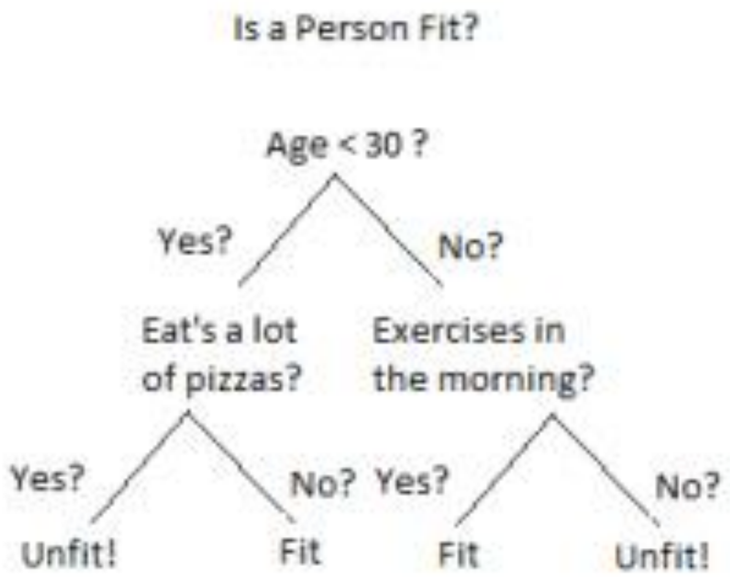
Connect, Message, Like,Follow&Share, 100% Free Counselling➔ThankYou

Follow me on Linkedin

Decision trees

Decision Trees are a type of Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or the final outcomes. And the decision nodes are where the data is split. An example of a decision tree can be explained using above binary tree. Let's say you want to predict whether a person is fit given their information like age, eating habit, and physical activity, etc. The decision nodes here are questions like 'What's the age?', 'Does he exercise?', 'Does he eat a lot of pizzas'? And the leaves, which are outcomes like either 'fit', or 'unfit'. In this case this was a binary classification problem (a yes no type problem).



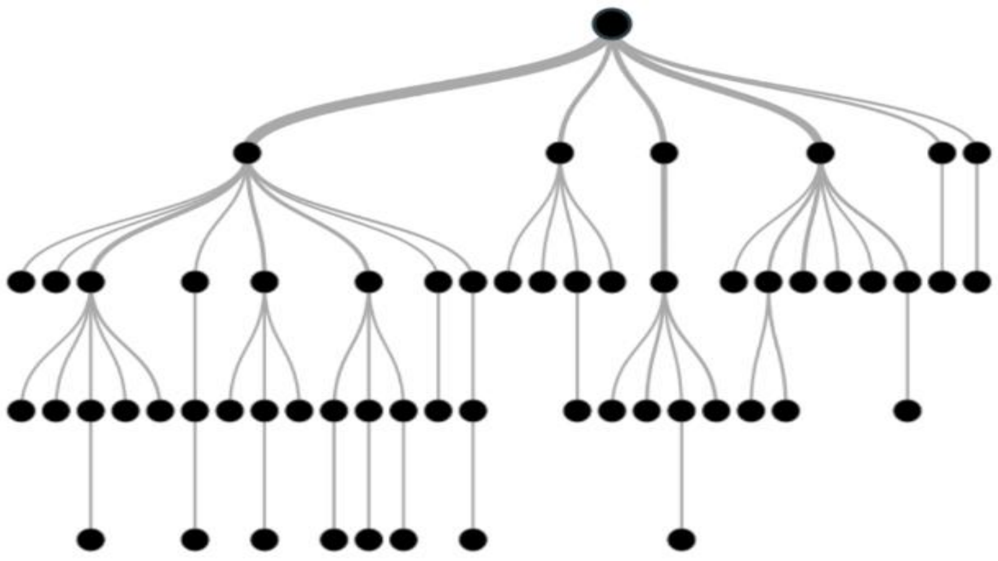
There are two main types of Decision Trees:

1. **Classification trees** (Yes/No types)

What we’ve seen above is an example of classification tree, where the outcome was a variable like ‘fit’ or ‘unfit’. Here the decision variable is **Categorical**.

2. **Regression trees** (Continuous data types)

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.



Before learning more about decision trees let’s get familiar with some of the terminologies.

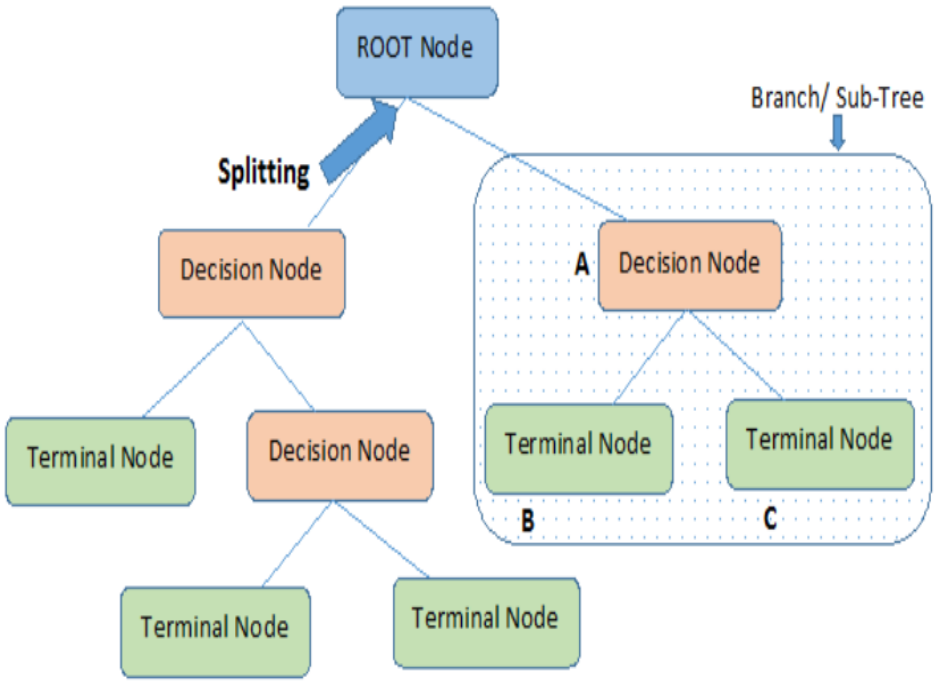
Root Nodes – It is the node present at the beginning of a decision tree from this node the population starts dividing according to various features.

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.

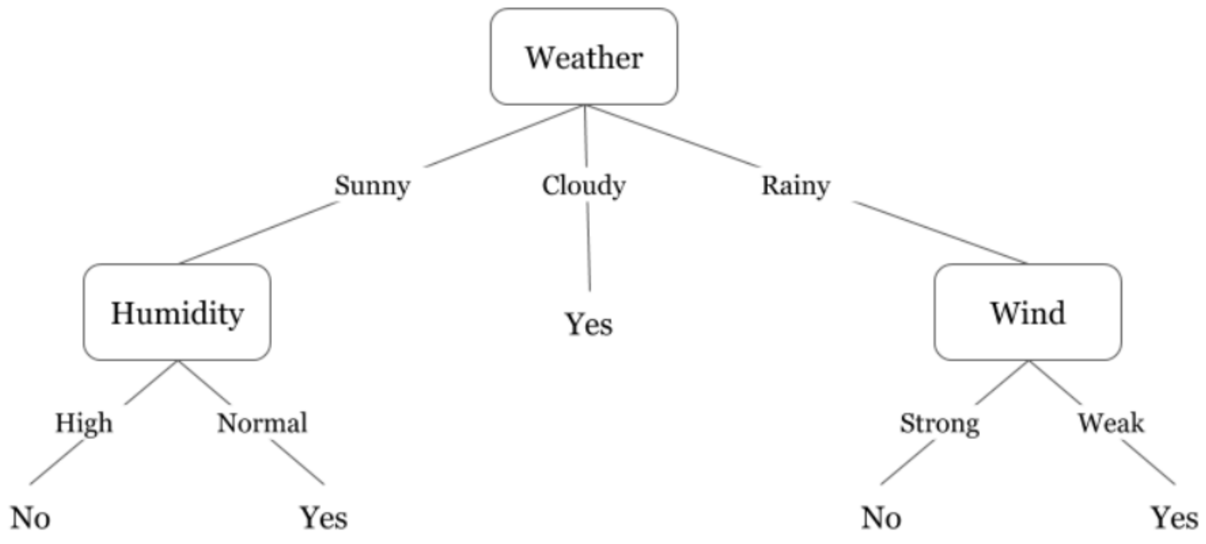


Example of a decision tree

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

Day	Weather	Temperature	Humidity	Wind	Play?
1	Sunny	Hot	High	Weak	No
2	Cloudy	Hot	High	Weak	Yes
3	Sunny	Mild	Normal	Strong	Yes
4	Cloudy	Mild	High	Strong	Yes
5	Rainy	Mild	High	Strong	No
6	Rainy	Cool	Normal	Strong	No
7	Rainy	Mild	High	Weak	Yes
8	Sunny	Hot	High	Strong	No
9	Cloudy	Hot	Normal	Weak	Yes
10	Rainy	Mild	High	Strong	No

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.



Did you notice anything in the above flowchart? We see that if the weather is cloudy then we must go to play. Why didn't it split more? Why did it stop there? 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. Let me try to explain this with the help of an example.

Suppose you have a group of friends who decides which movie they can watch together on Sunday. There are 2 choices for movies, one is ***"Lucy"*** and the second is ***"Titanic"*** and now everyone has to tell their choice. After everyone gives their answer we see that *"Lucy" gets 4 votes* and *"Titanic" gets 5 votes*. Which movie do we watch now? Isn't it hard to choose 1 movie now because the votes for both the movies are somewhat equal.

This is exactly what we call disorderness, there is an equal number of votes for both the movies, and we can't really decide which movie we should watch. It would have been much easier if the votes for "Lucy" were 8 and for "Titanic" it was 2. Here we could easily say that the majority of votes are for "Lucy" hence everyone will be watching this movie.

In a decision tree, the output is mostly "yes" or "no"

The formula for Entropy is shown below:

$$E(S) = -p_{(+)}\log p_{(+)} - p_{(-)}\log p_{(-)}$$

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?

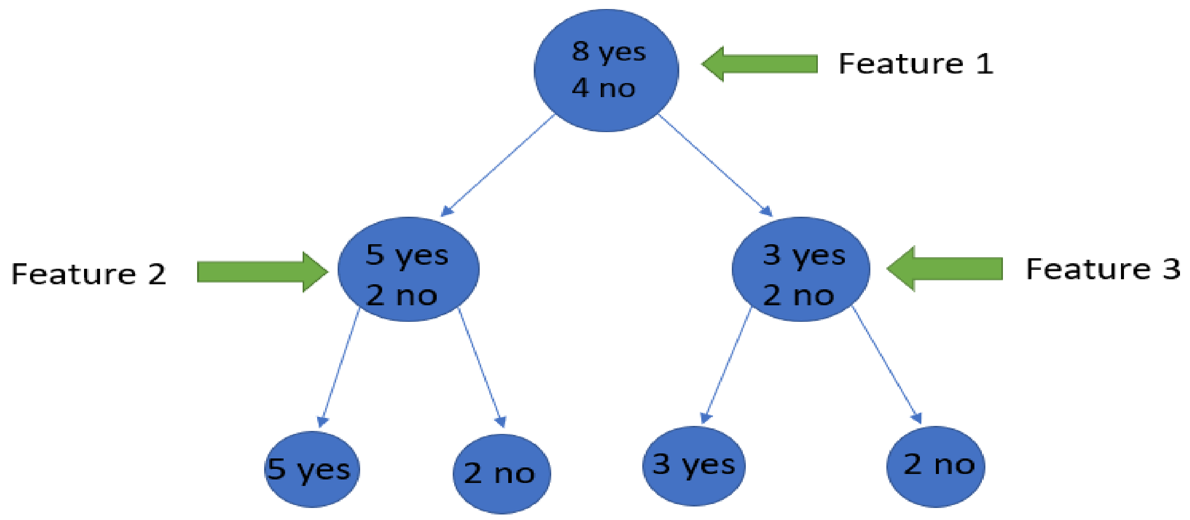
Now we know what entropy is and what is its formula, Next, we need to know that how exactly does it work in this algorithm.

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.



$$\Rightarrow \quad -\left(\frac{5}{7}\right)log_2\left(\frac{5}{7}\right) - \left(\frac{2}{7}\right)log_2\left(\frac{2}{7}\right)$$

$$\Rightarrow \quad -(0.71 * -0.49) - (0.28 * -1.83)$$

$$\Rightarrow \quad -(-0.34) - (-0.51)$$

$$\Rightarrow \quad 0.34 + 0.51$$

$$\Rightarrow \quad 0.85$$

For feature 3,

$$\Rightarrow \quad -\left(\frac{3}{5}\right)log_2\left(\frac{3}{5}\right) - \left(\frac{2}{5}\right)log_2\left(\frac{2}{5}\right)$$

$$\Rightarrow \quad -(0.6 * -0.73) - (0.4 * -1.32)$$

$$\Rightarrow \quad -(-0.438) - (-0.528)$$

$$\Rightarrow \quad 0.438 + 0.528$$

$$\Rightarrow \quad 0.966$$

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.

$$Information\ Gain\ =\ E(Y)\ -\ E(Y|X)$$

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

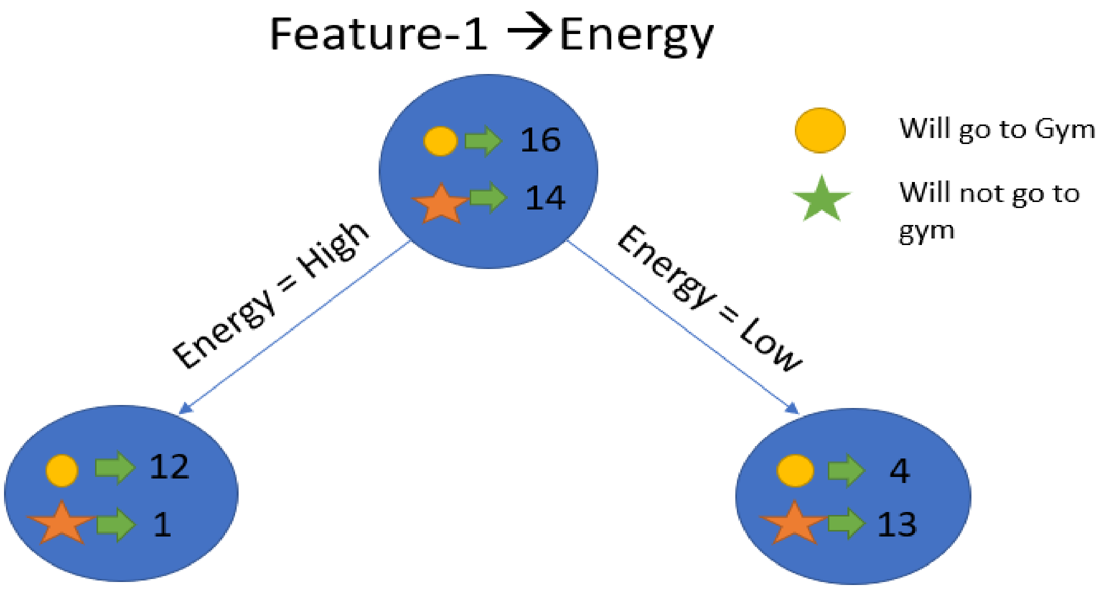
To understand this better let's consider an example:
Suppose our entire population has a total of 30 instances. The dataset is to predict whether the person will go to the gym or not. Let's say 16 people go to the gym and 14 people don't

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.



Let’s calculate the entropy:

$$E(Parent) = -\left(\frac{16}{30}\right)\log_2\left(\frac{16}{30}\right) - \left(\frac{14}{30}\right)\log_2\left(\frac{14}{30}\right) \approx 0.99$$

$$E(Parent|Energy = \text{"high"}) = -\left(\frac{12}{13}\right)\log_2\left(\frac{12}{13}\right) - \left(\frac{1}{13}\right)\log_2\left(\frac{1}{13}\right) \approx 0.39$$

$$E(Parent|Energy = \text{"low"}) = -\left(\frac{4}{17}\right)\log_2\left(\frac{4}{17}\right) - \left(\frac{13}{17}\right)\log_2\left(\frac{13}{17}\right) \approx 0.79$$

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

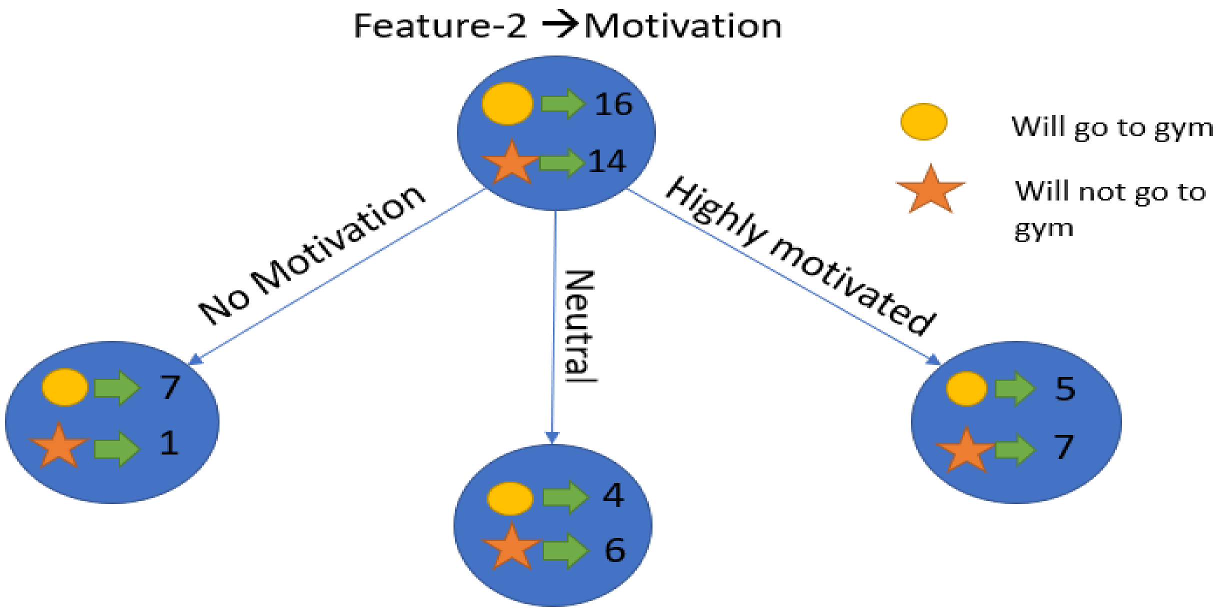
$$E(Parent|Energy) = \frac{13}{30} * 0.39 + \frac{17}{30} * 0.79 = 0.62$$

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

$$Information\ Gain = E(parent) - E(parent|energy)$$
$$= 0.99 - 0.62$$
$$= 0.37$$

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.



Let’s calculate the entropy here:

$$E(Parent) = 0.99$$

$$E\left(Parent|Motivation = "No\ motivation"\right) = -\left(\frac{7}{8}\right)log_2\left(\frac{7}{8}\right) - \frac{1}{8}\log_2\left(\frac{1}{8}\right) = 0.54$$

$$E(Parent|Motivation = "Neutral") = -\left(\frac{4}{10}\right)log_2\left(\frac{4}{10}\right) - \left(\frac{6}{10}\right)\log_2\left(\frac{6}{10}\right) = 0.97$$

$$E(Parent|Motivation = "Highly\ motivated") = -\left(\frac{5}{12}\right)log_2\left(\frac{5}{12}\right) - \left(\frac{7}{12}\right)\log_2\left(\frac{7}{12}\right) = 0.98$$

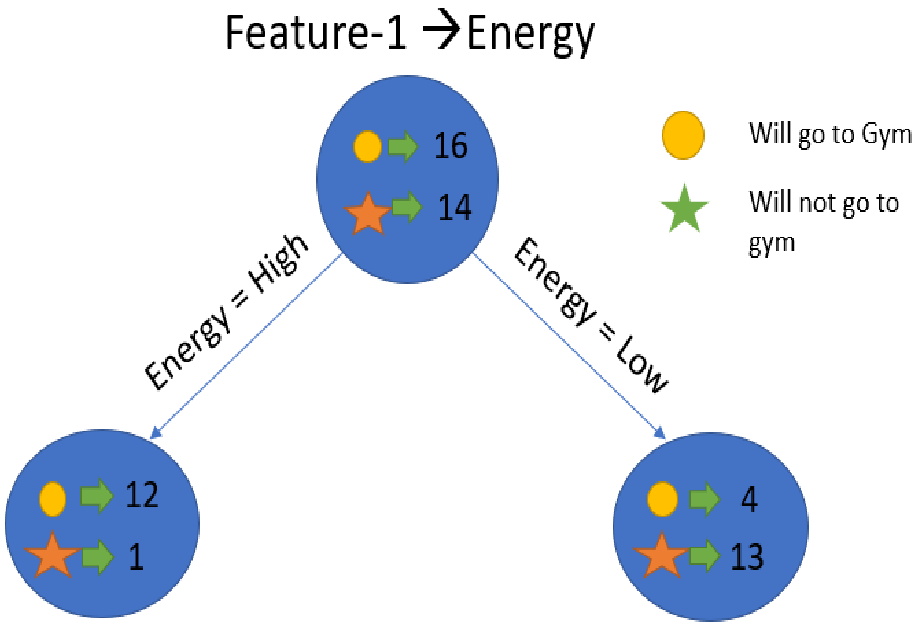
To see the weighted average of entropy of each node we will do as follows:

$$E(Parent|Motivation) = \frac{8}{30} * 0.54 + \frac{10}{30} * 0.97 + \frac{12}{30} * 0.98 = 0.86$$

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

$$\begin{aligned} Information\ Gain &= E(Parent) - E(Parent|Motivation) \\ &= 0.99 - 0.86 \\ &= 0.13 \end{aligned}$$

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.



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.

Here the decision or the outcome variable is **Continuous**, e.g. a number like 123. **Working** Now that we know what a Decision Tree is, we'll see how it works internally. There are many algorithms out there which construct Decision Trees, but one of the best is called as **ID3 Algorithm**. ID3 Stands for **Iterative Dichotomiser 3**. Before discussing the ID3 algorithm, we'll go through few definitions.

- **Entropy:**

Entropy, also called as Shannon Entropy is denoted by H(S) for a finite set S, is the measure of the amount of uncertainty or randomness in

$$H(S) = \sum_{x \in X} p(x) \log_2 \frac{1}{p(x)}$$

data. Intuitively, it tells us about the predictability of a certain event. Example, consider a coin toss whose probability of heads is 0.5 and probability of tails is 0.5. Here the entropy is the highest possible, since there's no way of determining what the outcome might be. Alternatively, consider a coin which has heads on both the sides, the entropy of such an event can be predicted perfectly since we know beforehand that it'll always be heads. In other words, this event has **no randomness** hence it's entropy is zero. In particular, lower values imply less uncertainty while higher values imply high uncertainty.

- **Information Gain:**

nformation gain is also called as Kullback-Leibler divergence denoted by IG(S,A) for a set S is the effective change in entropy after deciding on a particular attribute A. It measures the relative change in entropy with respect to the independent

variables. $IG(S,A) = H(S) - H(S,A)$ Alternatively,

$$IG(S,A) = H(S) - \sum_{i=0}^{n} P(x) * H(x)$$

where IG(S, A) is the information gain by applying feature A. H(S) is the Entropy of the entire set, while the second term calculates the Entropy after applying the feature A, where P(x) is the probability of event x.

Let's understand this with the help of an example. Consider a piece of data collected over the course of 14 days where the features are Outlook, Temperature, Humidity, Wind and the outcome variable is whether Golf was played on the day. Now, our job is to build a predictive model which takes in above 4 parameters and predicts whether Golf will be played on the day. We'll build a decision tree to do that using **ID3 algorithm.**

Day	Outlook	Temperature	Humidity	Wind	Play Golf
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

ID3 Algorithm will perform following tasks recursively

1. Create root node for the tree
2. If all examples are positive, return leaf node 'positive'
3. Else if all examples are negative, return leaf node 'negative'
4. Calculate the entropy of current state H(S)
5. For each attribute, calculate the entropy with respect to the attribute 'x' denoted by H(S, x)
6. Select the attribute which has maximum value of IG(S, x)
7. Remove the attribute that offers highest IG from the set of attributes

8. Repeat until we run out of all attributes, or the decision tree has all leaf nodes.

Now, let's go ahead and grow the decision tree. The initial step is to calculate H(S), the Entropy of the current state. In the above example, we can see in total there are 5 No’s and 9 Yes’s.

Yes	No	Total
9	5	14

$$Entropy(S) = \sum_{x \in X} p(x) \log_2 \frac{1}{p(x)}$$

$$Entropy(S) = -\left(\frac{9}{14}\right)\log_2\left(\frac{9}{14}\right) - \left(\frac{5}{14}\right)\log_2\left(\frac{5}{14}\right)$$

$$= 0.940$$

Remember that the Entropy is 0 if all members belong to the same class, and 1 when half of them belong to one class and other half belong to other class that is perfect randomness. Here it’s 0.94 which means the distribution is fairly random. **Now, the next step is to choose the attribute that gives us highest possible Information Gain** which we’ll choose as the root node. Let’s start with

$$IG(S, Wind) = H(S) - \sum_{i=0}^{n} P(x) * H(x)$$

‘Wind’

where ‘x’ are the possible values for an attribute. Here, attribute ‘Wind’ takes two possible values in the sample data, hence x = {Weak, Strong} We’ll have to calculate:

1. $H(S_{weak})$
2. $H(S_{strong})$
3. $P(S_{weak})$
4. $P(S_{strong})$
5. $H(S) = 0.94$ which we had already calculated in the previous example

_Amongst all the 14 examples we have **8 places where the wind is weak and 6 where the wind is Strong**.

Wind = Weak	Wind = Strong	Total
8	6	14

$$P(S_{weak}) = \frac{Number\ of\ Weak}{Total}$$

$$= \frac{8}{14}$$

$$P(S_{strong}) = \frac{Number\ of\ Strong}{Total}$$

$$= \frac{6}{14}$$

Now, out of the 8 Weak examples, 6 of them were 'Yes' for Play Golf and 2 of them were 'No' for 'Play Golf'. So, we have,

$$Entropy(S_{weak}) = -\left(\frac{6}{8}\right)\log_2\left(\frac{6}{8}\right) - \left(\frac{2}{8}\right)\log_2\left(\frac{2}{8}\right)$$
$$= 0.811$$

Similarly, out of 6 Strong examples, we have **3 examples where the outcome was 'Yes' for Play Golf and 3 where we had 'No' for Play Golf**.

$$Entropy(S_{strong}) = -\left(\frac{3}{6}\right)\log_2\left(\frac{3}{6}\right) - \left(\frac{3}{6}\right)\log_2\left(\frac{3}{6}\right)$$
$$= 1.000$$

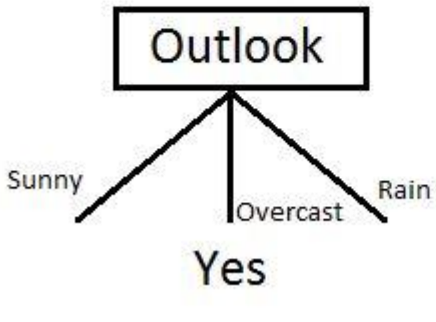
Remember, here half items belong to one class while other half belong to other. Hence we have perfect randomness. Now we have all the pieces required to calculate the Information Gain,

$$IG(S, Wind) = H(S) - \sum_{i=0}^{n} P(x) * H(x)$$
$$IG(S, Wind) = H(S) - P(S_{weak}) * H(S_{weak}) - P(S_{strong}) * H(S_{strong})$$
$$= 0.940 - \left(\frac{8}{14}\right)(0.811) - \left(\frac{6}{14}\right)(1.00)$$
$$= 0.048$$

Which tells us the Information Gain by considering 'Wind' as the feature and give us information gain of **0.048**. Now we must similarly calculate the Information Gain for all the features.

$$IG(S, Outlook) = 0.246$$
$$IG(S, Temperature) = 0.029$$
$$IG(S, Humidity) = 0.151$$
$$IG(S, Wind) = 0.048 \text{ (Previous example)}$$

We can clearly see that IG(S, Outlook) has the highest information gain of 0.246, **hence we chose Outlook attribute as the root node**. At this point, the decision tree looks like.



Here we observe that whenever the outlook is Overcast, Play Golf is always 'Yes', it's no coincidence by any chance, the simple tree resulted because of **the highest information gain is given by the attribute Outlook**. Now how do we proceed from this point? We can simply apply **recursion**, you might want to look at the algorithm steps described earlier. Now that we've used Outlook, we've got three of them remaining Humidity, Temperature, and Wind. And, we had three possible values of Outlook: Sunny, Overcast, Rain. Where the Overcast node already ended up having leaf node 'Yes', so we're left with two subtrees to compute: Sunny and Rain. Next step would be computing $H(S_{sunny})$. Table where the value of Outlook is Sunny looks like:

Temperature	Humidity	Wind	Play Golf
Hot	High	Weak	No
Hot	High	Strong	No
Mild	High	Weak	No
Cool	Normal	Weak	Yes
Mild	Normal	Strong	Yes

$$H(S_{sunny}) = \left(\frac{3}{5}\right)\log_2\left(\frac{3}{5}\right) - \left(\frac{2}{5}\right)\log_2\left(\frac{2}{5}\right) = 0.96$$

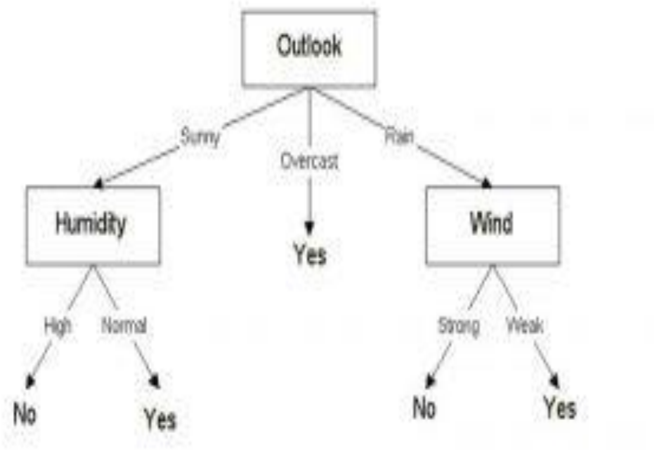
In the similar fashion, we compute the following values

$$IG(S_{sunny}, Humidity) = 0.96$$

$$IG(S_{sunny}, Temperature) = 0.57$$

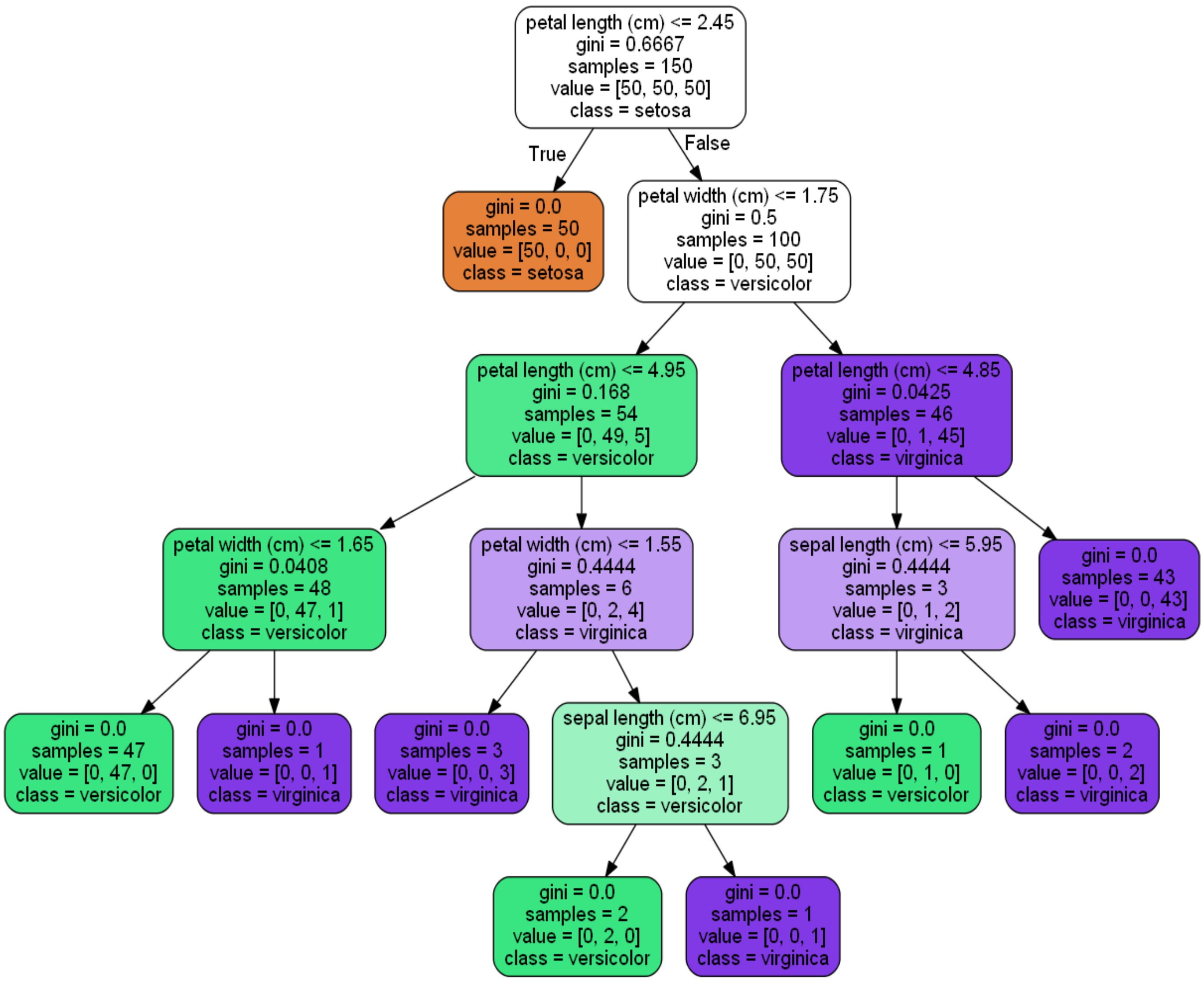
$$IG(S_{sunny}, Wind) = 0.019$$

As we can see the **highest Information Gain is given by Humidity**. Proceeding in the same way with S_{rain} will give us Wind as the one with highest information gain. The final Decision Tree looks something like this.



Code: Let's see an example in Python

```
import pydotplus
 from sklearn.datasets import load_iris
 from sklearn import tree
 from IPython.display import Image, display
__author__ = "Mayur Kulkarni <mayur.kulkarni@xoriant.com>"

 def load_data_set():
    """
    Loads the iris data set

    :return:        data set instance
    """
   iris = load_iris()
    return iris

 def train_model(iris):
    """
    Train decision tree classifier

    :param iris:    iris data set instance
    :return:        classifier instance
    """
   clf = tree.DecisionTreeClassifier()
    clf = clf.fit(iris.data, iris.target)
    return clf

 def display_image(clf, iris):
    """
    Displays the decision tree image

    :param clf:     classifier instance
    :param iris:    iris data set instance
    """
   dot_data = tree.export_graphviz(clf, out_file=None,
                                   feature_names=iris.feature_names,
                                   class_names=iris.target_names,
                                   filled=True, rounded=True)

    graph = pydotplus.graph_from_dot_data(dot_data)
    display(Image(data=graph.create_png()))

 if __name__ == '__main__':
    iris_data = load_iris()
    decision_tree_classifier = train_model(iris_data)
    display_image(clf=decision_tree_classifier, iris=iris_data)
```



Conclusion: Below is the summary of what we've studied in this blog:

1. Entropy to measure discriminatory power of an attribute for classification task. It defines the amount of randomness in attribute for classification task. Entropy is minimal means the attribute appears close to one class and has a good discriminatory power for classification
2. Information Gain to rank attribute for filtering at given node in the tree. The ranking is based on high information gain entropy in decreasing order.
3. The recursive ID3 algorithm that creates a decision tree.
4. ***A Decision Tree is a supervised Machine learning algorithm. It is used in both classification and regression algorithms***. The decision tree is like a tree with nodes. The branches depend on a number of factors. It splits data into branches like these till it achieves a threshold value. A decision tree consists of the root nodes, children nodes, and leaf nodes.

5. Let’s Understand the decision tree methods by Taking one Real-life Scenario
6. Imagine that you play football every Sunday and you always invite your friend to come to play with you. Sometimes your friend actually comes and sometimes he doesn’t.
7. The factor on whether or not to come depends on numerous things, like weather, temperature, wind, and fatigue. We start to take all of these features into consideration and begin tracking them alongside your friend’s decision whether to come for playing or not.
8. You can use this data to predict whether or not your friend will come to play football or not. The technique you could use is a decision tree. Here’s what the decision tree would look like after implementation:

Elements Of a Decision Tree

Every decision tree consists following list of elements:

a Node

b Edges

c Root

d Leaves

a) Nodes: It is The point where the tree splits according to the value of some attribute/feature of the dataset

b) Edges: It directs the outcome of a split to the next node we can see in the figure above that there are nodes for features like outlook, humidity and windy. There is an edge for each potential value of each of those attributes/features.

c) Root: This is the node where the first split takes place

d) Leaves: These are the terminal nodes that predict the outcome of the decision tree

How to Build Decision Trees from Scratch?

While building a Decision tree, the main thing is to select the best attribute from the total features list of the dataset for the root node as well as for sub-nodes. The selection of best attributes is being achieved with the help of a technique known as the Attribute selection measure (ASM).

With the help of ASM, we can easily select the best features for the respective nodes of the decision tree.

There are two techniques for ASM:

a) Information Gain

b) Gini Index

a) Information Gain:

1 Information gain is the measurement of changes in entropy value after the splitting/segmentation of the dataset based on an attribute.

2 It tells how much information a feature/attribute provides us.

3 Following the value of the information gain, splitting of the node and decision tree building is being done.

4 decision tree always tries to maximize the value of the information gain, and a node/attribute having the highest value of the information gain is being split first. Information gain can be calculated using the below formula:

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

Entropy: Entropy signifies the randomness in the dataset. It is being defined as a metric to measure impurity. 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.

b) Gini Index:

Gini index is also being defined as a measure of impurity/ purity used while creating a decision tree in the CART(known as Classification and Regression Tree) algorithm.

An attribute having a low Gini index value should be preferred in contrast to the high Gini index value.

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- $\sum_j P_j^2$

Where pj stands for the probability

4. How Does the Decision Tree Algorithm works?

The basic idea behind any decision tree algorithm is as follows:

1. Select the best Feature using Attribute Selection Measures(ASM) to split the records.

2. Make that attribute/feature a decision node and break the dataset into smaller subsets.

3 Start the tree-building process by repeating this process recursively for each child until one of the following condition is being achieved :

a) All tuples belonging to the same attribute value.

b) There are no more of the attributes remaining.

c) There are no more instances remaining.

Decision Trees and Random Forests

Decision trees and Random forest are both the tree methods that are being used in Machine Learning.

Decision trees are the Machine Learning models used to make predictions by going through each and every feature in the data set, one-by-one.

Random forests on the other hand are a collection of decision trees being grouped together and trained together that use random orders of the features in the given data sets.

Instead of relying on just one decision tree, the random forest takes the prediction from each and every tree and based on the majority of the votes of predictions, and it gives the final output. In other words, the random forest can be defined as a collection of multiple decision trees.

Advantages of the Decision Tree

1 It is simple to implement and it follows a flow chart type structure that resembles human-like decision making.

2 It proves to be very useful for decision-related problems.

3 It helps to find all of the possible outcomes for a given problem.

4 There is very little need for data cleaning in decision trees compared to other Machine Learning algorithms.

5 Handles both numerical as well as categorical values

Disadvantages of the Decision Tree

1 Too many layers of decision tree make it extremely complex sometimes.

2 It may result in overfitting (which can be resolved using the **Random Forest algorithm)**

3 For the more number of the class labels, the computational complexity of the decision tree increases.

Python Code Implementation

Numerical computing libraries and Loading Data

Exploratory data analysis

```
raw_data.info()
sns.pairplot(raw_data, hue = 'Kyphosis')
```

Split the data set into training data and test data

```
from sklearn.model_selection import train_test_split
```

```
x = raw_data.drop('Kyphosis', axis = 1)
y = raw_data['Kyphosis']
x_training_data, x_test_data, y_training_data, y_test_data = train_test_split(x, y,
test_size = 0.3)
```

Train the decision tree model

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model.fit(x_training_data, y_training_data)
predictions = model.predict(x_test_data)
```

Measure the performance of the decision tree model

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
print(classification_report(y_test_data, predictions))
print(confusion_matrix(y_test_data, predictions))
```

Before creating and training our model, first, we have to preprocess our dataset.

Let us start by Importing some important basic libraries.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.simplefilter("ignore")
```

To just ignore warnings that we come across during model creation, just import warnings and set it to ignore.

Seaborn and Matplotlib are two famous libraries that are used for visualizations.

Coming to the dataset, here is the link. Click on it to download.

https://bit.ly/3FxCSC4

Now import it from your local desktop. Paste the path to the news file in the read_csv. Use pandas for it.

```
#import dataset
df=pd.read_csv(r"C:UsersAdminDownloadsnews.csv")
```

Now view it. To view the data frame that you have created simply enter the name of the data frame and run. Instead of showing all the rows, it will give the first five rows and the last five rows.

```
df
```

View the info of the data frame.

```
df.info()
RangeIndex: 6335 entries, 0 to 6334
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  6335 non-null   int64
 1   title       6335 non-null   object
 2   text        6335 non-null   object
 3   label       6335 non-null   object
dtypes: int64(1), object(3)
memory usage: 198.1+ KB
```

Check for null values. Use the is null method for it.

```
df.isnull().sum()
Unnamed: 0    0
title         0
text          0
label         0
```

```
dtype: int64
```

We can see that all values are 0. It shows that there are no null values in the dataset. So we can move further without changing anything in the dataset.

View the shape. This shaping method gives the number of rows and the number of columns in it.

```
print("There are {} rows and {} columns.".format(df.shape[0],df.shape[1]))
```

```
There are 6335 rows and 4 columns.
```

View the statistical description of the data frame. The statistical description shows details like count (non-null values in the column), mean, standard deviation, minimum value, maximum values, and finally percentiles of the values in that column. In our data frame, we have only one numerical column so it will give a description for only one column. But we don’t need this column for building our model.

```
df.describe()
```

	Unnamed: 0
count	6335.000000
mean	5280.415627
std	3038.503953
min	2.000000
25%	2674.500000
50%	5271.000000
75%	7901.000000
max	10557.000000

Drop this unnamed column using the drop method.

```
#Let's drop unimportant columns
```

```
df=df.drop(['Unnamed: 0'],axis=1)
```

Now view the final updated data frame.

```
df
```

Let us see how much Real news and how much Fake news was there in the dataset. The value counts method actually returns the count of a particular variable in a given feature.

```
df['label'].value_counts()
REAL    3171
FAKE    3164
Name: label, dtype: int64
```

There are 3171 Real news and 3164 Fake news. OMG!! probability is almost 0.5 for the news being fake so, be aware of them.

Visualizations

Visualize the count plot using the seaborn library.

```
plt.figure(figsize=(5,10));
sns.countplot(df['label']);
```

In the graph, we can see that both are almost similar.

Define x and y.

For creating a model, first, we have to define x and y in which x contains all independent features of the dataset and y contains a dependent feature that is labelled.

```
x = df.iloc[ : , :-1].values
y = df.iloc[ : , -1].values
```

Import CountVectorizer

As we are working with text data, we have to use a count vectorizer for it. It is actually used to transform our text into a vector on the basis of the frequency of each word which means how many times the word is repeated in the entire text. It is available in the sci-kit learn library.

Fit_transform is used on the training dataset because it will scale our training data and learn the scaling parameters. With this, our model will learn the mean and variance of the features that are there in this training dataset. These parameters are used to work with test data.

```
from sklearn.feature_extraction.text import CountVectorizer
vect=CountVectorizer(stop_words="english",max_features=1000)
x1=vect.fit_transform(x[:,0]).todense()
x2=vect.fit_transform(x[:,1]).todense()
x_mat=np.hstack((x1,x2))
```

Let’s view the final matrix that was created.

```
x_mat
matrix([[0, 0, 0, ..., 0, 2, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

Split the data into train and test datasets

For this, you have to import train_test_split which is available in the scikit learn library.

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test= train_test_split(x_mat,y,random_state=1000)
```

Create Model

Import DecisionTreeClassifier and set criterion to entropy so that our model will build a decision tree taking entropy as criteria and assigning it to a variable called a model. So from now on whenever we need to use DecisionTreeClassifier(criterion=”entropy”), we just have to use the model.

```
from sklearn.tree import DecisionTreeClassifier
model=DecisionTreeClassifier(criterion=”entropy”)
```

Train the model

Train the model using the fit method and pass training datasets as arguments in it.

```
model.fit(x_train,y_train)
```

```
DecisionTreeClassifier(criterion=’entropy’)
```

It’s time to predict the results. Use predict method for it.

```
y_pred=model.predict(x_test)
```

View the predicted results.

```
y_pred
```

```
array([‘FAKE’, ‘REAL’, ‘REAL’, …, ‘REAL’, ‘REAL’, ‘REAL’], dtype=object)
```

Find Accuracy of the Model

Now we will see the accuracy and confusion matrix for the model that we built. For that import accuracy_score and confusion_matrix from the sci-kit learn library.

```
from sklearn.metrics import accuracy_score,confusion_matrix
```

```
accuracy=accuracy_score(y_pred,y_test)*100
print("Accuracy of the model is {:.2f}".format(accuracy))
```

```
The accuracy of the model is 79.61
```

```
confusion_matrix(y_pred,y_test)
```

```
array([[632, 174],
       [149, 629]], dtype=int64)
```

We can see that accuracy of our model is 79.61 per cent.