Decision Trees

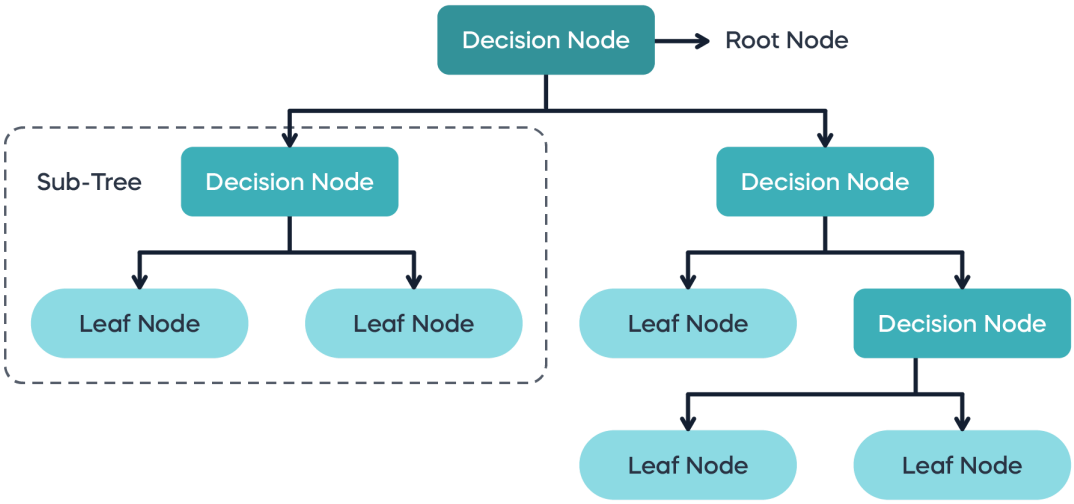
The problem

Given an input dataset consisting in a cartesian product of m attributes $X = A_1 \times A_2 \times ... \times A_m$, and a set of classes C, we want to find a function such that $f : X \to C$.

What is a decision tree

A decision tree is made by:

- **Nodes**, which are the tests for attributes.
- **Branches**, which represent the possibile values for each attribute.
- **Leaves**, which represent the classes.



We define H as the hypothesis space, namely it is the set of all possible decision trees.

A great advantage of decision tree is that they are *expleainable*, meaning that their policy is fully exposed, and can be represented as a formula made by conjunctive rules disjoint by each other.

For example:

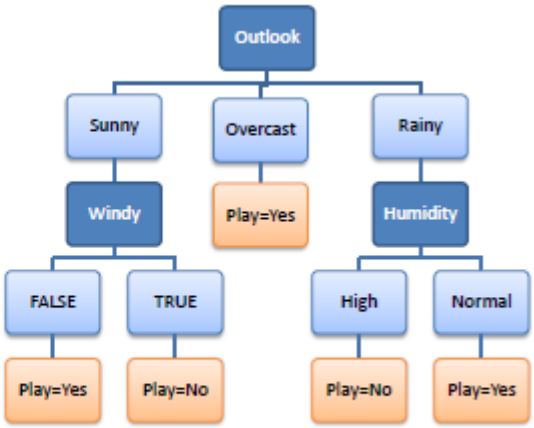
R_1: IF (Outlook=Sunny) AND (Windy=FALSE) THEN Play=Yes

R_2: IF (Outlook=Sunny) AND (Windy=TRUE) THEN Play=No

R_3: IF (Outlook=Overcast) THEN Play=Yes

R_4: IF (Outlook=Rainy) AND (Humidity=High) THEN Play=No

R_5: IF (Outlook=Rain) AND (Humidity=Normal) THEN Play=Yes



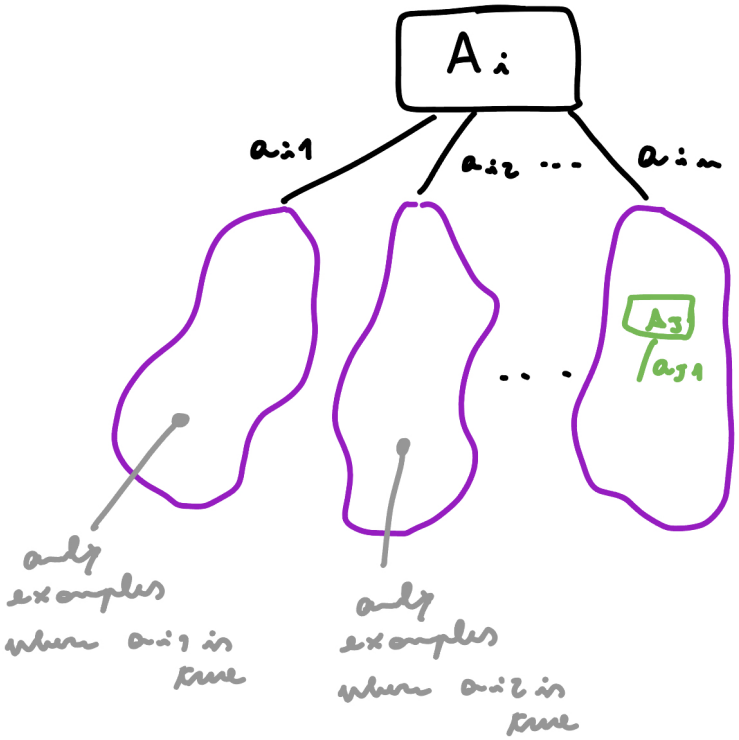
Computing a decision tree: the ID3 algorithm

The ID3 algorithm takes as an *input* three sets: Examples (dataset), Target Attributes (classes), Input Attributes (input) and outputs a decision tree. The algorithm works in a recursive way.

The base cases for this algorithm are.

- All examples are positive → return a leaf node with a positive label.
- All examples are negative → return a lead node with a negative label.
- There are positive and negative examples. but the list of attribute is empty (An example could be the absence of a person among the population with age over 100 years)→ choose the leaf node giving it the label that is most abundant in the dataset (if we have more positives than negatives, then put a positive label, and vice-versa).
- In the case that we have both positives and negatives examples and the list of attributes is not empty, we follow these steps:
  1. Choose the optimal attribute A_i (optimality will be discussed)
  2. Create a node with A_i
  3. Add branches and childrend with that attribute

4. Recursively repeat these steps for its childrens

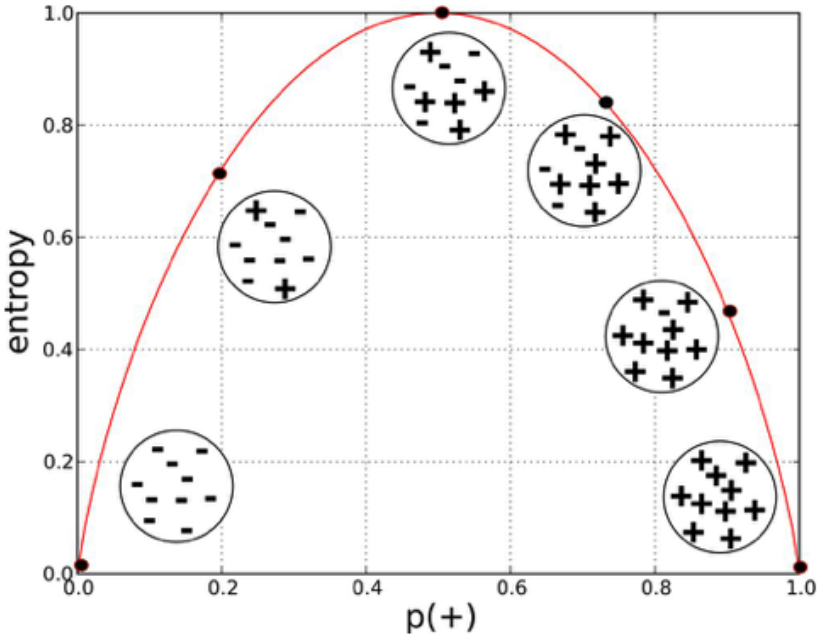


Finding the best split

The problem of choosing each time the best attribute for a split is crucial, since the same decision tree could be shorter or much longer due to this choice. We want to find the optimal one given that our policy is that the best decision tree is the most compact one.

In order to achieve this, we refer to two metrics: entropy and information gain.

Entropy gives an idea of how much examples are balanced inside a certain population. For example if we have two classes, entropy follows this curve:



Notice that entropy is maximum (1) when there is exactly the same number of positive and negative examples, while it is minimum (0), when we have just positives or negatives.

When choosing the best split we seek the minimum entropy, since this will direction us towards the most compact decision tree.

Information gain

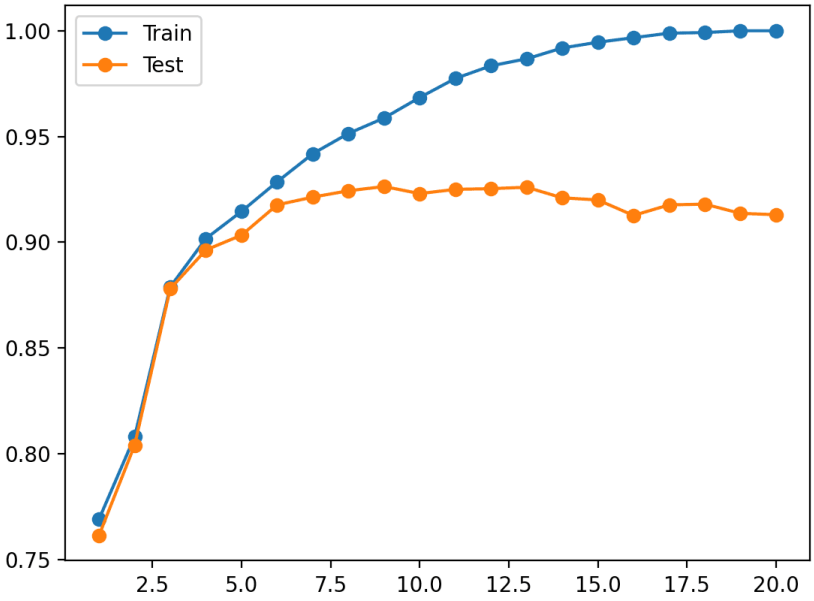
Given a dataset S and an attribute A, the information gain measures the reduction of entropy when splitting S according to A:

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

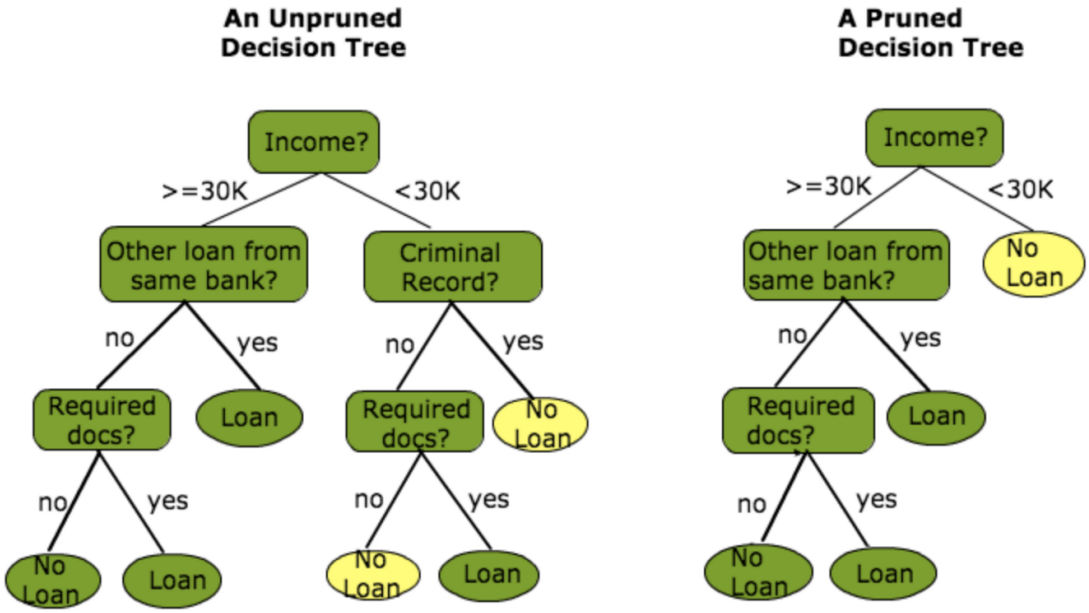
Our algorithm will choose each time the attribute that brings the highest information gain, thus the “best split”.

Evaluating decision trees: the problem of overfitting

When feeding a decision tree, we will always obtain that train accuracy will increase steadily. It happens that there is always a certain point where test accuracy will start dropping, no matter if train accuracy keeps growing. Starting from this point, we happen to be in an overfitting region, meaning that our decision tree will lose its ability to generalize predictions:



A possible approach to correct this phenomena is to perform **post-pruning**. Basically we let the training run, and once finished we start cutting parts of the tree, thus performing backtracking on the error. The logic is that we will keep pruning each time we will obtain a new test accuracy that is higher than the previous one. We will stop only when performance will start dropping once again (we reached a local minimum).



Remember that pruning actually means removing constraints form the rule representation of a decision tree.