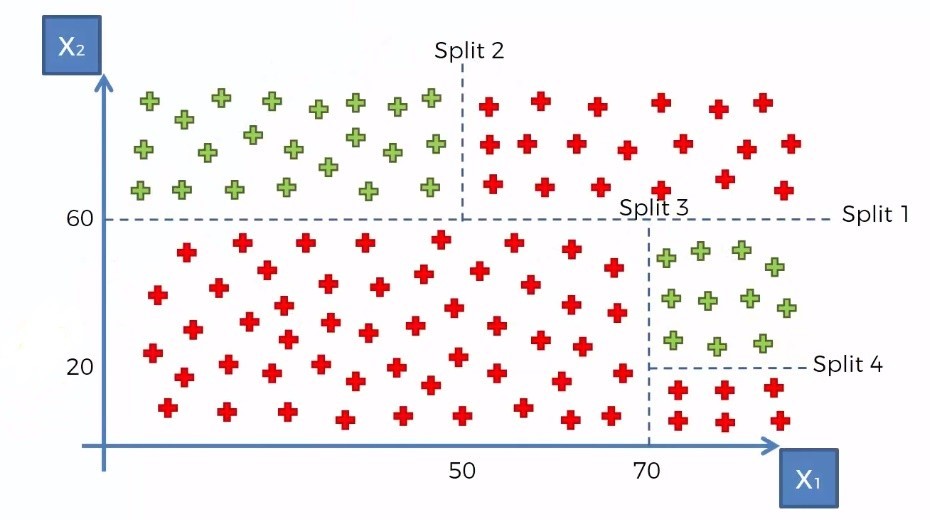
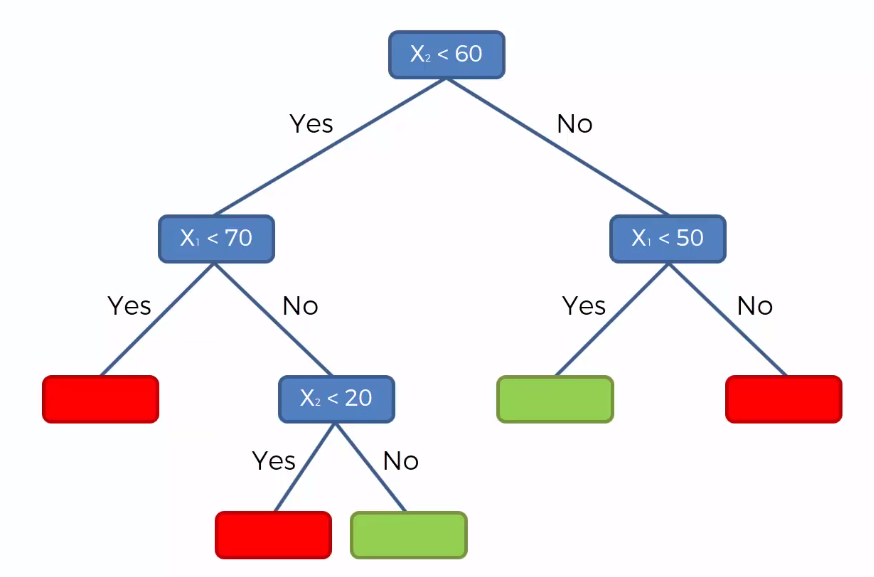
**Decision Trees**

**Introduction To Decision Trees**

Decision trees are tree-based models that help in making a decision in both regression and classification problems. To make a decision, they use a hierarchical structure and split the dataset into smaller subsets.

**For example:** Suppose there is a dataset with two independent variables, X1 and X2, and one dependent variable with two classes, red and green, as shown in the below picture.



Let's look at what the decision tree looks like for this example.  


The decision tree will make the first split where X2<60, the second split is where X1<70, and similarly the third and fourth split are where X2<20 and X1<50, respectively. After making splits 1 and 2, there are only red data points in the left region where X2<60 and X1<70, this region is pure as it has only one type of class. If a region is not pure, it is further split into sub-regions.

Formally, the decision tree splits the data based on different splitting methods. One of the most commonly used methods is Entropy and Information Gain. They are defined as follows:

**Entropy:**Entropy is the measure of randomness or impurity contained in a dataset. Mathematically, it can be written as:

LaTeX: H\left(Y\right)\:=\:-\sum p_i\log_2\left(p_i\right)

Where, LaTeX: Y is the target variable and LaTeX: p_{i}  is the probability of a class LaTeX: i in the data.

**Information gain:** It is the measure of the information gained by adding a feature/independent variable or, in other words, reduction in the impurity after adding a feature. We simply subtract the entropy of Y given X from the entropy of Y to calculate the reduction of impurity about Y given an additional piece of information X.

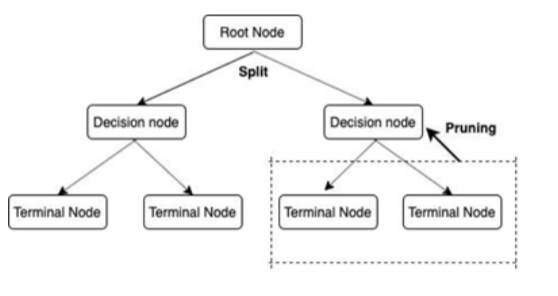
Mathematically, it is given as:

LaTeX: Information\:Gain\:=\:H\left(Y\right)\:-\:H\left(Y\mid X\right)

Where, LaTeX: Y is the target variable, LaTeX: X is the independent variable.

**Important terminology:**

1. **Root Node:**The root node is from where the decision tree starts. It represents the entire population or samples which get divided into two or more branches.
2. **Branch or Sub-Tree:**A part of the entire decision tree is called a branch or sub-tree.
3. **Splitting:**Dividing a node into two or more sub-nodes based on if-else conditions.
4. **Decision Node:**A sub-node that splits into further sub-nodes. In simple terms, every node is a decision node, except for leaf nodes.
5. **Leaf or Terminal Node:**This is the end of the decision tree where it cannot be split into further sub-nodes.
6. **Depth of the tree:**The depth of a decision tree is the number of nodes from the root node down to the furthest leaf node. The below tree has a depth equal to 2.



Construction of Decision Trees

In the previous pre-read, we understood what are decision trees and some important concepts related to decision trees. In this pre-read, we are going to see how a decision tree is constructed using dummy data.

Let’s say we want to solve a classification problem, where we are predicting whether the patient is diabetic or not. The below dataset shows 10 rows with 4 columns where Calories Intake (per day), Sugar Level, and Daily Activity are the independent variables, and Diabetic is the target variable.

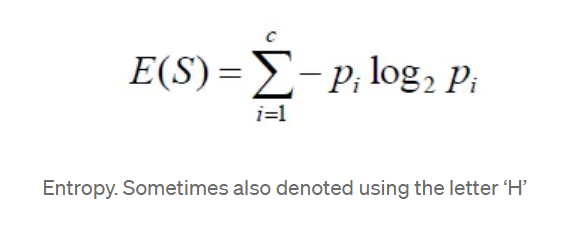
The objective is to classify whether a person is diabetic or not. We can use a decision tree to solve such a problem.



**How do we construct the tree (on which feature/column should we make the first split)?**

As we know from the previous pre-read, we have different splitting methods to decide the split in the decision tree. The feature which has the lowest entropy, i.e., highest information gain is selected as our root node.

We know that the formula of entropy is given as:



Where, LaTeX: p_i is the probability of LaTeX: i^{th} class in the dataset

As we have only 2 classes in the target variable, this formula can also be given as:

LaTeX: H\left(Y\right)\:=\:-p\log_2\left(p\right)-\left(1-p\right)\log_2\left(1-p\right)

Where,

LaTeX: p = The probability of the +ve class

LaTeX: \left(1-p\right) = The probability of the -ve class

By using the above formula, we can calculate LaTeX: H\left(Y\right) for the dependent feature **Diabetic** in the given dataset.

LaTeX: H\left(Y\right)\:=\:-\left(\frac{5}{10}\right)\log_2\left(\frac{5}{10}\right)-\left(1-\frac{5}{10}\right)\log_2\left(1-\frac{5}{10}\right)=1

**Now, let’s calculate the entropy and the information gain for all the independent features in the dataset one by one.**

As we have 3 categories in**Calories Intake(per day),** i.e., High, Moderate, and Low, we can make 3 splits. Hence, for each split, we calculate the entropy.

The entropy of Calories Intake (per day) for a split on High,



The entropy of Calories Intake (per day) for a split on Moderate,



The entropy of Calories Intake (per day) for a split on Low,



Weighted Entropy of Calories Intake (per day) can be given as,



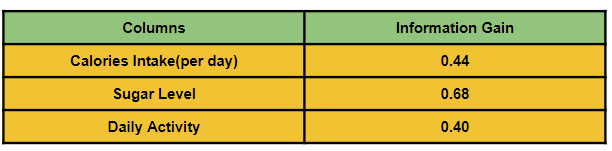
Now, we calculate the Information Gain as follows:



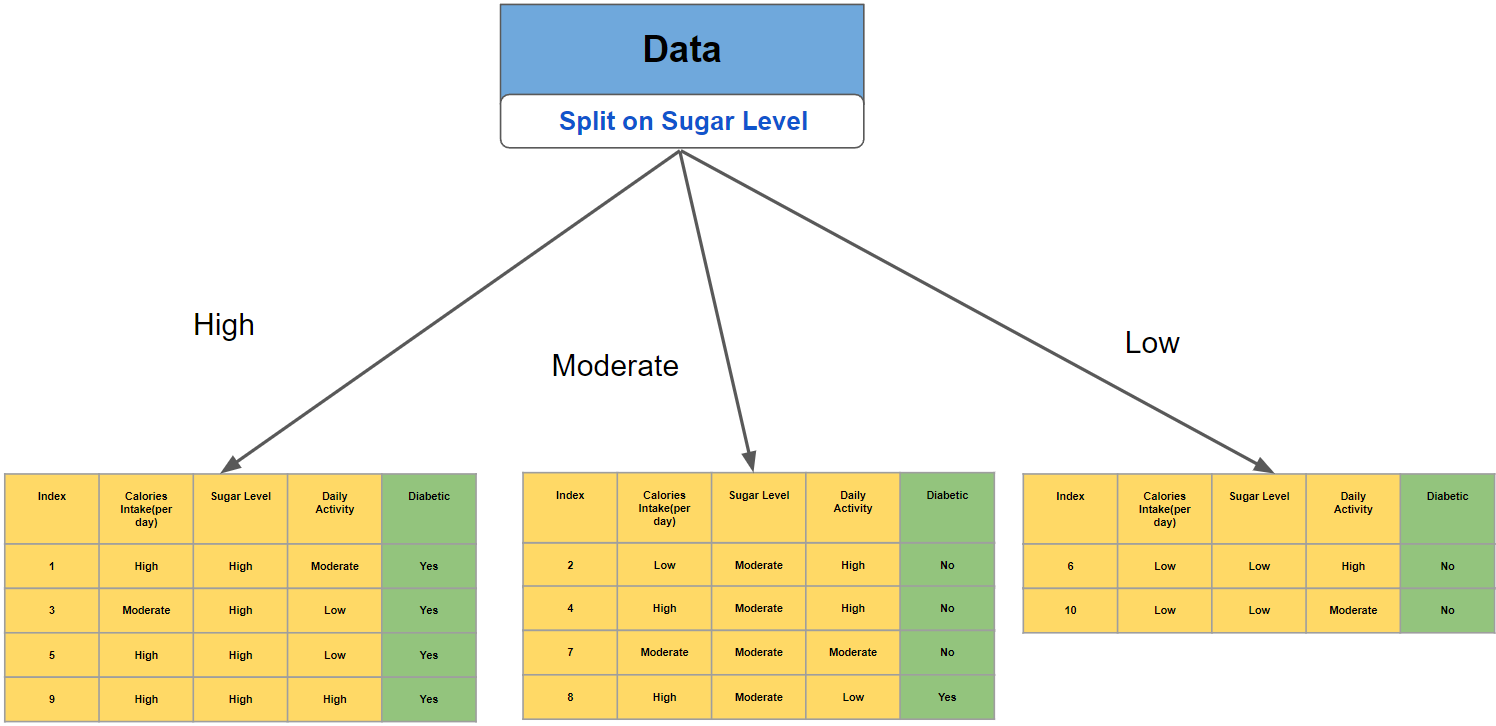
**Similarly, we can calculate the information gain for columns Sugar Level and Daily Activity and we get the following results.**

Information Gain (Sugar Level) = 0.68

Information Gain (Daily Activity) = 0.40



Since the feature Sugar Level has the highest Information Gain, we select Sugar Level as our root node and split the data accordingly.



Likewise, the whole calculation of information gain and splitting of nodes is done recursively for each node in the decision tree to construct the tree. By default, splitting will stop when the leaf nodes become homogeneous/pure (meaning all the data points in that leaf node belong to the same class).

Decision trees only make the optimal split at each node, and the algorithm does not consider the larger problem as a whole. Also, once a split has been made, it is never reconsidered. Decision trees are hence considered **Greedy Algorithms,**which is a Computer Science term for any algorithm that tries to approximate the globally optimal solution to a problem by finding the locally optimal solution at each step of the problem instead.

New Package Introduction - Decision Trees

Now, let’s go through some of the common functions used in this LVC from the Scikit-learn library

#### ****Decision Tree****

A decision tree is a non-parametric supervised learning approach that can be used for classification as well as regression problems. It has a tree structure that is hierarchical and consists of a root node, branches, internal nodes, and leaf nodes.

**Decision Tree Regression**

sklearn.tree.DecisionTreeRegressor(\*, criterion='squared\_error', splitter='best', max\_depth=None, min\_samples\_split=2, min\_samples\_leaf=1, min\_weight\_fraction\_leaf=0.0, max\_features=None, random\_state=None, max\_leaf\_nodes=None, min\_impurity\_decrease=0.0, ccp\_alpha=0.0)[source]

**Example**

from sklearn.tree import DecisionTreeRegressor  
dtree\_regressor = **DecisionTreeRegressor()**  
dtree\_regressor.fit(x\_train**,** y\_train)

You can refer to the decision tree regressor sklearn documentation for a better understanding of the parameters and attributes [here](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html).

Similarly, we can implement the decision tree classifier through the sklearn library.

**Decision Tree Classification**

sklearn.tree.DecisionTreeClassifier(\*, criterion='gini', splitter='best', max\_depth=None, min\_samples\_split=2, min\_samples\_leaf=1, min\_weight\_fraction\_leaf=0.0, max\_features=None, random\_state=None, max\_leaf\_nodes=None, min\_impurity\_decrease=0.0, class\_weight=None, ccp\_alpha=0.0

**Example**

from sklearn.tree import DecisionTreeClassifier  
dtree\_classifier = **DecisionTreeClassifier()**  
dtree\_classifier.fit(x\_train, y\_train)

You can refer to the decision tree regressor sklearn documentation for a better understanding of the parameters and attributes [here](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html).

#### ****Label Encoding****

Sklearn provides a very efficient tool for encoding the levels of categorical features into numeric values. LabelEncoder encodes labels with a value between 0 and n\_classes-1, where n is the number of distinct labels. If a label repeats, it assigns the same value as assigned earlier.

**Example**

from sklearn.preprocessing import LabelEncoder  
ln = **LabelEncoder()**  
ln.fit\_transform(data[‘column’]) # **column** should be any categorical variable

You can refer to the label encoder sklearn documentation for a better understanding of the parameters and attributes [here](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html#:~:text=LabelEncoder%20can%20be%20used%20to%20normalize%20labels.&text=It%20can%20also%20be%20used,and%20comparable)%20to%20numerical%20labels.).

Happy Learning!