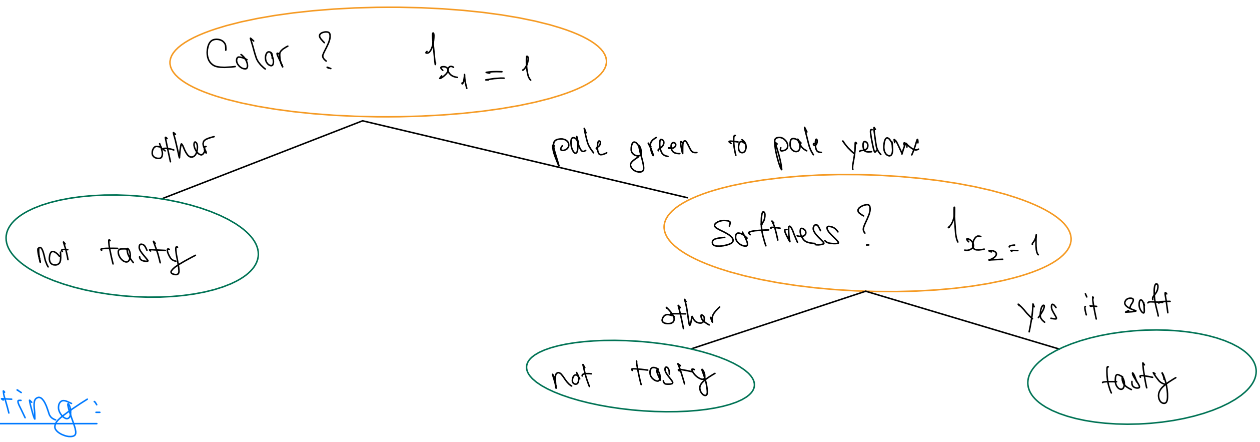
DECISION TREE

Definition:

- A decision tree is a predictor, $h: X \to Y$, that predicts the label associated with instance x by travelling from the root node to a leaf.
- In that tree there are : $\begin{cases} \text{. decision nodes: decided by features} \\ \text{. leaf nodes: are the labels} \end{cases}$

Visualize Let take example of papayas classified as tasty based of the color and softness.



Overfitting:

- Allow arbitrary size can lead to overfitting:

A common splitting rule is based on a single feature:

$$1_{x_i} < \theta \quad \text{where} \begin{cases} i \in [d] & \text{is the index of the feature} \\ \theta & \text{is the threshold} \end{cases}$$

Using this method is like splitting instance space, $X = \mathbb{R}^d$, into subspaces, where each subspace correspond to one leaf.

With that in mind, if we have $\begin{cases} \text{a tree of } k \text{ leaves} \\ k \text{ instances} \end{cases} \Rightarrow$ Overfitting

- Avoid overfitting with decision tree

We aim at learning the decision tree that satisfies both: $\begin{cases} \text{fits the data} \\ \text{not too large} \end{cases}$

One way to do this is to rely on Minimum Description Length (MDL)

Sample complexity:

- Block and bit:

Lets explain encode using block and bits, where

- block encodes the state of a node (leaf, children, splitter, ...)
- bits (can only have value 0 or 1) is the computing unit needed to represent the states in a block

Simple example: Consider a block has 4 states to represent, we would need 2 bits to represent 4 states, in particular:

"00" : state 1

"01" : state 2

"10" : state 3

"11" : state 4

- Represent a tree as block and bits

A tree with n nodes can be described using n+1 blocks, each of size $\log_2(d+3)$ bits. So the description length is $(n+1)\log_2(d+3)$

Why n+1 blocks?

Each block correspond with one node, plus an extra block to mark an end of the code

Why $\log_2(d+3)$ bits?

d : maximum number of children a node can have

3 : three additional states, no children, combination of children, end of code

$d+3$: number of states to represent

$\log_2(d+3)$: number of bits needed

Simple example: Consider binary tree, so $d = 2$,

and $d+3 = 5$ states to represent,

which requires $\log_2(5)$ bits

- <u>Upper bound using PAC - Bayesian</u>

We can prove that with probability of at least $1-\delta$ over a sample size m, for every n and every decision tree $h \in \mathcal{H}$ with n nodes:

$$L_D(h) \leq L_S(h) + \sqrt{\frac{(n+1)\log_2(d+3) + \log(2/\delta)}{m}}$$

This bound performs a tradeoff:

- When n is large (bigger tree), then $L_S(h)$ is small
- When n is small (smaller tree), then $L_S(h)$ is large

$\Rightarrow$ Our goal is to find a tree with low $L_S(h)$ and not too large (reasonable number of nodes n)

Decision Tree Algorithm

A general framework

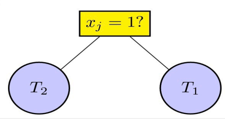
- Start with a single leaf and assign label based off the majority over the training set
- Start iterate, in each iterate:
  - Examine the effect of splitting (using some "gain" measurement)
  - Among all the splits, we either: $\rightarrow$ choose one that maximize "gain" $\searrow$ not to split

An example: Iterative Dichotomizer 3 (ID3)

For simplicity, we discuss the case of binary features, $X = \{0,1\}^d$ therefore, each splitting rule has the form $1_{x_i = 1}$

ID3(S, A)

INPUT: training set S, feature subset $A \subseteq [d]$
if all examples in S are labeled by 1, return a leaf 1
if all examples in S are labeled by 0, return a leaf 0
if $A = \emptyset$, return a leaf whose value = majority of labels in S
else :
  Let $j = \text{argmax}_{i \in A}$ Gain(S, i)
  if all examples in S have the same label
    Return a leaf whose value = majority of labels in S
  else
    Let T_1 be the tree returned by ID3($\{(\mathbf{x}, y) \in S : x_j = 1\}, A \setminus \{j\}$).
    Let T_2 be the tree returned by ID3($\{(\mathbf{x}, y) \in S : x_j = 0\}, A \setminus \{j\}$).
    Return the tree:



How is Gain Measure is implemented? (Potential function)

Difference algorithms use different implementations, here are 3 common ones:

- **Train error:**

  If the training error decrease, then that is a good split

  - Formally, let $\phi(a) = \min\{a, 1-a\}$ be the function that picks the error rate of the majority vote. i.e if a is the majority, then error is $1-a$ and vice versa
  - Before splitting, assuming majority class is 1, i.e $[y=1]$ the error rate is $\phi(\mathbb{P}_S[y=1])$
  - After splitting on feature x_i, the error rate is:

$$\phi(\mathbb{P}_S[y=1 \mid x_i=1]) \cdot \mathbb{P}_S[x_i=1] + \phi(\mathbb{P}_S[y=1 \mid x_i=0]) \cdot \mathbb{P}_S[x_i=0]$$

  - To see if there any "gain", calculate the difference:

$$\text{Gain}(S,i) := \text{Error Before} - \text{Error After}$$

- **Information Gain:**
  - Calculate by the difference between entropy of label before and after the split, used by ID3 and C4.5 algorithms of Quinlan
  - The formula is the same as above but with different error rate function $\phi(a) = -a\log(a) - (1-a)\log(1-a)$

- **Gini Index:**

  Used by CART algorithm

  The formula is the same as above but with different error rate function $\phi(a) = 2a(1-a)$

Compare train error, information gain and gini index?

Both the information gain and Gini index are smooth and concave upper bound of the train error

More on error rate, (potential function)

Assume we have a tree of just 1 leaf, a training set of 5 positives and 10 negatives. $\phi(a) = \min(a, 1-a)$ is the potential function.

Pick a literal (feature) x_i and compute $\phi\left(P_{(x,y)\sim S}(y=0)\right)$.

$$\phi\left(P_{(x,y)\sim S}(y=0)\right) = \phi\left(\frac{1}{3}\right) = \min\left(\frac{1}{3}, \frac{2}{3}\right) = \frac{1}{3} \quad \Rightarrow \text{this is the error rate of the tree}$$

Pruning the tree

- There are 2 ways to prevent a tree from growing too large:
  - Limit the number of iterations
  - Prune the tree after build

  The latter is more commonly used.
- Usually the pruning start from bottom up, where each node might be replaced with one of its subtrees or with a leaf, based on some bound or estimate of $L_D(h)$ - like the PAC-Bayesian Upper bound

Pseudo code Pruning:

Input:
- function $f(T, m)$ - Bounding function for generalize error like PAC-Bayesian
- tree T

For each node j in a bottom-up walk on T (leaves to root)
- find T' that minimizes $f(T', m)$, T' can be any of the following:
  - the current tree, replacing node j with a leaf 1
  - the current tree, replacing node j with a leaf 0
  - the current tree, replacing node j with its left subtree
  - the current tree, replacing node j with its right subtree
  - the current tree
- let $T := T'$

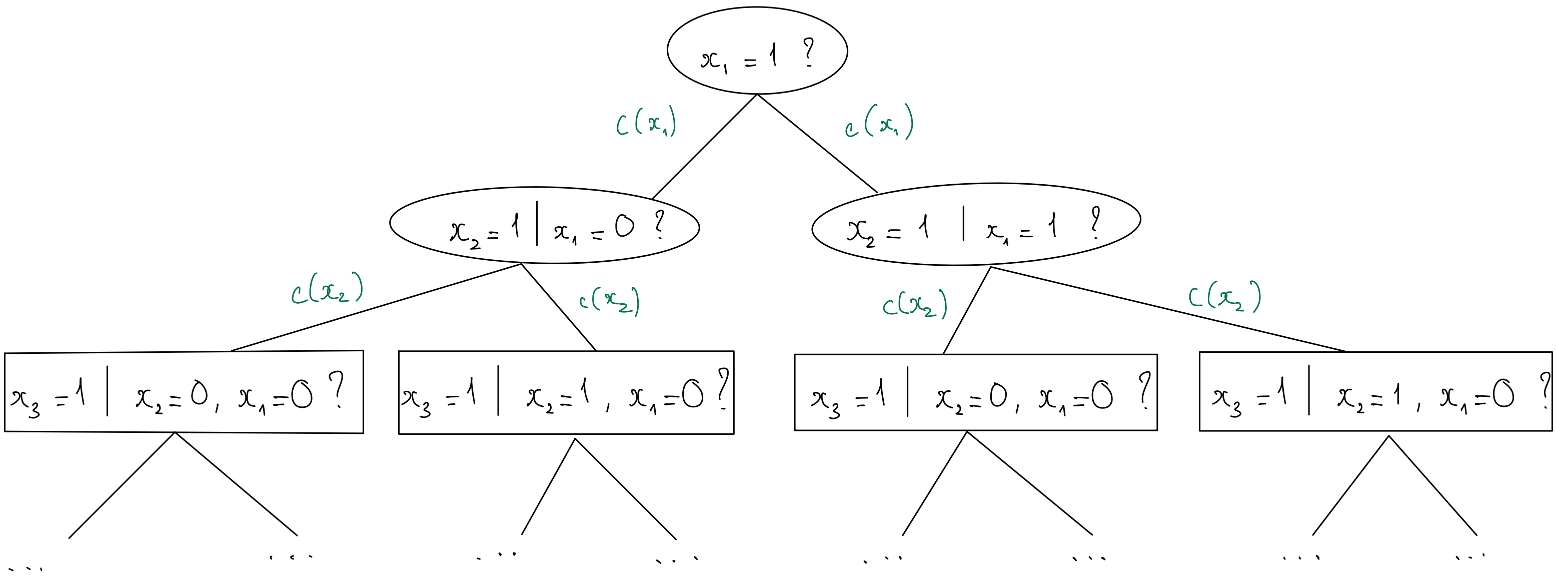
Visualize the tree building process

Lets say we try to build a decision tree over the variables

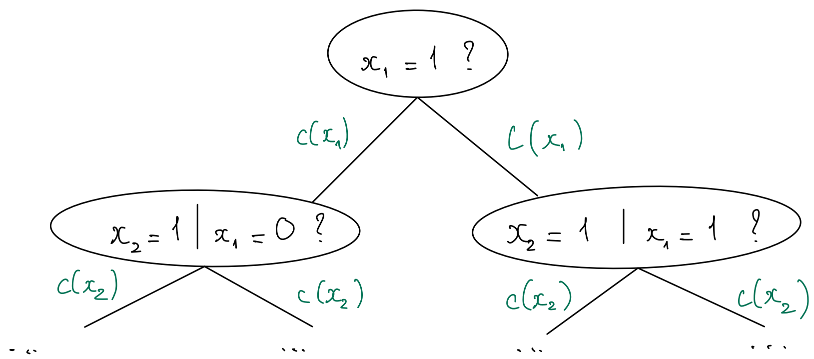
$$x = (x_1, \dots, x_n) \in \{-1, 1\}^n$$

Assume the Gain of the features is as follows: $x_1 > x_2 > \dots > x_n$

The tree would look like this:



Lets say we prune the last level of the tree, and now the tree becomes:



The error rate of this tree is:

$$\phi\left[P_S(x_2 = 1 \mid x_1 = 0)\right] \cdot P_S(x_1 = 0) + \phi\left[P_S(x_2 = 1 \mid x_1 = 1)\right] \cdot P_S(x_1 = 1)$$

What is that function c ?

It's the "condition path" function used to map boolean values $\{-1, 1\}$ to indicators $\{0, 1\}$, c is defined as:

$$c(x) = \begin{cases} \frac{1 + x}{2} & \text{if } x = 1 \\ \frac{1 - x}{2} & \text{if } x = -1 \end{cases}$$

Confusion notes:

Input values are usually booleans, $\{-1, 1\}$

But a splitting node only accepts indicators, $\{0, 1\}$

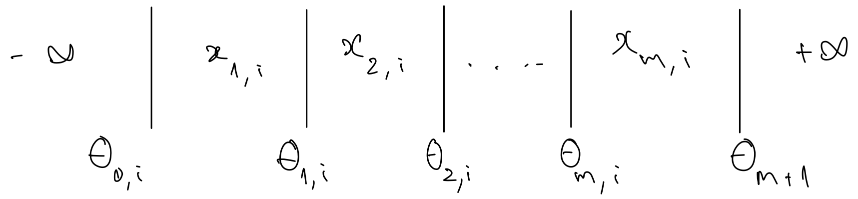
Hence, we use "condition path" function to convert from booleans to indicators

Threshold-based Splitting Rules for Real-Valued Features

Often, the features are not binary, $\mathbb{1}_{[x_i = 1]}$, but real-valued.

Therefore, we use threshold-based splitting rules, $\mathbb{1}_{[x_i < \theta]}$,
where θ is the threshold

The basic idea is to reduce the problem to binary features, for any feature i:



From this setup, we can create a splitting node in the form of:

$$\mathbb{1}_{[x_i < \theta_{j,i}]}$$

How does threshold-based splitting rule affects Gain's performance?

- If we have d features and m examples. Then based on the graph above, we would have dm binary features.
- And since Gain requires going through the examples to decide the majority vote, it takes $O(m)$ operations per binary feature. Therefore, in total it costs $O(dm^2)$ to go over all features, which is not ideal
- However, there are more clever implementation which costs $O(dm \log(m))$

Example build a decision tree

Given this traing set :

and let the potential function be:

$\phi(a) = 2a(1-a)$

x_1	x_2	$y=1$ Positive	$y=0$ Negative
0	0	1	1
0	1	2	1
1	0	3	1
1	1	4	2

- What is the current error rate ?

$$\phi\left(P_S[\text{negative}]\right) = \phi\left(\frac{5}{15}\right) = \phi\left(\frac{1}{3}\right)$$
$$= 2 \cdot \frac{1}{3}\left(1 - \frac{1}{3}\right) = \frac{4}{9}$$

- Pick x_1 or x_2 to be the root ?
  - Calculate Gain of x_1 = Current Error - Error condition on x_1.

Error condition on x_1 :=
$$\phi\left(P_S[\text{Negative} \mid x_1 = 0]\right) \cdot P_S[x_1 = 0] + \phi\left(P_S[\text{Negative} \mid x_1 = 1]\right) \cdot P_S[x_1 = 1]$$
$$= \phi\left(\frac{2}{5}\right) \cdot \frac{5}{15} + \phi\left(\frac{3}{10}\right) \cdot \frac{10}{15}$$
$$= \frac{11}{25}$$

So the Gain for x_1 is $\left(\frac{4}{9} - \frac{11}{25}\right) > 0$

  - Calculate Gain of x_2 = Current Error - Error condition on x_2

Error condition on x_2 :=
$$\phi\left(P_S[\text{Negative} \mid x_2 = 0]\right) \cdot P_S[x_2 = 0] + \phi\left(P_S[\text{Negative} \mid x_2 = 1]\right) \cdot P_S[x_2 = 1]$$
$$= \phi\left(\frac{2}{6}\right) \cdot \frac{6}{15} + \phi\left(\frac{3}{9}\right) \cdot \frac{9}{15}$$
$$= \frac{4}{9}$$

So the Gain for x_2 is $\left(\frac{4}{9} - \frac{4}{9}\right) = 0$

- Since the Gain of x_1 > Gain of x_2, pick x_1 to be at the root