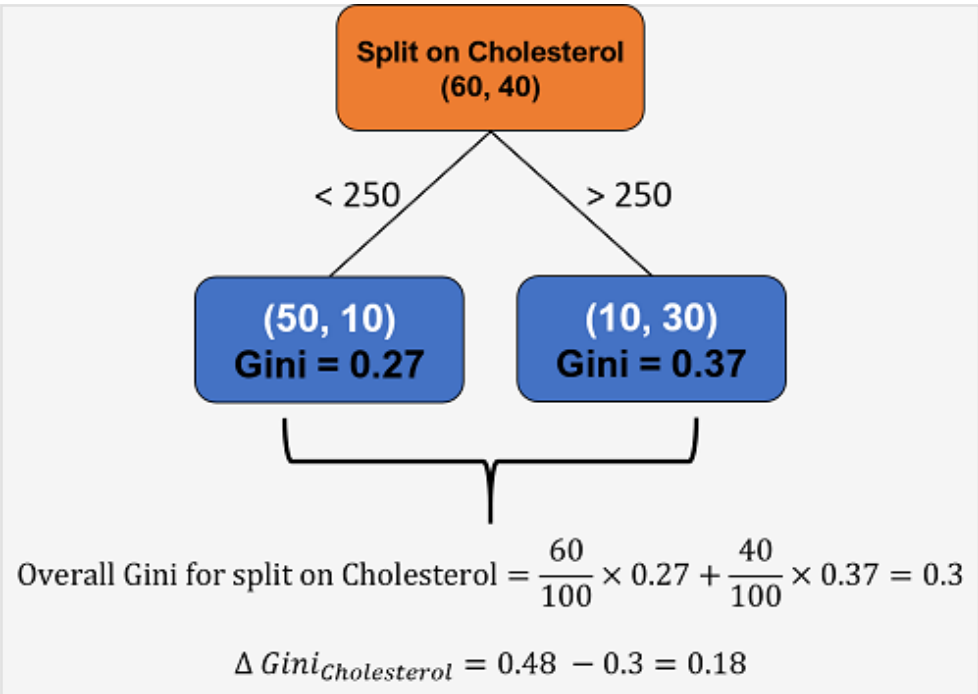
This gives the Gini impurity after the split based on cholesterol as:

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

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.

Let's summarise all the steps you performed**.**

1. Calculate the Gini impurity before any split on the whole dataset.
2. Consider any one of the available attributes.
3. Calculate the Gini impurity after splitting on this attribute for each of the levels of the attribute. In the example above, we considered the attribute 'Sex' and then calculated the Gini impurity for both males and females separately.
4. Combine the Gini impurities of all the levels to get the Gini impurity of the overall attribute.
5. Repeat steps 2-5 with another attribute till you have exhausted all of them.
6. Compare the decrease in Gini impurity across all attributes and select the one which offers maximum reduction.

You can also perform the same exercise using Entropy instead of Gini index as your measure.

Important: Please note that Gini index is also often referred to as Gini impurity. Also, some sources/websites/books might have mentioned a different formula for the Gini index. There is nothing wrong in using either of the formulas (because the ultimate interpretation regarding the impurity of the feature remains unchanged across both the formulas), but in order to avoid any confusion, we would recommend you to stick to the one mentioned in this

session as this is the formula that we will be consistently using throughout the whole module. Here's it again for you!

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

In the next segment, you will learn about the property of feature importance in decision 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.

In this video, let's understand the notion of variable importance in decision trees.

Note that in the video below, reduction in Gini Impurity based on Gender and cholesterol is 0.06<0.18 instead of 0.06<0.018.

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**.

In the previous example, the variable 'cholesterol' resulted in higher reduction in impurity than 'gender'. This implies that cholesterol will help distinguish between the two classes better than gender. This is intuitive as well: splitting on cholesterol is expected to be more important and informative than gender; this is because medically, people with high cholesterol have higher chances of developing heart disease than the ones with low cholesterol.

Summary:-

In this session, you learnt about the learning algorithms behind decision trees. In a step-by-step manner, you pick an attribute and a rule to split data into multiple partitions to increase the homogeneity of the data set.

You also learnt about the various ways in which you can measure the homogeneity of a data set, such as the Gini index, entropy and MSE.

Now, let's summarise your learnings so far:

- A decision tree first decides on an attribute to split on.

- To select this attribute, it measures the homogeneity of the nodes before and after the split.
- You can measure homogeneity in various ways with metrics like Gini index and entropy.
- The attribute that results in the increase of homogeneity the most is then selected for splitting.
- Then, this whole cycle is repeated until you obtain a sufficiently homogeneous data set.

Module 3 : Hyper-parameter Tuning in Decision Trees

Introduction:-

Welcome to the session on '**Hyperparameter Tuning in Decision Trees**'. In the previous session, you learnt about the concepts of homogeneity and its various measures. In this session, you will first understand the disadvantages of decision trees. Then, you will learn about various truncation and pruning strategies that are used to overcome one of the biggest disadvantages of trees, that is, overfitting.

In this session:

- Disadvantages of decision trees
- Tree truncation
- Tree pruning
- Hyperparameter tuning
- Decision tree regression

Disadvantages of Decision Trees:-

Decision trees are quite intuitive and promising algorithms for dealing with both continuous and categorical attributes. Does this mean that they should be used for every case? Not really. Let's watch the upcoming video to understand the problems associated with decision trees.

Note that you will learn about model selection and other related concepts of overfitting and bias variance trade off in the upcoming modules.

The following is a summary of the disadvantages of decision trees:

- They 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 (or rather, mugged up) all the data points in the training set.
- They tend to be quite **unstable**, which is an implication of overfitting.

A few changes in the data can considerably change a tree.

Additional Readings:
You will learn about these concepts in detail in the Model Selection module, but in case you are curious about them now, you may refer to the following links.

- Overfitting
- Bias-Variance Tradeoff

Tree Truncation:-

Earlier, you learnt that decision trees have a strong tendency to overfit data, which is a serious problem. So, you need to pay attention to the size of the tree. A large tree will have a leaf only to cater to a single data point.

There are two broad strategies to control overfitting in decision trees: **truncation and pruning**. In this video, you will learn how these two techniques help control overfitting.

There are two ways to control overfitting in trees:

1. **Truncation** - Stop the tree while it is still growing so that it may not end up with leaves containing very few data points. Note that truncation is also known as **pre-pruning**.
2. **Pruning** - Let the tree grow to any complexity. Then, cut the branches of the tree in a bottom-up fashion, starting from the leaves. It is more common to use pruning strategies to avoid overfitting in practical implementations.

Please note that this session covers mainly the "Truncation" techniques. In case you also want to learn about "Pruning" techniques (which is completely independent), we have provided an optional segment that can be accessed here. We recommend going through it after you have finished this whole session on hyperparameter tuning.

Let's look into various ways in which you can truncate a tree:

Though there are various ways to truncate or prune trees, the DecisionTreeClassifier() function in sklearn provides the following hyperparameters which you can control:

1. **criterion (Gini/IG or entropy):** It defines the homogeneity metric to measure the quality of a split. Sklearn supports "Gini" criteria for Gini Index & "entropy" for Information Gain. By default, it takes the value of "Gini".
2. **max_features:** It defines the no. of features to consider when looking for the best split. We can input integer, float, string & None value.
    - If an integer is inputted then it considers that value as max

features at each split.
    - If float value is taken then it shows the percentage of features at each split.
    - If “auto” or “sqrt” is taken then max_features=sqrt(n_features).
    - If “log2” is taken then max_features= log2(n_features).
    - If None, then max_features=n_features. By default, it takes “None” value.
3. **max_depth:** The max_depth parameter denotes the maximum depth of the tree. It can take any integer value or None. If None, then nodes are expanded until all leaves contain just one data point (leading to overfitting) or until all leaves contain less than "min_samples_split" samples. By default, it takes “None” value.
4. **min_samples_split:** This tells about the minimum no. of samples required to split an internal node. If an integer value is taken then consider min_samples_split as the minimum no. If float, then min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the minimum number of samples for each split. By default, it takes the value "2".
5. **min_samples_leaf:** The minimum number of samples required to be at a leaf node. If an integer value is taken then consider min_samples_leaf as the minimum no. If float, then it shows the percentage. By default, it takes the value "1".

There are other hyperparameters as well in DecisionTreeClassifier. You can read the documentation in python using:

```
help(DecisionTreeClassifier)
```

Later in the session, you will go through a model building exercise in Python which will show you how these hyperparameters affect the tree structure.

Building Decision Trees in Python:-

In the previous session, we built a decision tree on default hyperparameters. Let’s now learn how to tune some of these hyperparameters and check what difference they make to the model performance.

You can download the dataset and Jupyter Notebook from the link given below and practice along.

You are also required to download Graphviz that will be used to visualise decision trees in Python.

Let’s first create some helper functions to evaluate your model, as you would be doing it multiple times and creating these functions would ease the job.

Now that you have created some helper functions, let’s change some of the default hyperparameters, such as max_depth, min_samples_split, min_samples_leaf and criterion (Gini/IG or entropy), and understand how they impact the model performance.

In this video, you learnt how changing the default hyperparameters improve the model performance. Let's now watch the following video to look at how entropy can be used instead of Gini to measure the quality of a split.

Now, what you did so far was just exploring the hyperparameters and see how they affected the model performance. The numbers you chose for the hyperparameters just now was more or less random. However, there should be some way to choose the optimal values for the hyperparameters, right? In the next segment, you will learn how to tune the hyperparameters to find their optimal values using k-fold cross-validation.

Choosing Tree Hyper-parameters in Python:-

So far, you have learnt how to tune different hyperparameters manually. However, you cannot always choose the best set of hyperparameters for the model manually. Instead, you can use **gridsearchcv()** in Python, which uses the **cross-validation** technique. Now, what exactly is cross-validation, and how is it helpful? Let’s watch the next video and hear what Rahim has to say about it.

As you learnt in this video, the problems with manual hyperparameter tuning are as follows:

- **Split into train and test sets**: Tuning a hyperparameter makes the model 'see' the test data. Also, the results are dependent upon the specific train-test split.
- **Split into train, validation and test sets**: The validation data would eat into the training set.

However, in the cross-validation technique, you split the data into train and test sets and train multiple models by sampling the train set. Finally, you can use the test set to test the hyperparameter once.

Specifically, you can apply the **k-fold cross-validation** technique, where you can divide the training data into k-folds/groups of samples. If k = 5, you can use k-1 folds to build the model and test it on the kth fold.

It is important to remember that k-fold cross-validation is only applied on the train data. The test data is used for the final evaluation. One extra step that we perform in order to execute cross-validation is that we divide the train data itself into train and test (or validation) data and keep changing it across "k" no.

of folds so that the model is more generalised. This is depicted in the image below.

The green and orange boxes constitute the training data. Here, the green ones are the actual training data and orange ones are the test (or validation) data points selected within the training dataset. As you can see, the training data is divided into 5 blocks or folds, and each time 4 blocks are being used as training data and the remaining one block is being used as the validation data. Once the training process is complete, you jump to model evaluation on the test data depicted by the blue box.

Now, coming back to the question, how do you control the complexity (or size) of a tree? A very 'big' or complex tree will result in overfitting. On the other hand, if you build a relatively small tree, it may not be able to achieve a good enough accuracy, i.e., it will underfit. So, what values of hyperparameters should you choose? As you would have guessed, you can use grid search with cross-validation to find the optimal hyperparameters.

In the next video, Rahim will explain how each hyperparameter affects a tree and how you should choose the optimal set of hyperparameters.

You played around with different values of different hyperparameters using GridSearchCV(). This function helped you try out different combinations of hyperparameters which ultimately eased your process of figuring out these best values. It is, however, important to note that the values tried out in the demonstration above may have not necessarily given the best results in terms of accuracy. You may go ahead and try out different combinations as well and see if you can surpass Rahim's test score.

Phew! That was surely a lot to take in the past couple of segment. Let's now watch the next video to recall and summarise what you have learnt so far while building decision trees.

You are required to read the documentation of sklearn DecisionTreeClassifier and understand the meaning of hyperparameters. The following questions are based on the documentation.

If you are comfortable with building decision trees now, we recommend you to attempt some optional model building coding exercises on the DoSelect console. The questions can be accessed here.

Comprehension - Hyperparameters:-

Consider a decision tree classification model that has a very high training accuracy and a low test accuracy. The training accuracy is 98%, and the test accuracy is 55%. The 'min_samples_split' for this model is 5, and the 'max_depth' is 20.

Please note that, unless explicitly specified, 'node' does not mean the terminal node (i.e. leaf) - it refers to an internal node.

Decision Tree Regression:-

So far, you have learnt about splits for discrete target variables. But how is splitting performed for continuous output variables? You calculate the weighted mean square error (WMSE) of data sets (before and after splitting) in a similar manner as you do for linear regression models. In this video, you will learn about this in detail.

In decision tree regression, each leaf represents the average of all the values as the prediction as opposed to taking an majority vote in classification trees. This **average** is calculated using the following formula:

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

This is nothing but the sum of all data points divided by the total number of data points.

Also, 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:

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

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:

1. Calculate the MSE of the target variable.
2. Split the data set based on different rules obtained from the attributes and calculate the MSE for each of these nodes.
3. The resulting MSE is subtracted from the MSE before the split. This result is called the **MSE reduction**.
4. The attribute with the largest MSE reduction is chosen for the decision node.
5. 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.
6. 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.

This module includes an optional demonstration on regression trees for those who want to explore and understand the process in detail.

Additional Readings:
In the next segment, you will learn how to implement decision tree regression in Python. If you wish to read more upon decision tree regression, we have curated some links below which you can go through.

- Decision Tree Regression
- Regression Trees Calculated Example (Note that in the given example, standard deviation is computed to calculate the homogeneity of a node which is nothing but the square root of MSE).

Decision Tree Regression in Python:-

Now that you have learnt how decision trees can be used to solve regression problems, let's understand how regression trees are built in Python. For this, you will use the same housing data set that you used in multiple linear regression to predict house prices based on various factors such as area, number of bedrooms, parking space etc.

Essentially, the aim is to:

- Identify the variables affecting house prices, e.g., the area, the number of rooms, bathrooms, etc.
- Create a linear model that quantitatively relates house prices with variables, such as the number of rooms, area, number of bathrooms; and
- Know the variables that significantly contribute towards predicting house prices.

The data set used in the following videos can be downloaded from below.

You can download the Python code file used in the next video to practise along. Please ensure that you go through the initial Python code file that you used in multiple linear regression to recall the initial steps and model building that you did earlier. This will help you have a better understanding of the further steps that this segment will cover.

Let's build the model using a **DecisionTreeRegressor()** with some arbitrary

parameters for the sake of simplicity and more accurate prediction results.

In order to see what the model looks like, let's plot our decision tree and try to interpret what it conveys about house prices. We also need to evaluate how our decision tree performs. From the video given below, let's understand model interpretation and evaluation.

So now, you have a good understanding of how decision trees can be used for decision-making whenever you have continuous target variables. As an exercise, you can definitely perform more hyperparameter tuning here and improve the performance of the models that you built right now.

Summary:-

In this session, you looked at one of the major drawbacks of decision trees, that is, overfitting, and the various methods that can be used to avoid it.

You learnt that decision trees are prone to overfitting. There are two ways to avoid overfitting: truncation and pruning.

In truncation, you let the tree grow only to a certain size, while in pruning, you let the tree grow to its logical end and then you chop off the branches that do not increase the accuracy on the validation set.

There are various hyperparameters in the DecisionTreeClassifier that let you truncate the tree, such as minsplit, max_depth, etc.
You also learnt about the effect of various hyperparameters on the decision tree construction.

You can download the lecture notes for decision trees from the link given below.

Let's attempt some graded questions in the next page. Make sure you have gained an understanding of all the concepts before attempting.

Graded Questions:-

The questions given below are **graded**. Best of luck!

Please download the training data set and validation data set that are attached, to answer the following questions.

Delhi Delights!
Suppose you work at Delhi Delights! which is a food delivery company in Delhi. It offers a premium membership called 'Delighted Members', with which there is no delivery cost for your order. Lately, the number of purchases of this premium membership has been going down. Now, based on past data, Delhi Delights!

wants to predict which of the customers will buy the 'Delighted Members' membership and which ones will not. For this, you have been provided with the following dataset.

Delhi Delights! Dataset - The first two columns in the training set file represent the two attributes, Average Delivery Rating (a1) and Average Orders per Month (a2). The third column (column C) is the class label associated with each data point in the training set. There are two class labels: "Yes" and "No" indicating whether the person has bought the premium membership. In total, there are 30 data points/observations in the training set. You need to build a decision tree using this file. Use all 30 points for training.

Please note that you mustn't use DecisionTreeClassifier() or write codes to solve the following questions. You can use Excel to split and filter the data set according to the test conditions specified in the questions. The DecisionTreeClassifier function uses a slightly different test on each node, and hence, the final answer to the following questions may not match.

Please use the following hyperparameters to solve the following questions:

- max_depth = 5
- min_samples_split = 10
- min_samples_leaf = 5
- Homogeneity measure = Gini

Note: **(13, 5)**, written on a node, implies that 13 data points belong to class label "No", and 5 data points belong to class label "Yes". This syntax is true for all the nodes shown in the tree.