upGrad

SUMMARY

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 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.

Limitations of Logistic Regression

Logistic regression is a popular machine learning model, but it has a few limitations. It requires the following tasks to be performed on the given data for it to function properly:

1. Preprocessing the data before use
2. Normalising the data
3. Scaling the features to the same level

This makes the process tedious and complicated when a huge volume of data is involved. Therefore, it is not always the most suitable method. Apart from the aforementioned factors, logistic regression does not work well in the following cases:

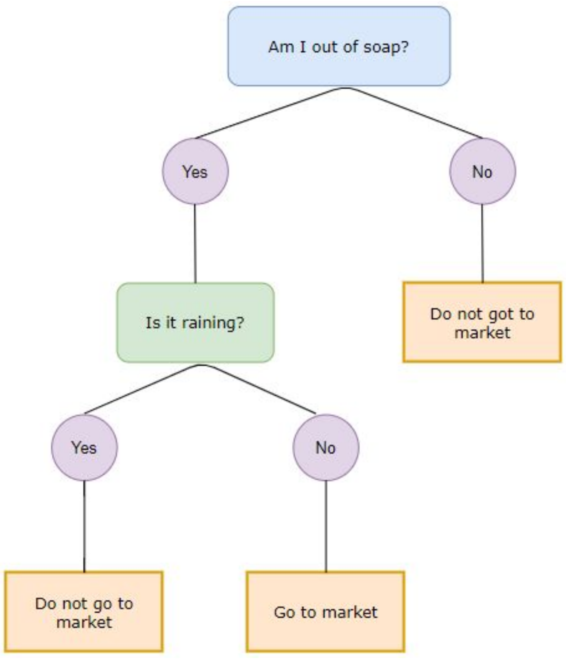
1. When multiclass data is involved
2. When a large number of entries are missing in the data

Due to these limitations, there is a requirement for another machine learning model that can work efficiently in such scenarios. This is where the decision tree model comes into the picture. Although the logistic regression technique helps you interpret the dependency of the dependent and the independent variables, you may not be able to visualise it. Decision tree outcomes are interactive and comparatively easy to understand and visualise, and, unlike logistic regression, decision Trees do not require large-size training data.

Decision Trees: Introduction

As the name suggests, a decision tree is a flowchart-like structure that helps in making 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. A parametric algorithm is an algorithm in which data can be divided over a certain set of parameters. These parameters are used to restrict the flexibility of the model, as models can only be built around parameters. Therefore, we can conclude that decision trees are more flexible than logistic regression.

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 is that decision trees is that they can be used for both classification and regression tasks. The two types of decision trees are as follows:

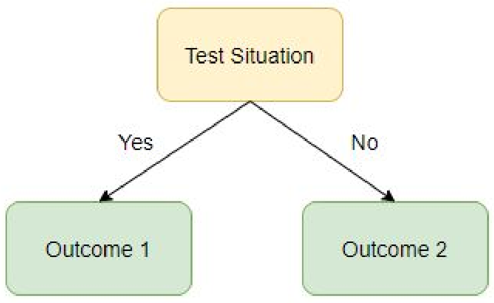
1. Classification decision trees
2. Regression decision trees

Components of a Decision Tree

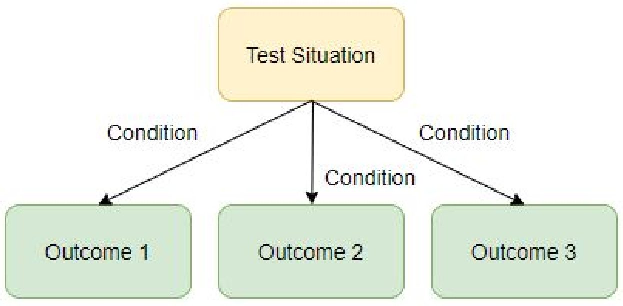
The other two types of decision trees based on the split-criteria are as follows:

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.



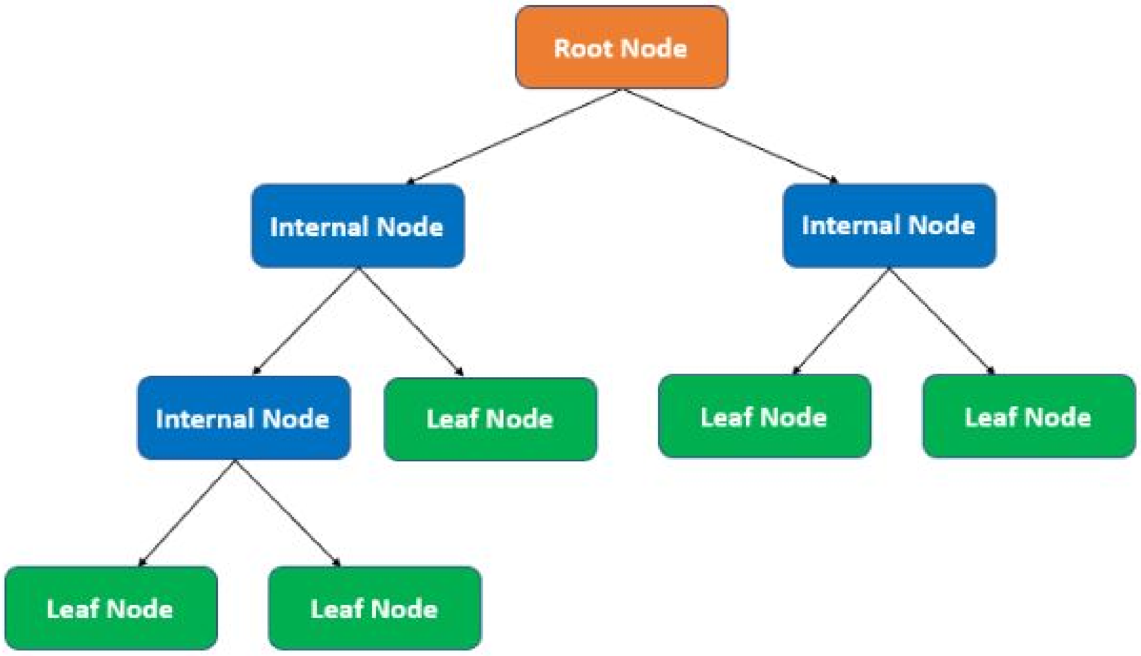
Binary Decision Tree



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 you did in the previous segment.

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.

2. **Splitting:** This is the process through which a single node breaks down into more nodes.

3. **Edge:** An edge represents possible values into which a parent node splits.

4. **Leaf/Terminal node:** The node that does not split into more nodes is known as a leaf node or a terminal node.

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

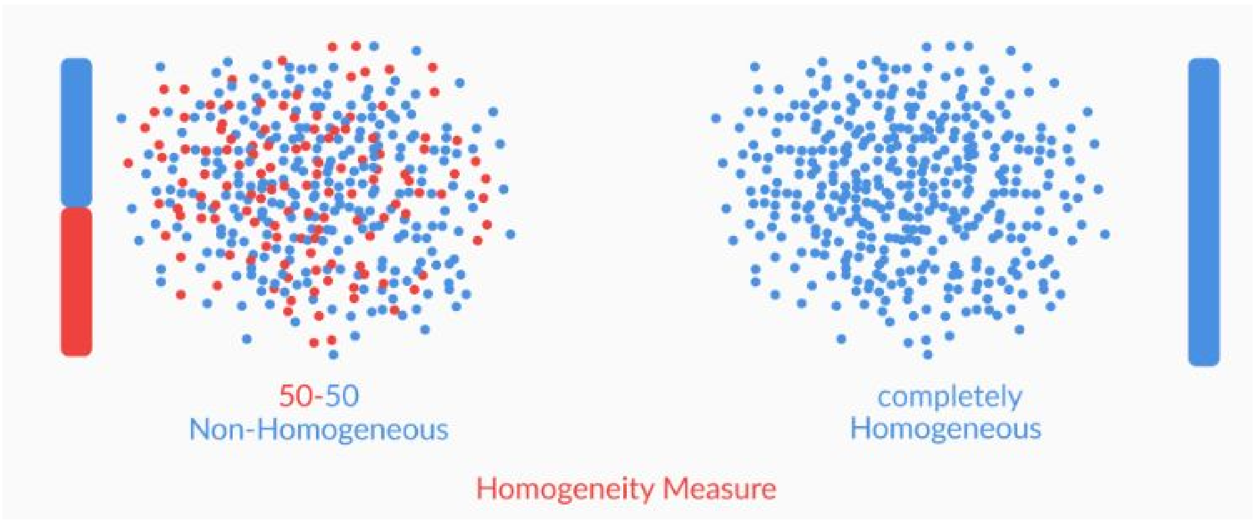
6. **Parent and child nodes:** A node that is divided into subnodes is called a parent node of the subnodes, whereas the subnodes are considered the children of the parent node.

Homogeneity

In order to construct a decision tree, you must know how to select the node that will lead to the best possible solution. There are two main points to be kept in mind while constructing a decision tree. They are as follows:

1. Simplicity
2. Homogeneity/Purity

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 data set increases 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.

Building a Decision Tree: Classification

Building a classification tree involves the following two steps:

1. **Learning step:** In this step, the model is trained on the existing data, which is known as the training data.
2. **Prediction step:** In this step, the model is used to predict the result for a data set, which is known as the test data.

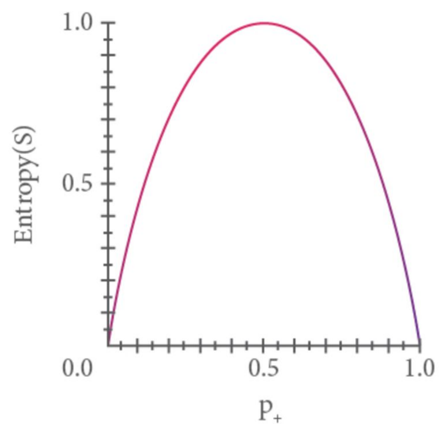
Entropy: In the case of classification decision trees, the splitting criteria depend upon entropy. Entropy is the measure of the randomness of a variable. Thus, a higher entropy means more randomness, which leads to less purity. Entropy quantifies the degree of disorder in the given data. Its value varies 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. The entropy of a variable in a given class can be calculated using the following formula:

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

Where, p_i is the probability of the feature/attribute under consideration.

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.

Following are the steps in creating a categorical decision tree:

1. Obtaining a tabular representation of the given data

2. Calculating the entropy for all the features in the data set

3. Calculating the information gain, i.e., the change in entropy

4. Selecting the node with the highest information gain as the root node

5. Dividing the tree on the root node and repeating the aforementioned process to further split until leaf nodes are obtained

Next, you solved the classification problem for the coronavirus case study using decision trees on the data set given in the table provided below.

S. No	Symptoms	International travel	Interaction with Infected Person	Corona Test Result
1	Fever	More than 14 days	Without mask	No
2	Fever	More than 14 days	With mask	No
3	Breathing problem	More than 14 days	Without mask	Yes
4	Cough	More than 14 days	Without mask	Yes
5	Cough	Less than 14 days	Without mask	Yes
6	Cough	Less than 14 days	With mask	No
7	Breathing problem	Less than 14 days	With mask	Yes
8	Fever	More than 14 days	Without mask	No
9	Fever	Less than 14 days	Without mask	Yes
10	Cough	Less than 14 days	Without mMask	Yes
11	Fever	Less than 14 days	With mask	Yes
12	Breathing problem	More than 14 days	With mask	Yes
13	Breathing problem	Less than 14 days	Without mask	Yes
14	Cough	More than 14 days	With mask	No

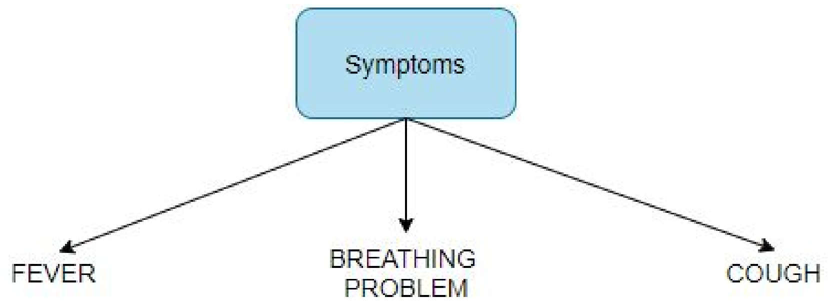
In a stepwise manner, let's analyse the process and create a decision tree to find out whether or not a person has coronavirus:

- Step 1: Calculate the entropy at source, i.e, 'Corona test result'.
    - P(Yes) = 9/14, P(No) = 5/14
    - Entropy = -P(Yes) $\log_2$P(Yes) - P(No) $\log_2$P(No)
    - Entropy = -9/14 $\log_2$P(9/14) - P(5/14) $\log_2$P(5/14) = 0.94

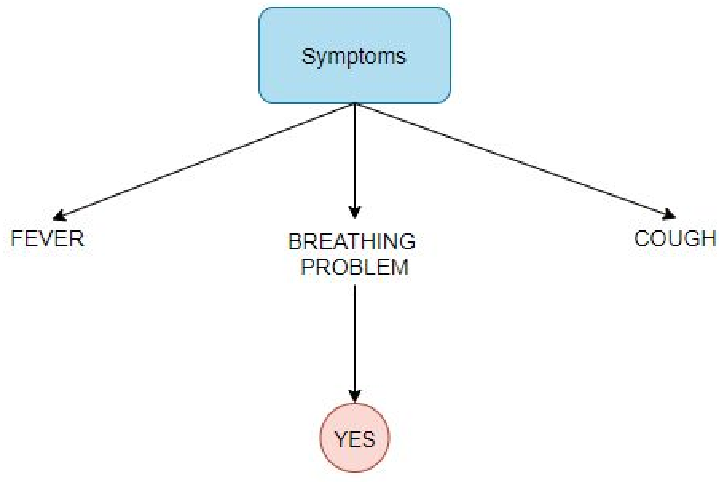
- Step 2: Calculate the entropy for all the features, i.e., Symptoms, International travel and Interaction with an infected person.
    - Now, calculate the entropy for each feature. Note that here, you are assuming that the split is happening three ways: The 'Symptoms' feature splits into Fever, Breathing Problem and Cough.
        - Entropy for 'Fever' = 0.97
        - Entropy for 'Breathing Problem' = 0
        - Entropy for 'Cough' = 0.97
        - The total entropy of the Symptoms = 5/14 * 0.97 + 4/14 * 0 + 5/14 * 0.97 = 0.69
    - Similarly, calculate the entropy for the other features, International travel and Interaction with infected people.

- Step 3: Calculate the information gain for all the features, i.e., Symptoms, International travel, Interaction with an infected person.
    - IG for 'Symptoms' = 0.25
    - IG for 'Interaction with infected person' = 0.048
    - IG for 'International travel' = 0.152

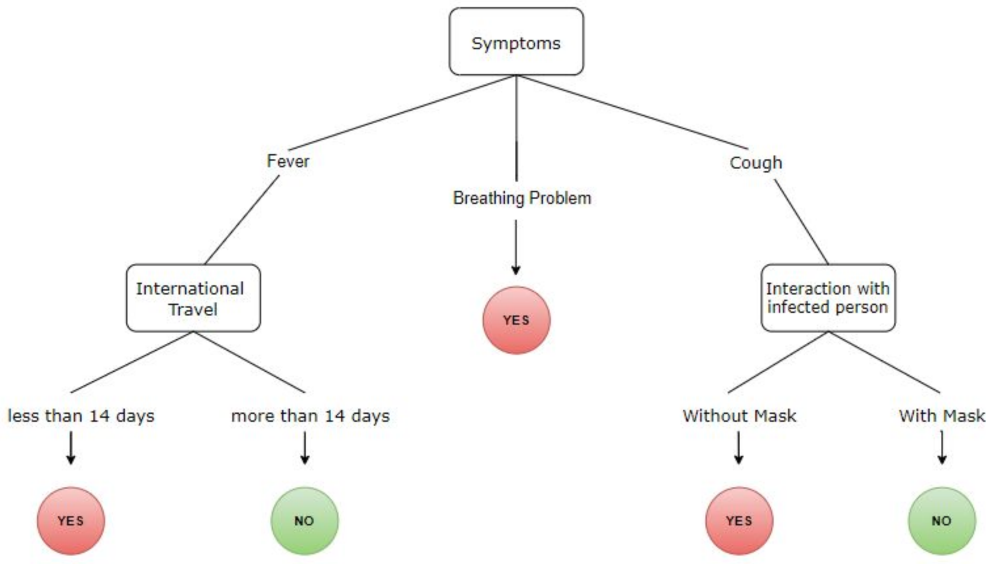
- Step 4: The information gain is maximum for 'Symptoms'; hence, it is chosen as the root node. At this point, the decision tree would look like the one shown in the image provided below.



- Step 5: The entropy for 'Breathing Problem' is zero, as all the outcomes in the case of this feature lead to a positive case of coronavirus according to the given data set. Therefore, the feature is not split further.However, 'Fever' and 'Cough' will be further split depending on the information gain. So, after selecting the root node, the tree structure would look similar to the one shown in the image given below.

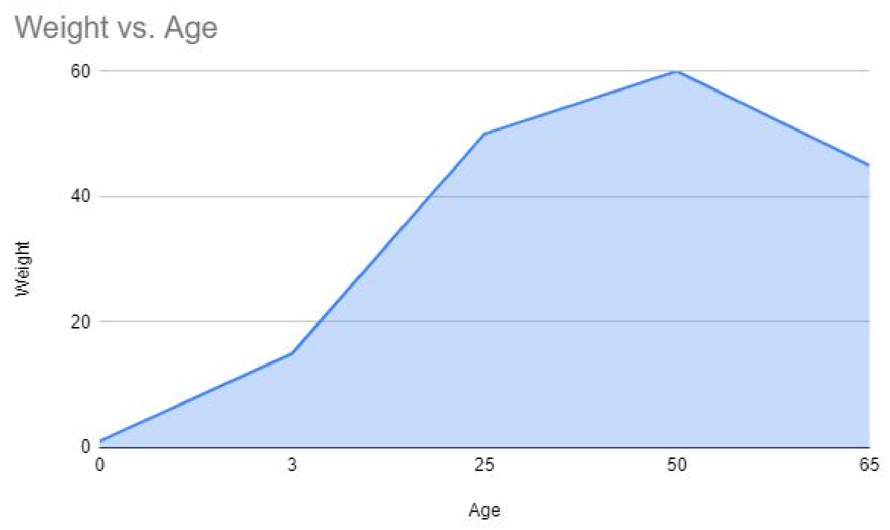


- Step 6: Calculate the information gain for 'International travel' and 'Interaction with infected person' for both 'Fever' and 'Cough', and split on them until you obtain the final tree. The final decision tree is shown in the image given below.

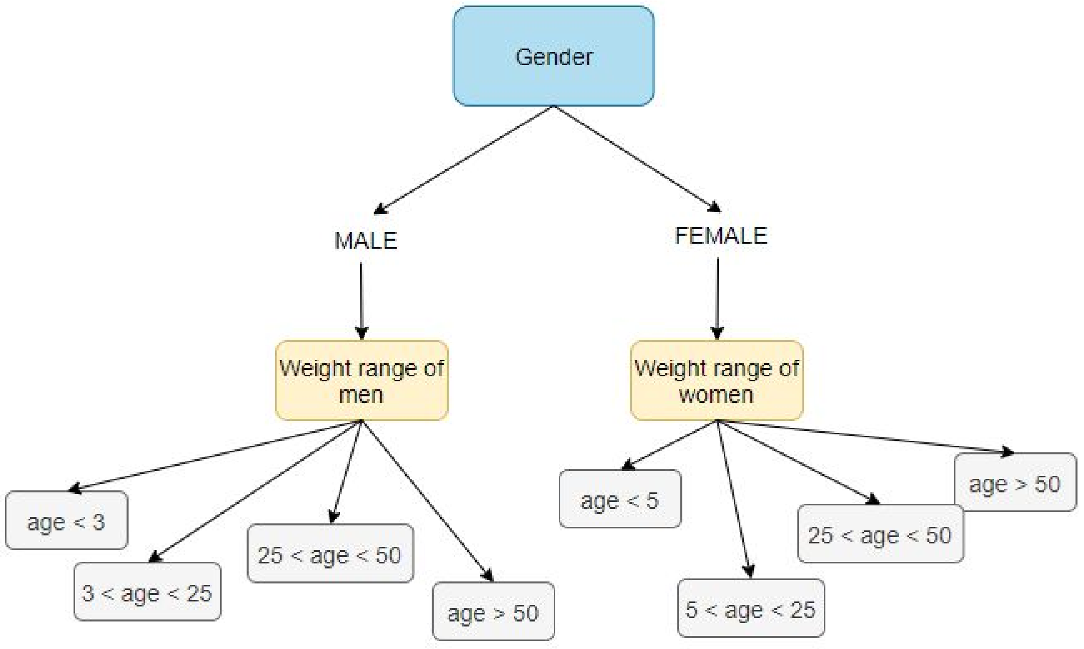


Building a Decision Tree: Regression

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 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, you can use standard deviationThe reduction in the standard deviation of the data set becomes the measure of its purity. The formula used for calculating standard deviation is as follows:

$$s_N = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2}.$$

where,
N is the number of data points,
xi is ith data, and
x̅ is average(mean) of all data.

While selecting the splitting value in the case of continuous variables, the factor that is measured is the reduction in the variance. The formula for calculating variance is as follows:

$$\text{Variance} = \frac{\Sigma(X - \overline{X})^2}{n}$$

As you can see, the variance is simply the square of the standard deviation, and hence, it would not matter if you used standard deviation. The key point that you need to focus on is how to find the splitting value for continuous variables. While working with continuous feature variables, sort them in ascending order and then find the midpoint of all the data points. Take a look at the coronavirus case study observation table given below, which now has continuous target variables.

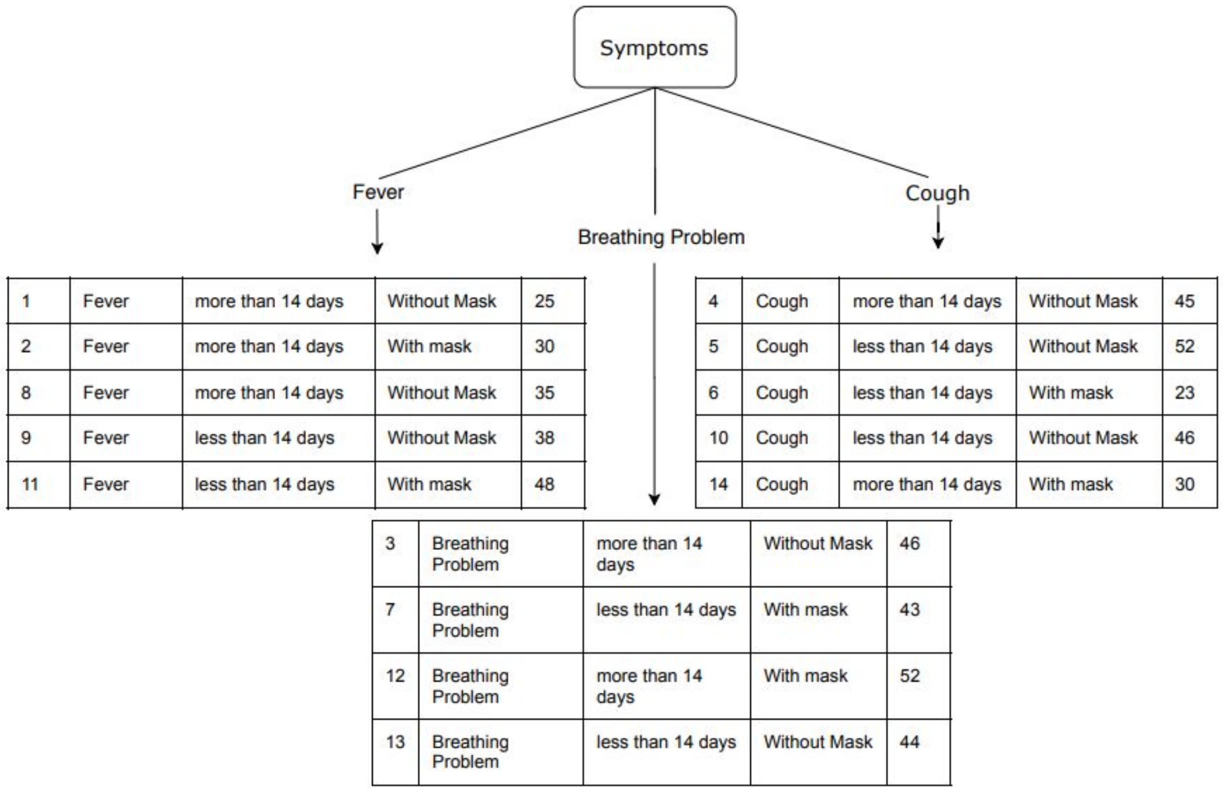
S. No	**Symptoms**	**International travel**	**Interaction with Infected Person**	**Corona Test Result**
1	Fever	More than 14 days	Without mask	25
2	Fever	More than 14 days	With mask	30
3	Breathing problem	More than 14 days	Without mask	46
4	Cough	More than 14 days	Without mask	45
5	Cough	Less than 14 days	Without mask	52
6	Cough	Less than 14 days	With mask	23
7	Breathing problem	Less than 14 days	With mask	43
8	Fever	More than 14 days	Without mask	35
9	Fever	Less than 14 days	Without mask	38
10	Cough	Less than 14 days	Without mask	46
11	Fever	Less than 14 days	With mask	48
12	Breathing problem	More than 14 days	With mask	52
13	Breathing problem	Less than 14 days	Without mask	44
14	Cough	LMore than 14 days	With mask	30

Use the coronavirus data set given in the table provided above to create a decision tree using the following steps:

- Calculate the standard deviation of percentage infection.
    - Calculate the mean/average of the values.
    - Apply the standard deviation formula and compute the result.

- Calculate the standard deviation for the features Symptoms, International travel and Interaction with an infected person.
    - For each feature, calculate the mean and standard deviation as you did in the step.
    - Calculate the reduction in the standard deviation as the difference between the standard deviation of percentage infection and the standard deviation of the feature under consideration.
    - Select the feature maximum value of reduction in the standard deviation as the root node.

- As the maximum value of reduction in the standard deviation is for 'Symptoms', it is selected as the root node. Now, the tree looks like the one shown in the image given below.



1	Fever	more than 14 days	Without Mask	25
2	Fever	more than 14 days	With mask	30
8	Fever	more than 14 days	Without Mask	35
9	Fever	less than 14 days	Without Mask	38
11	Fever	less than 14 days	With mask	48

3	Breathing Problem	more than 14 days	Without Mask	46
7	Breathing Problem	less than 14 days	With mask	43
12	Breathing Problem	more than 14 days	With mask	52
13	Breathing Problem	less than 14 days	Without Mask	44

4	Cough	more than 14 days	Without Mask	45
5	Cough	less than 14 days	Without Mask	52
6	Cough	less than 14 days	With mask	23
10	Cough	less than 14 days	Without Mask	46
14	Cough	more than 14 days	With mask	30

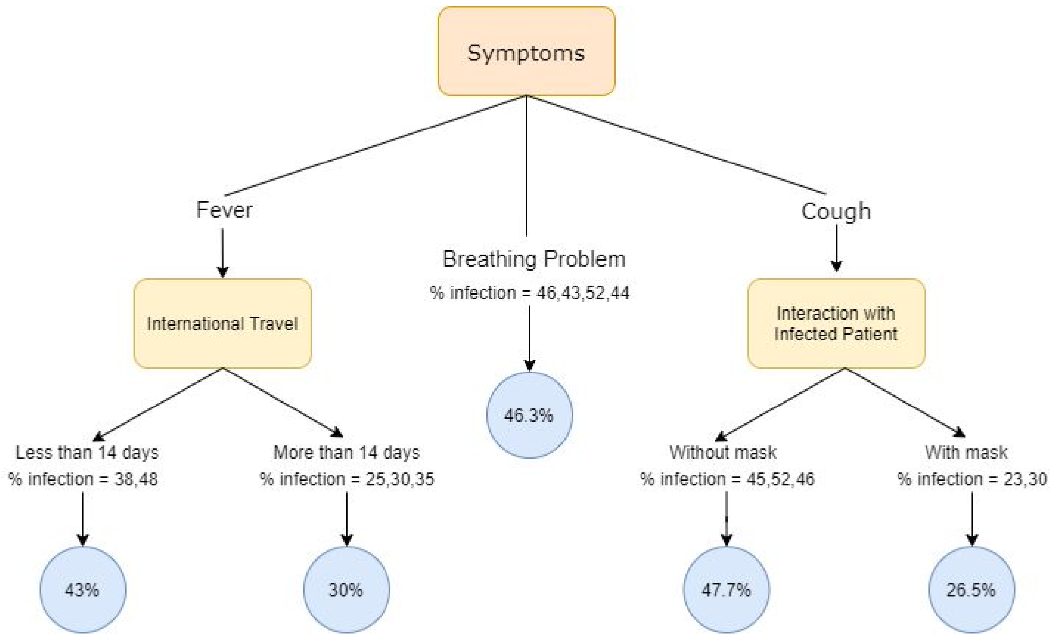
- Now, calculate the reduction in the standard deviation for 'International travel' and 'Interaction with infected patients' by keeping 'Symptoms' as the root node.

- If the number of leaf nodes is large, calculating the average for each leaf node becomes a tedious task. You can use the maxBins function for this purpose.

- Repeat the same steps for the other features until you reach the leaf nodes.

- Once you have reached the point where you cannot split the tree any further, calculate the average of values at the leaf node.

- The structure of the final decision tree is shown in the image given below.



CART

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.

You have already learnt about the method of measuring purity in the case of both regression and classification trees. 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:

$$\text{Gini} = 1 - \sum_{i=1}^{n} (p_i)^2$$

where, p_i stands for the probability of the classes within the variable under consideration.

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 impurity before splitting on the whole data set.
2. Consider any one of the available attributes.
3. Calculate the Gini impurity after splitting on that particular attribute for each of the levels of the attribute.
4. Combine the Gini impurities of all the levels to obtain the Gini impurity of the overall attribute.

5. Repeat steps 2–5 with another attribute until you have exhausted all of them.

6. Compare the Gini impurity across all the attributes and select the one that has the minimum Gini impurity.

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.

Now that you have a fair understanding of both the ID3 and CART algorithms, let’s summarise your key learnings regarding them:

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.

Summary

Implementing Decision Trees

In this session, you learnt about the advantages and disadvantages of decision trees, their implementation in Python and Spark and the suitable methods to overcome some of these disadvantages.

Advantages and Disadvantages of Decision Trees

Decision trees are actively used in the following industries:

- Healthcare
- FinanceE-commerce

You can choose between decision trees and logistic regression based on the following conditions:

- Use decision trees:
    - If the type of data is categorical, as logistic regression will not be able to work with it
    - If the data is linearly non-separable
    - If the data contains outliers
    - If the data set has numerous missing values

- Use logistic regression
    - If the data is continuous
    - After removing outliers from the data set

<u>The advantages of decision trees can be summarised as follows:</u>

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 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.

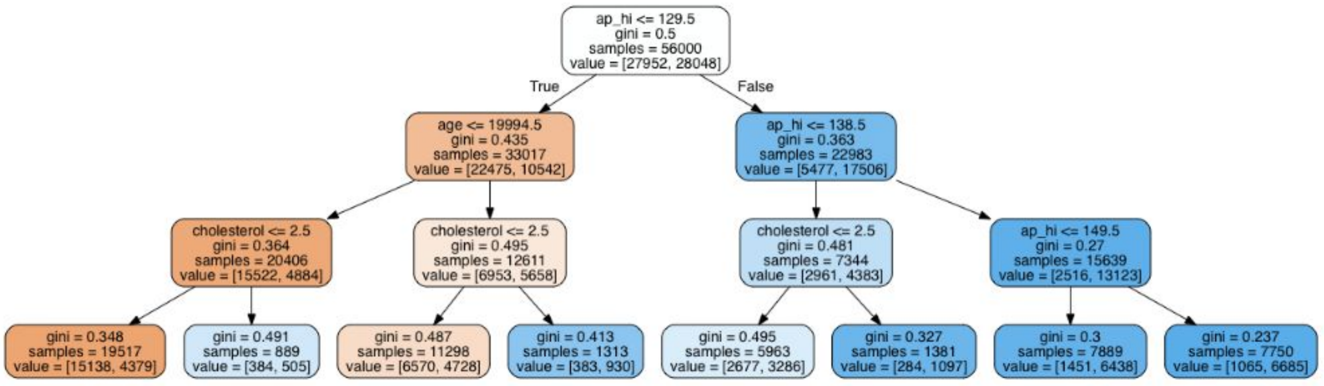
4. Greedy algorithms do not always give a globally optimal decision tree. The process is called greedy because you have to optimise at every split and not overall. The overall information gain in the tree, i.e., the difference between the entropy at a source and that at the leaf nodes, may not be the maximum if you follow the greedy approach, i.e., find the best split at each node. This can be handled using random forests, which will be covered in the next module.

Implementation of Decision Trees in Python

So, you built a decision tree to predict whether a person has a heart disease or not. The data set included the following features:

- id: ID of the patient
- age: Age of the patient (in days)
- gender: Gender of the patient
- height: Height of the patient
- weight: Weight of the patient
- ap_hi: Arterial pressure (upper value)
- ap_lo: Arterial pressure (lower value)
- cholesterol: Patient’s cholesterol level
- gluc: Patient’s blood glucose level
- smoke: Whether the patient smokes or not (0/1)
- alco: 0: Whether the patient drinks alcohol or not (0/1)
- active: Whether the patient is active or not (0/1)
- cardio: Whether the patient has a heart disease or not (0/1)

In this case study, you built a decision tree that was quite huge due to the large number of entries in the data set. Such a tree is prone to overfitting. The final decision tree obtained after using hyperparameters to avoid overfitting is shown in the image given below.



Overfitting and Pruning

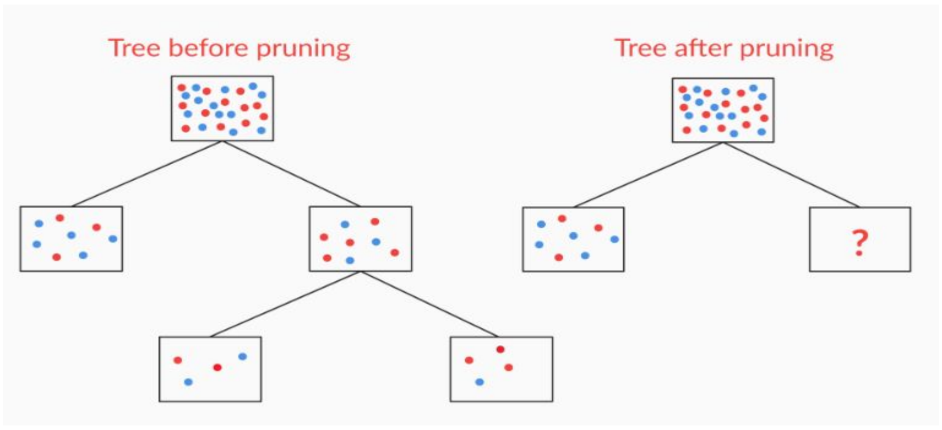
Earlier, you observed that decision trees have a strong tendency to overfit data, which is a major problem. So, you have to pay attention to the size of the tree. A very large tree will have only one leaf to cater to a single data point. Overfitting in decision trees can be controlled using the following 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)
```

Case Study: LIBSVM

In this segment, you used a LIBSVM data set for the case study, which is a regression problem. The LIBSVM data set contains a lot of missing values. Hence, implementing the data set using decision trees is preferable to doing the same using logistic regression. In this case study, you also calculated the mean square error (MSE) using the following formula :

$$MSE = \frac{1}{n}\Sigma\underbrace{\left(y - \widehat{y}\right)^2}_{\text{The square of the difference between actual and predicted}}$$

MSE, in simple terms, is the square of the difference between the actual values and the predicted values divided by the number of observations. You implemented a regression tree in this segment and visualised it.

Case Study: Credit Card Fraud Detection

In this segment, you implemented a classification decision tree case study on Spark. As mentioned at the beginning of this session, decision trees are actively used in the financial sector. You were given a classification problem to predict whether a credit card transaction is fraudulent or not. You dealt with the following features:

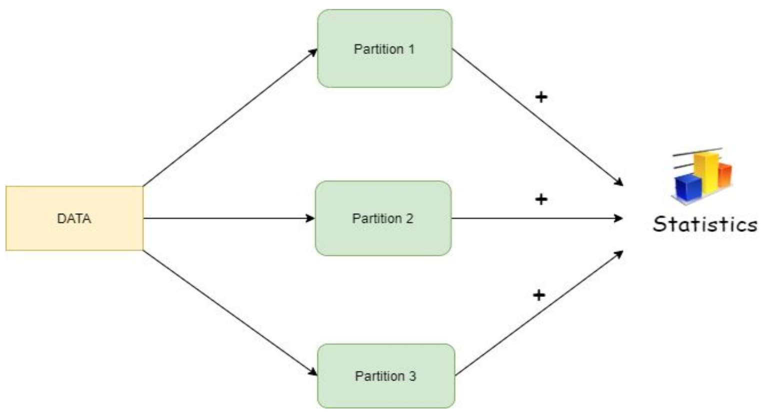
1. type
2. amount
3. nameOrig
4. oldbalanceOrg
5. newbalanceOrig
6. nameDest
7. oldbalanceDest
8. newbalanceDest
9. isFraud
10. isFlaggedFraud

The first step you performed was using a String Indexer. A **string indexer** converts string values into numerical indexes. Next, you created dummy variables (for replacing categorical variables) in models, such as linear and logistic regression. Note that creating dummy variables is also called **one-hot encoding**. Until this point, you converted the string values into indexes and assigned them dummy variables. Next, you created a vector of all the columns/features that you would use to construct a decision tree. This can be done using **Vector Assembler**. Once you prepared the data in the required format, you built a classification tree and visualised the result.

Decision Tree Optimisation in Spark

Working with a large data set may lead to the creation of a huge decision tree. Such a large tree requires a lot of time to execute. To prevent this from happening, the following three main types of optimisations are available in Spark:

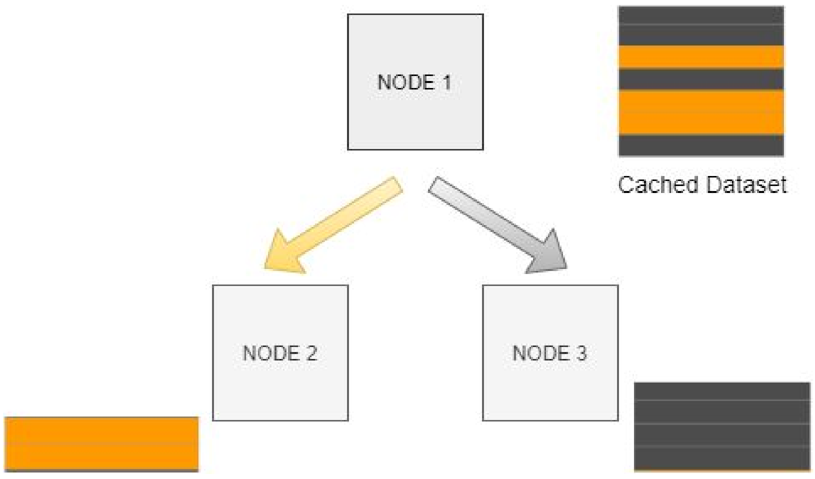
1. **Distributed decision trees:** In the case of a distributed tree, the data is divided into partitions that run in parallel to in worker nodes. These worker nodes compute the statistics and send it to the drive node, which, in turn, aggregates them and decides a splitting criterion for the tree. This process is visualised in the image given below:



Distributed trees divide the load and the data into different partitions that run parallel to one another, thereby reducing the overall time as compared with a single decision tree, which runs alone.

Level-wise training:

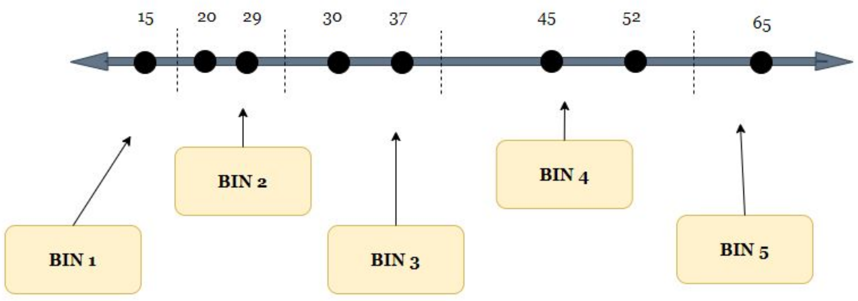
Level-wise training in parallel, as the name suggests, divides the data into different levels that run in parallel. Generally, while using a decision tree, you choose a root node, decide on a splitting criterion and then divide the data based on the splitting criterion as shown in the image given below.



However, when Spark uses level-wise training, rather than passing the data to one node at a time, it passes the data to all the nodes at the same level. This reduces the number of data passes drastically, saving a lot of computation time and resources.

Binning:

Binning comes into play when regression decision trees are used. While working with continuous values, midpoints are computed. If the number of columns with continuous variables is more, then the time required for computation increases drastically. Binning is used at this stage. As the name suggests, binning is the process of creating bins to contain values (a bin may contain more than one value) rather than taking each value separately. Suppose you have the following points: [15,20,29,30,37,45,52,65]. With binning, you divide the points into bins of data. Take a look at the image given below to understand this process better.



Now, rather than computing splitting values for all the bins, you can do the same for each bin. The maxBins parameter is used in a decision tree to decide which value is the most optimal. The optimal value of maxBins can be decided by hyperparameter-tuning. The decision is made based on the following criteria:

1. If maxBins is less: less training time and decreased accuracy
2. If maxBins is large: more training time and increased accuracy