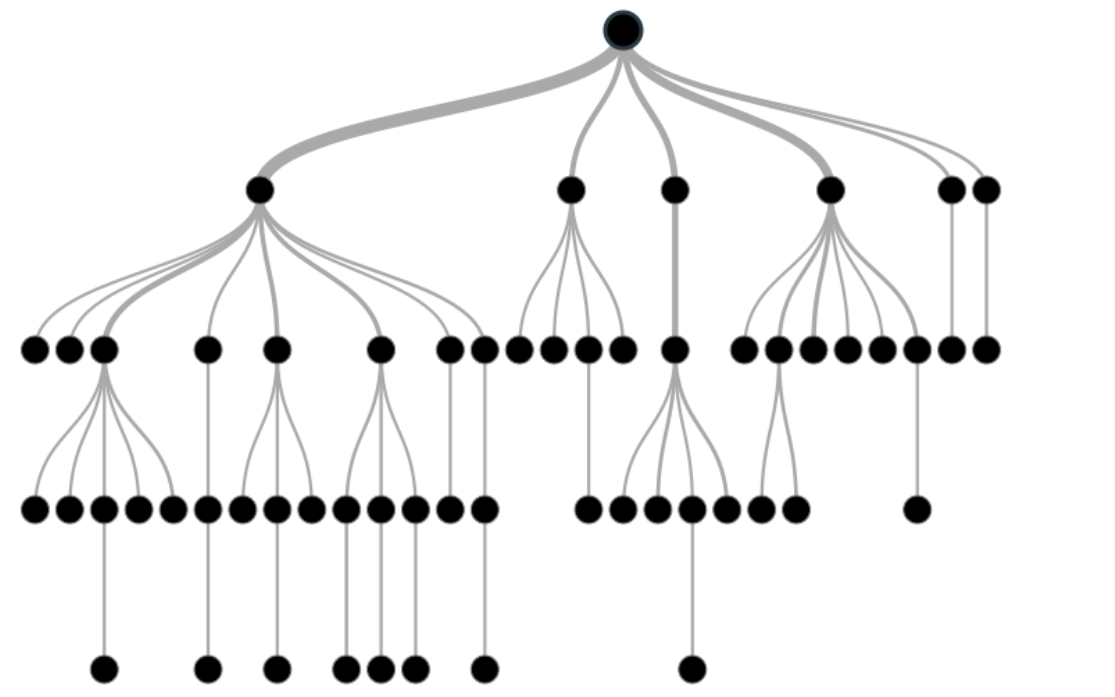
**Decision Tree**

**What is a Decision Tree?**

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





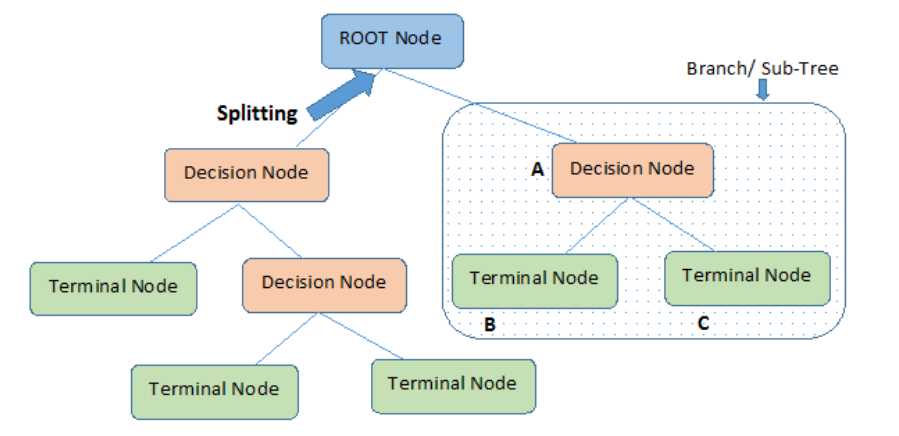
***Root Nodes***– It is the node present at the beginning of a decision tree from this node the population starts dividing according to various features.

***Decision Nodes*** – the nodes we get after splitting the root nodes are called Decision Node

***Leaf Nodes***– the nodes where further splitting is not possible are called leaf nodes or terminal nodes

***Sub-tree*** – just like a small portion of a graph is called sub-graph similarly a sub-section of this decision tree is called sub-tree.

***Pruning* – is nothing but cutting down some nodes to stop overfitting.**



## Contents

1. What is a Decision Tree?

2. Example of a Decision Tree

3. Entropy

4. Information Gain

5. When to stop Splitting?

6. How to stop overfitting?

* max\_depth
* min\_samples\_split
* min\_samples\_leaf
* max\_features

7. Pruning

* Post-pruning
* Pre-pruning

## Example of a decision tree

Let’s understand decision trees with the help of an example.

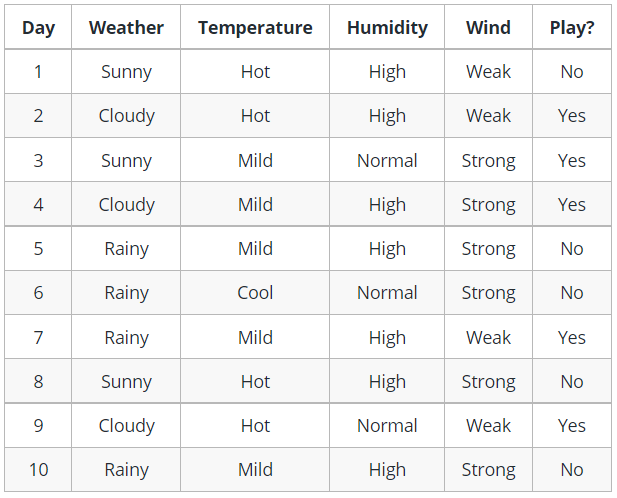


Image 3

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

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

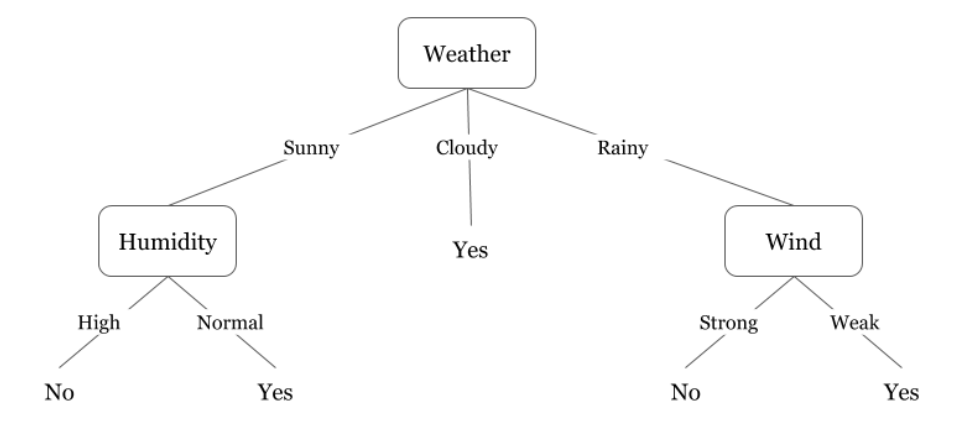


Image 4

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

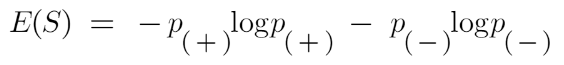
The goal of machine learning is to decrease uncertainty or disorders from the dataset and for this, we use decision trees.

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

## Entropy

Entropy is nothing but the uncertainty in our dataset or measure of disorder.

The formula for Entropy is shown below:



Here p+ is the probability of positive class

p– is the probability of negative class

S is the subset of the training example

## How do Decision Trees use Entropy?

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

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

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

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

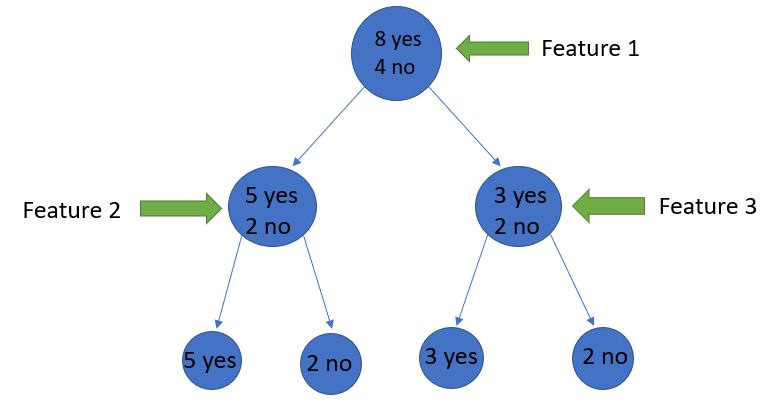
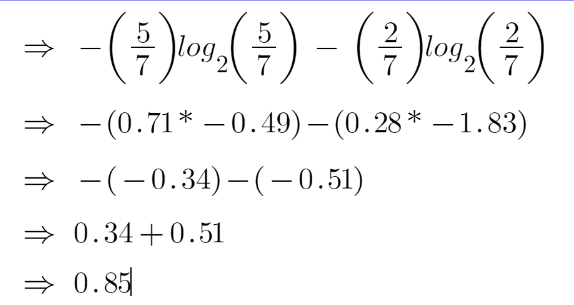
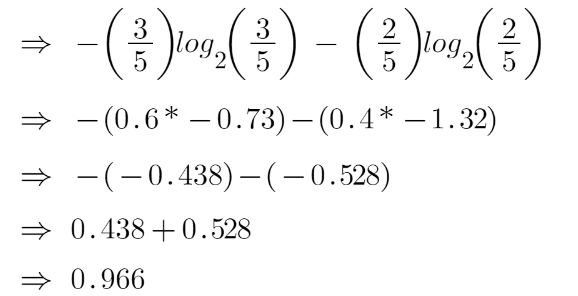


Image Source: Author



For feature 3,



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

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

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

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

## Information Gain

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

information gain Decision tree algorithm

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

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

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

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

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

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

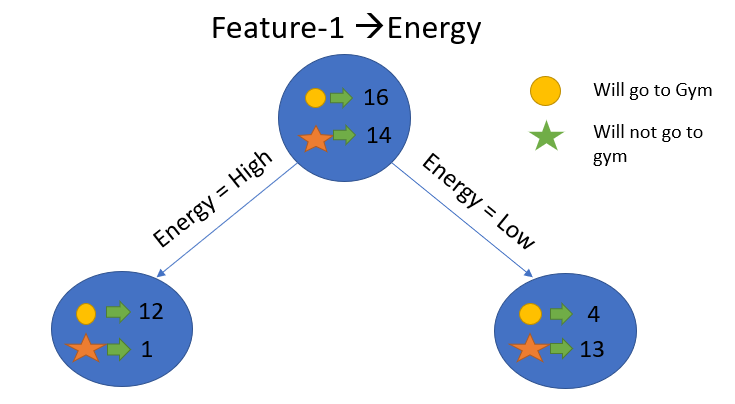
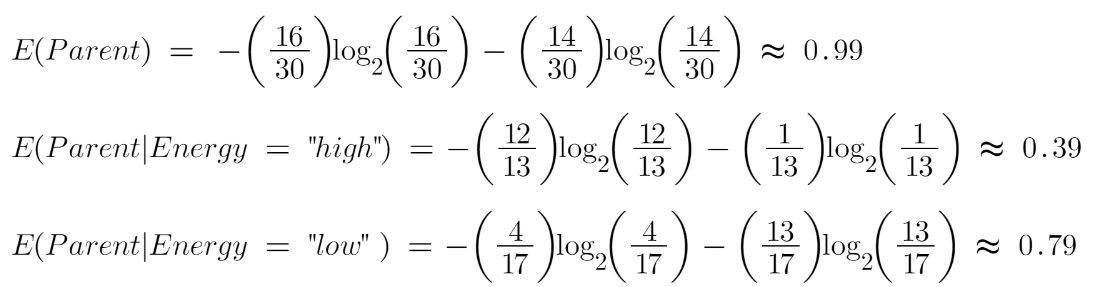
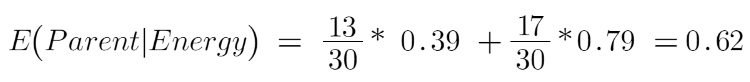


Image Source: Author

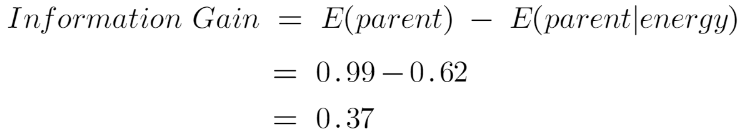
Let’s calculate the entropy:



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



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



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

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

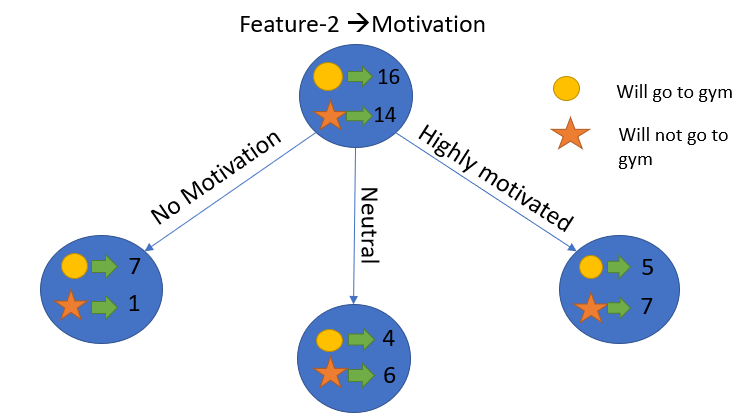
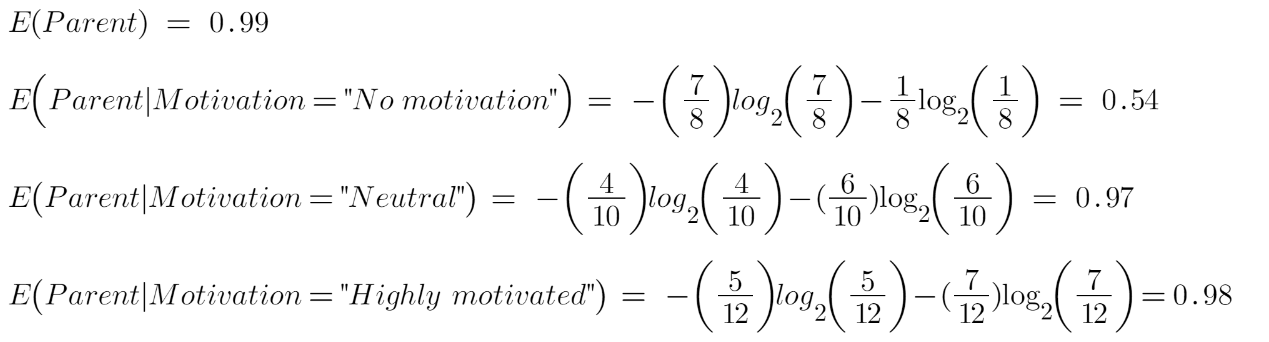
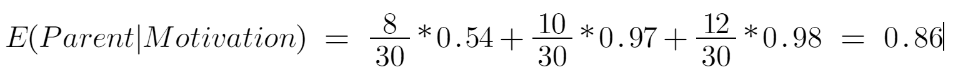


Image Source: Author

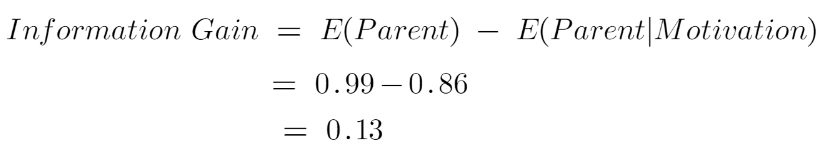
Let’s calculate the entropy here:



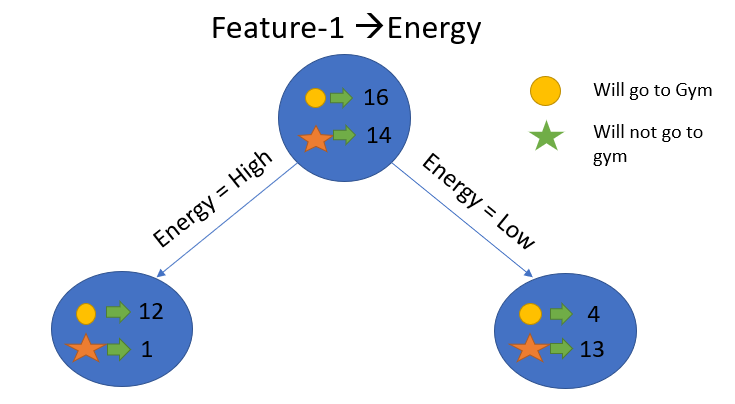
To see the weighted average of entropy of each node we will do as follows:



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



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

Image Source: Author

In this example “Energy” will be our root node and we’ll do the same for sub-nodes. Here we can see that when the energy is “high” the entropy is low and hence we can say a person will definitely go to the gym if he has high energy, but what if the energy is low? We will again split the node based on the new feature which is “Motivation”.

## When to stop splitting?

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

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

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

There are more hyperparameters such as :

***min\_samples\_leaf*** – represents the minimum number of samples required to be in the leaf node. The more you increase the number, the more is the possibility of overfitting.

***max\_features*** – it helps us decide what number of features to consider when looking for the best split.

## Pruning

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

There are mainly 2 ways for pruning:

(i) **Pre-pruning** – we can stop growing the tree earlier, which means we can prune/remove/cut a node if it has low importance **while growing** the tree.

(ii) **Post-pruning** – once our **tree is built to its depth**, we can start pruning the nodes based on their significance.

Gini impurity is an important measure used to construct the decision trees. Gini impurity is **a function that determines how well a decision tree was split**. Basically, it helps us to determine which splitter is best so that we can build a pure decision tree. Gini impurity ranges values from 0 to 0.5

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM.**By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

* **Information Gain**
* **Gini Index**

### **1. Information Gain:**

* Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
* It calculates how much information a feature provides us about a class.
* According to the value of information gain, we split the node and build the decision tree.
* A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

1. Information Gain= Entropy(S)- [(Weighted Avg) \*Entropy(each feature)

**Entropy:** Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

Entropy(s)= -P(yes)log2 P(yes)- P(no) log2 P(no)

**Where,**

* **S= Total number of samples**
* **P(yes)= probability of yes**
* **P(no)= probability of no**

### **2. Gini Index:**

* Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.
* An attribute with the low Gini index should be preferred as compared to the high Gini index.
* It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
* Gini index can be calculated using the below formula:

Gini Index= 1- ∑jPj2

Pruning: Getting an Optimal Decision tree

*Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.*

A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree **pruning**technology used:

* **Cost Complexity Pruning**
* **Reduced Error Pruning.**

Advantages of the Decision Tree

* It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
* It can be very useful for solving decision-related problems.
* It helps to think about all the possible outcomes for a problem.
* There is less requirement of data cleaning compared to other algorithms.

Disadvantages of the Decision Tree

* The decision tree contains lots of layers, which makes it complex.
* It may have an overfitting issue, which can be resolved using the **Random Forest algorithm.**
* For more class labels, the computational complexity of the decision tree may increase.

from sklearn.tree import DecisionTreeClassifier