Nearest neighbors and decision trees

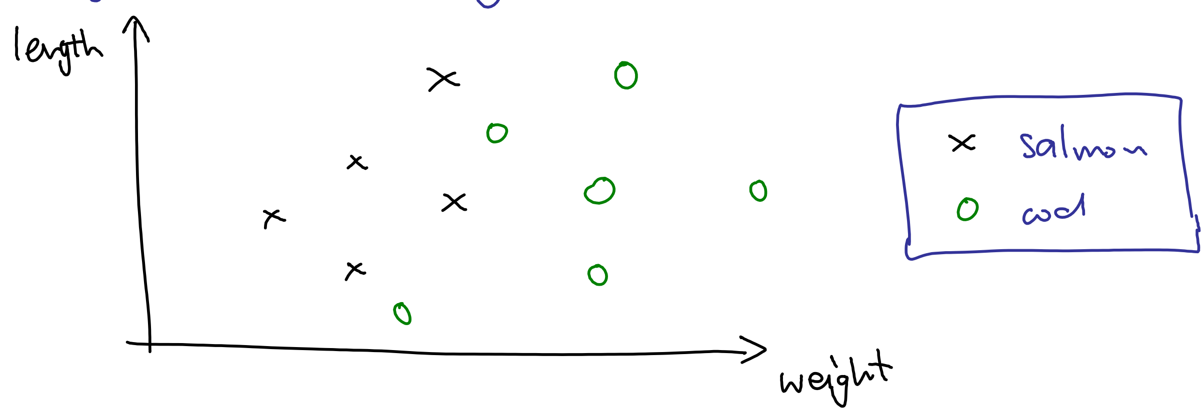
Two topics: (A) nearest neighbor classifiers
(B) decision trees

Note: for some easy background reading, see the chapter from instructor's book, "9 Algorithms ...".

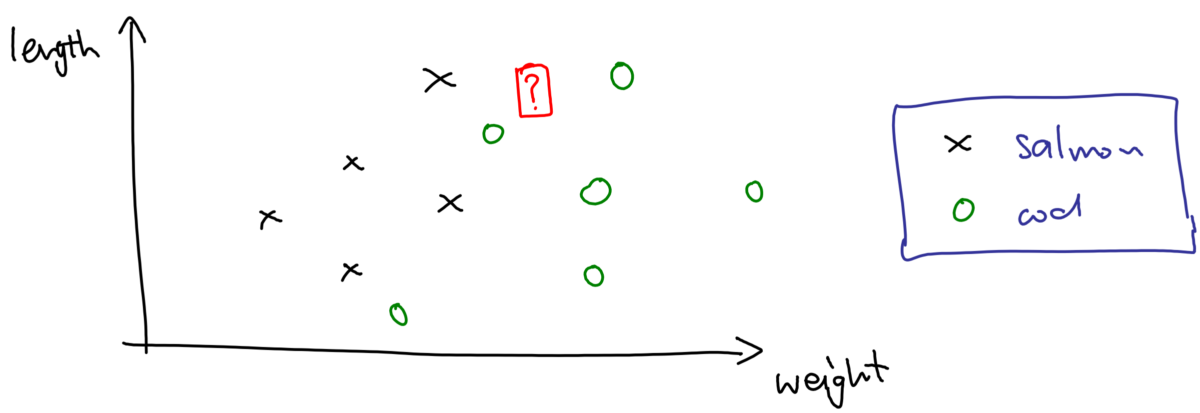
(A) Nearest neighbor classifiers

Example: classify a fish as salmon or cod based on its length and weight.

We are given the <u>training data</u>:



Given a new, unclassified fish at $\boxed{?}$:



we can return the class of the nearest neighbor (Cod, in this case).
Or, we can take the majority vote out of, say, the 5 nearest neighbors (still chooses cod in this case)

When we vote using the k nearest neighbors, this is called the <u>k-nearest-neighbors classifier</u>.

<u>Remarks</u>: ① Nearest neighbors requires a meaningful distance function. Defining this could be difficult. For example, what if we switch from measuring weight in kg to weight in grams, in the above example? Then the 'size' attribute becomes much more significant — perhaps too significant

② For large datasets, classification is expensive. e.g.: with 10^7 training examples, need to compute 10^7 distances just to classify one test example. (This can be improved, of course).

(B) Decision trees

Recall expected value : e.g.

x	3	4	5
$p(x)$	0.2	0.7	0.1

$$E(X) = \sum_x x\, p(x)$$

$$=$$ ← exercise

New concept: entropy

$$H(X) = \sum_x p(x) \log_2 \frac{1}{p(x)}$$

$$= -\sum_x p(x) \log_2 p(x)$$

e.g.

x	apple	banana	lemon	grape	plum
$p(x)$	1/16	1/2	1/16	1/8	1/4

$$H(X) = \frac{1}{16} \times 4 + \frac{1}{2} \times 1 + \frac{1}{16} \times 4 + \frac{1}{8} \times 3 + \frac{1}{4} \times 2$$

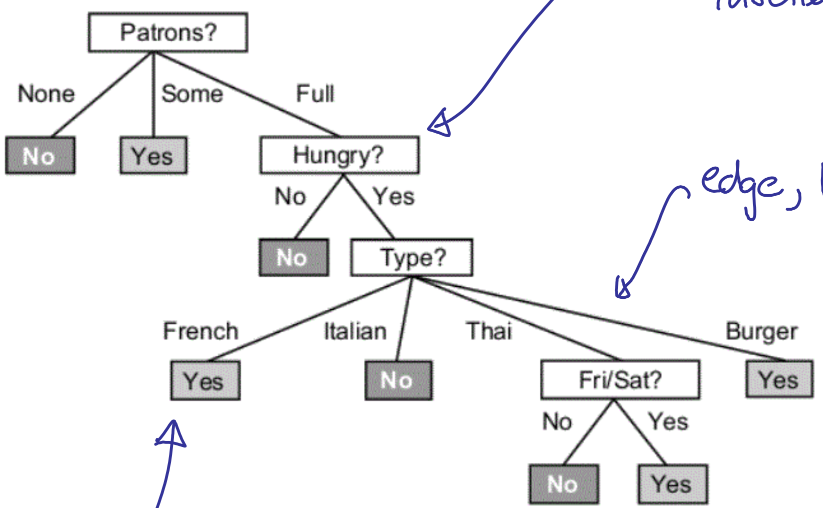
$$= \frac{1}{4} + \frac{1}{2} + \frac{1}{4} + \frac{3}{8} + \frac{1}{2}$$

$$= 15/8$$

Example from text book, used to decide whether or not to eat at a given restaurant:

internal node, labelled with <u>split attribute</u>

edge, labelled with <u>attribute value</u>



leaf, labelled with <u>decision</u>

Algorithm for building a decision tree:

- choose the attribute whose split gives the highest <u>information gain</u>, defined as:

  old entropy − new expected entropy

  (but note that 'old entropy' is constant, so we really just choose the attribute with lowest expected entropy)

- cascade just the examples that match the path from the root, and apply same algorithm.

Example:

attributes : color {Blue, Green, Yellow}

sound {Quiet, Loud}

texture {Rough, Smooth}

(class vble) → material {Wood, Metal, Fibreglass}

Color	Sound	Texture	Material
B	Q	R	W
B	L	R	W
B	Q	S	M
B	L	S	M
G	Q	S	W
G	L	S	W
Y	Q	R	F
Y	L	R	F

Step 1 : Calculate attribute to split on at the root:

Try splitting on each attribute and see which results in lowest expected entropy.

(i) color

	W	M	F	tot	entropy	weight
B	2	2	0	4	1	4/8
G	2	0	0	2	0	2/8
Y	0	0	2	2	0	2/8

$\therefore$ expected entropy $= \frac{4}{8}\times 1 + \frac{2}{8}\times 0 + \frac{2}{8}\times 0 = 1/2$

(ii) texture

	W	M	F	tot	entropy	weight
R	2	0	2	4	1	4/8
S	2	2	0	4	1	4/8

$\therefore$ expected entropy $= 4/8 \times 1 + 4/8 \times 1 = 1$

(iii) sound

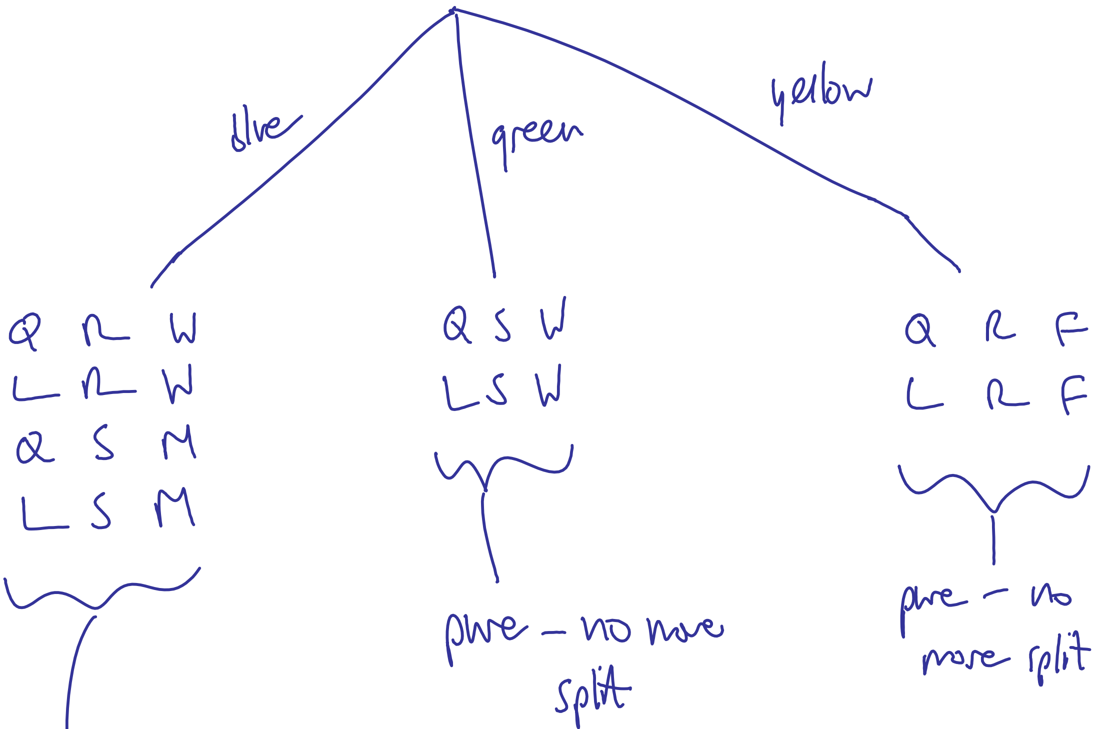
	W	M	F	tot	entropy	weight
Q	2	1	1	4	3/2	4/8
L	2	1	1	4	3/2	4/8

$\therefore$ expected entropy $= \frac{4}{8}\times 3/2 + \frac{4}{8}\times 3/2 = 3/2$

$\Rightarrow$ Color split has lowest expected entropy ($\frac{1}{2}$ bit)

... so split on color first.

tree so far:



check each attribute for lowest expected entropy after split:

Step 2: calculate split attribute for the (one remaining) impure node:

sound

	W	M	F	tot	entropy	weight
Q	1	1	0	2	1	2/4
L	1	1	0	2	1	2/4

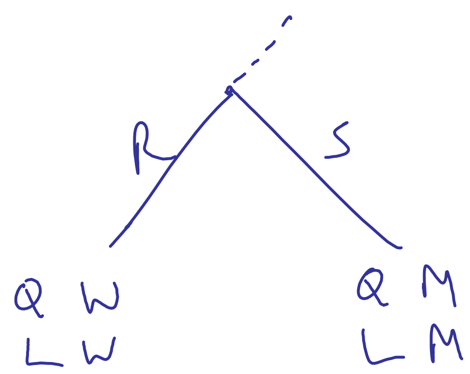
$\therefore$ total expected entropy $= \frac{2}{4} \times 1 + \frac{2}{4} \times 1 = 1$

texture

	W	M	F	tot	entropy	weight
R	2	0	0	2	0	2/4
S	0	2	0	2	0	2/4

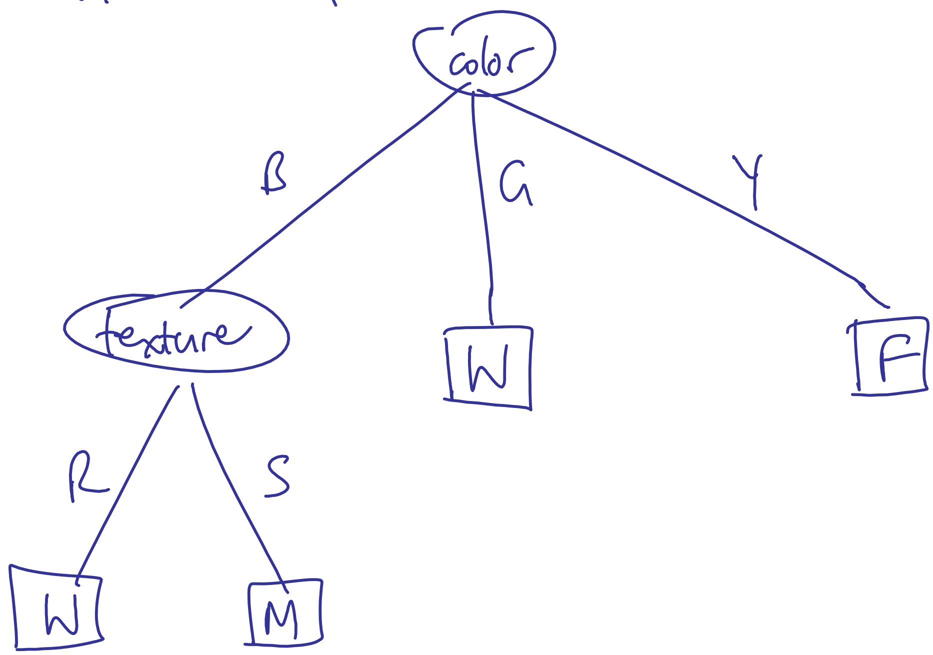
$\therefore$ expected entropy $= \frac{2}{4} \times 0 + \frac{2}{4} \times 0 = 0$

so texture has lowest expected entropy. Splitting this node gives



– both nodes pure, so we are done.

Final decision tree:



Appendix: Mathematical details for choosing split attribute (needed for programming assignment).

In section 18.3.4 (p703), the textbook describes how to choose which attribute to split on, but only for a 2-class classification problem (i.e. with only positive and negative examples). Here we describe a generalization of that procedure for multi-class problems.

Suppose we have:

- N training samples $x_1, x_2, \ldots x_N$
- an attribute A with D distinct values, divide the training set into subsets $S_1, S_2, \ldots S_D$. The number of elements in S_d is n_d.

  $$\left(\text{So } \sum_{d=1}^{D} n_d = N \right)$$

- The proportion of training samples in S_d is π_d, so

  $$\pi_d = \frac{n_d}{N} . \qquad d = 1, 2, \ldots D$$

- There are C classes: $1, 2, \ldots C$.

- The number of elements from the set S_d in class c is denoted $n_{d,c}$.

  Thus, $$\sum_{c=1}^{C} n_{d,c} = n_d \quad , \text{for } d = 1, 2, \ldots D .$$

- The proportion of elements from the set S_d in class c is denoted $\pi_{d,c}$

  Thus,

$$\pi_{d,c} = \frac{n_{d,c}}{n_d}$$

- The entropy of the distribution of classes in S_d, written H_d, can be computed as

$$H_d = \sum_{c=1}^{C} -\pi_{d,c} \log_2 \pi_{d,c}$$

- The expected entropy for attribute A, denoted $E(A)$ is given by

$$E(A) = \sum_{d=1}^{D} \pi_d H_d$$

We want to choose the attribute A^* with the highest information gain. But info gain = (current entropy) − (expected entropy), so this is equivalent to choosing the attribute with lowest expected entropy.

Thus, we choose

$$A^* = \underset{A}{\operatorname{argmin}}\ E(A)$$