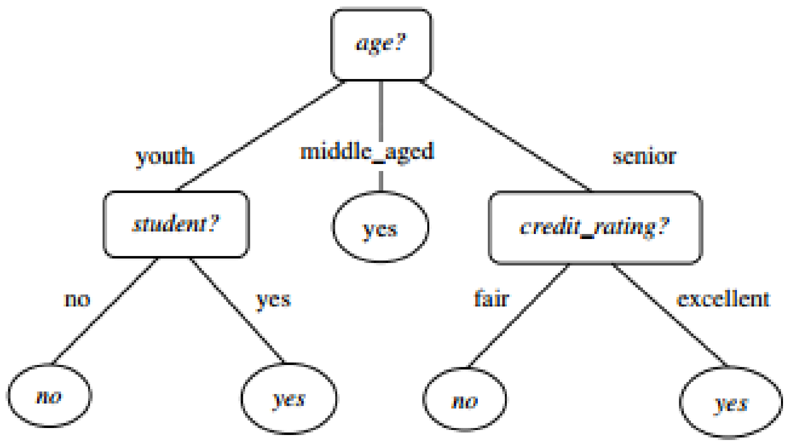
Decision Tree

A tree has many analogies in real life, and turns out that it has influenced a wide area of **machine learning**, covering both **classification and regression**. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions.

A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

For example, below is the decision tree for the concept ***buys_computer***.



Common terms used with Decision trees:

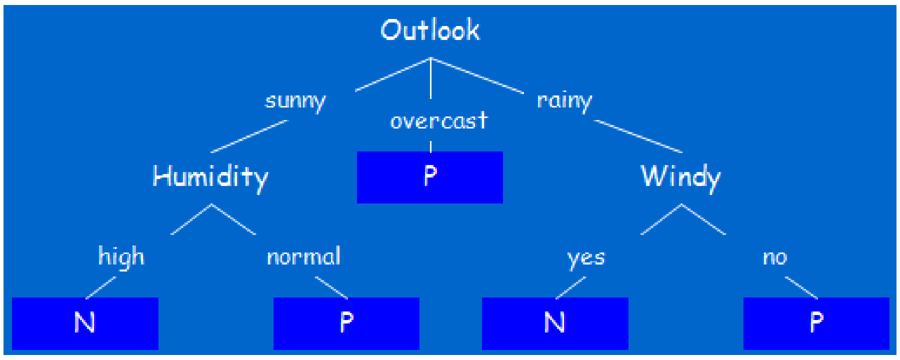
1. **Root Node:** It represents entire population or sample and this further gets divided into two or more homogeneous sets.
2. **Splitting:** It is a process of dividing a node into two or more sub-nodes.
3. **Decision Node:** When a sub-node splits into further sub-nodes, then it is called decision node.
4. **Leaf/ Terminal Node:** Nodes do not split is called Leaf or Terminal node.
5. **Pruning:** When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.
6. **Branch / Sub-Tree:** A sub section of entire tree is called branch or sub-tree.
7. **Parent and Child Node:** A node, which is divided into sub-nodes is called parent node of sub-nodes whereas sub-nodes are the children of parent node.

Example of a Decision Tree:

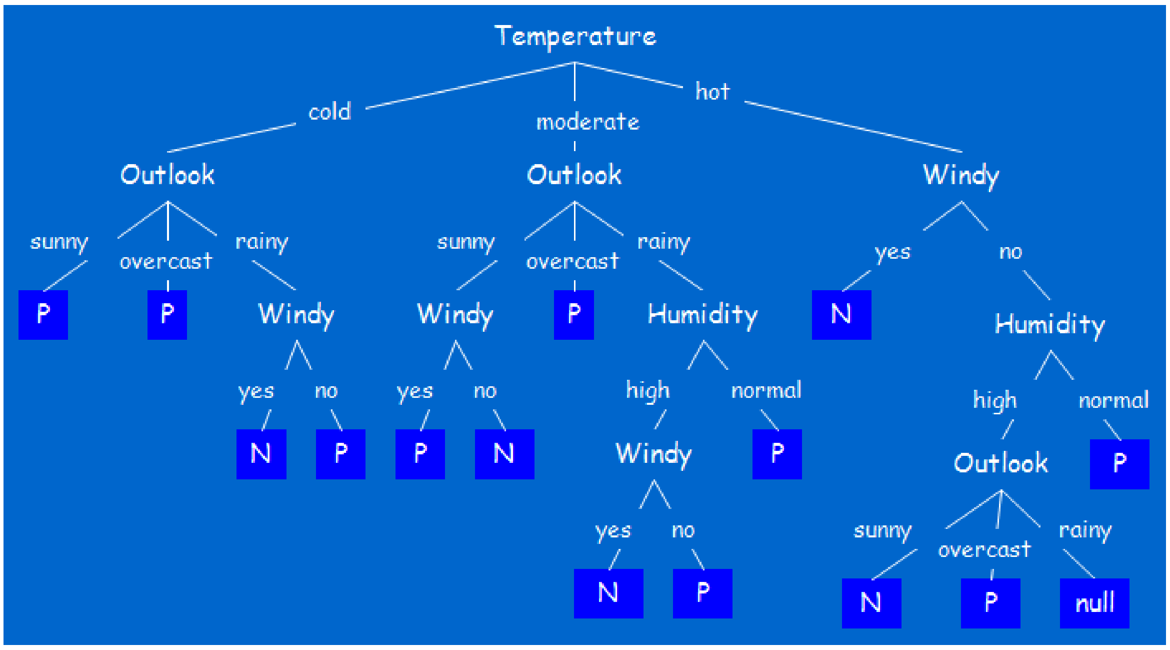
Let’s consider the following data. There can be n number of decision trees that can be created from this data.

#	Attribute				Class
	Outlook	Temperature	Humidity	Windy	Play
1	sunny	hot	high	no	N
2	sunny	hot	high	yes	N
3	overcast	hot	high	no	P
4	rainy	moderate	high	no	P
5	rainy	cold	normal	no	P
6	rainy	cold	normal	yes	N
7	overcast	cold	normal	yes	P
8	sunny	moderate	high	no	N
9	sunny	cold	normal	no	P
10	rainy	moderate	normal	no	P
11	sunny	moderate	normal	yes	P
12	overcast	moderate	high	yes	P
13	overcast	hot	normal	no	P
14	rainy	moderate	high	yes	N

Tree 1:



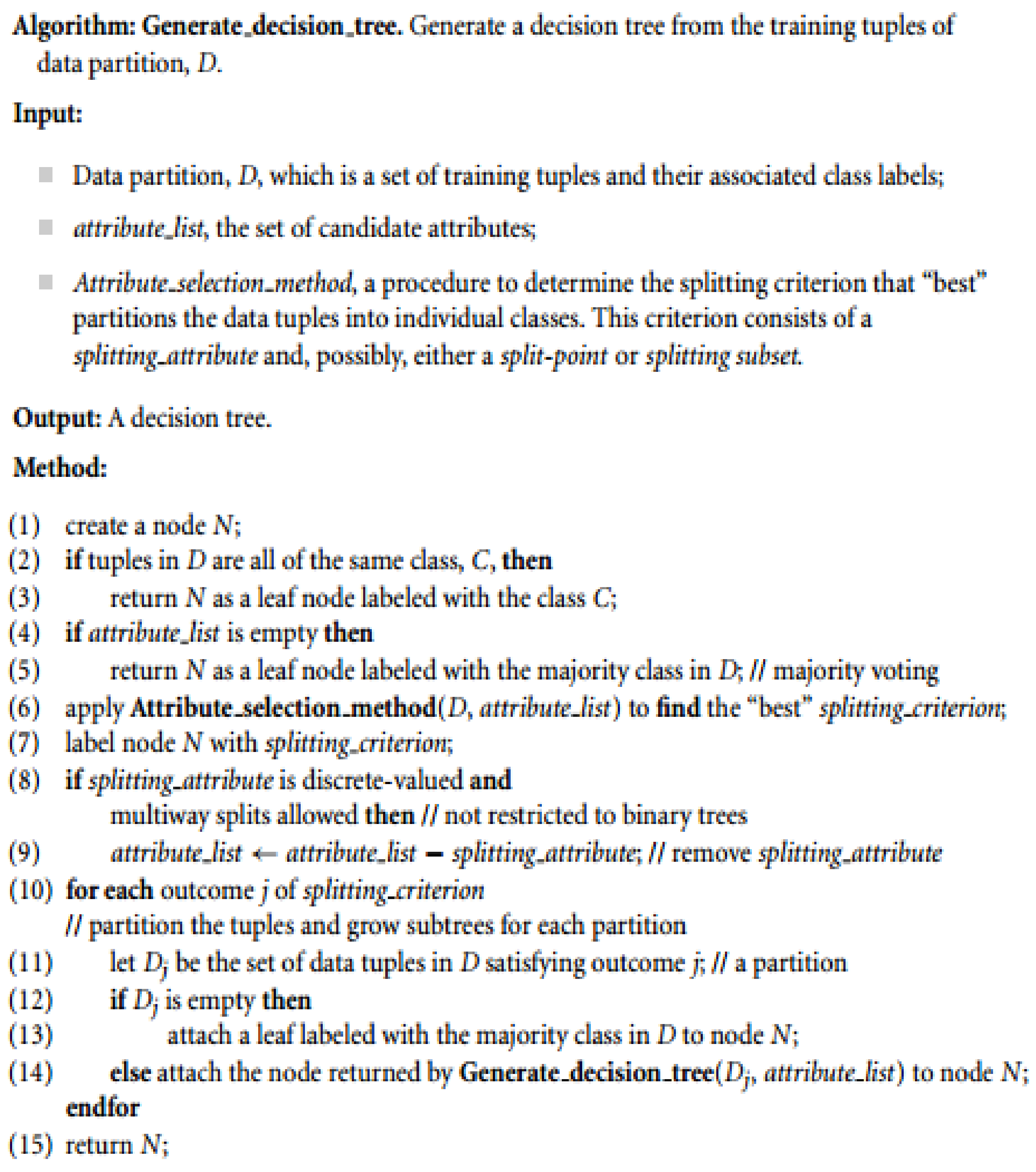
Tree 2:



Decision Tree Induction:

Most of the existing systems to generate decision tree are based on Hunt's algorithm called "Top-Down Induction of Decision Tree (TDIDT)". This algorithm adopts a greedy (i.e., nonbacktracking) approach in which decision trees are constructed in a top-down recursive divide-and-conquer manner.

Basic algorithm for inducing a decision tree from training tuples



Algorithm: Generate_decision_tree. Generate a decision tree from the training tuples of data partition, *D*.

Input:

- Data partition, *D*, which is a set of training tuples and their associated class labels;
- *attribute_list*, the set of candidate attributes;
- *Attribute_selection_method*, a procedure to determine the splitting criterion that "best" partitions the data tuples into individual classes. This criterion consists of a *splitting_attribute* and, possibly, either a *split-point* or *splitting subset*.

Output: A decision tree.

Method:

```
(1)  create a node N;
(2)  if tuples in D are all of the same class, C, then
(3)      return N as a leaf node labeled with the class C;
(4)  if attribute_list is empty then
(5)      return N as a leaf node labeled with the majority class in D; // majority voting
(6)  apply Attribute_selection_method(D, attribute_list) to find the "best" splitting_criterion;
(7)  label node N with splitting_criterion;
(8)  if splitting_attribute is discrete-valued and
         multiway splits allowed then // not restricted to binary trees
(9)      attribute_list ← attribute_list − splitting_attribute; // remove splitting_attribute
(10) for each outcome j of splitting_criterion
     // partition the tuples and grow subtrees for each partition
(11)     let $D_j$ be the set of data tuples in D satisfying outcome j; // a partition
(12)     if $D_j$ is empty then
(13)         attach a leaf labeled with the majority class in D to node N;
(14)     else attach the node returned by Generate_decision_tree($D_j$, attribute_list) to node N;
     endfor
(15) return N;
```

- The algorithm is called with three parameters: D, attribute list, and Attribute selection method. We refer to D as a data partition. Initially, it is the complete set of training tuples and their associated class labels. The parameter attribute list is a list of attributes describing the tuples. Attribute selection method specifies a heuristic procedure for selecting the attribute that "best" discriminates the given tuples according to class. This procedure employs an attribute selection measure such as information gain or the Gini index. Whether the tree is strictly binary is generally driven by the attribute selection measure. Some attribute selection measures, such as the Gini index, enforce the resulting tree to be binary. Others, like information gain, do not, therein allowing multiway splits (i.e., two or more branches to be grown from a node).
- The tree starts as a single node, N, representing the training tuples in D (step 1).
- If the tuples in D are all of the same class, then node N becomes a leaf and is labelled with that class (steps 2 and 3). Note that steps 4 and 5 are terminating conditions. All terminating conditions are explained at the end of the algorithm.
- Otherwise, the algorithm calls Attribute selection method to determine the splitting criterion. The splitting criterion tells us which attribute to test at node N by determining the "best" way to separate or partition the tuples in D into individual classes (step 6). The splitting criterion also tells us which branches to grow from node N with respect to the outcomes of the chosen test. More specifically, the splitting criterion indicates the splitting attribute and may also indicate either a split-point or a splitting subset. The splitting criterion is determined so that, ideally, the resulting partitions at each branch are as "pure" as possible. A partition is pure if all the tuples in it belong to the same class. In other words, if we split up the tuples in D according to the mutually exclusive outcomes of the splitting criterion, we hope for the resulting partitions to be as pure as possible.
- The node N is labelled with the splitting criterion, which serves as a test at the node (step 7). A branch is grown from node N for each of the outcomes of the splitting criterion. The tuples in D are partitioned accordingly (steps 10 to 11). There are three possible scenarios, as illustrated in Figure 1. Let A be the splitting attribute. A has v distinct values, {a1, a2,..., av}, based on the training data.

  1) A is discrete-valued: In this case, the outcomes of the test at node N correspond directly to the known values of A. A branch is created for each known value, a_j, of A and labelled with that value (Figure 1a). Partition D_j is the subset of class-labelled tuples in D having value aj of A. Because all the tuples in a given partition have the same value for A, A need not be considered in any future partitioning of the tuples. Therefore, it is removed from attribute list (steps 8 and 9).
  2) A is continuous-valued: In this case, the test at node N has two possible outcomes, corresponding to the conditions A ≤ split point and A > split point, respectively, where split point is the split-point returned by Attribute selection method as part of the splitting criterion. (In practice, the split-point, a, is often taken as the midpoint of two known adjacent values of A and therefore may not actually be a pre-existing value of A from the training data.) Two branches are grown from N and labelled according to the previous outcomes (Figure 1b). The tuples are partitioned such that D1 holds the subset of class-labelled tuples in D for which A ≤ split point, while D2 holds the rest.

3) A is discrete-valued and a binary tree must be produced (as dictated by the attribute selection measure or algorithm being used): The test at node N is of the form “A ∈ S_A?,” where S_A is the splitting subset for A, returned by Attribute selection method as part of the splitting criterion. It is a subset of the known values of A. If a given tuple has value a_j of A and if aj ∈ S_A, then the test at node N is satisfied. Two branches are grown from N (Figure 1c). By convention, the left branch out of N is labelled yes so that D1 corresponds to the subset of class-labelled tuples in D that satisfy the test. The right branch out of N is labelled no so that D2 corresponds to the subset of class-labelled tuples from D that do not satisfy the test.

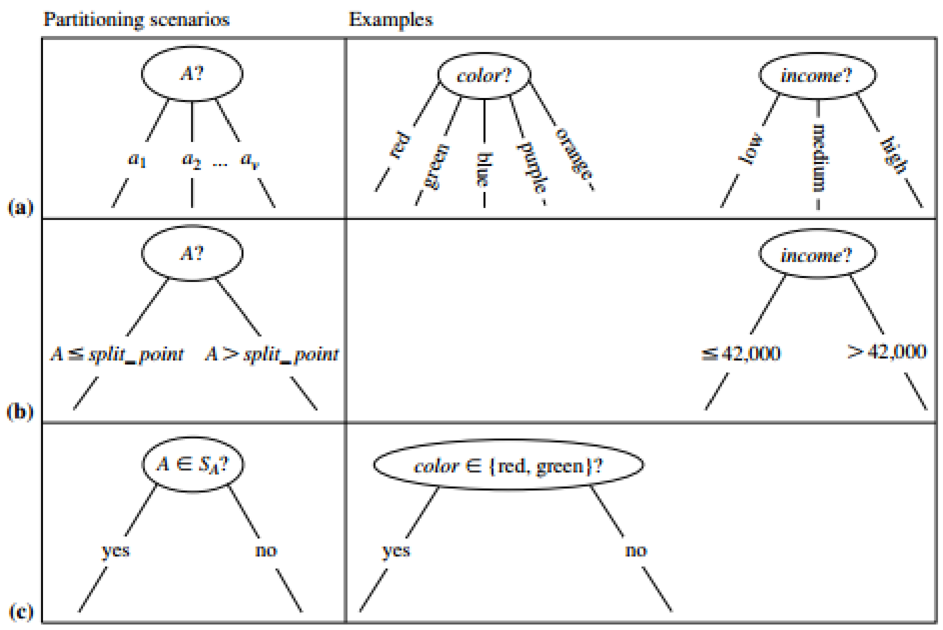


Figure 1

- The algorithm uses the same process recursively to form a decision tree for the tuples at each resulting partition, D_j, of D (step 14).
- The recursive partitioning stops only when any one of the following terminating conditions is true:
  1) All the tuples in partition D (represented at node N) belong to the same class (steps 2 and 3).
  2) There are no remaining attributes on which the tuples may be further partitioned (step 4). In this case, majority voting is employed (step 5). This involves converting node N into a leaf and labelling it with the most common class in D. Alternatively, the class distribution of the node tuples may be stored.
  3) There are no tuples for a given branch, that is, a partition D_j is empty (step 12). In this case, a leaf is created with the majority class in D (step 13).
- The resulting decision tree is returned (step 15).