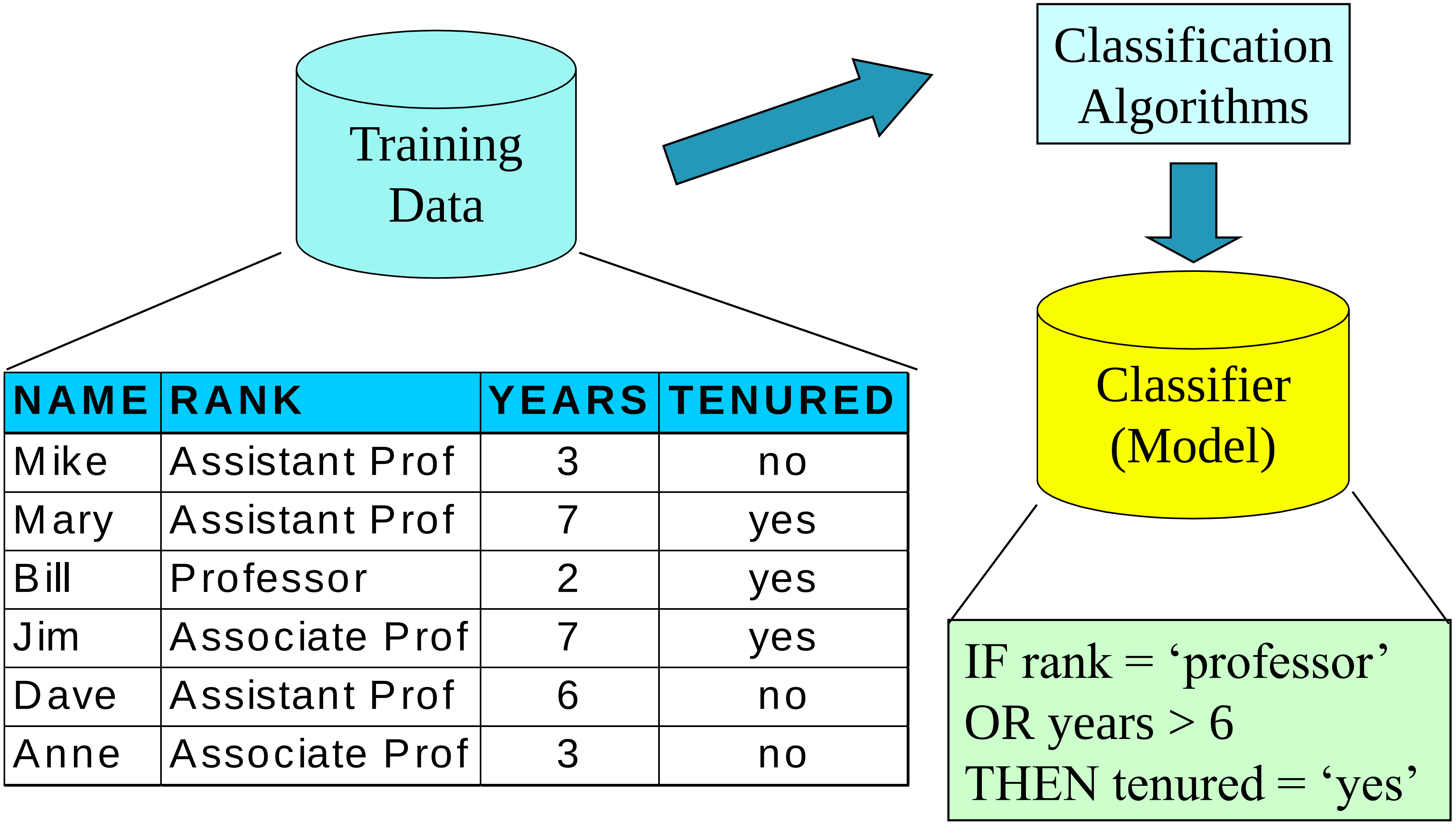
Supervised vs. Unsupervised Learning

- Supervised learning (classification)
  - Supervision: The training data (observations, measurements, etc.) are accompanied by **labels** indicating the class of the observations
  - New data is classified based on the training set
- Unsupervised learning (clustering)
  - The class labels of training data is unknown
  - Given a set of measurements, observations, etc. with the aim of establishing the existence of classes or clusters in the data

Classification—A Two-Step Process

- Model construction: describing a set of predetermined classes
  - Each tuple/sample is assumed to belong to a predefined class, as determined by the class label attribute
  - The set of tuples used for model construction is training set
  - The model is represented as classification rules, decision trees, or mathematical formulae
- Model usage: for classifying future or unknown objects
  - Estimate accuracy of the model
    - The known label of test sample is compared with the classified result from the model
    - Accuracy rate is the percentage of test set samples that are correctly classified by the model
    - Test set is independent of training set
  - If the accuracy is acceptable, use the model to classify new data
- Note: If *the test set* is used to select models, it is called validation (test) set

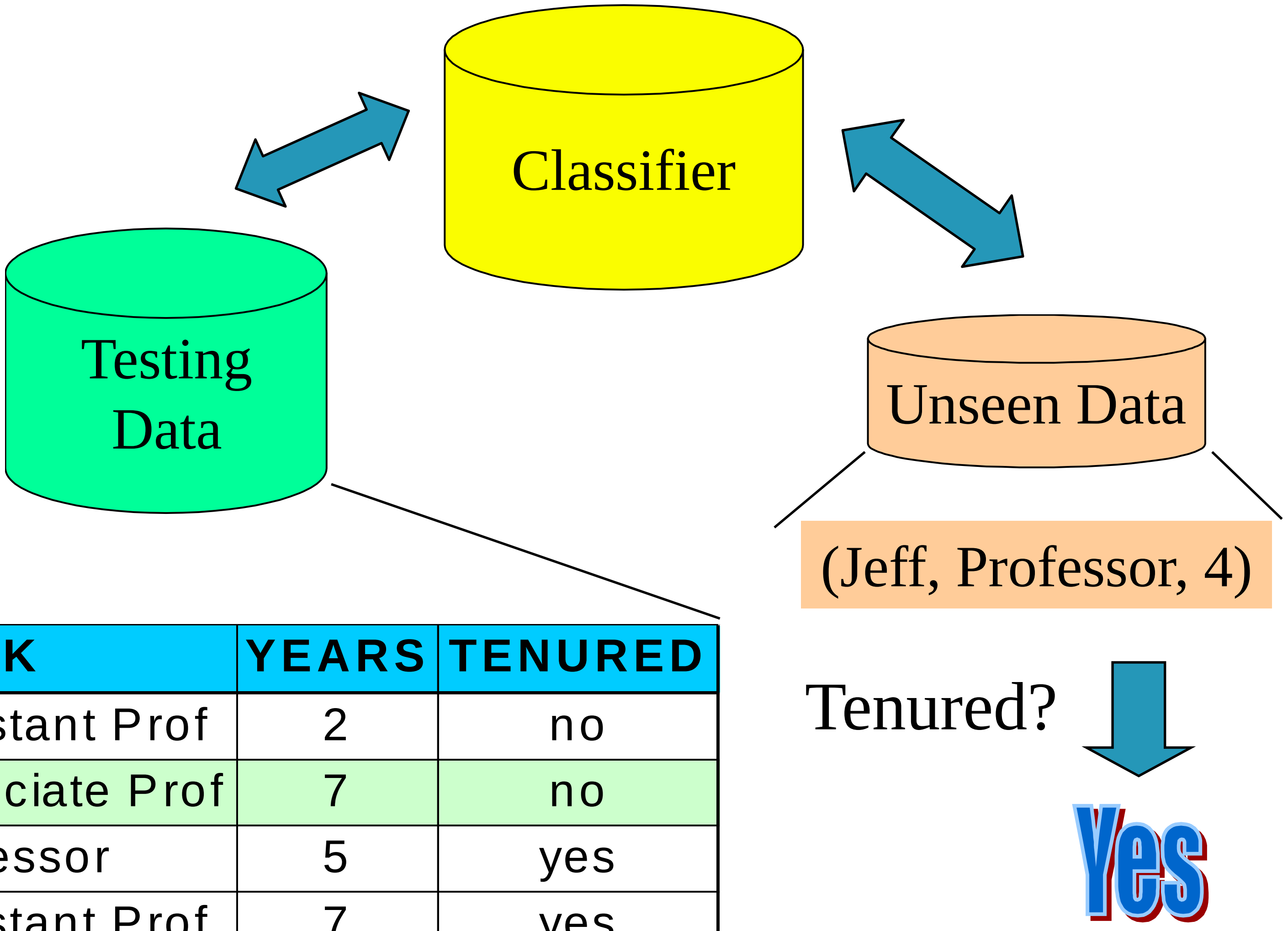
Process (1): Model Construction



NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

IF rank = ‘professor’
OR years > 6
THEN tenured = ‘yes’

Process (2): Using the Model in Prediction



NAME	RANK	YEARS	TENURED
Tom	Assistant Prof	2	no
Merlisa	Associate Prof	7	no
George	Professor	5	yes
Joseph	Assistant Prof	7	yes

Chapter 8. Classification: Basic Concepts

- Classification: Basic Concepts
- Decision Tree Induction
- Bayes Classification Methods
- Rule-Based Classification
- Model Evaluation and Selection
- Techniques to Improve Classification Accuracy: Ensemble Methods

Decision Tree Induction

- Decision tree induction is the learning of decision trees from class-labeled training tuples.
- A decision tree is a flowchart-like tree structure, where each internal node (non-leaf node) denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (or *terminal node) holds a class label.*

Decision Tree Induction

- *The topmost node in* a tree is the root node.
- Internal nodes are denoted by rectangles, and leaf nodes are denoted by ovals.
- Some decision tree algorithms produce only *binary trees (where* each internal node branches to exactly two other nodes), whereas others can produce nonbinary trees.
- Decision trees can easily be converted to classification rules.

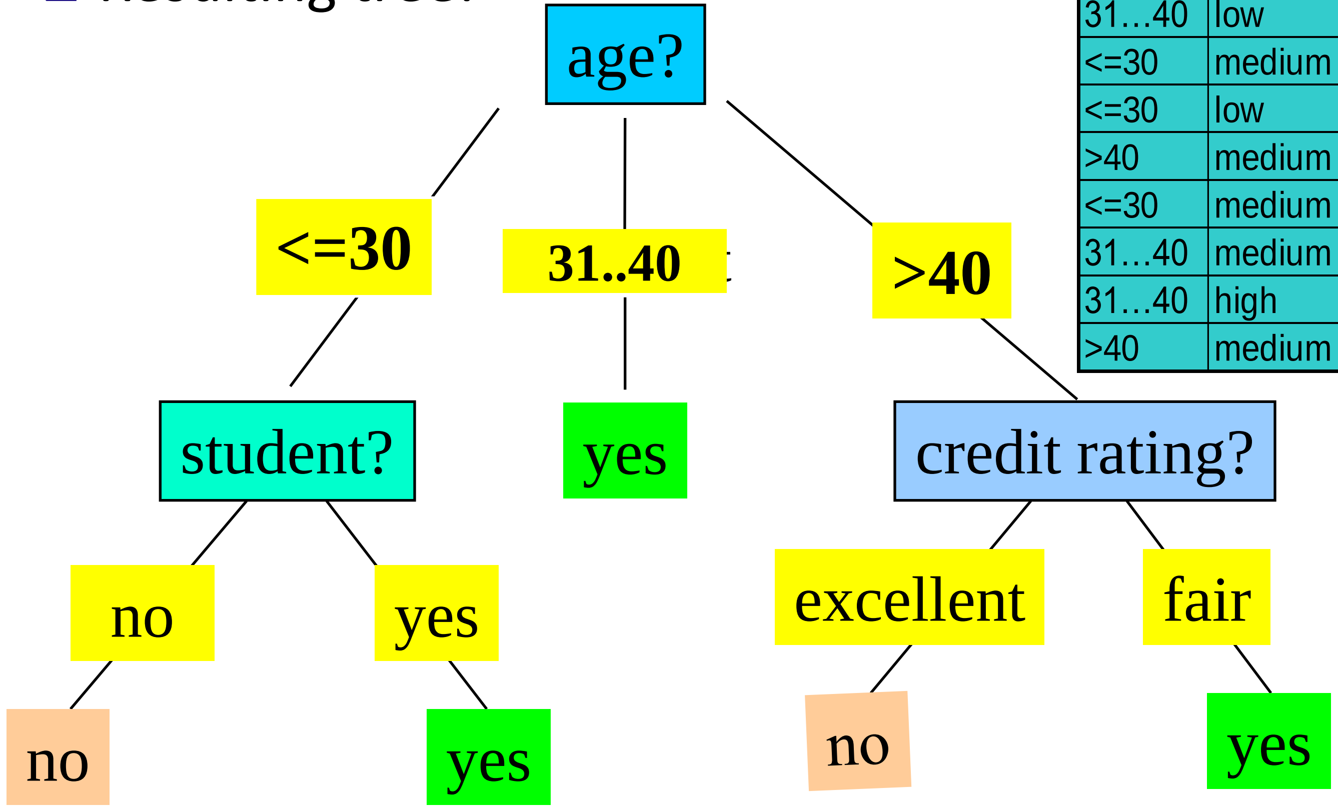
Why are decision tree classifiers so popular?

- The construction of decision tree classifiers ***does not require any domain knowledge*** or parameter setting, and therefore is appropriate for exploratory knowledge discovery.
- Decision trees ***can handle multidimensional data***. Their representation of acquired knowledge in tree form is intuitive and generally ***easy to assimilate by humans***.
- The learning and classification steps of decision tree induction are ***simple and fast***.
- In general, decision tree classifiers have ***good accuracy***.
- However, successful use may depend on the data at hand.
- Decision tree induction algorithms have been used for classification in many application areas such as medicine, manufacturing and production, financial analysis, astronomy, and molecular biology

Decision Tree Induction: An Example

- Training data set: Buys_computer
- The data set follows an example of Quinlan's ID3 (Playing Tennis)
- Resulting tree:

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no



Algorithm for Decision Tree Induction

- Basic algorithm (a greedy algorithm)
  - Tree is constructed in a top-down recursive divide-and-conquer manner
  - At start, all the training examples are at the root
  - Attributes are categorical (if continuous-valued, they are discretized in advance)
  - Examples are partitioned recursively based on selected attributes
  - Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)
- Conditions for stopping partitioning
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
  - There are no samples left

Algorithm for Decision Tree Induction

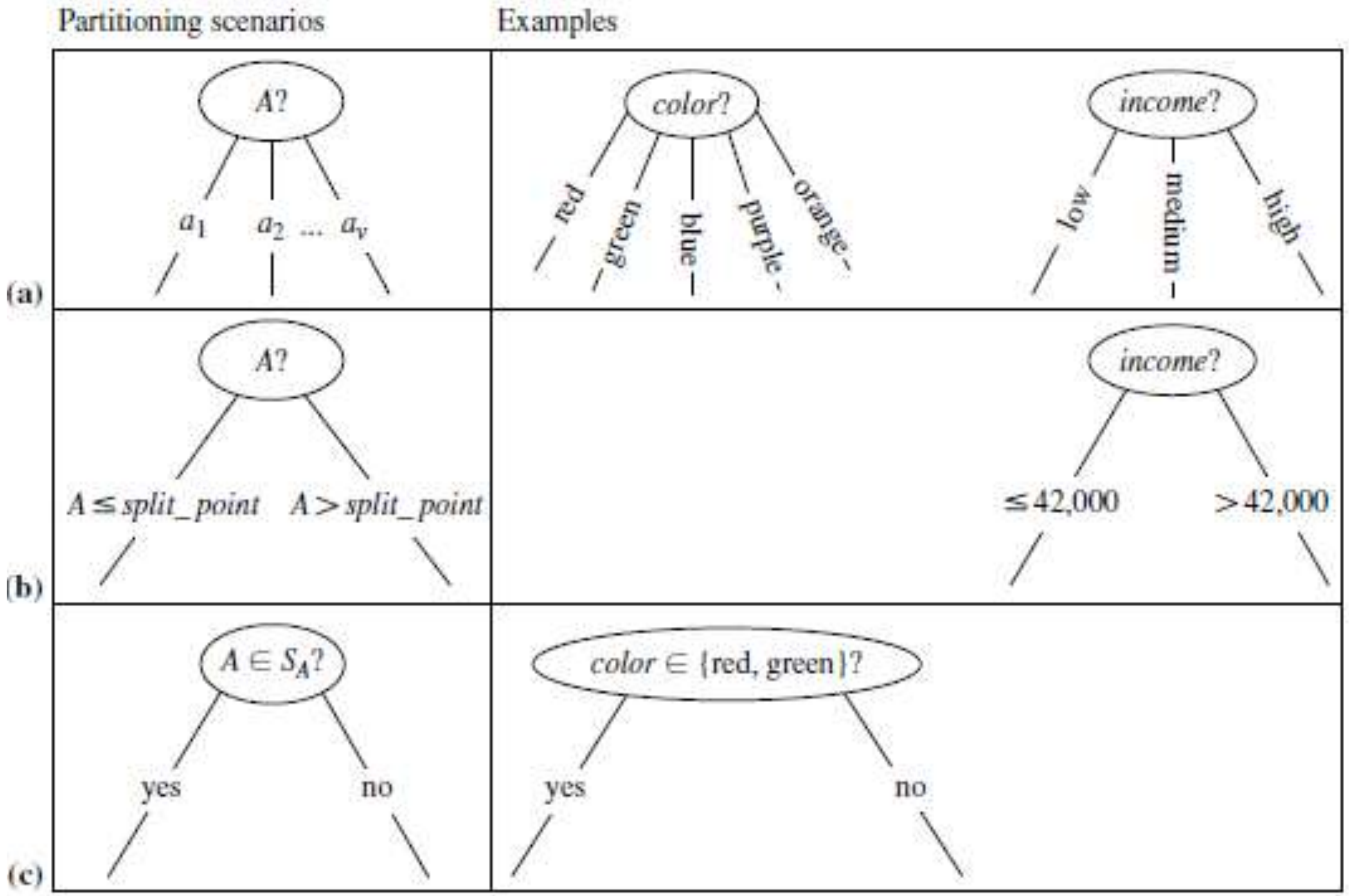
- **Algorithm:** Generate a decision tree from the training tuples of data partition, *D.*
- **Input:**
  - Data partition***, D****, which is a set of training tuples and their associated class labels;*
  - ***attribute list****, the set of candidate attributes;*
  - ***Attribute selection method****, 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
(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;
```

Algorithm for Decision Tree Induction

- 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.

Algorithm for Decision Tree Induction



Attribute Selection Measure: Information Gain (ID3/C4.5)

- Select the attribute with the *highest information gain*
- Let p_i be the probability that an arbitrary tuple in D belongs to class C_i, estimated by $|C_{i, D}|/|D|$
- Expected information (Entropy) needed to classify a tuple in D:

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

- Information needed (after using A to split D into v partitions) to classify D:

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

- Information gained by branching on attribute A

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

Attribute Selection: Information Gain

- Class P: buys_computer = "yes"
- Class N: buys_computer = "no"

$$Info(D) = I(9,5) = -\frac{9}{14}\log_2(\frac{9}{14}) - \frac{5}{14}\log_2(\frac{5}{14}) = 0.940$$

age	p_i	n_i	$I(p_i, n_i)$
<=30	2	3	0.971
31…40	4	0	0
>40	3	2	0.971

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

$$Info_{age}(D) = \frac{5}{14}I(2,3) + \frac{4}{14}I(4,0) + \frac{5}{14}I(3,2) = 0.694$$

$\frac{5}{14}I(2,3)$ means "age <=30" has 5 out of 14 samples, with 2 yes'es and 3 no's. Hence

$$Gain(age) = Info(D) - Info_{age}(D) = 0.246$$

Similarly,

$$Gain(income) = 0.029$$
$$Gain(student) = 0.151$$
$$Gain(credit\_rating) = 0.048$$

Computing Information-Gain for Continuous-Valued Attributes

- Let attribute A be a continuous-valued attribute
- Must determine the best split point for A
  - Sort the value A in increasing order
  - Typically, the midpoint between each pair of adjacent values is considered as a possible *split point*
    - $(a_i+a_{i+1})/2$ is the midpoint between the values of a_i and a_{i+1}
  - The point with the *minimum expected information requirement* for A is selected as the split-point for A
- Split: $Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j).$ $\quad \mathrm{Info}_x(D) = \frac{|D_1|}{|D|}\mathrm{Entropy}(D_1) + \frac{|D_2|}{|D|}\mathrm{Entropy}(D_2)$
  - D1 is the set of tuples in D satisfying A ≤ split-point, and D2 is the set of tuples in D satisfying A > split-point

Information-Gain Drawback

- The information gain measure is biased toward tests with many outcomes. That is, it prefers to select attributes having a large number of values.
- For example, consider an attribute that acts as a unique identifier such as *product ID. A split on product ID would* result in a large number of partitions (as many as there are values), each one containing just one tuple. Because each partition is pure, the information required to classify dataset *D based on this partitioning would be*

$$Info_{product\_ID}(D) = 0$$

- *Therefore, the information* gained by partitioning on this attribute is maximal. Clearly, such a partitioning is useless for classification

Gain Ratio for Attribute Selection (C4.5)

- C4.5 (a successor of ID3) uses gain ratio to overcome the problem (normalization to information gain)

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

  - GainRatio(A) = Gain(A)/SplitInfo(A)
- Ex.

$$SplitInfo_{income}(D) = -\frac{4}{14} \times \log_2\left(\frac{4}{14}\right) - \frac{6}{14} \times \log_2\left(\frac{6}{14}\right) - \frac{4}{14} \times \log_2\left(\frac{4}{14}\right) = 1.557$$

  - gain_ratio(income) = 0.029/1.557 = 0.019
- The attribute with the maximum gain ratio is selected as the splitting attribute

Gini Index (CART, IBM IntelligentMiner)

- If a data set *D* contains examples from *n* classes, gini index, *gini*(*D*) is defined as

$$gini(D) = 1 - \sum_{j=1}^{n} p_j^2$$

  where p_j is the relative frequency of class *j* in *D*

- If a data set *D* is split on A into two subsets D_1 and D_2, the *gini* index *gini*(*D*) is defined as

$$gini_A(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$$

- Reduction in Impurity:

$$\Delta gini(A) = gini(D) - gini_A(D)$$

- **The attribute provides the smallest *gini*$_{split}$(*D*)** (or the largest reduction in impurity) **is chosen to split the node** (*need to enumerate all the possible splitting points for each attribute*)

Gini Index

- 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, {a1, a2, : : : , av}, occurring* in *D.*
- ***To determine the best binary split on A****, we examine all 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 belongs 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*

Gini Index

- 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.*
- When considering a binary split, we compute a weighted sum of the impurity of each resulting partition. For example, if a binary split on *A partitions D into D1 and D2, the* Gini index of *D given that partitioning is*

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

Gini Index

- For each attribute, each of the possible binary splits is considered. For a discrete-valued attribute, the subset that gives the minimum Gini index for that attribute is selected as its splitting subset.
- For continuous-valued attributes, each possible split-point must be considered. The strategy is similar to that described earlier 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, D1 is the* set of tuples in *D satisfying A split point, and D2 is the set of tuples in D satisfying A > split point.*

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

Computation of Gini Index

- Ex. D has 9 tuples in buys_computer = "yes" and 5 in "no"

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

- Suppose the attribute income partitions D into 10 in D_1: {low, medium} and 4 in D_2

$$gini_{income \in \{low,medium\}}(D) = \left(\frac{10}{14}\right) Gini(D_1) + \left(\frac{4}{14}\right) Gini(D_2)$$

$$= \frac{10}{14}\left(1 - \left(\frac{7}{10}\right)^2 - \left(\frac{3}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2\right)$$

$$= 0.443$$

$$= Gini_{income \in \{high\}}(D).$$

$Gini_{\{low,high\}}$ is 0.458; $Gini_{\{medium,high\}}$ is 0.450. Thus, split on the {low,medium} (and {high}) since it has the lowest Gini index

- All attributes are assumed continuous-valued
- May need other tools, e.g., clustering, to get the possible split values

Computation of Gini Index

- Evaluating age , we obtain { youth, senior } (or { middle aged}) as the best split for age with a Gini index of 0.357;
- the attributes student and credit rating are both binary, with Gini index values of 0.367 and 0.429, respectively.
- The attribute age and splitting subset {youth, senior} therefore give the minimum Gini index overall, with a reduction in impurity of 0.459 −0.357=0.102.
- The binary split "age∈{youth, senior?}" results in the maximum reduction in impurity of the tuples in D and is returned as the splitting criterion.
- Node N is labeled with the criterion, two branches are grown from it, and the tuples are partitioned accordingly.

Comparing Attribute Selection Measures

- The three measures, in general, return good results but
  - **Information gain**:
    - biased towards multivalued attributes
  - **Gain ratio**:
    - tends to prefer unbalanced splits in which one partition is much smaller than the others
  - **Gini index**:
    - biased to multivalued attributes
    - has difficulty when # of classes is large
    - tends to favor tests that result in equal-sized partitions and purity in both partitions

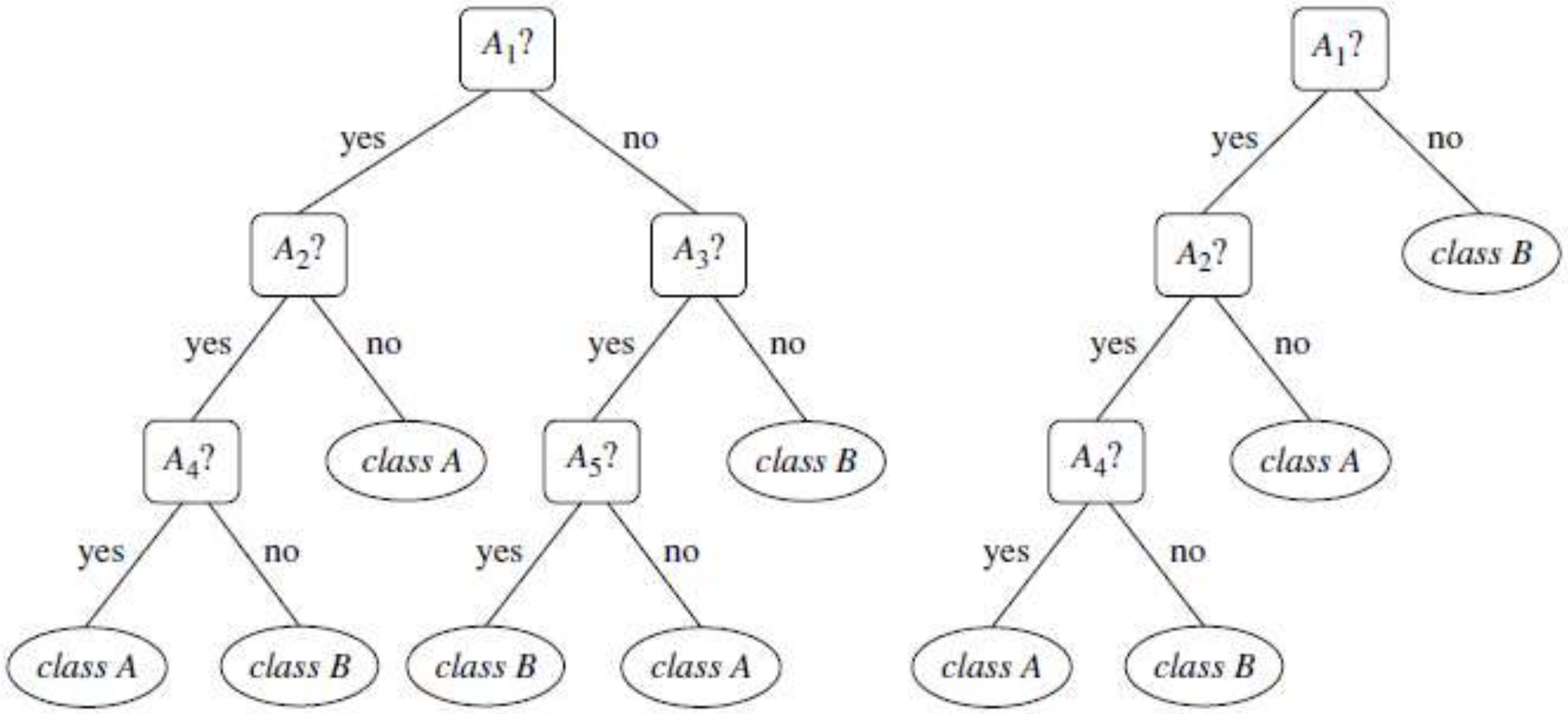
Other Attribute Selection Measures

- CHAID: a popular decision tree algorithm, measure based on χ^2 test for independence
- C-SEP: performs better than info. gain and gini index in certain cases
- G-statistic: has a close approximation to χ^2 distribution
- MDL (Minimal Description Length) principle (i.e., the simplest solution is preferred):
  - The best tree as the one that requires the fewest # of bits to both (1) encode the tree, and (2) encode the exceptions to the tree
- Multivariate splits (partition based on multiple variable combinations)
  - CART: finds multivariate splits based on a linear comb. of attrs.
- Which attribute selection measure is the best?
  - Most give good results, none is significantly superior than others

Overfitting and Tree Pruning

- Overfitting: An induced tree may overfit the training data
  - Too many branches, some may reflect anomalies due to noise or outliers
  - Poor accuracy for unseen samples
- Pruned trees tend to be smaller and less complex and, thus, easier to comprehend. They are usually faster and better at correctly classifying independent test data (i.e., of previously unseen tuples) than unpruned trees.

Tree Pruning



Overfitting and Tree Pruning

- Two approaches to avoid overfitting
  - Prepruning: *Halt tree construction early*-do not split a node if this would result in the goodness measure falling below a threshold
    - Difficult to choose an appropriate threshold
  - Postpruning: *Remove branches* from a "fully grown" tree—get a sequence of progressively pruned trees
    - Use a set of data different from the training data to decide which is the "best pruned tree"

Overfitting and Tree Pruning

- Two approaches to avoid overfitting
  - <u>Prepruning</u>: *Halt tree construction early*-do not split a node if this would result in the goodness measure falling below a threshold
    - When constructing a tree, measures such as statistical significance, information gain, Gini index, and so on, can be used to assess the goodness of a split. If partitioning the tuples at a node would result in a split that falls below a prespecified threshold, then further partitioning of the given subset is halted.High thresholds could result in oversimplified trees, whereas low thresholds could result in very little simplification.
    - Difficult to choose an appropriate threshold
  - <u>Postpruning</u>: *Remove branches* from a “fully grown” tree—get a sequence of progressively pruned trees
    - A subtree at a given node is pruned by removing its branches and replacing it with a leaf. The leaf is labeled with the most frequent class among the subtree being replaced.
    - Use a set of data different from the training data to decide which is the “best pruned tree”