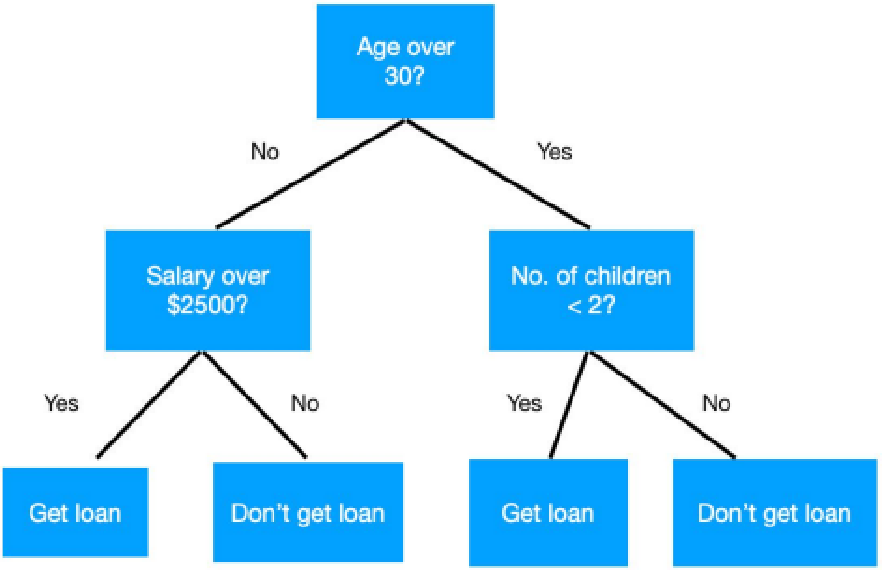
A Decision Tree (DT) is a supervised learning algorithm used for classification (spam/not spam) as well as regression (pricing a car or a house) problems.

A decision tree is like a **flow chart** where each **internal node represents a test** on an attribute and each branch represents the outcome of that test. In a classification problem, each **leaf node** represents a class label, i.e., the decision was taken after computing all attributes, and the path from the first node to a leaf represents **classification rules**, also called **decision rules.**

Let's consider a simple example of **"who gets a loan"**? Here, the decision tree represents a sequence of questions that the bank might ask an applicant in order to figure out whether that applicant is eligible for the loan or not. Each internal node represents a question and each leaf node represents the class label - Get loan/ Don't get loan. As we move along the edges, we get decision rules, for example, if an applicant's age is under 30 and has a salary less than $2500, then the applicant won't get a loan.



Decision trees are one of the most famous supervised learning algorithms. They have many advantages but a few limitations as well.

Advantages of Decision Trees:

- Human-Algorithm Interaction
    - Simple to understand and interpret
    - Mirrors human decision making more closely
    - Uses an open-box model i.e. can visualize and understand the machine learning logic (as opposed to a black-box model)

- Versatile
  - Able to handle both numerical and categorical data
  - Powerful - can model arbitrary functions as long as we have sufficient data

  - Requires little data preparation
  - Performs well with large datasets

- Built-in feature selection
  - Naturally de-emphasizes irrelevant features
  - Develops a hierarchy in terms of the relevance of features

- Testable: Possible to validate a model using statistical tests

Limitations of decision trees:

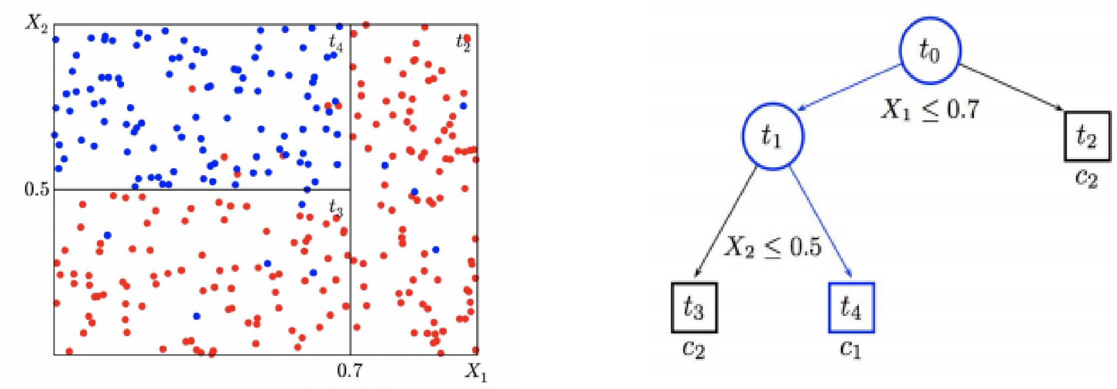
- Trees can be non-robust: A small change in the training data can result in a large change in the tree and consequently the final predictions.

- The problem of learning an optimal decision tree is known to be NP-Complete
  - Practical decision-tree learning algorithms are based on heuristics (greedy algorithms)
  - Such algorithms cannot guarantee to obtain the globally optimal decision tree.

- Overfitting: Decision-tree learners can create over-complex trees that do not generalize well from the training data.

Before we move further, let's answer a simple question: **Why do we need decision trees? Why can't we use linear classifiers?**

Consider a scenario for classification. Let two continuous independent variables be X1 and X2 and the dependent variable be either Red or Blue points. The decision tree is built top-down from a root node and involves partitioning the data into subsets that contain instances with similar values (homogeneous). The data points,

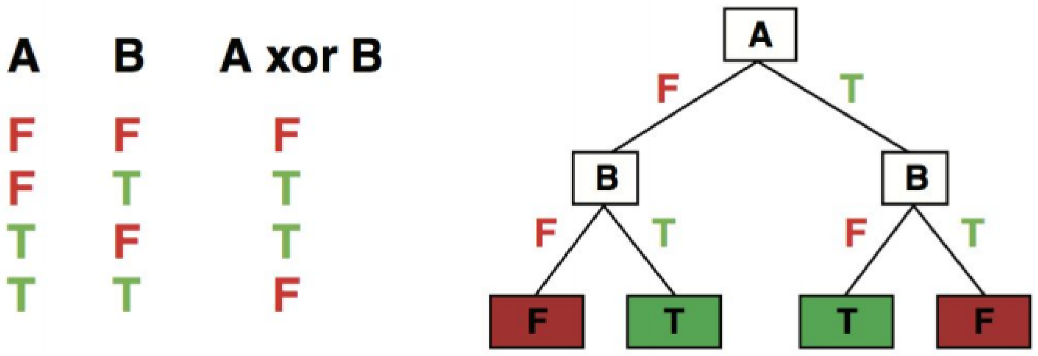


sample decision tree, and homogeneous subsets are given in the below diagram which shows a tree with a depth equal to 2.

From the above figure, we can see that a simple 2 level tree for classification would result in **fewer misclassifications than any linear classifier** than we can fit to separate these two classes - blue and red.

How powerful are Decision Trees?

A decision tree can realize any Boolean Function. The below diagram shows a decision tree for the XOR boolean function (where a statement is false if A and B are both true or both false, otherwise, false).



A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F

Note: Realization is not unique as there can be many trees for the same function.

What are the steps to build a decision tree?

The algorithm follows the below steps to build a decision tree:

1. Pick a feature
2. Split the data based on that feature such that the outcome is binary, i.e., no data point belongs to both sides of the split.
3. Define the new decision rule
4. Repeat the process until each leaf node is homogeneous, i.e., all the data points in a leaf node belong to the same class

Remark: At each split, we need to try all the different combinations for all the features which make the algorithm **computationally expensive**.

Now, the question arises: **how do we find the best split?** How to identify the feature and the value of that feature to split the data at each level? This motivates the use of **entropy as an impurity measure**.

Entropy: It is a **measure of uncertainty or diversity** embedded in a random variable.

Suppose Z is a random variable with the probability mass function $P(Z)$, then the entropy of Z is given as:

$$H(Z) = -\sum_{z \epsilon Z} P(Z) log P(Z) = -E(log(P(Z)))$$

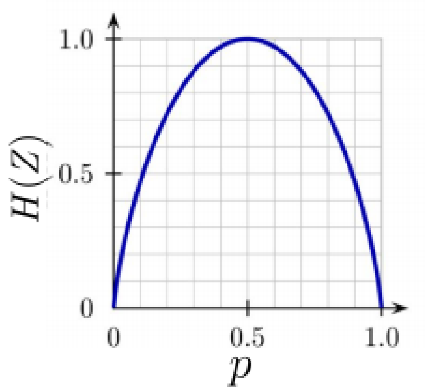
Where E represents the expected value. For all calculations, we will use a **log with base 2**.

Note: Since the log of values between 0 and 1 is negative, the minus sign helps to avoid negative values of entropy.

Let's consider an **example of a coin flip** where the probability of heads is equal to p and the probability of tails is equal to $1 - p$. Then, the entropy is given as:

$$-p\, log p - (1-p)\, log(1-p)$$

If we plot the entropy as a function of p, we get a graph like this:



From the above graph, we can observe that the entropy is minimum i.e. 0 when $p = 0$ or $p = 1$ i.e. when we can only get heads or tails which implies that **the entropy is minimum when the outcome is homogeneous**. Also, the entropy is maximum i.e. 1 when $p = 0.5$ which implies that **the entropy is maximum when both outcomes are equally likely**. So, we can say that the **lower the impurity, the lesser the entropy**.

In a similar way, we can calculate the entropy of two variables i.e. the entropy for the joint distribution of X and Y. This is called **joint entropy** of two variables and we can extend the formula of the entropy for a single variable to two variables as shown below:

$$H(X,Y) = -\sum_{(x,y)\, \epsilon\, XxY} P(x,y)\, log(P(x,y))$$

Where $P(x, y)$ represents the joint distribution of X and Y.

So, we have seen the entropy of a single random variable and the joint distribution of two variables but in decision trees, we also want to find out the entropy of the target variable for a given split. This is called **conditional entropy**. It is denoted as $H(Y \mid X)$.

In a decision tree, our aim is to find the feature and the corresponding value such that if we split the data the reduction of entropy in the target variable given the split is highest i.e. the difference between the entropy of Y and the conditional entropy $H(Y \mid X)$ is maximized. This is called **information gain**. Mathematically, it is written as:

$$IG(Y \mid X) = H(Y) - H(Y \mid X)$$

Our aim is to **maximize information gain** at each split, or in other words, minimizing $H(Y \mid X)$ as $H(Y)$ is constant.

Empirical Computation of Entropy

Now that we know the theory behind entropy and information gain, let's go through the **steps to find the best split** in a decision tree.

1. Start with the complete training dataset
2. Pick a feature, say $X(m)$
3. Describe the data based on that feature i.e. $\{(x_i(m),\ y_i),\ i = 1, 2,...., N\}$
4. Split the data into two nodes, say S_1 and S_2, based on one of the values of the same feature
5. Compute $H(S_1)$ and $H(S_2)$
6. Find the entropy of the split using the formula $P(S_1) * H(S_1) + P(S_2) * H(S_1)$
7. Compute information gain for the split
8. Repeat the process for other features and pick the one that maximizes information gain

Remark: We can see that the algorithm is making the locally optimal choice - choosing the split that maximizes the information gain at that stage, and not trying to find the global optimal solution. Hence, the decision tree is considered a **Greedy Algorithm**.

Finally, let's see a simple **example to understand the computations** involved in the steps mentioned above. Consider the following dataset with 2 independent variables, X_1 and X_2, and one target variable, Y, where all the variables can only take Boolean values.

X_1	X_2	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F
F	T	F
F	F	F

Here, we have two choices to split the data. We can split the data on X_1= T and X_1= F or X_2 = T and X_2 = F. We need to find out which feature maximizes the information gain to find the best split.

First, let's compute $H(Y)$(recall that we use log with base 2 for computations). In the target variable, we have only classes with True occurring 5 out of 8 times and False occurring 3 out of 8 times. So,

$$H(Y) = \ - p\, logp \ - \ (1 - p)\, log(1 - p) = \ - \ (5/8) * log(5/8) \ - \ (3/8) * log(3/8) = 0.954$$

Now, the split on X_1will divide the target variable data into two disjoint sets, one for X_1= T (say S_1) and the second for X_1= F (say S_2). The rows in the below table show the count of target classes in each node after the split.

	Y= T	Y= F
X_1= T	4	0
X_1= F	1	3

Computing $H(S_1)$ and $H(S_2)$,

$$H(S_1) = -(4/4) * log(4/4) - (0/4) * log(0/4) = 0, \text{ because } (log1 = 0\ and\ 0 * log0 = 0)$$

$$H(S_2) = -(1/4) * log(1/4) - (3/4) * log(3/4) = 0.811$$

So, the entropy of the split is,

$$P(S_1) * H(S_1) + P(S_2) * H(S_1) = (4/8) * 0 + (4/8) * 0.811 = 0.4055$$

The information gain by splitting on feature X_1 is

$$IG = H(Y) - 0.4055 = 0.954 - 0.4055 = 0.5485$$

Similarly, the split on X_2 will divide the target variable data into two disjoint sets, S_1 and say S_2.

	Y= T	Y= F
X_2= T	3	1
X_2= F	2	2

Computing $H(S_1)$ and $H(S_2)$ for X_2,

$$H(S_1) = -(3/4) * log(3/4) - (1/4) * log(1/4) = 0.811$$

$$H(S_2) = -(2/4) * log(2/4) - (2/4) * log(2/4) = 1$$

So, the entropy of the split is,

$$P(S_1) * H(S_1) + P(S_2) * H(S_1) = (4/8) * 0.811 + (4/8) * 1 = 0.9055$$

The information gain by splitting on feature $X_2$is

$$IG = H(Y) - 0.9055 = 0.954 - 0.9055 = 0.0485$$

Since the information gain is maximized while splitting on the feature X_1, it is the best split for the data.

Remark: Observe that the entropy of a split will always be less than or equal to the entropy of the target variable. Both the entropies will be equal (and consequently, the information gain will be zero) if the feature we are splitting on is independent of the target variable and provides no information about the target variable.

Appendix

Notations and Definitions

- **Feature Space:** A vector of independent variables - X
- **Outcome Class (categorical):** Y
- **Decision Rule**: It is a function from $f: X \rightarrow Y$. It is the rule to identify which data point will belong to which class.
- **Misclassification error (Empirical error):** It is equal to the number of misclassifications divided by the total number of observations. Mathematically, it can be written as:

$$R(f) = \frac{1}{N} \sum_{i}^{N} I(f(x_i) \neq y_i)$$

Where x_i is a data point, y_i is the actual class, $f(x_i)$ is the prediction made by the decision tree, and I is a function such that:

$$I(x) = 1 \, if \; x \neq 0, \; otherwise \; it \; is \; 0$$

- **A Probabilistic Model:** We can assume that X⊂ X and Y⊂ $Y$are random variables and each class is characterized by some joint distribution of a subset of independent variables.
- **Sub-Class:** A sub-class is the set of data points that have the same decision rule for a subset of features and belong to the same class. Mathematically, it can be written as:

$$C = \{(x, y) \,|\, x(k) = v_{k'} \; k \; subset \; of \; all \; feature \; indices\}$$