**function** \textsc{Decision-Tree-Learning}(examples, attributes, parent\_examples)\hspace{1em}**returns** \hspace{1em} a tree

if \textit{examples} is empty then return \textsc{Plurality-Value}(parent\_examples)
else if all \textit{examples} have the same classification then return the classification
else if \textit{attributes} is empty then return \textsc{Plurality-Value}(examples)
else
\[ A \leftarrow \text{argmax}_{a \in \text{attributes}} \text{Importance}(a, \text{examples}) \]
\[ \text{tree} \leftarrow \text{a new decision tree with root test } A \]
for each value \( v_k \) of \( A \) do
\[ \text{exs} \leftarrow \{ e : e \in \text{examples} \text{ and } e.A = v_k \} \]
\[ \text{subtree} \leftarrow \textsc{Decision-Tree-Learning}(\text{exs}, \text{attributes} - A, \text{examples}) \]
add a branch to \textit{tree} with label \( (A = v_k) \) and subtree \textit{subtree}
return \textit{tree}

\textbf{Figure 18.4} The decision-tree learning algorithm. The function \textsc{Importance} is described in Section ???. The function \textsc{Plurality-Value} selects the most common output value among a set of examples, breaking ties randomly.
function CROSS-VALIDATION.WRAPPER(Learner, k, examples) returns a hypothesis

local variables: errT, an array, indexed by size, storing training-set error rates
errV, an array, indexed by size, storing validation-set error rates

for size = 1 to ∞ do
  errT[size], errV[size] ← CROSS-VALIDATION(Learner, size, k, examples)
  if errT has converged then do
    best_size ← the value of size with minimum errV[size]
  return Learner(best_size, examples)

Figure 18.7 An algorithm to select the model that has the lowest error rate on validation data by building models of increasing complexity, and choosing the one with best empirical error rate on validation data. Here errT means error rate on the training data, and errV means error rate on the validation data. Learner(size, examples) returns a hypothesis whose complexity is set by the parameter size, and which is trained on the examples. PARTITION(examples, fold, k) splits examples into two subsets: a validation set of size N/k and a training set with all the other examples. The split is different for each value of fold.

function CROSS-VALIDATION(Learner, size, k, examples) returns two values:

  average training set error rate, average validation set error rate

fold_errT ← 0; fold_errV ← 0
for fold = 1 to k do
  training_set, validation_set ← PARTITION(examples, fold, k)
  h ← Learner(size, training_set)
  fold_errT ← fold_errT + ERROR-RATE(h, training_set)
  fold_errV ← fold_errV + ERROR-RATE(h, validation_set)
return fold_errT/k, fold_errV/k

Figure 18.7 An algorithm to select the model that has the lowest error rate on validation data by building models of increasing complexity, and choosing the one with best empirical error rate on validation data. Here errT means error rate on the training data, and errV means error rate on the validation data. Learner(size, examples) returns a hypothesis whose complexity is set by the parameter size, and which is trained on the examples. PARTITION(examples, fold, k) splits examples into two subsets: a validation set of size N/k and a training set with all the other examples. The split is different for each value of fold.

function DECISION-LIST-LEARNING(examples) returns a decision list, or failure

if examples is empty then return the trivial decision list No

  t ← a test that matches a nonempty subset examples₁ of examples
  such that the members of examples₁ are all positive or all negative

if there is no such t then return failure

if the examples in examples₁ are positive then o ← Yes else o ← No

return a decision list with initial test t and outcome o and remaining tests given by

DECISION-LIST-LEARNING(examples − examples₁)

Figure 18.10 An algorithm for learning decision lists.
function BACK-PROP-LEARNING(examples, network) returns a neural network

inputs: examples, a set of examples, each with input vector x and output vector y
network, a multilayer network with L layers, weights $w_{i,j}$, activation function $g$

local variables: $\Delta$, a vector of errors, indexed by network node

repeat
  for each weight $w_{i,j}$ in network do
    $w_{i,j} \leftarrow$ a small random number
  for each example $(x, y)$ in examples do
    /* Propagate the inputs forward to compute the outputs */
    for each node $i$ in the input layer do
      $a_i \leftarrow x_i$
    for $\ell = 2$ to $L$ do
      for each node $j$ in layer $\ell$ do
        $in_j \leftarrow \sum_i w_{i,j} a_i$
        $a_j \leftarrow g(in_j)$
    /* Propagate deltas backward from output layer to input layer */
    for each node $j$ in the output layer do
      $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
    for $\ell = L - 1$ to $1$ do
      for each node $i$ in layer $\ell$ do
        $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$
    /* Update every weight in network using deltas */
    for each weight $w_{i,j}$ in network do
      $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
  until some stopping criterion is satisfied

return network

Figure 18.23 The back-propagation algorithm for learning in multilayer networks.
Figure 18.33  The AdaBoost variant of the boosting method for ensemble learning. The algorithm generates hypotheses by successively reweighting the training examples. The function Weighted-Majority generates a hypothesis that returns the output value with the highest vote from the hypotheses in \( h \), with votes weighted by \( z \).