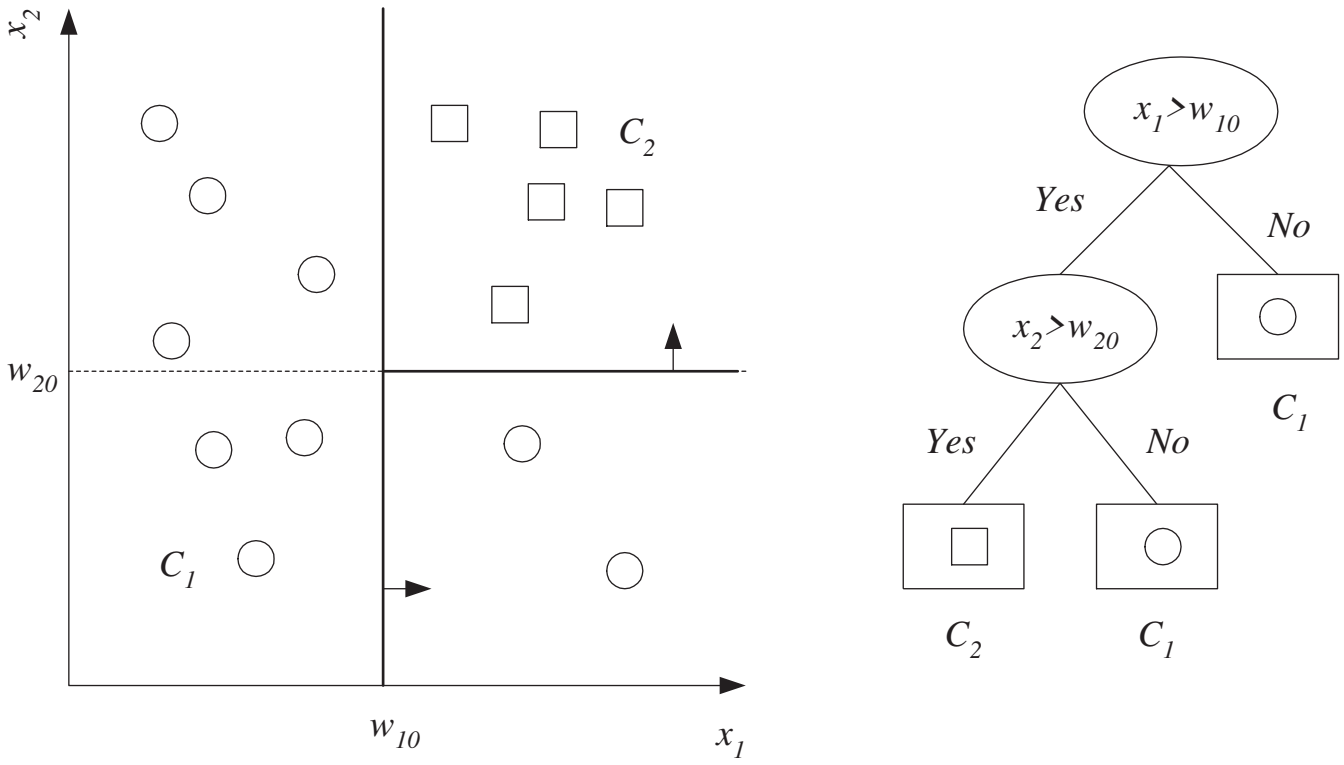
10 Decision trees

- Applicable to classification and regression.
- Can use continuous and discrete (categorical) features. $x \in \mathbb{R}$ or $x \in$ {red,green,blue}.
- Efficient at test time:
  - An input instance follows a single root-leaf path in the tree, ignoring the rest of it.
  - This path (and the whole tree) may not even use all the features in the training set.
- A decision tree is a model that (as long as it is not too big) can be *interpreted* by people (unlike black-box models such as neural nets):
  - We can inspect the tree visually regardless of the dimensionality of the feature vector.
  - We can track the root-leaf path followed by a particular input instance to understand how the tree made its decision.
  - The tree can be transformed into a set of IF-THEN rules.
- Widely used in practice, sometimes preferred over more accurate but less interpretable models.
- They define class regions as a sequence of recursive splits.
- The decision tree consists of:
  - *Internal decision nodes*, each having ≥ 2 children. Decision node m selects one of its children based on a test (a *split*) applied to the input $\mathbf{x}$.
    - Continuous feature x_d: "go right if $x_d > s_m$" for some $s_m \in \mathbb{R}$.
    - Discrete feature x_d: n-way split for the n possible values of x_d.
  - *Leaves*, each containing a value (class label or output value y). Instances $\mathbf{x}_n$ falling in the same leaf should have identical (or similar) output values y_n.
    - Classification: class label $y \in \{1, \ldots, K\}$ (or proportion of each class $p_1, \ldots, p_K$).
    - Regression: numeric value $y \in \mathbb{R}$ (average of the outputs for the leaf's instances).

  The predicted output for an instance $\mathbf{x}$ is obtained by following a path from the root to a leaf. In the best case (balanced tree) for binary trees, the path has length $\log_2 L$ if there are L leaves.
- Having learned a tree, we discard the training set $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$ and keep only the tree nodes (and associated split and output values). The resulting tree may be considered:
  - Nonparametric: if the tree is very big, having $\Theta(N)$ nodes if each leaf represents one (or a few) instances. Still, inference in the tree is much faster than, say, in kernel regression or k-nearest-neighbors classification (no need to scan the whole training set).
  - Parametric: if the tree is much smaller than the training set N.

  In practice, the size of the tree depends on the application.

Univariate trees



- The test at node m compares one feature with a threshold: "$x_d > s_m$" for some $d \in \{1, \ldots, D\}$ and $s_m \in \mathbb{R}$. This defines a *binary split* into two regions: $\{\mathbf{x} \in \mathbb{R}^D\colon\ x_d \leq s_m\}$ and $\{\mathbf{x} \in \mathbb{R}^D\colon\ x_d > s_m\}$. The overall tree defines box-shaped, axis-aligned regions in input space.

  Simplest and most often used. More complex tests exist, e.g. "$\mathbf{w}_m^T\mathbf{x} > s_m$", which define oblique regions (*multivariate trees*).

- With discrete features, the number of children equals the number n_d of values the feature can take, and the test selects the child corresponding to the value of x_d (*n-way split*).

  Ex: $x_d \in \{\text{red}, \text{green}, \text{blue}\} \Rightarrow n_d = 3$ children.

- *Tree induction* is learning the tree from a training set $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$, i.e., determining its nodes and structure:

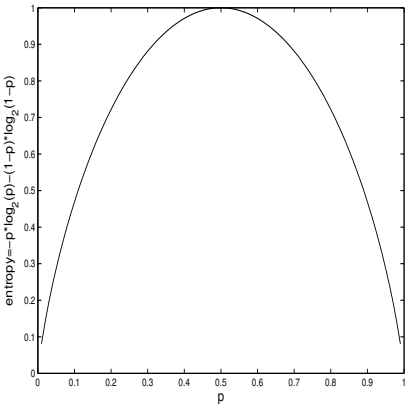
  - For each internal node, its test (feature $d \in \{1, \ldots, D\}$ and threshold s_m).
  - For each leaf, its output value y.

- For a given sample, many (sufficiently big) trees exist that code it with zero error. We want to find the smallest such tree. This is NP-hard so an approximate, greedy algorithm is used: Fig. 9.3 pseudocode

  - Starting at the root with the entire training set, select a best split according to a *purity criterion*: $(N_{\text{left}}\phi_{\text{left}} + N_{\text{right}}\phi_{\text{right}})/(N_{\text{left}} + N_{\text{right}})$, where $N_\bullet$ is the number of instances going to child $\bullet$. Associate each child with the subset of instances that fall in it.
  - Continue splitting each child (with its subset of the training set) recursively until each child is pure (hence a leaf) and no more splits are necessary.
  - Prevent overfitting by either early stopping or pruning.

  Ex. algorithms: CART, ID3, C4.5.

Classification trees



- *Purity criterion*: a node is pure if it contains instances of the same class. Consider a node and all the training set instances that reach it, and call p_k the proportion of instances of class k, for $k = 1, \dots, K$ (so $p_k \geq 0$ and $\sum_{k=1}^{K} p_k = 1$). We can measure impurity as the *entropy* of $\mathbf{p} = (p_1, \dots, p_K)$: $\phi(\mathbf{p}) = -\sum_{k=1}^{K} p_k \log_2 p_k$ (where $0 \log_2 0 \equiv 0$[?]). This is maximum if $p_1 = \cdots = p_K = \frac{1}{K}$, and minimum ("pure") if one $p_k = 1$ and the rest are 0[?].

  Other measures satisfying those conditions are possible: Gini index $\phi(\mathbf{p}) = \sum_{i \neq j}^{K} p_i p_j = \sum_{i=1}^{K} p_i(1 - p_i)$, misclassification error $\phi(\mathbf{p}) = 1 - \max(p_1, \dots, p_K)$.

- If a node is pure, i.e., all its instances are of the same class k, there is no need to split it. *It becomes a leaf with output value k.*

  We can also store the proportions $\mathbf{p} = (p_1, \dots, p_K)$ in the node (e.g. to compute risks).

- If a node m is not pure, we split it. We evaluate $(N_{\text{left}}\phi_{\text{left}} + N_{\text{right}}\phi_{\text{right}})/(N_{\text{left}} + N_{\text{right}})$ for all possible features $d = 1, \dots, D$ and all possible split thresholds s_m for each feature, and pick the split with minimum impurity.
  - If the number of instances that reach node m is N_m, there are are $N_m - 1$ possible thresholds (the midpoints between consecutive values of x_d, assuming we have sorted them).
  - For discrete features, there is no threshold but an n-way split.

Regression trees

- *Purity criterion*: the squared error $E(g) = \sum_{n \in \text{node}} (y_n - g)^2$ (where $g \in \mathbb{R}$) is minimal when g is the mean of the y_n values[?]. Then $\phi = E(g)$ is the variance of the y_n values at a node.

  If there is much noise or outliers, it is preferable to set g to the median of the y_n values.

- We consider a node to be pure if $E \leq \theta$ for a user threshold $\theta > 0$. In that case, we do not split it. *It becomes a leaf with output value g.*

- If a node m is not pure, we split it. We evaluate all possible features $d = 1, \dots, D$ and all possible split thresholds s_m for each feature and pick the split with minimum impurity (= sum of the variances E of each of the children), as in the classification case.

- Rather than assigning a constant output value to a leaf, we can assign it a regression function (e.g. linear), as in a running-mean smoother.

Early stopping and pruning

- Growing the tree until each leaf is pure will produce a large tree with high variance (sensitive to the training sample) that will overfit when there is noise.
  How to learn smaller trees that generalize better to unseen data?

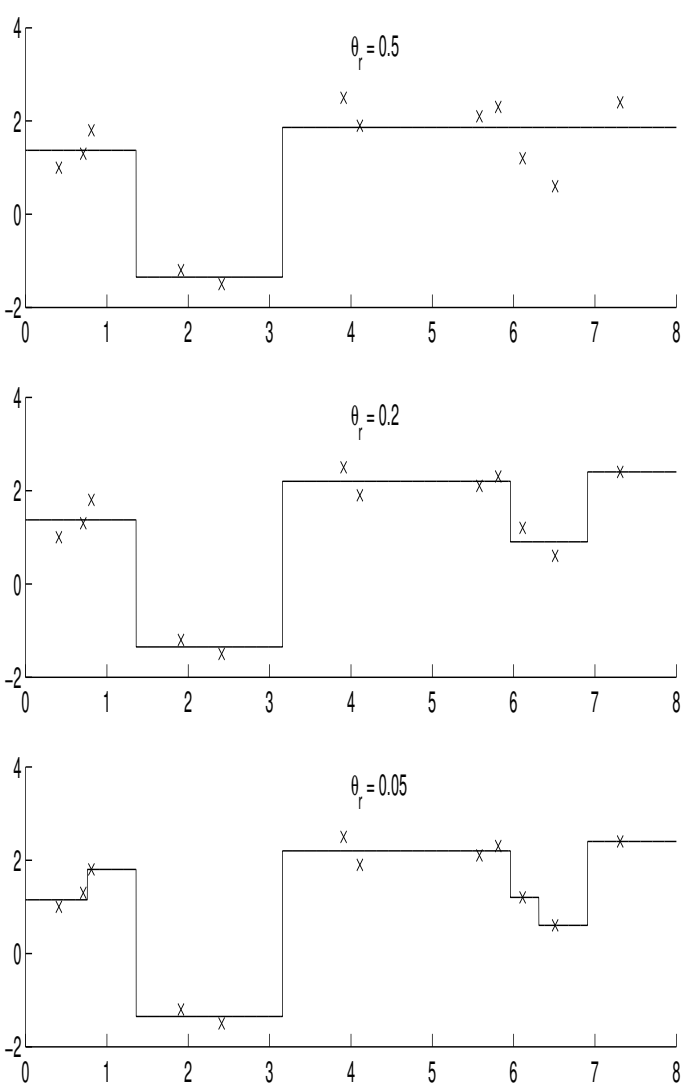
- *Early stopping*: we stop splitting if the impurity is below a user threshold $\theta > 0$.
  $\theta \downarrow$ low bias, high variance, large tree; $\theta \uparrow$ high bias, low variance, small tree.

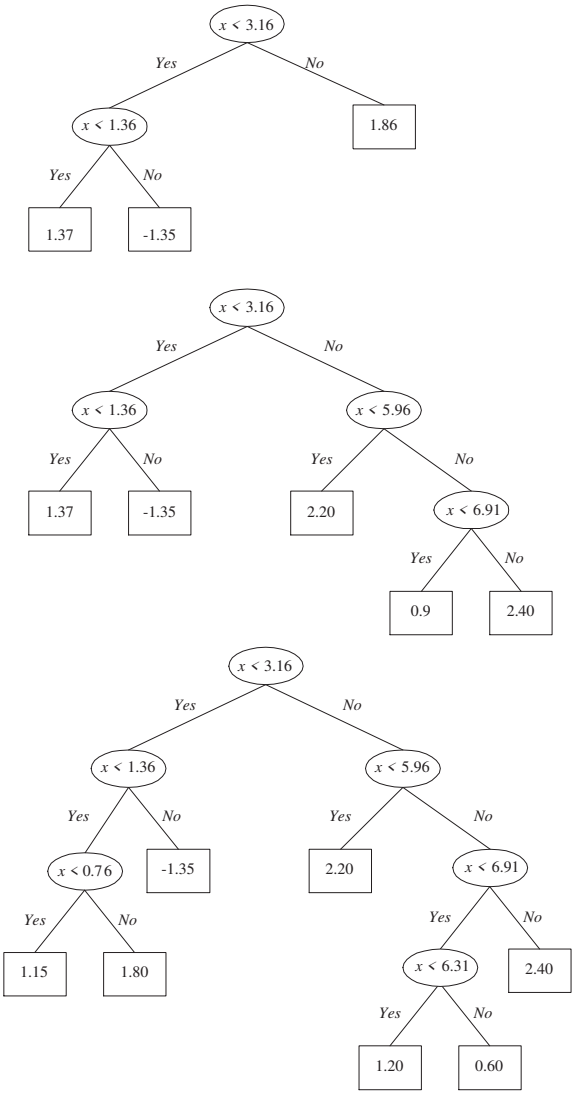
- *Pruning*: we grow the tree in full until all leaves are pure and the training error is zero. Then, we find subtrees that cause overfitting and prune them.

  Keep aside a subset from the training set ("pruning set"). For each possible subtree, try replacing it with a leaf node labeled with the training instances covered by the subtree. If the leaf node performs no worse than the subtree on the pruning set, we prune the subtree and keep the leaf node because the additional complexity of the subtree is not justified; otherwise, we keep the subtree.

- Pruning is slower than early stopping but it usually leads to trees that generalize better.

  An intuitive reason why is as follows. In the greedy algorithm used to grow the tree, once we make a decision at a given node (to select a split) we never backtrack and try a different, maybe better, possibility.





Rule extraction from trees

- A decision tree does its own feature extraction: the final tree may not use all the D features.
- Features closer to the root may be more important globally.
- Path root $\rightsquigarrow$ leaf = conjunction of tests. This and the leaf's output value give a rule.
- The set of extracted rules allows us to extract knowledge from the dataset.

R1: IF (age $>$ 38.5) AND (years-in-job $>$ 2.5) THEN $y = 0.8$
R2: IF (age $>$ 38.5) AND (years-in-job $\leq$ 2.5) THEN $y = 0.6$
R3: IF (age $\leq$ 38.5) AND (job-type = 'A') THEN $y = 0.4$
R4: IF (age $\leq$ 38.5) AND (job-type = 'B') THEN $y = 0.3$
R5: IF (age $\leq$ 38.5) AND (job-type = 'C') THEN $y = 0.2$