Classification II

- Previously we looked at Bayesian classification
  - main issue: obtain likelihoods $P(\mathbf{x}|\omega)$
  - **parametric** technique: choose model (e.g., multivariate Gaussian) and estimate parameters (e.g., mean and covariance matrix) from training set
  - use decision rule (e.g., ML or MAP) to classify

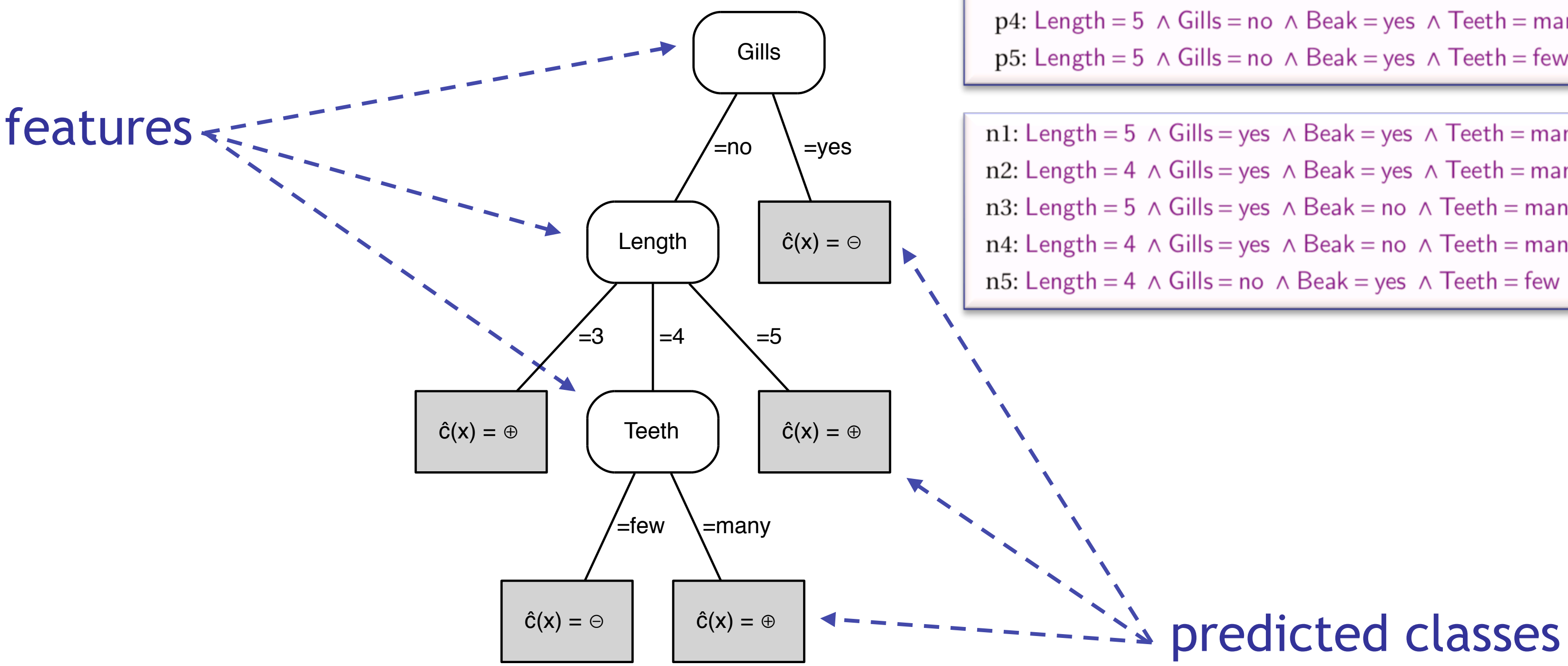
- In this lecture we look at a **non-parametric** technique called **decision trees**
  - deterministically build a model that separates classes on training set
  - then evaluate and adapt the model on a separate **test set** to avoid overfitting and improve generalisation

Decision trees

- Partitions the instance space into regions of (near-)uniform class membership
  - internal nodes are labelled with features (aka splits)
  - leaves are labelled with classes

p1: Length = 3 ∧ Gills = no ∧ Beak = yes ∧ Teeth = many
p2: Length = 4 ∧ Gills = no ∧ Beak = yes ∧ Teeth = many
p3: Length = 3 ∧ Gills = no ∧ Beak = yes ∧ Teeth = few
p4: Length = 5 ∧ Gills = no ∧ Beak = yes ∧ Teeth = many
p5: Length = 5 ∧ Gills = no ∧ Beak = yes ∧ Teeth = few

n1: Length = 5 ∧ Gills = yes ∧ Beak = yes ∧ Teeth = many
n2: Length = 4 ∧ Gills = yes ∧ Beak = yes ∧ Teeth = many
n3: Length = 5 ∧ Gills = yes ∧ Beak = no ∧ Teeth = many
n4: Length = 4 ∧ Gills = yes ∧ Beak = no ∧ Teeth = many
n5: Length = 4 ∧ Gills = no ∧ Beak = yes ∧ Teeth = few



Growing trees

Algorithm 5.1: GrowTree(D, F) – grow a feature tree from training data.

Input : data D; set of features F.
Output : feature tree T with labelled leaves.

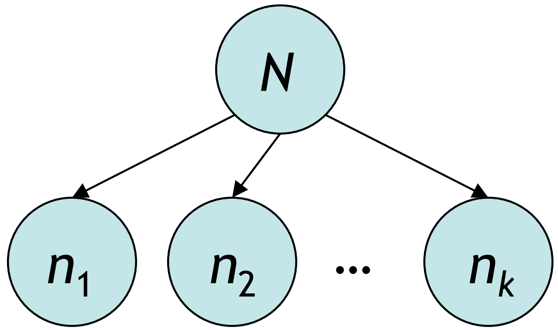
```
if Homogeneous(D) then return Label(D) ;
S ←BestSplit(D, F) ;
split D into subsets D_i according to the literals in S;
for each i do
    if D_i ≠ ∅ then T_i ←GrowTree(D_i, F) else T_i is a leaf labelled with Label(D);
end
return a tree whose root is labelled with S and whose children are T_i
```

Homogeneous(D) returns true if the instances in D are homogeneous enough to be labelled with a single label, and false otherwise;

Label(D) returns the most appropriate label for a set of instances D;

BestSplit(D, F) returns the best set of literals to be put at the root of the tree.

Finding the best split



- Suppose a node with N instances is split into k nodes with n_i instances ($1 \leq i \leq k$, $\Sigma_i\, n_i = N$)

- **Information gain** of a split can be calculated as the decrease in impurity going from parent to children:

$$Imp(Parent) - \sum_{i=1}^{k} \frac{n_i}{N} Imp(Child_i)$$

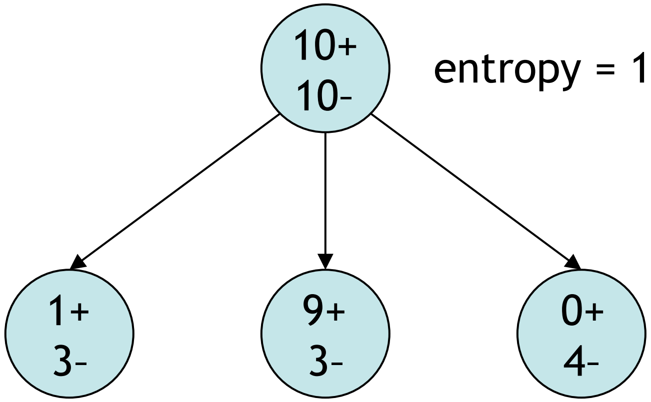
- If we have c classes and the proportion of class j in a set of instances is p_j, then the impurity of that set of instances can be measured by **entropy**

$$\sum_{j=1}^{c} -p_j \log_2 p_j$$

Entropy

- $-\log_2 p_j$ measures the information content in bits of an event occurring with probability p_j
  - can be used to design codes: cannot do better than assigning $-\log_2 p_j$ bits (Shannon)
- $\Sigma_j -p_j \log_2 p_j$ is the average information content per event
  - or the 'randomness' of the distribution
  - entropy is maximal for uniform distributions, and zero if all but one p_j are zero
  - e.g., for two events entropy is $-p_1 \log_2 p_1 - (1-p_1) \log_2 (1-p_1)$
- Here, we use it to measure how much additional information a particular split provides us about the class

An example



entropy = 1

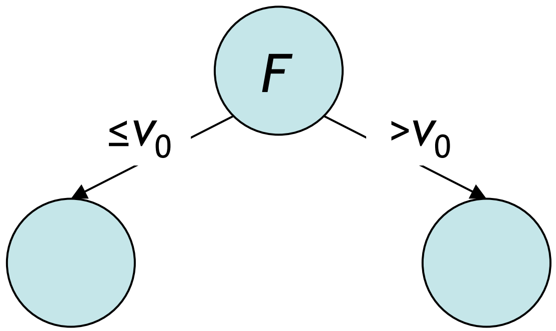
entropy = $-1/4 \log_2 1/4 - 3/4 \log_2 3/4 = 0.5 + 0.31 = 0.81$

entropy = 0

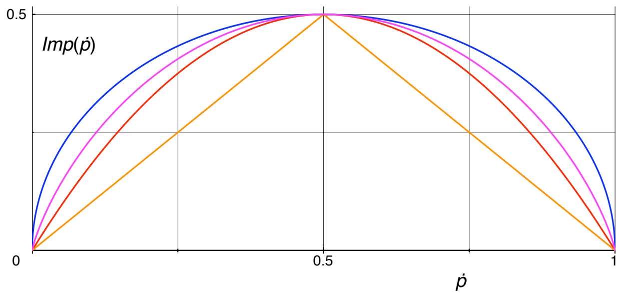
entropy = $-9/12 \log_2 9/12 - 3/12 \log_2 3/12 = 0.31 + 0.5 = 0.81$

Information gain = $1 - (4/20)0.81 - (12/20)0.81 - (4/20)0 = 0.35$

- For numeric features we create binary splits by setting a threshold that maximises information gain



Other impurity functions



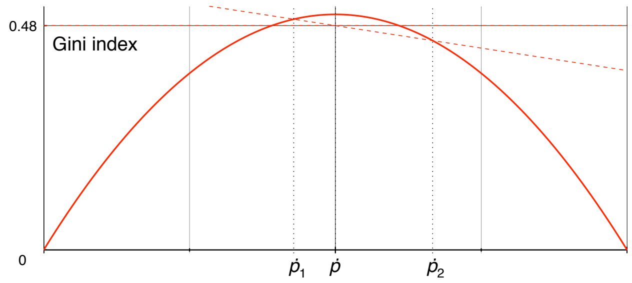
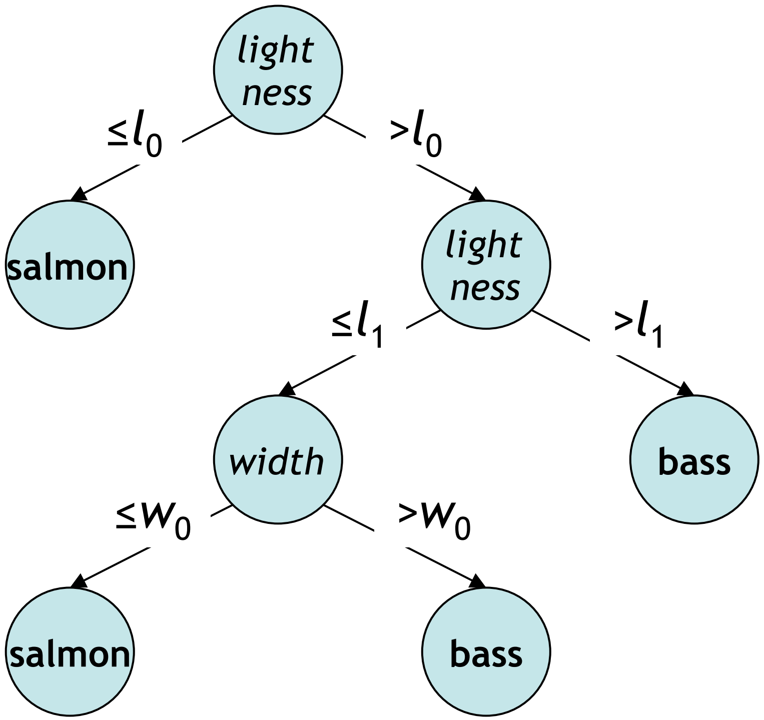
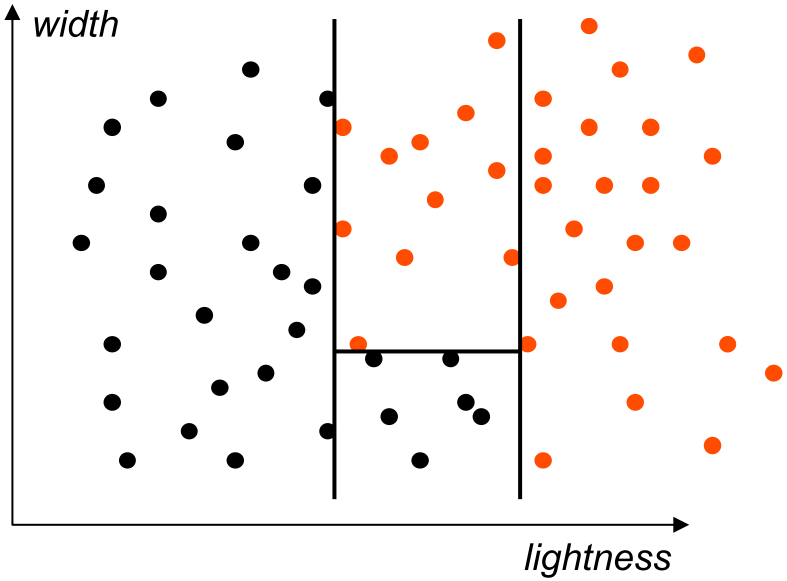


Figure 5.2. (left) Impurity functions plotted against the empirical probability of the positive class. From the bottom: the relative size of the minority class, $\min(\dot{p}, 1-\dot{p})$; the Gini index, $2\dot{p}(1-\dot{p})$; entropy, $-\dot{p}\log_2 \dot{p} - (1-\dot{p})\log_2(1-\dot{p})$ (divided by 2 so that it reaches its maximum in the same point as the others); and the (rescaled) square root of the Gini index, $\sqrt{\dot{p}(1-\dot{p})}$ – notice that this last function describes a semi-circle. **(right)** Geometric construction to determine the impurity of a split ($\mathrm{Teeth} = [\mathrm{many}, \mathrm{few}]$ from Example 5.1): $\dot{p}$ is the empirical probability of the parent, and $\dot{p}_1$ and $\dot{p}_2$ are the empirical probabilities of the children.

Trees, branches and leaves





- The instance space is divided into axis-parallel rectangles
  - or hyper-rectangles in *d*-dimensional space

Pruning trees

- The GrowTree algorithm will achieve very high accuracy on the training set
  - 100% accuracy if features can be re-used
  - but many of the leaves will contain only a few instances —> overfitting!
- In order to improve generalisation, we prune the tree back from leaves to root, using a separate test set
  1. Let *N* be the parent of a current leaf
  2. Let α be the test set accuracy of the current tree
  3. Let β be the test set accuracy if *N* and its children are replaced by a leaf, labelled with the most frequent class among *N*'s training instances ('majority class')
  4. If $\alpha < \beta$ then prune (i.e., replace the subtree with the leaf)
  5. If all parents of leaves have been tried then stop else go to 1.

Trees and probabilities

- Consider a branch of a decision tree from root to a leaf, let **x** be the conjunction of conditions on that branch, and let p_i be the relative frequency of class j instances in the leaf.
- Then p_j can be interpreted as an estimate of $P(\omega_j|\mathbf{x})$, i.e., the posterior probability of class ω_j given **x**
- Hence, assigning the majority class $\text{argmax}_\omega P(\omega_j|\mathbf{x})$ to the leaf corresponds to the MAP decision rule

    - Finally, it is worth noticing that decision trees can represent almost any posterior probability distribution (they have low **bias**) but tend to be sensitive to small variations in the training data (they have high **variance**)