Decision Tree

**Decision tree** is a tree that thresholds features of an instance $x$ at internal nodes (in a top-down manner) until $x$ reaches a leaf node which will assign $x$ to one class. (Figure 1)

![An Example Decision Tree](ELS, Figure 9.2)

For a fixed decision tree, an internal node thresholds one feature of $x$ and passes it to two of the child nodes accordingly. For example, in Fig 1, $x$ is passed to the left child node if $X_1 \leq t_1$ and to the right child node if $X_1 > t_1$. A leaf node assigns any arriving $x$ to one class, e.g., $R_4$ will assign all arriving examples to class 3 and $R_2$ will assign all arriving examples to class 1.

To construct/learn a decision tree, we need to decide (1) which feature to threshold at an internal node; (2) what class to assign at a leaf node. Different algorithms have different methods. We will introduce one algorithm called **CART** (Classification And Regression Tree).

CART continuously grows a tree, by splitting undesired leaf nodes, until all leaf nodes are pure. A node is pure if most arriving instances have the same label – it allows us to confidently assign that label to future arriving instances without making significant prediction error. The purity of any node $m$ can be measured by its entropy, i.e.,

$$H(m) = -\sum_{k=1}^{K} p_{mk} \cdot \log p_{mk}, \quad (1)$$

where $p_{mk}$ is the probability that an instance from class $k$ arrives at node $m$. It can be estimated from all training examples that arrive at node $m$.

If $H(m)$ is small, then node $m$ is pure and CART will treat it as a leaf node. The label of $m$ is the class of its most arriving instances, e.g., if most instances are from class 4, then $m$ will assign future arriving instances to class 4.

If $H(m)$ is big, then node $m$ is not pure and CART will split it by picking up a feature for further thresholding arriving instances. It will selected an unused feature $X_j$ based on some cost...
function (e.g., how pure are the resulted child nodes). It will examine the cost of all features and (for numerical features) a set of candidate thresholds.

After a tree is generated, we can prune it to avoid overfitting. Pruning cuts branches to reduce the size of the tree. It continuously does so until classification accuracy is no longer improved. Another way to avoid overfitting is to fix the maximum depth of the tree.

We can have some simple verification on why purity matters. Suppose a node \( m \) assigns arriving instances to class \( k \). Let \( m_{/k} \) be the set of arriving instances not from class \( k \) and \( m_k \) be the set of instances from class \( k \). The classification error of \( m \) is

\[
er_m = \frac{1}{|m|} \sum_{(x,y) \in m} 1_{f(x) \neq y} = \frac{1}{|m|} \left( \sum_{(x,y) \in m_{/k}} 1_{f(x) \neq y} + \sum_{(x,y) \in m_k} 1_{f(x) \neq y} \right)
\]

\[
= \frac{1}{|m|} \left( \sum_{(x,y) \in m_{/k}} 1_{f(x) \neq y} \right)
\]

\[
= \frac{|m_{/k}|}{|m|}.
\]

We see if \( k \) is the class of most arriving instances, then \(|m_{/k}| \) is minimized (among all choices of class) and thus error will be minimized.