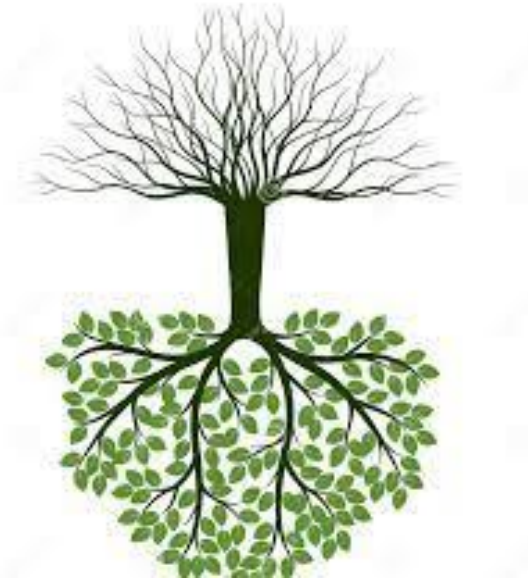
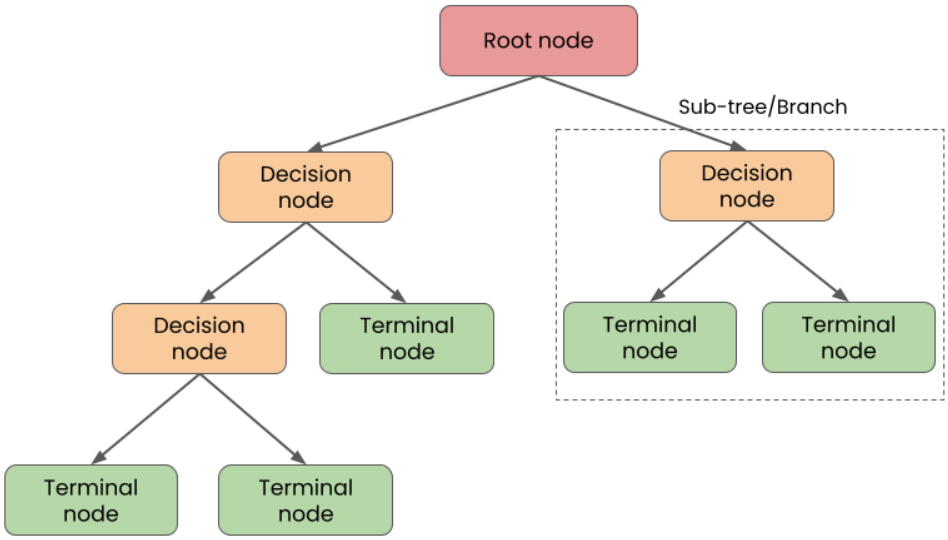
Decision Trees

Hello everyone. I am a self-taught data scientist, and today's topic is decision trees. I will share what I have learned, and I hope you can also gain some insights along with me.

Note: I would like to thank Onur Koç, who has played a significant role in helping me understand the topic as I studied.

Decision trees are non-parametric supervised machine learning algorithms that can be employed for both classification and regression tasks. They are widely used and robust machine learning algorithms in today's context. Visually, they can be represented as upside-down trees.





Before we start, I would like to provide information about the terms used in the diagram on the right:

Root node:

- It is the starting point of the decision tree.
- It is the first node that forms the foundation of the entire tree.
- It is created by dividing the dataset based on a condition related to a specific feature.

Decision node:

- It tests the dataset on a specific feature and divides the data into two or more subsets based on the test result.
- Each subset can be further divided into more subsets with another test at the next node.
- Decision nodes represent the decision rules used for classifying or regressing the dataset.

Terminal/Leaf node:

- After the dataset is divided based on a specific rule or condition, classification or regression results are obtained in these terminal nodes.
- Leaf nodes are the bottommost nodes of the tree and produce the final outcomes. They contain a class or regression value.

Let’s build a decision tree and visualize it to understand the process. We are going to use one of the most popular datasets, the iris dataset.

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from matplotlib import pyplot as plt
from sklearn import tree
import pandas as pd
import numpy as np
```

Firstly, we import the necessary libraries. Then, we split our dataset into independent features and the dependent variable we want to predict. The only parameter we set in our model is max_depth. I will explain this parameter and more in the following.

```
iris = load_iris()
X = iris.data
y= iris.target

tree_clf = DecisionTreeClassifier(max_depth = 3) # max_depth is a parameter we will discuss later on
tree_clf.fit(X, y)
```

```
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df['target'] = iris.target
print(iris_df.head())
```

If you are not familiar with the iris dataset, don't worry. When you run the code on the side, you will receive details about the dataset in the output format on the side.

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   target
0       0
1       0
2       0
3       0
4       0
```

The 'Target' column represents the flower types. We have three different flower types:

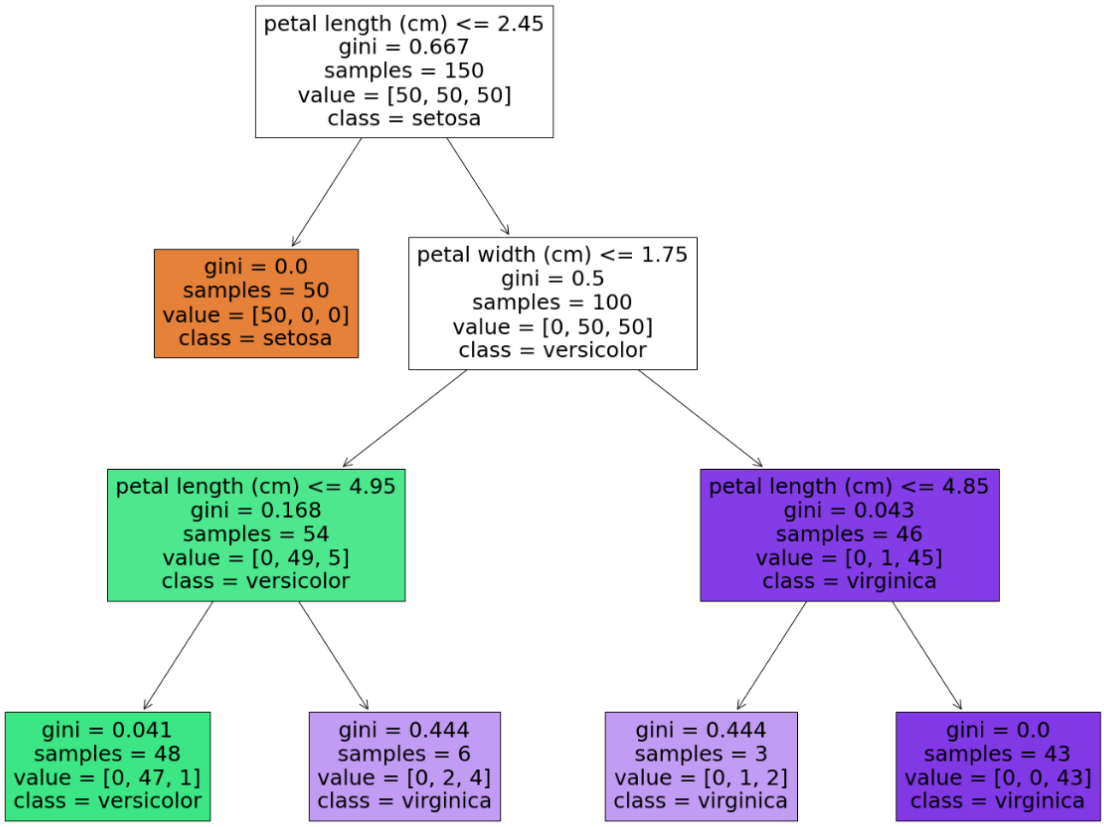
0 = Iris-setosa

1 = Iris-versicolor

2 = Iris-virginica

We've built our first tree. Now, let's visualize it and discuss the visualization.

```
fig = plt.figure(figsize=(25,20))
d = tree.plot_tree(tree_clf,
                   feature_names = iris.feature_names[2:],
                   class_names = iris.target_names,
                   filled = True)
```



Now, with an understanding of the decision tree's working principle, we begin at the root node (depth 0, the first one): this node checks if the petal length (cm) feature is less than or equal to 2.45. If so, we move to the left child node of the root (depth 1, left). In this case, this node serves as a leaf node, indicating it doesn't ask further questions but produces a result. The result classifies data with petal length (cm) values equal to or less than 2.45 as the setosa type.

Exploring data with petal length (cm) values greater than 2.45, we examine the right child node of the root (depth 1, right), which is a decision node introducing a new question. Does our petal width (cm) value exceed 1.75? This question leads us to new decision nodes (depth 2) that, in turn, ask more questions, eventually reaching leaf nodes to classify all our data.

The 'Samples' value indicates the number of examples in that node, while the 'value' list shows the class affiliation of the examples. For example, when observing the depth 1, left node, it informs us that a total of 50 samples are divided, with all 50 belonging to the first class (as seen in the 'class' section).

Understanding the logic of the decision tree, let's now address potential questions that might arise, helping us delve deeper into the working principle of the decision tree, where we will find answers.

- When asking questions, how does it decide which feature to select? For example,
  - Why did it choose the petal length feature at the root node instead of sepal width or petal width?
- When asking questions, how does it decide which feature value to choose? For example,
  - Why did it not choose other values like 1.7 or 2.3 instead of the value 2.45?
- What is the Gini value, and why is it important?

Parameters of Decision Tree

1. Criterion

Criterion : {“gini”,”entropy”,”log_loss”}, default = “gini”

Determines the criterion used to measure the splitting quality of the decision tree. Today, we will talk about Gini and Entropy.

1.1 Gini

Gini Index is a measure of splitting quality used by a Decision Tree algorithm. This index helps assess how homogeneous (containing examples from the same class) or heterogeneous (containing examples from different classes) a dataset is. The Gini Index calculates the homogeneity of a node when it is split using a specific feature and threshold value. Ideally, a node's Gini Index is zero, indicating that all examples belong to the same class. The lower the Gini Index, the better the split, as it signifies higher homogeneity. Gini Impurity ranges between 0 and 0.5.

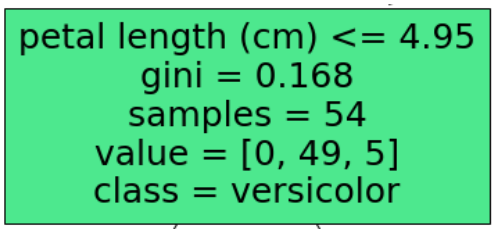
$$Gini = 1 - \sum_{i=1}^{n} p_i^2$$

n: The number of classes

p_i: The percentage of each class in the node

Let’s compute the Gini of the depth 2 left node:

$Gini = 1 - [(0/54)^2 + (49/54)^2 + (5/54)^2 = \mathbf{0.168}$



1.2 Entropy

From a very general perspective, we can define entropy as a measure of the disorder of a system. From this point of view, in our case, it is the measure of impurity in a split.

$$Entropy\ (S) = \sum_{i=0}^{n} - p_i log_2(p_i)$$

n: The number of classes p_i: The percentage of each class in the node

If we had chosen entropy as the criterion, we would have needed to perform the following calculation for the same node (depth 2, left).

petal length (cm) <= 4.95
gini = 0.168
samples = 54
value = [0, 49, 5]
class = versicolor

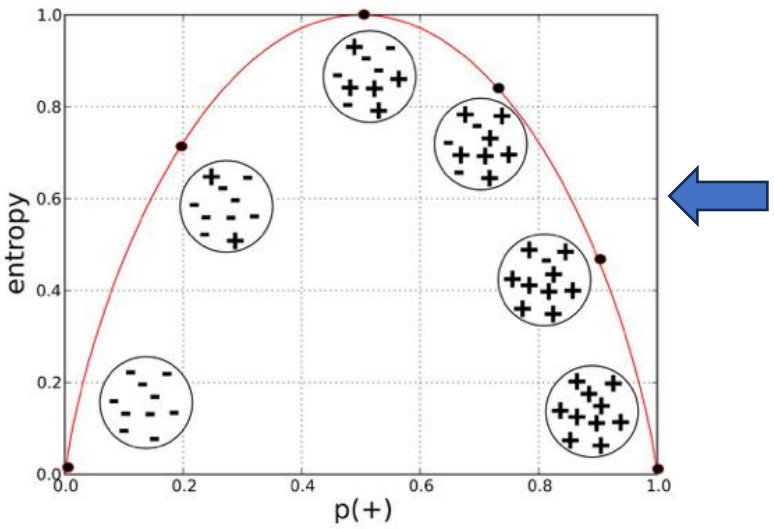
$$Entropy\ (S) = -[(0/54)log_2(0/54) + (49/54)log_2(49/54) + (5/54)log_2(5/54) \approx \mathbf{0,445}$$

To verify, this time we instruct our code to operate with the entropy by inputting the criterion parameter.

```
1  iris = load_iris()
2  X = iris.data
3  y= iris.target
4
5  tree_clf = DecisionTreeClassifier(max_depth = 3, criterion="entropy") # max_depth is a parameter we will discuss later on
6  tree_clf.fit(X, y)
```

petal length (cm) <= 4.95
entropy = 0.445
samples = 54
value = [0, 49, 5]
class = versicolor

As seen, we have obtained the same result. Entropy ranges from 0 to 1. If entropy is 0, consider it a pure sub-tree, and entropy becomes 1 if all labels are equally distributed in a leaf. The obtained value of 0.45 indicates a moderate level of disorder.

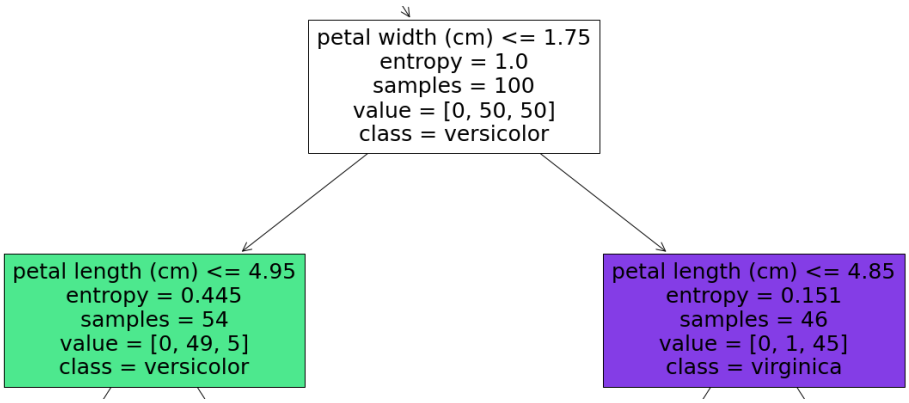


On the adjacent graph, we can better understand the level of disorder corresponding to the varying entropy values.

Information Gain

Information Gain measures how well a feature or a set of features can split or classify a dataset.

$$Gain(S, A) = Entropy(S) - \sum_{i \in values(A)} \frac{|S_i|}{|S|} Entropy(S_i)$$



Let's calculate the information gain for the second split.

$$Information\ Gain = 1 - [(\tfrac{54}{100}) \times 0{,}445 + (\tfrac{46}{100}) \times 0{,}151)] \approx 0{,}69$$

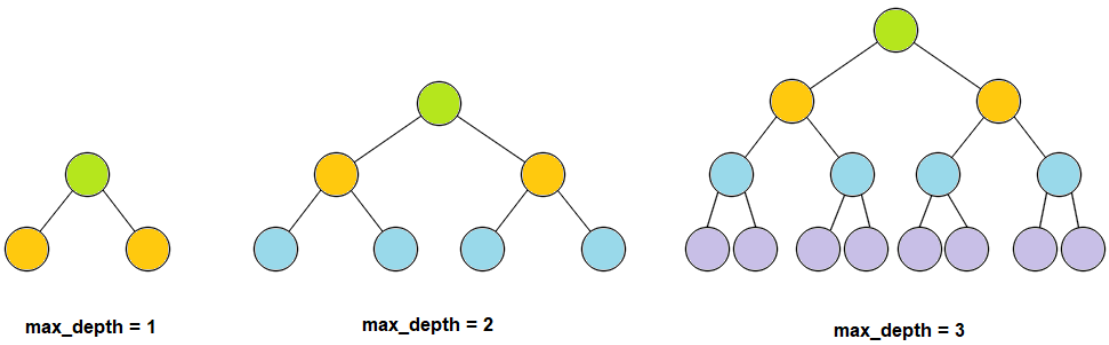
Information Gain value:

- 0: The classes in the node are completely homogeneous, indicating a clear separation.
- 1: The classes in the node are completely mixed and not homogeneous.

The Information Gain value of 0.69 indicates that it has slightly reduced the uncertainty between the classes in the node, but the classes are still not entirely homogeneous.

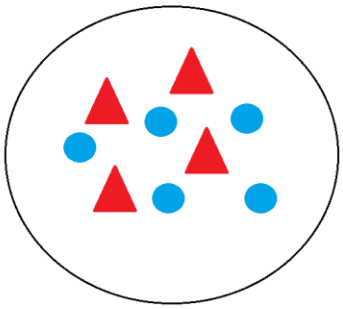
2. max_depth

As you may recall, we used the max_depth parameter at the beginning of the text. This parameter determines the maximum depth of the decision tree. It controls how deep the tree can grow. A larger max_depth results in a more complex and detailed tree, but it may increase the risk of overfitting.



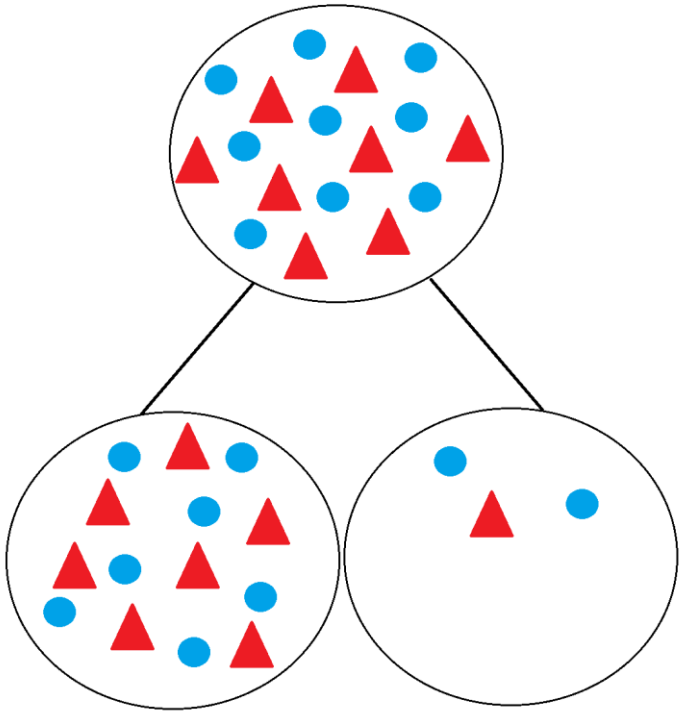
3. min_samples_split

It sets the minimum number of samples required to split a node. This parameter restricts further divisions in the tree and can help reduce the risk of overfitting.



min_samples_split = 10

NO SPLIT



min_samples_split = 10

4. max_features

It is a parameter that determines the maximum number of features to consider at each split step of a decision tree. This parameter is used to control how many features the model will consider in each split step. It is particularly useful for large datasets. Let's say our dataset has 50 different features, and we set our parameter as max_features = 10. Before each split, the model randomly selects 10 features and chooses the best one from these 10 features. It's a parameter that can be adjusted to prevent overfitting.

5. class_weight (for classification problems)

The main reasons for using `class_weight` are as follows:

- Balancing classes in imbalanced datasets: If some classes in your dataset have fewer examples than others, you can use class weights to assign more weight to minority classes, allowing the model to better learn these classes.
- Giving more importance to specific classes: If you want certain classes to have a greater impact on the model's learning, you can assign higher weights to these classes.

Typically, the `class_weight` parameter is used in two ways:

- `class_weight="balanced"`: This option automatically determines class weights. The weights are calculated inversely proportional to the frequency of each class in the dataset. This ensures the automatic assignment of appropriate weights when there is an imbalance among classes.
- Manual Specification (`class_weight={0: 1, 1: 2}`): Users can manually set the weights of classes. This is useful, especially when prioritizing a specific class or correcting an imbalance situation.

6. sample_weight

The sample_weight parameter is used to determine the importance of each individual example (data point). For instance, when developing a medical diagnosis model, you may believe that the diagnosis for some patients is more critical than others.

For example, consider the following scenarios:

Example 1 (Patient A): He/she has a critical condition, and accurate diagnosis is crucial.

Example 2 (Patient B): He/she has a less critical condition, and accurate diagnosis is important but not a top priority.

By using sample_weight , you can assign higher weight to Example 1, which helps the model pay more attention to diagnosing critical cases.

The pros and coins of decision trees:

Some advantages of decision trees are:	**The disadvantages of decision trees include:**
Simple to understand and to interpret. Trees can be visualized.	Decision-tree learners can create over-complex trees that do not generalize the data well. This is called overfitting. Mechanisms such as pruning, setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
Requires little data preparation. Other techniques often require data normalization, dummy variables need to be created and blank values to be removed. Some tree and algorithm combinations support missing values.	Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.
The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.	Predictions of decision trees are neither smooth nor continuous, but piecewise constant approximations. Therefore, they are not good at extrapolation.
Able to handle both numerical and categorical data.	The problem of learning an optimal decision tree is NP-complete, so practical algorithms often make locally optimal decisions. These algorithms cannot guarantee the globally best decision tree.
Able to handle multi-output problems.	There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.
Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic.	Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.
Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.	
Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.	