Unit III

Classification

- **Classification**
  - **Basic Concepts**
  - **General Approach to solving a classification problem**
- **Decision Tree Induction**
  - **Working of decision tree**
  - **Building a decision tree**
  - **Methods for expressing an attribute test conditions**
  - **Measures for selecting the best split**
  - **Algorithm for decision tree induction**

Classification

Classification is the task of assigning objects to one of the several predefined categories. This includes wide applications. For example detecting spam messages, detecting malignant cells, classifying galaxies etc.

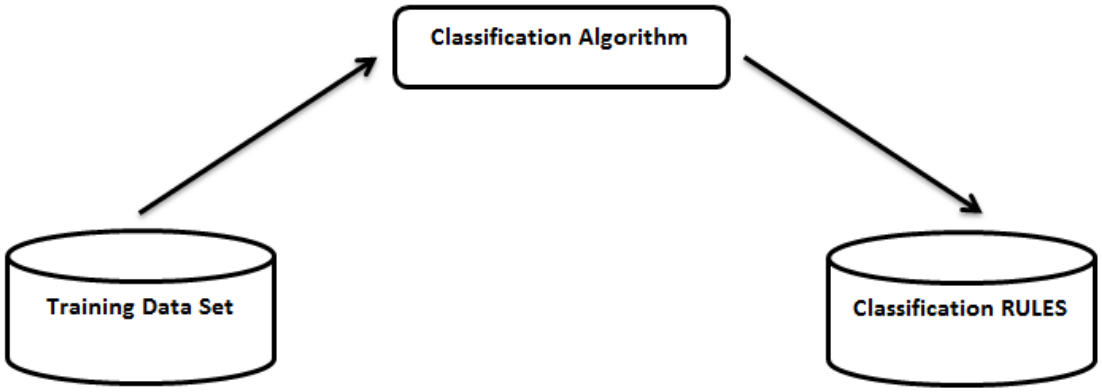
The input for classification task is a collection of records. The records are characterized by a tuple (X, y) where X denotes the attribute set and y denotes the class label (or target attribute).

Definition: Classification is the task of learning a target function f that maps each attribute set X to one of the predefined class labels y.

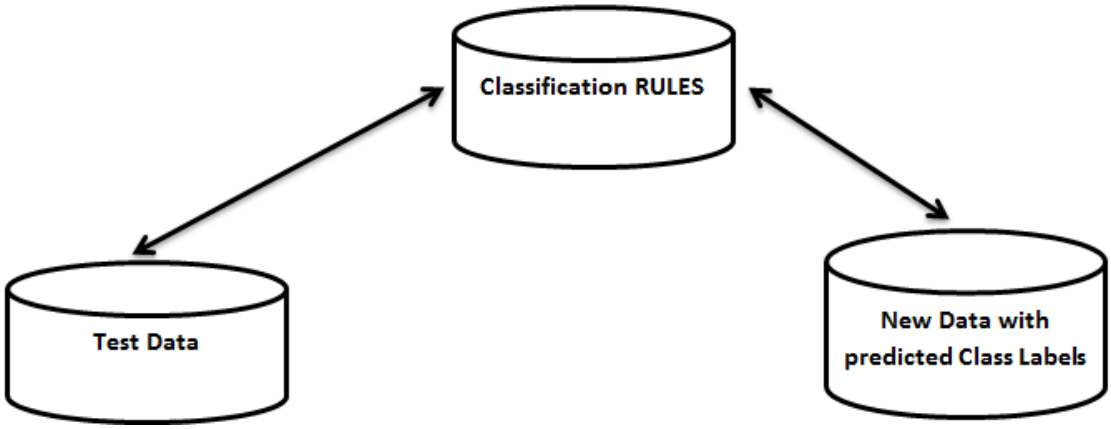
The target function is also known as classification model.

A classification model is useful for the following purposes (OR Classification is a two-step process)

- ✓ **Descriptive Modeling (OR Learning Step):** A classification model can serve as an explanatory tool to distinguish between objects of different classes. In this step a model is constructed for describing the classes (class labels or target classes). This is achieved by analyzing the database tuples (or records) which were described by attributes (attributes other than class attribute). Each tuple (or record) belongs to one class of class label attribute. Data set used for building the model for describing the classes is called as ***Training Data Set.*** Data tuples are also called as samples or objects or examples or records. As class label is associated with each tuple in training data set, this step is also known as **Supervised Learning.**



- ✓ **Predictive Modeling (OR Classification):** A classification model is used to predict the class labels of tuples with no class labels in Test Data Set. The predictive accuracy of the classifier is estimated. The accuracy is estimated by using methods like holdout, random sampling, cross validation & bootstrap.



Evaluation of the performance of a classification model *is based on the counts of test records (from test data set) correctly and incorrectly predicted by the model. These counts are tabulated in a table known as a* ***confusion matrix.*** The following table depicts a confusion matrix for a 2-class problem.

		Predicted Class	
		Class = 1	Class = 2
Actual Class	Class = 1	f_{11}	f_{10}
	Class = 2	f_{01}	f_{00}

For each entry f_{ij} in confusion matrix denotes the number of records of class i predicted to be of class j. ($f_{11} + f_{00}$) denotes correct number of predictions. ($f_{01} + f_{10}$) denotes incorrect number of predictions.

Performance metric like **accuracy** is defined as:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}}$$

Equivalently the performance of classification model is expressed in terms of **error rate.** This is defined as:

$$Error\ Rate = \frac{Number\ of\ incorrect\ predictions}{Total\ number\ of\ predictions} = \frac{f_{01} + f_{10}}{f_{11} + f_{10} + f_{01} + f_{00}}$$

Note: Prediction of Class Labels (discrete values) is known as *Classification* and prediction of continuous values is known as *Prediction.*

Classification and Prediction methods are compared and evaluated according to the following criteria:

- **Predictive accuracy:** This refers to the ability of the model to correctly predict the class label.
- **Speed:** This refers to the computation costs involved in generating and using the model.
- **Robustness:** This is the ability of the model to make correct predictions even if noisy data or missing values exits.
- **Scalability:** This refers to the ability to construct the model efficiently even if large amounts of data are present.
- **Interpretability:** This refers to the level of understanding that is provided by the model.

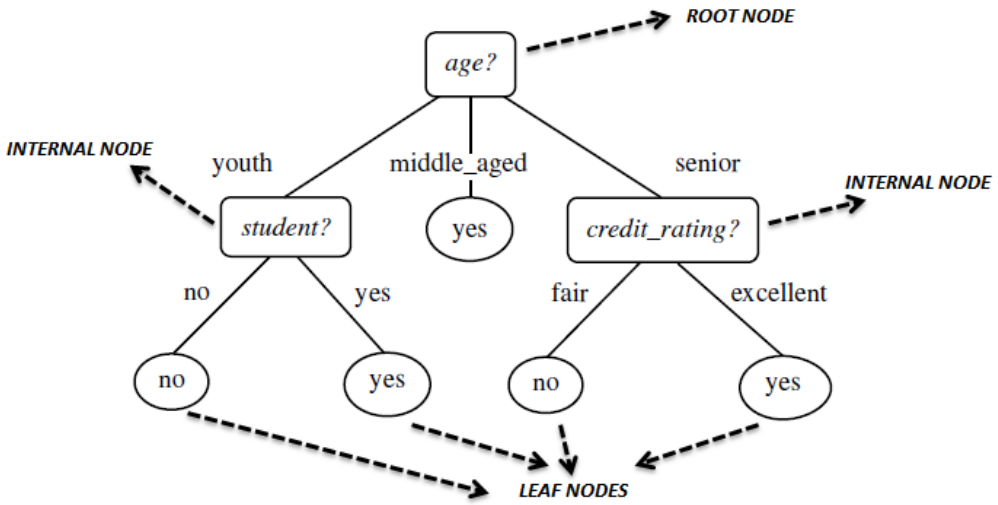
DECISION TREE INDUCTION

How a Decision Tree Works?

Decision tree consists of three types of nodes. They are:

- A **root node** has no incoming edges and zero or more outgoing edges.
- **Internal Nodes,** each has exactly one incoming edge and two or more outgoing edges.
- **Leaf or Terminal Nodes,** each has one incoming edge and no outgoing edges.

In a decision tree, each leaf node is assigned a **class label.** The **non-terminal nodes including root node** performs test and based on the outcomes the records are separated.



How to build a Decision Tree?

For building decision tree most of the algorithms follow a Greedy approach. For example ID3, C4.5, CART follows this approach. In these top-down recursive divide-and-conquer (partitioning strategy) manner is followed for constructing decision tree. The basis for all these decision tree induction algorithms is **Hunt's Algorithm.**

Hunt's Algorithm:

In Hunt's Algorithm, a decision tree is grown in a recursive fashion by partitioning the training records into successively purer subsets.

Let, D be the set of training records that are associated with node t

$y = \{y_1, y_2, \ldots..y_c\}$ be the class labels

Step 1:

If all the records in D belong to the same class y_t, the t is a leaf node (class label) labeled as y_t.

Step 2:

If D contains records that belong to more than one class, an ***attribute test condition*** is selected to partition the records into smaller subsets. A child node is created for each outcome of the test condition and the records in D are distributed to the children based on the outcomes. The algorithm is then recursively applied to each child node.

Hunt's algorithm work if every record of training data set has a unique class label.

Additional conditions for the following cases:

1. It is possible for some of the child nodes created in step 2 to be empty i.e. there are no records associated with these nodes. In such cases the node is declared as leaf node with the same class label.
2. In step 2, if all the records associated with D have identical attribute values then these records cannot be partitioned further and the node is declared as leaf node with same class label.

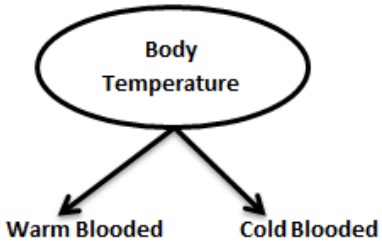
Design Issues of Decision Tree Induction

Two issues:

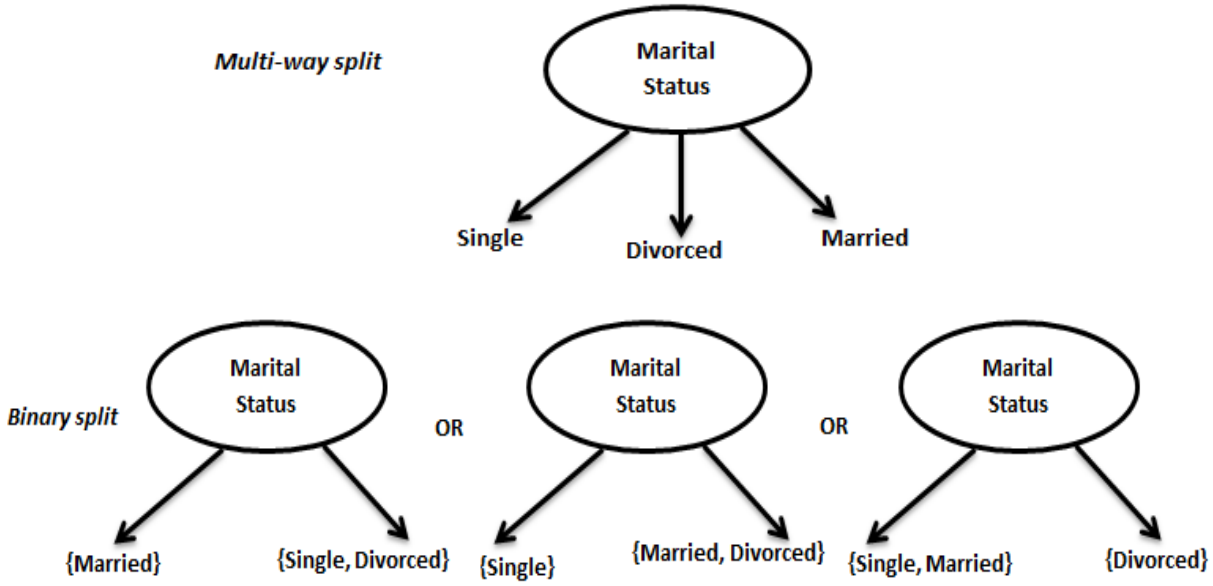
1. *How should the training records be split?*
   Each recursive step of the tree-growing process must select an attribute test condition to divide the records into smaller subsets. To implement this step, the algorithm must provide a method for specifying the test condition for different attribute types as well as an objective measure for evaluating the goodness of each test condition.
2. *How should the splitting procedure stop?*
   A stopping condition is needed to terminate the tree-growing process. A possible strategy is to continue expanding a node until either all the records belong to the same class or all records have identical attribute values.

Methods for Expressing Attribute Test Conditions

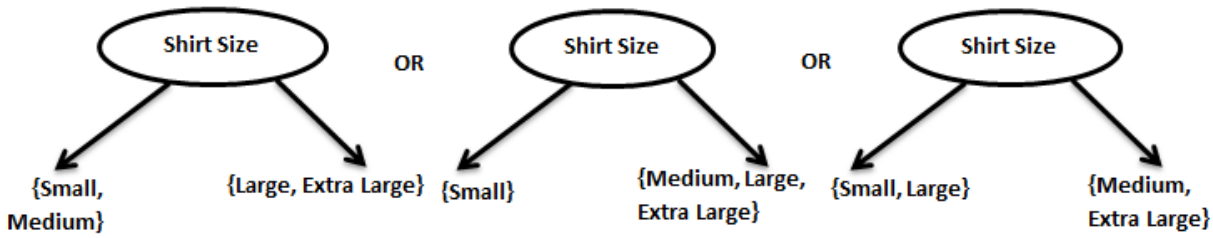
Binary Attributes: The test condition for a binary attribute generates two potential outcomes.



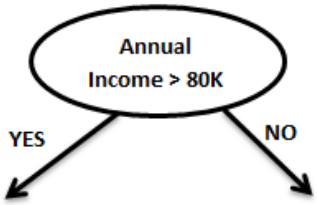
Nominal Attributes: A nominal attribute can have many values; its test condition can be expressed in two ways: *Multi-way split* and *Binary split* (by grouping attribute values). Decision tree algorithm CART considers binary splits. (It considers all $2^k - 1$ ways of binary splits for k attribute values).

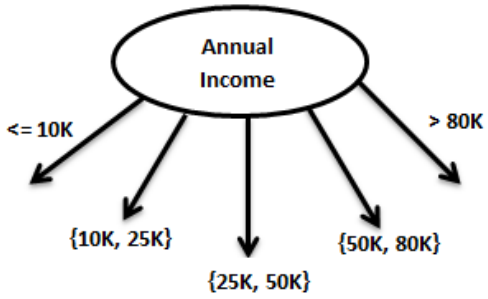


Ordinal Attributes: Ordinal attributes can also produce binary or multi-way splits. Ordinal attribute values can be grouped as along as the grouping does not violate the order property of the attribute values.



Continuous Attributes: For continuous attributes, the test condition can be expressed as a comparison test ($A < v$) or ($A >= v$) with binary outcomes or a range query with outcomes of the form $v_i <= A < v_{i+1}$.





Algorithm for DECISION TREE INDUCTION:

Most of the classification methods adopt Greedy method (non-backtracking) for constructing the Decision Tree in a top-down recursive divide-and-conquer manner. These algorithms start with a training data set with class labels. The training data set is recursively partitioned into smaller subsets as the tree is constructed.

The basic algorithm for decision tree is as follows:

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.*

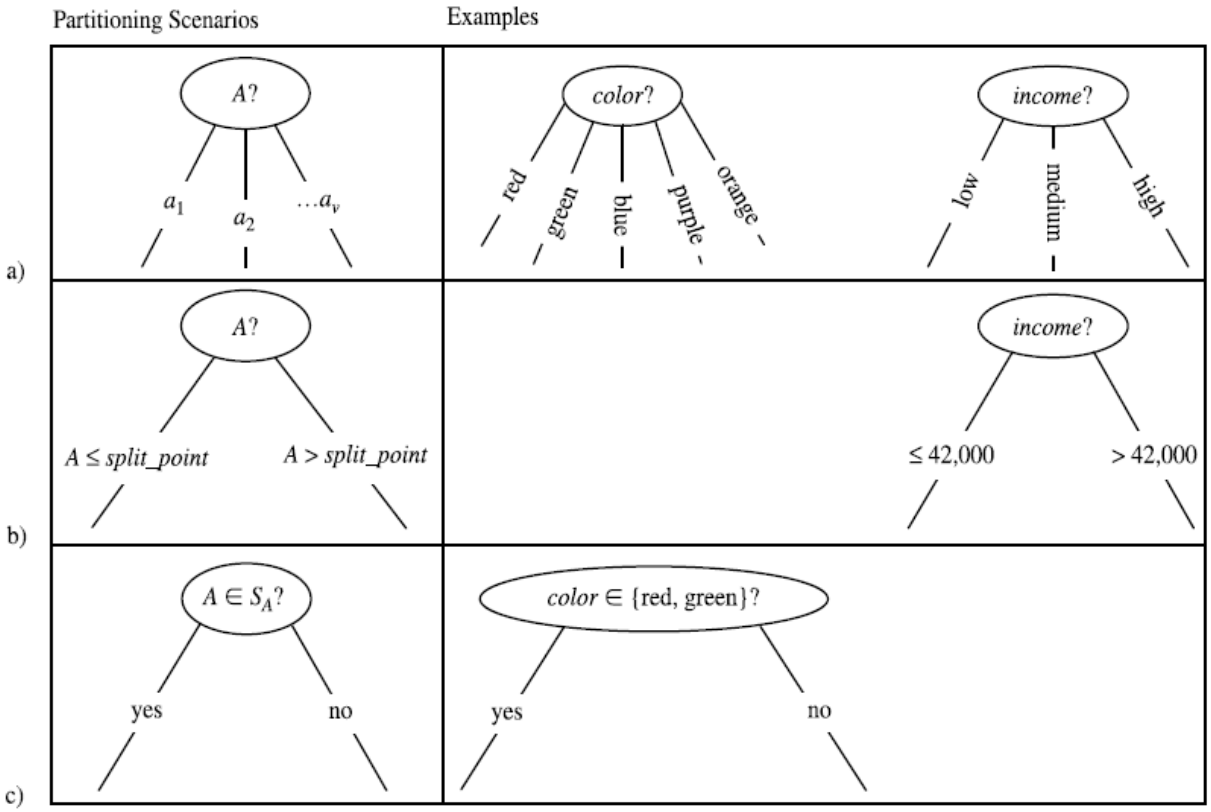
Output: *A decision tree.*

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 *Dj* be the set of data tuples in *D* satisfying outcome *j*; // a partition
(12) if *Dj* 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(*Dj*, *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. Some attribute selection measures, such as the gini index, enforce the resulting tree to be binary. Others, like information gain, do not, allowing multi-way 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 labeled with that class (steps 2 and 3). Note that steps 4 and 5 are terminating conditions. All of the 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. More specifically, the splitting criterion indicates the splitting attribute and may also indicate either a split-point or a splitting subset.
- The node *N* is labeled 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 below. Let *A* be the splitting attribute. *A* has *v* distinct values, $\{a_1, a_2, \ldots, a_v\}$, 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 labeled with that value. Partition D_j is the subset of class-labeled tuples in *D* having value a_j of *A*. Because all of the tuples in a given partition have the same value for *A*, then *A* need not be considered in any future partitioning of the tuples. Therefore, it is removed from *attribute list* (steps 8 to 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. The tuples are partitioned such that*D*1 holds the subset of class-labeled tuples in *D* for which *A* <= *split point*, while *D*2 holds the rest.
    3. ***A is discrete-valued* and a *binary tree* must be produced:** The test at node *N* is of the form "$A \in S_A$?". 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 $a_j \in SA$, then the test at node *N* is satisfied. Two branches are grown from *N*. By convention, the left branch out of *N* is labeled *yes* so that *D*1 corresponds to the subset of class-labeled tuples in *D* that satisfy the test.

The right branch out of *N* is labeled *no* so that *D*2 corresponds to the subset of class-labeled tuples from *D* that do not satisfy the test.



- 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 of the tuples in partition *D* (represented at node *N*) belong to the same class (steps 2 and 3), or

  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 labeling 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 *Dj* is empty (step 12). In this case, a leaf is created with the majority class in *D* (step 13).

1. The resulting decision tree is returned (step 15).

The computational complexity of the algorithm given training set *D* is $O(n \times |D| \times log(|D|))$, where *n* is the number of attributes describing the tuples in *D* and $|D|$ is the number of training tuples in *D*. This means that the computational cost of growing a tree grows at most $n \times |D| \times log(|D|)$ with $|D|$ tuples.

Attribute Selection Measures:

An attribute selection measure is a heuristic for selecting the splitting criterion that "best" separates a given data partition, *D*, of class-labeled training tuples into individual classes. If we were to split *D* into smaller partitions according to the outcomes of the

splitting criterion, ideally each partition would be pure. Attribute selection measures are also known as splitting rules because they determine how the tuples at a given node are to be split. The attribute selection measure provides a ranking for each attribute describing the given training tuples. The attribute having the best score for the measure is chosen as the *splitting attribute* for the given tuples. If the splitting attribute is continuous-valued or if we are restricted to binary trees then, respectively, either a *split point* or a *splitting subset* must also be determined as part of the splitting criterion. The tree node created for partition *D* is labeled with the splitting criterion, branches are grown for each outcome of the criterion, and the tuples are partitioned accordingly.

This section describes three popular attribute selection measures:

1. *Information gain*
2. *gain ratio*, and
3. *gini index*.

The notation used herein is as follows. Let *D*, the data partition, be a training set of class-labeled tuples. Suppose the class label attribute has *m* distinct values defining *m* distinct classes, C_i (for $i = 1, : : : , m$). Let $C_{i,D}$ be the set of tuples of class C_i in *D*. Let $|D|$ and $|C_{i,D}|$ denote the number of tuples in *D* and $C_{i,D}$, respectively.

Information Gain:

Let node *N* represent or hold the tuples of partition *D*. The attribute with the highest information gain is chosen as the splitting attribute for node *N*. This attribute minimizes the information needed to classify the tuples in the resulting partitions and reflects the least randomness or "impurity" in these partitions. The expected information needed to classify a tuple in *D* is given by,

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

where p_i is the probability that an arbitrary tuple in *D* belongs to class C_i and is estimated by $|C_{i,D|}/|D|$. A log function to the base 2 is used, because the information is encoded in bits. *Info(D)* is just the average amount of information needed to identify the class label of a tuple in *D*. Note that, at this point, the information we have is based solely on the proportions of tuples of each class. *Info(D)* is also known as the entropy of *D*.

Now, suppose we were to partition the tuples in *D* on some attribute *A* having *v* distinct values, $\{a_1, a_2, \ldots, a_v\}$, as observed from the training data. If *A* is discrete-valued, these values correspond directly to the *v* outcomes of a test on *A*. Attribute *A* can be used to split *D* into *v* partitions or subsets, $\{D_1, D_2, \ldots, D_v\}$, where D_j contains those tuples in *D* that have outcome a_j of *A*. These partitions would correspond to the branches grown from node *N*. Ideally, we would like this partitioning to produce an exact classification of the tuples. That is, we would like for each partition to be pure. However, it is quite likely that the partitions will be impure (e.g., where a partition may contain a collection of tuples from different classes rather than from a single class). This amount is measured by

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j).$$

The term $|D_j|/|D|$ acts as the weight of the jth partition. $Info_A(D)$ is the expected information required to classify a tuple from D based on the partitioning by A. The smaller the expected information (still) required, the greater the purity of the partitions.
Information gain is defined as the difference between the original information requirement (i.e., based on just the proportion of classes) and the new requirement (i.e., obtained after partitioning on A). That is,

$$Gain(A) = Info(D) - Info_A(D)$$

The attribute A with the highest information gain, ($Gain(A)$), is chosen as the splitting attribute at node N.
Consider the following data set,

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

Number of classes: 02 ({yes, no})

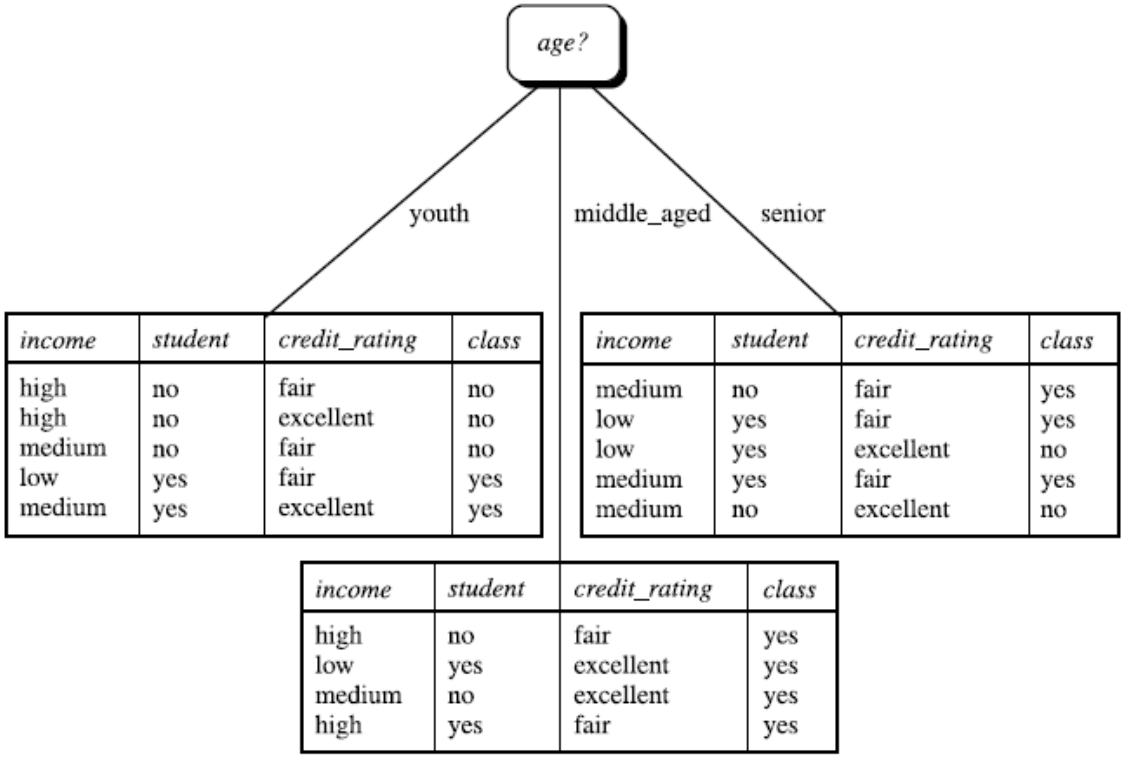
$$Info(D) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940 \text{ bits.}$$

$$Info_{age}(D) = \frac{5}{14} \times (-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}) + \frac{4}{14} \times (-\frac{4}{4}\log_2\frac{4}{4} - \frac{0}{4}\log_2\frac{0}{4}) + \frac{5}{14} \times (-\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}) = 0.694 \text{ bits.}$$

Hence, the gain in information from such a partitioning would be

$$Gain(age) = Info(D) - Info_{age}(D) = 0.940 - 0.694 = 0.246 \text{ bits.}$$

Similarly, we can compute *Gain*(*income*) = 0.029 bits, *Gain*(*student*) = 0.151 bits, and *Gain*(*credit rating*) = 0.048 bits. Because *age* has the highest information gain among the attributes, it is selected as the splitting attribute. Node *N* is labeled with *age*, and branches are grown for each of the attribute's values.



income	*student*	*credit_rating*	*class*
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

income	*student*	*credit_rating*	*class*
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

income	*student*	*credit_rating*	*class*
high	no	fair	yes
low	yes	excellent	yes
medium	no	excellent	yes
high	yes	fair	yes

Suppose, instead, that we have an attribute *A* that is continuous-valued, rather than discrete-valued. (For example, suppose that instead of the discretized version of *age* above, we instead have the raw values for this attribute.) For such a scenario, we must determine the "best" split-point for *A*, where the split-point is a threshold on *A*. We first sort the values of *A* in increasing order. Typically, the midpoint between each pair of adjacent values is considered as a possible split-point. Therefore, given *v* values of *A*, then *v*-1 possible splits are evaluated. For example, the midpoint between the values a_i and a_{i+1} of *A* is

$$\frac{a_i + a_{i+1}}{2}$$

If the values of *A* are sorted in advance, then determining the best split for *A* requires only one pass through the values. For each possible split-point for *A*, we evaluate $Info_A(D)$, where the number of partitions is two, that is *v* = 2 (or *j* = 1, 2). The point with the minimum expected information requirement for *A* is selected as the *split_point* for *A*. *D*1 is the set of tuples in *D* satisfying *A* <= *split_point*, and *D*2 is the set of tuples in *D* satisfying *A* > *split point*.

Gain Ratio

An extension to information gain known as *gain ratio*, which attempts to overcome the bias i.e. whenever there is a tie in information gain values of attributes. It applies a kind of normalization to information gain using a "split information" i.e.

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

This value represents the potential information generated by splitting the training data set, *D*, into *v* partitions, corresponding to the *v* outcomes of a test on attribute *A*. Note that, for each outcome, it considers the number of tuples having that outcome with respect to the total number of tuples in *D*. It differs from information gain, which measures the information with respect to classification that is acquired based on the same partitioning. The gain ratio is defined as

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo(A)}$$

The attribute with the maximum gain ratio is selected as the splitting attribute.

To compute the gain ratio of *income*, we obtain

$$SplitInfo_A(D) = -\frac{4}{14} \times \log_2\left(\frac{4}{14}\right) - \frac{6}{14} \times \log_2\left(\frac{6}{14}\right) - \frac{4}{14} \times \log_2\left(\frac{4}{14}\right).$$

$$= 0.926.$$

Gain(*income*) = 0.029.
Therefore, *GainRatio*(*income*) = 0.029/0.926 = 0.031.

Gini Index

The Gini index measures the impurity of *D*, a data partition or set of training tuples, as

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2$$

where *pi* is the probability that a tuple in *D* belongs to class *Ci* and is estimated by $|C_{i,D}|/|D|$. The sum is computed over *m* classes.

The Gini index considers a binary split for each attribute. Let's first consider the case where *A* is a discrete-valued attribute having *v* distinct values, $\{a_1, a_2, \ldots, a_v\}$, occurring in *D*. To determine the best binary split on *A*, we examine all of the possible subsets that can be formed using known values of *A*. Each subset, *SA*, can be considered as a binary test for attribute *A* of the form "*A* 2 *SA*?". Given a tuple, this test is satisfied if the value of *A* for the tuple is among the values listed in *SA*. If *A* has *v* possible values, then there are 2*v* possible subsets. For example, if *income* has three possible values, namely {*low, medium, high*}, then the possible subsets are {*low, medium, high}*, {*low, medium}*, {*low, high}*, {*medium, high}*, {*low}*, {*medium}*, {*high}*, and { }. We exclude the power set, {*low, medium, high*}, and the empty set from consideration since, conceptually, they do not represent a split. Therefore, there are 2^v-2 possible ways to form two partitions of the data, *D*, based on a binary split on *A*.

$$Gini_A(D) = \frac{|D_1|}{|D|}Gini(D_1) + \frac{|D_2|}{|D|}Gini(D_2)$$

For continuous-valued attributes, each possible split-point must be considered. The strategy is similar to that described above for information gain, where the midpoint between each pair of (sorted) adjacent values is taken as a possible split-point. The point giving the minimum Gini index for a given (continuous-valued) attribute is taken as the split-point of that attribute. Recall that for a possible split-point of *A*, *D*1 is the set of tuples in *D* satisfying *A* <= *split_point*, and *D*2 is the set of tuples in *D* satisfying *A* > *split_point*.

The reduction in impurity that would be incurred by a binary split on a discrete- or continuous-valued attribute *A* is

$$\Delta Gini(A) = Gini(D) - Gini_A(D)$$

The attribute that maximizes the reduction in impurity (or, equivalently, has the minimum Gini index) is selected as the splitting attribute. This attribute and either its splitting subset (for a discrete-valued splitting attribute) or split-point (for a continuous valued splitting attribute) together form the splitting criterion.

Gini index to compute the impurity of *D*

$$Gini(D) = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 = 0.459$$

To find the splitting criterion for the tuples in *D*, we need to compute the gini index for each attribute. Let's start with the attribute *income* and consider each of the possible splitting subsets. Consider the subset {*low, medium*}. This would result in 10 tuples in partition *D*1 satisfying the condition "*income* ∈ {*low, medium}*." The remaining four tuples of *D* would be assigned to partition *D*2. The Gini index value computed based on this partitioning is

$$
\begin{aligned}
&Gini_{income \in \{low,medium\}}(D) \\
&= \frac{10}{14} Gini(D_1) + \frac{4}{14} Gini(D_2) \\
&= \frac{10}{14}\left(1 - \left(\frac{7}{10}\right)^2 - \left(\frac{3}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2\right) \\
&= 0.450 \\
&= Gini_{income \in \{high\}}(D).
\end{aligned}
$$

Similarly, the Gini index values for splits on the remaining subsets are: 0.315 (for the subsets {*low, high}* and {*medium*}) and 0.300 (for the subsets {*medium, high}* and {*low}*). Therefore, the best binary split for attribute *income* is on {*medium, high}* (or {*low*}) because it minimizes the gini index.