CSCI567 Machine Learning (Spring 2021)

Sirisha Rambhatla

University of Southern California

March 5, 2021

Outline

Outline

Logistics

- HW 3 is due today.
- Solutions for Quiz 1 were released.

Outline

Decision tree

We have seen different ML models for classification/regression:

- linear models, neural nets and other nonlinear models induced by kernels

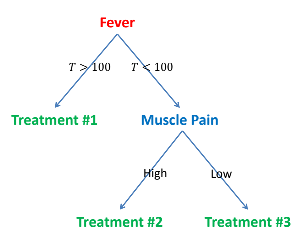
Decision tree is yet another one:

- nonlinear in general
- works for both classification and regression; we focus on classification
- one key advantage is good interpretability
- used to be very popular; ensemble of trees (i.e. "forest") can still be very effective

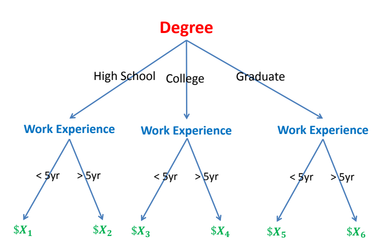
Example

Many decisions are made based on some tree structure

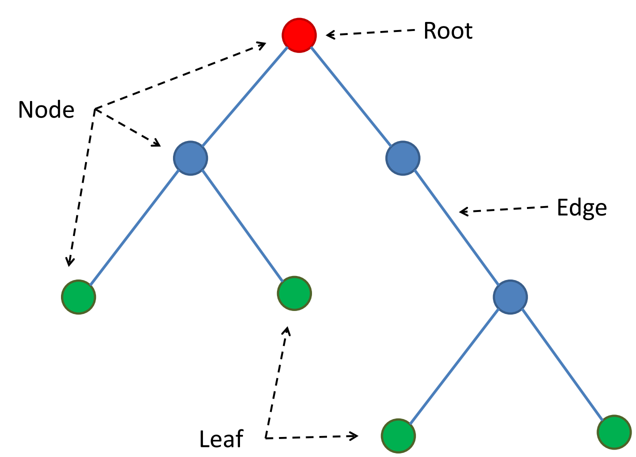
Medical treatment



Salary in a company



Tree terminology

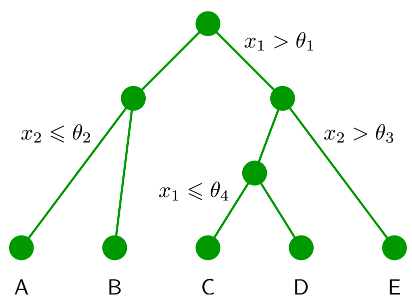


A more abstract example of decision trees

Input: $\boldsymbol{x} = (x_1, x_2)$

Output: $f(\boldsymbol{x})$ determined naturally by traversing the tree

- start from the root
- test at each node to decide which child to visit next
- finally the leaf gives the prediction $f(\boldsymbol{x})$

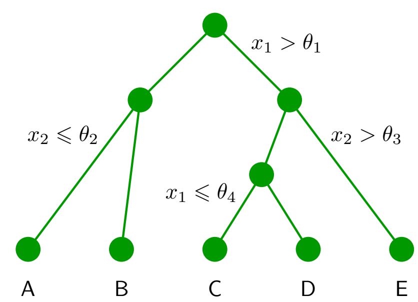


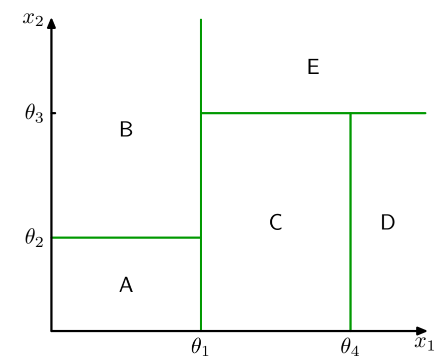
For example, $f((\theta_1 - 1, \theta_2 + 1)) = \text{B}$

Complex to formally write down, but easy to represent pictorially or as codes.

The decision boundary

Corresponds to a classifier with boundaries:

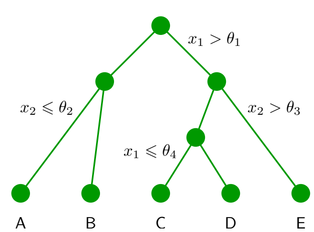




Parameters

Parameters to learn for a decision tree:

- the structure of the tree, such as the depth, #branches, #nodes, etc
  - some of them are sometimes considered as hyperparameters
  - unlike typical neural nets, the structure of a tree is *not fixed in advance, but learned from data*
- the test at each internal node
  - which feature(s) to test on?
  - if the feature is continuous, what threshold $(\theta_1, \theta_2, \ldots)$?



- the value/prediction of the leaves (A, B, ...)

Learning the parameters

So how do we *learn all these parameters?*

Recall typical approach is to find the parameters that minimize some loss.

This is unfortunately *not feasible for trees*

- suppose there are Z nodes, there are roughly $\#\text{features}^Z$ different ways to decide "which feature to test on each node", which is *a lot*.
- enumerating all these configurations to find the one that minimizes some loss is too computationally expensive.

Instead, we turn to some **greedy top-down approach**.

A running example

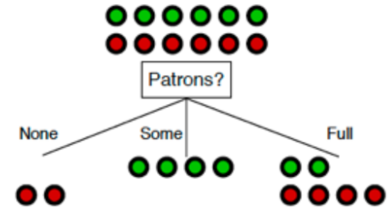
[Russell & Norvig, AIMA]

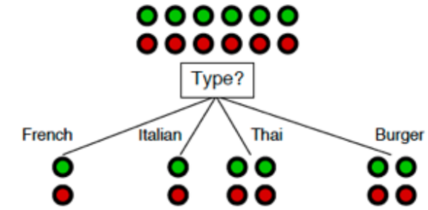
- predict whether a customer will wait for a table at a restaurant
- 12 training examples
- 10 features (all discrete)

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*T*	*F*	*F*	*T*	*Some*	*$$$*	*F*	*T*	*French*	*0–10*	*T*
X_2	*T*	*F*	*F*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*30–60*	*F*
X_3	*F*	*T*	*F*	*F*	*Some*	*$*	*F*	*F*	*Burger*	*0–10*	*T*
X_4	*T*	*F*	*T*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*10–30*	*T*
X_5	*T*	*F*	*T*	*F*	*Full*	*$$$*	*F*	*T*	*French*	*>60*	*F*
X_6	*F*	*T*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Italian*	*0–10*	*T*
X_7	*F*	*T*	*F*	*F*	*None*	*$*	*T*	*F*	*Burger*	*0–10*	*F*
X_8	*F*	*F*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Thai*	*0–10*	*T*
X_9	*F*	*T*	*T*	*F*	*Full*	*$*	*T*	*F*	*Burger*	*>60*	*F*
X_{10}	*T*	*T*	*T*	*T*	*Full*	*$$$*	*F*	*T*	*Italian*	*10–30*	*F*
X_{11}	*F*	*F*	*F*	*F*	*None*	*$*	*F*	*F*	*Thai*	*0–10*	*F*
X_{12}	*T*	*T*	*T*	*T*	*Full*	*$*	*F*	*F*	*Burger*	*30–60*	*T*

First step: how to build the root?

I.e., which feature should we test at the root? Examples:





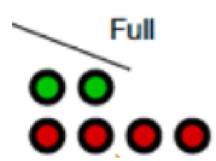
Which split is better?

- intuitively "patrons" is a better feature since it leads to "more pure" or "more certain" children
- how to quantify this intuition?

Measure of uncertainty of a node

It should be **a function of the distribution of classes**

- e.g. a node with 2 positive and 4 negative examples can be summarized by a distribution P with $P(Y=+1)=1/3$ and $P(Y=-1)=2/3$



One classic uncertainty measure of a distribution is its *(Shannon) entropy*:

$$H(Y) = -\sum_{k=1}^{\mathsf{C}} P(Y=k)\log P(Y=k)$$

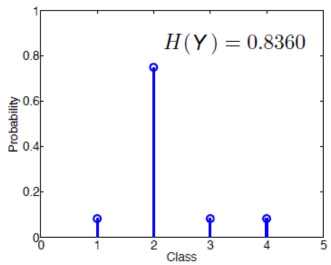
Properties of entropy

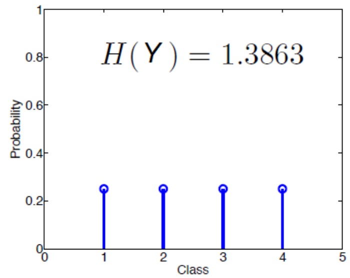
$$H(Y) = -\sum_{k=1}^{\mathsf{C}} P(Y=k)\log P(Y=k)$$

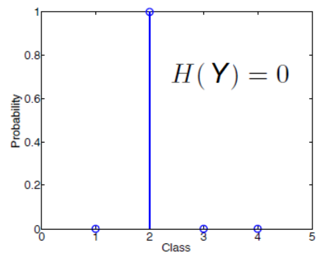
- the base of log can be 2, e or 10
- always non-negative
- it's the *smallest codeword length to encode symbols drawn from P*
- maximized if P is uniform (max $=\ln \mathsf{C}$): most uncertain case
- minimized if P focuses on one class (min $=0$): most certain case
  - e.g. $P=(1,0,\ldots,0)$
  - $0\log 0$ is defined naturally as $\lim_{z\to 0+} z\log z = 0$

Examples of computing entropy

With base e and 4 classes:







Another example

Entropy in each child if root tests on "patrons"

For "None" branch

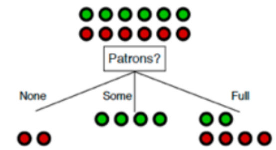
$$-\left(\frac{0}{0+2}\log\frac{0}{0+2}+\frac{2}{0+2}\log\frac{2}{0+2}\right)=0$$

For "Some" branch

$$-\left(\frac{4}{4+0}\log\frac{4}{4+0}+\frac{0}{4+0}\log\frac{0}{4+0}\right)=0$$

For "Full" branch

$$-\left(\frac{2}{2+4}\log\frac{2}{2+4}+\frac{4}{2+4}\log\frac{4}{2+4}\right)\approx 0.9$$



So how good is choosing "patrons" overall?

Very naturally, we take the weighted average of entropy:

$$\frac{2}{12}\times 0+\frac{4}{12}\times 0+\frac{6}{12}\times 0.9=0.45$$

Measure of uncertainty of a split

Suppose we split based on a discrete feature A, the uncertainty can be measured by the **conditional entropy**:

$$\begin{aligned}
&H(Y \mid A)\\
&=\sum_a P(A=a)H(Y \mid A=a)\\
&=\sum_a P(A=a)\left(-\sum_{Y_k=1}^{\mathsf{C}} P(Y_k \mid A=a)\log P(Y_k \mid A=a)\right)\\
&=\sum_a \text{“fraction of example at node } A=a\text{”} \times \text{“entropy at node } A=a\text{”}
\end{aligned}$$

Pick the feature that leads to the smallest conditional entropy.

Deciding the root

For "French" branch

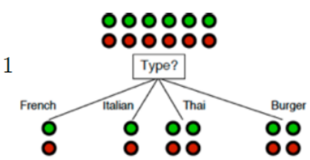
$$-\left(\frac{1}{1+1}\log\frac{1}{1+1}+\frac{1}{1+1}\log\frac{1}{1+1}\right)=1$$

For "Italian" branch

$$-\left(\frac{1}{1+1}\log\frac{1}{1+1}+\frac{1}{1+1}\log\frac{1}{1+1}\right)=1$$

For "Thai" and "Burger" branches

$$-\left(\frac{2}{2+2}\log\frac{2}{2+2}+\frac{2}{2+2}\log\frac{2}{2+2}\right)=1$$



The conditional entropy is $\frac{2}{12}\times 1+\frac{2}{12}\times 1+\frac{4}{12}\times 1+\frac{4}{12}\times 1=1>0.45$

So splitting with "patrons" is better than splitting with "type".

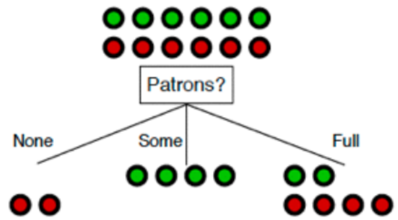
In fact by similar calculation "patrons" is the best split among all features.

We are now done with building the root (this is also called a **stump**).

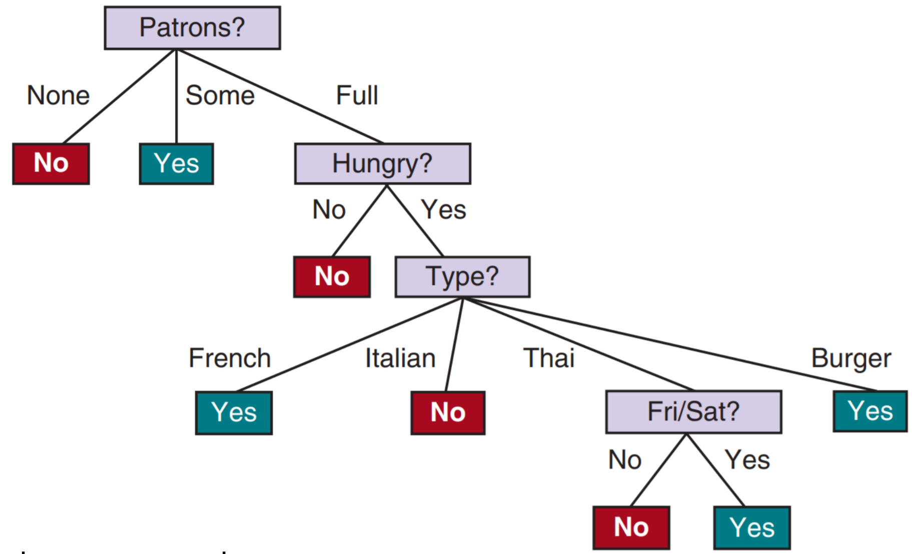
Repeat recursively

Split each child in the same way.

- but no need to split children "none" and "some": they are pure already and become leaves
- for "full", repeat, focusing on those 6 examples:



	Alt	Bar	Fri	Hun	Pat	$Price$	$Rain$	Res	$Type$	Est	WillWait
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T



Again, very easy to interpret.

Putting it together

DecisionTreeLearning(Examples, Features)

- if Examples have the same class, return a leaf with this class
- else if Features is empty, return a leaf with the majority class
- else if Examples is empty, return a leaf with majority class of parent
- else

  find the best feature A to split (e.g. based on conditional entropy)

  Tree $\leftarrow$ a root with test on A

  For each value a of A:

  Child $\leftarrow$ **DecisionTreeLearning**(Examples with $A = a$, Features$\setminus\{A\}$)
  add **Child** to **Tree** as a new branch
- return **Tree**

Variants

Popular decision tree algorithms (e.g. C4.5, CART, etc) are all based on this framework.

Variants:

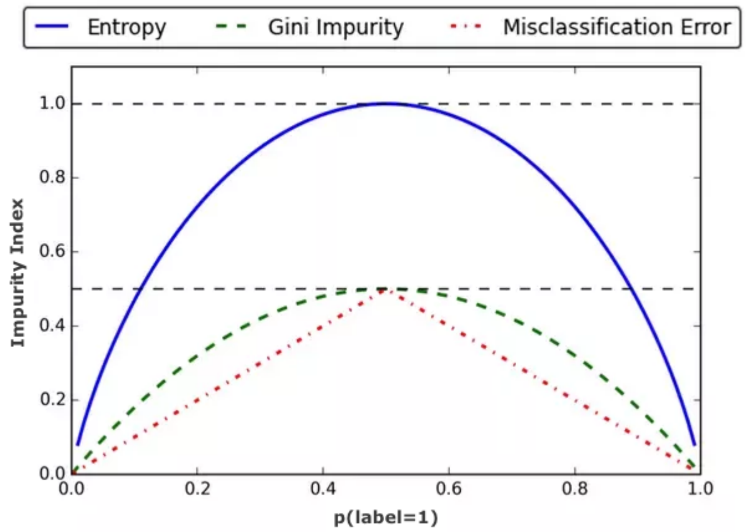
- replace entropy by **Gini impurity**:

$$G(P) = \sum_{k=1}^{C} P(Y = k)(1 - P(Y = k))$$

  meaning: *how often a randomly chosen example would be incorrectly classified if we predict according to another randomly picked example*

- if a feature is continuous, we need to find a threshold that leads to minimum conditional entropy or Gini impurity.

Variants



1

Image Credit: `https://medium.com/@jason9389/gini-impurity-and-entropy-16116e754b27`

Regularization

If the dataset has no contradiction (i.e. same $\boldsymbol{x}$ but different y), the training error of a tree is always zero, which might indicate overfitting.

Pruning is a typical way to prevent overfitting for a tree:

- restrict the depth or #nodes
- other more principled approaches
- all make use of a validation set