Data Mining

3.2 Decision Tree Classifier

Fall 2008

Instructor: Dr. Masoud Yaghini

Outline

- Introduction
- Basic Algorithm for Decision Tree Induction
- Attribute Selection Measures
- Information Gain
- Gain Ratio
- Gini Index
- Tree Pruning
- Scalable Decision Tree Induction Methods
- References

Introduction

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
  - each leaf node (or *terminal node)* holds a class label*.*
  - The topmost node in a tree is the root node.

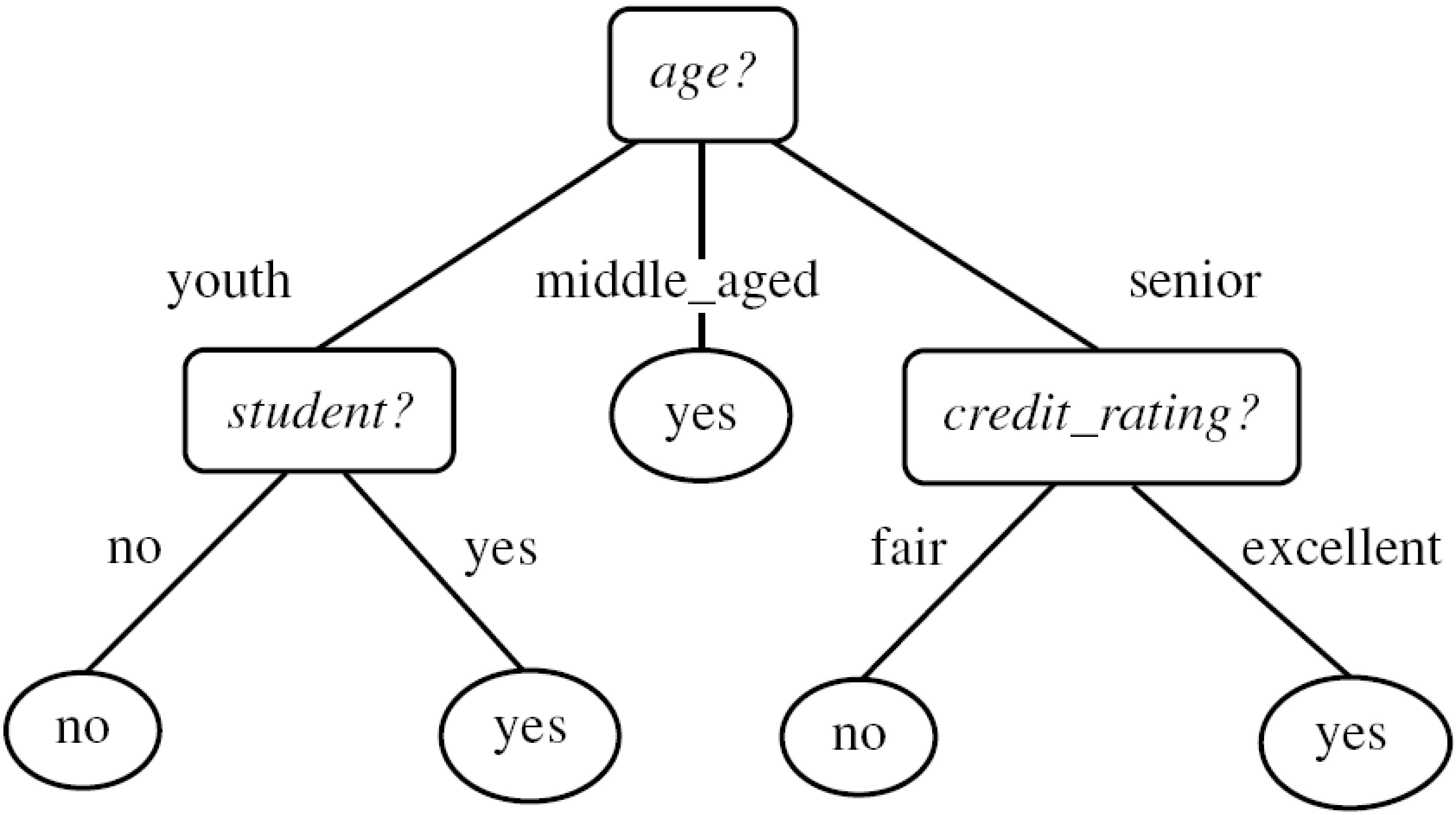
An example

- This example represents the concept *buys _computer,* that is, it predicts whether a customer at *AllElectronics is likely to purchase a computer.*

An example: Training Dataset

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

An example: A Decision Tree for "*buys_computer*"



Decision Tree Induction

- How are decision trees used for classification?
  - Given a tuple, ***X*,** for which the associated class label is unknown,
  - The attribute values of the tuple are tested against the decision tree
  - A path is traced from the root to a leaf node, which holds the class prediction for that tuple.

Decision Tree Induction

- Advantages of decision tree
  - The construction of decision tree classifiers does not require any domain knowledge or parameter setting.
  - Decision trees can handle high dimensional data.
  - Easy to interpret for small-sized trees
  - The learning and classification steps of decision tree induction are simple and fast.
  - Accuracy is comparable to other classification techniques for many simple data sets

Decision Tree Induction

- Decision tree induction algorithms have been used for classification in many application areas, such as:
  - Medicine
  - Manufacturing and production
  - Financial analysis
  - Astronomy
  - Molecular biology.

Decision Tree Induction

- Attribute selection measures
  - During tree construction, **attribute selection measures** are used to select the attribute that best partitions the tuples into distinct classes.
- Tree pruning
  - When decision trees are built, many of the branches may reflect noise or outliers in the training data.
  - Tree pruning attempts to identify and remove such branches, with the goal of improving classification accuracy on unseen data.
- Scalability
  - Scalability issues related to the induction of decision trees from large databases.

Decision Tree Induction Algorithms

- ID3 (Iterative Dichotomiser) algorithm
  - Developed by J. Ross Quinlan
  - During the late 1970s and early 1980s
- C4.5 algorithm
  - Quinlan later presented C4.5 (a successor of ID3)
  - Became a benchmark to which newer supervised learning algorithms are often compared.
  - Commercial successor: C5.0
- CART (Classification and Regression Trees) algorithm
  - The generation of binary decision trees
  - Developed by a group of statisticians

Decision Tree Induction Algorithms

- ID3, C4.5, and CART adopt a greedy (i.e., nonbacktracking) approach in which decision trees are constructed in a top-down recursive divide-and-conquer manner.
- Most algorithms for decision tree induction also follow such a top-down approach, which starts with a training set of tuples and their associated class labels.
- The training set is recursively partitioned into smaller subsets as the tree is being built.

Basic Algorithm for Decision Tree Induction

Basic 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)

Basic Algorithm for Decision Tree Induction

- 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

Basic Algorithm for Decision Tree Induction

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.

Basic Algorithm - Method

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* $\leftarrow$ *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;

Basic Algorithm for Decision Tree Induction

- Step 1
  - The tree starts as a single node, *N,* representing the training tuples in *D*
- Steps 2 and 3
  - If the tuples in *D* are all of the same class, then node *N* becomes a leaf and is labeled with that class.
- Steps 4 and 5
  - steps 4 and 5 are terminating conditions.
  - Allof the terminating conditions are explained at the end of the algorithm.

Basic Algorithm for Decision Tree Induction

- Step 6
  - 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
  - The splitting criterion is determined so that, ideally, the resulting partitions at each branch are as "pure" as possible.

Basic Algorithm for Decision Tree Induction

- Step 7
  - The node *N* is labeled with the splitting criterion, which serves as a test at the node
- Steps 10 to 11
  - A branch is grown from node *N* for each of the outcomes of the splitting criterion.
  - The tuples in *D* are partitioned accordingly

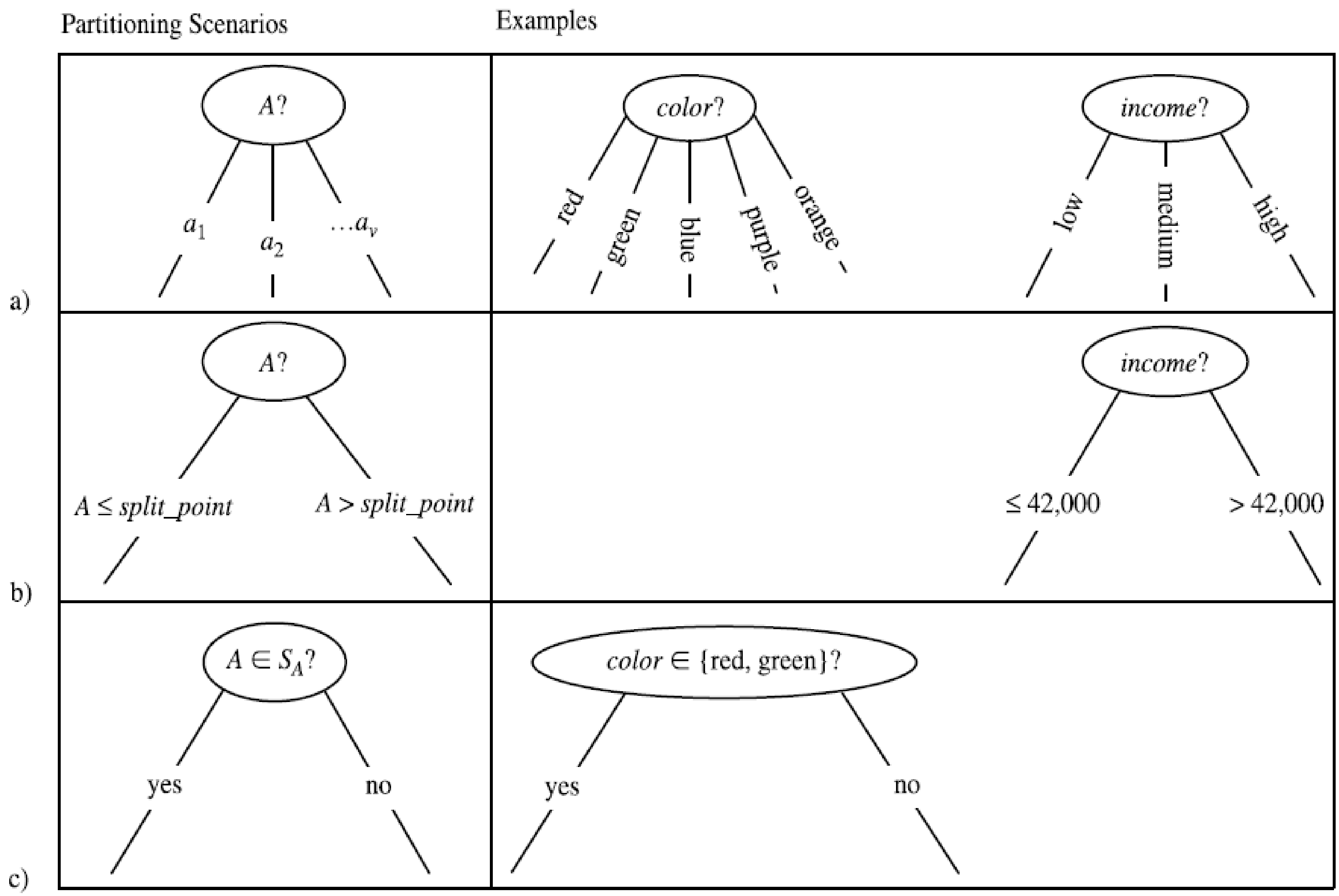
Basic Algorithm for Decision Tree Induction

- Different ways of handling continuous attributes
  - Discretization to form an ordinal categorical attribute
  - Binary Decision: $(A < v)$ or $(A \geq v)$
    - consider all possible splits and finds the best cut
    - can be more compute intensive

Basic Algorithm for Decision Tree Induction

- Let *A* be the splitting attribute. *A* has *v* distinct values, $\{a_1, a_2, \ldots, a_v\}$, based on the training data.
- There are three possible scenario for partitioning tuples based on the splitting criterion:
  a. *A* is *discrete-valued*
  b. *A* is *continuous-valued*
  c. *A* is *discrete-valued* and a *binary tree* must be produced

Basic Algorithm for Decision Tree Induction



Basic Algorithm for Decision Tree Induction

- In scenario *a* (*A* is *discrete-valued*)
  - the outcomes of the test at node N correspond directly to the known values 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).

Basic Algorithm for Decision Tree Induction

- Step 14
  - The algorithm uses the same process recursively to form a decision tree for the tuples at each resulting partition, *Dj,* of *D .*
- Step 15
  - The resulting decision tree is returned.

Basic Algorithm for Decision Tree Induction

- 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)
  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.*
  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).*

Decision Tree Induction

- Differences in decision tree algorithms include:
  - how the attributes are selected in creating the tree and
  - the mechanisms used for pruning

Attribute Selection Measures

Attribute Selection Measures

- Which is the best attribute?
  - Want to get the smallest tree
  - choose the attribute that produces the “purest” nodes
- Attribute selection measure
  - 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 (i.e., all of the tuples that fall into a given partition would belong to the same class).

Attribute Selection Measures

- Attribute selection measures are also known as splitting rules because they determine how the tuples at a given node are to be split.

Attribute Selection Measures

- 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 <u>must also be</u> determined as part of the splitting criterion.

Attribute Selection Measures

- This section describes three popular attribute selection measures:
  - Information gain
  - Gain ratio
  - Gini index

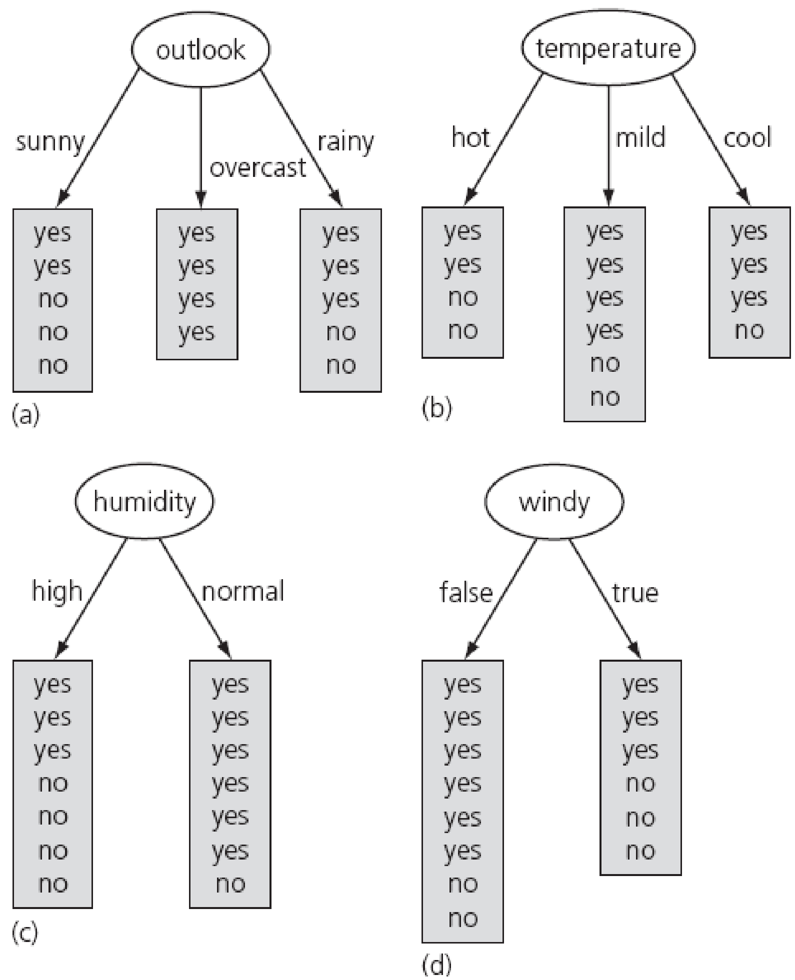
Attribute Selection Measures

- 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 $i = 1, \dots, m$)
  - *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.*

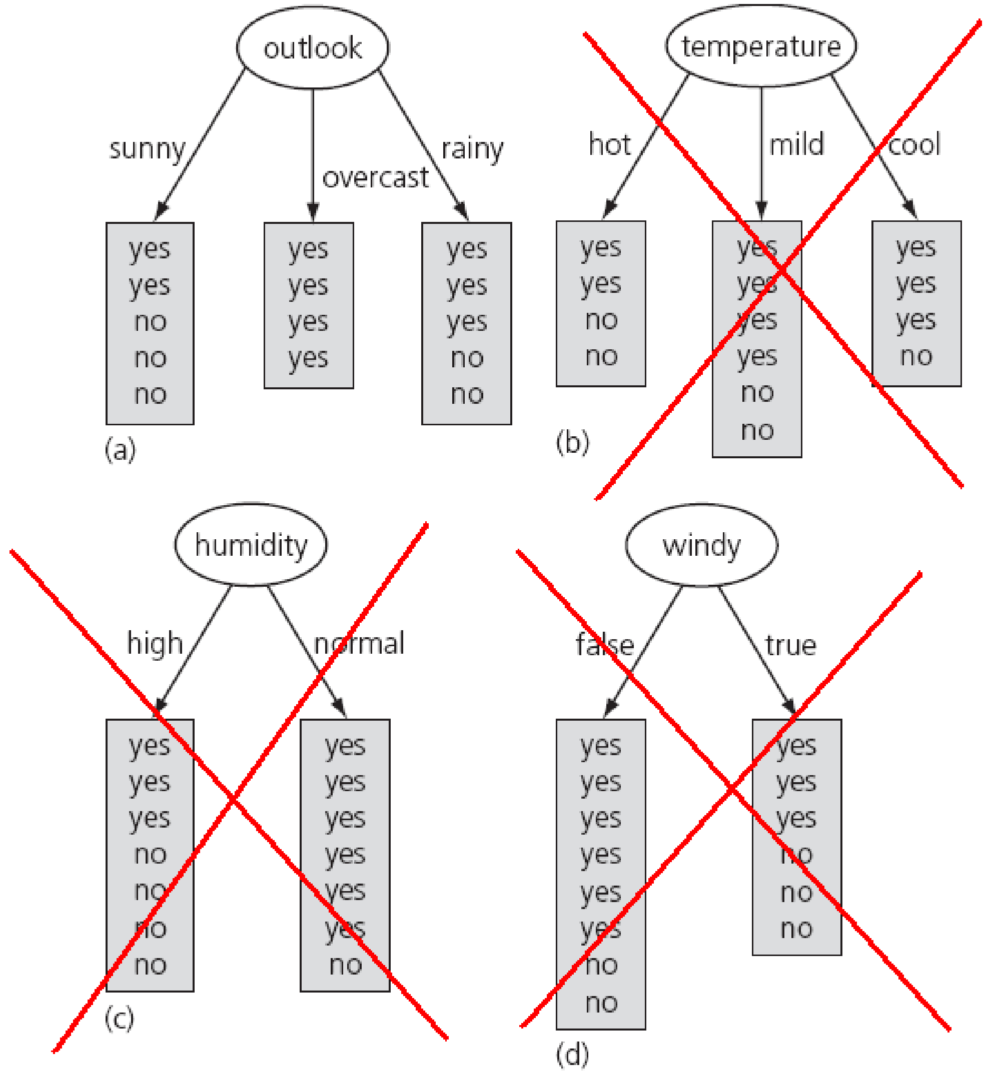
Information Gain

Information Gain

- Which attribute to select?



Information Gain



Information Gain

- Need a measure of node impurity:

C0: 5 C1: 5	C0: 9 C1: 1
Non-homogeneous, **High degree of impurity**	**Homogeneous,** **Low degree of impurity**

Attribute Selection Measures

- ID3 uses information gain as its attribute selection measure.
- 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 "impurity" in these partitions.

Attribute Selection Measures

- 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:

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

- *Info(D)* is just the average amount of information needed to identify the class label of a tuple in *D.* The smaller information required, the greater the purity.
- The information we have is based solely on the proportions of tuples of each class.
- A log function to the base 2 is used, because the information is encoded in bits (It is measured in bits).

Attribute Selection Measures

- *High Entropy* means X is from a uniform (boring) distribution
- *Low Entropy* means X is from a varied (peaks and valleys) distribution

Attribute Selection Measures

- Information needed (after using *A* to split *D)* to classify *D*:

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

- Attribute *A* can be used to split *D* into *v partitions or subsets,* {*D1, D2, … , Dv*}*,* where *Dj* contains those tuples in *D* that have outcome *aj* of *A.*
- This measure tell us how much more information would we still need (after the partitioning) in order to arrive at an exact classification
- The smaller the expected information (still) required, the greater the purity of the partitions.

Attribute Selection Measures

- Information gained by branching on attribute A

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

- Information gain increases with the average purity of the subsets
- Information gain: information needed before splitting – information needed after splitting

Example: *AllElectronics*

- This table presents a training set, *D.*

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

Example: *AllElectronics*

- The class label attribute, *buys_computer,* has two distinct values (namely, {yes, no}); therefore, there are two distinct classes (that is, *m = 2).*
- Let class *C1* correspond to *yes* and class *C2* correspond to *no*.
- The expected information needed to classify a tuple in *D*:

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

Example: *AllElectronics*

- Next, we need to compute the expected information requirement for each attribute.
- Let's start with the attribute *age.* We need to look at the distribution of *yes* and *no* tuples for each category of *age.*
  - For the age category youth, there are two yes tuples and three *no* tuples*.*
  - For the category *middle_aged,* there are four *yes* tuples and zero *no* tuples*.*
  - For the category *senior,* there are three *yes* tuples and two *no* tuples.

Example: *AllElectronics*

- The expected information needed to classify a tuple in *D* if the tuples are partitioned according to *age* is

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

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

Example: *AllElectronics*

- The gain in information from such a partitioning would be

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

- Similarly

$$Gain(income) = 0.029$$

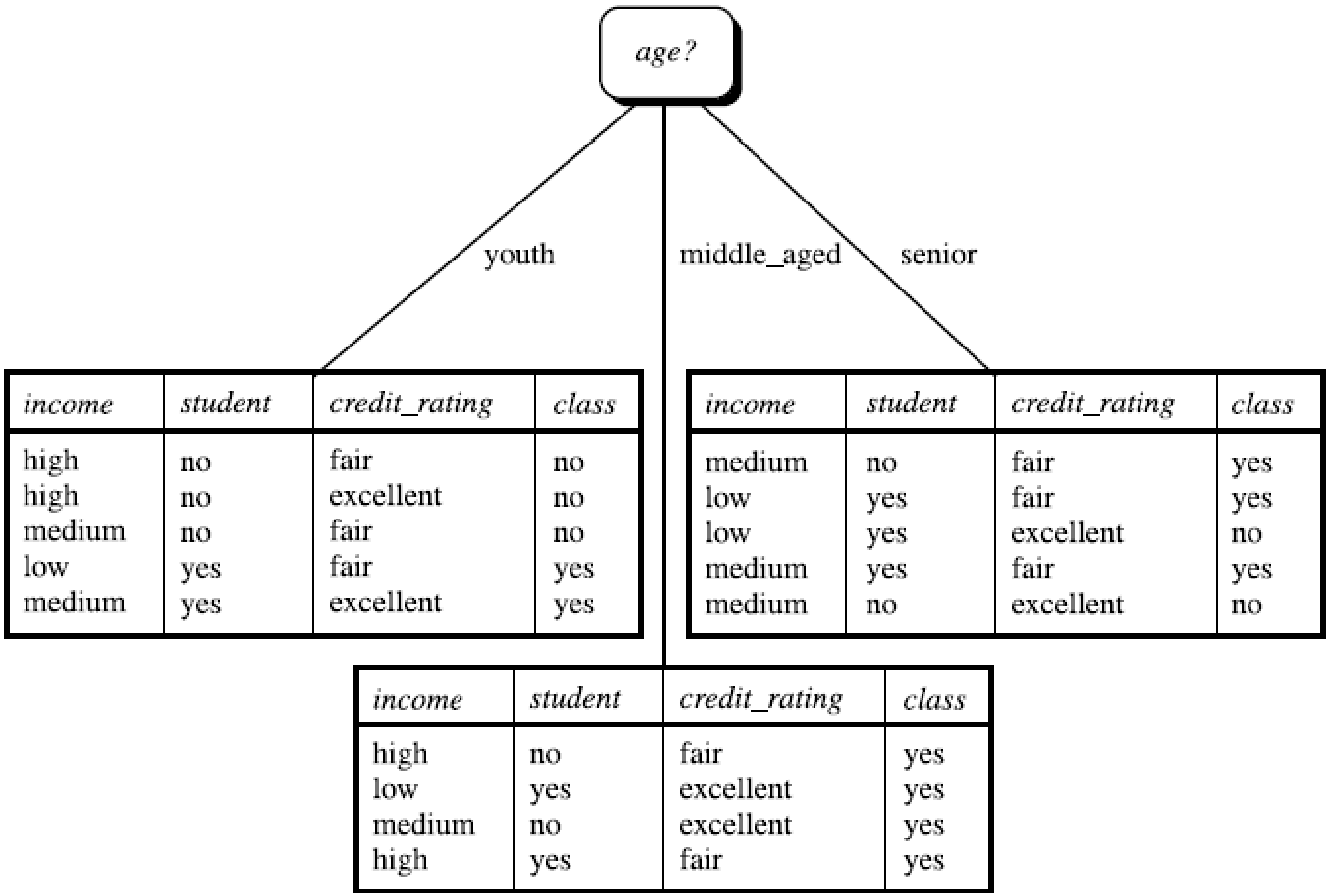
$$Gain(student) = 0.151$$

$$Gain(credit\_rating) = 0.048$$

- Because *age* has the highest information gain among the attributes, it is selected as the splitting attribute.

Example: *AllElectronics*

- Branches are grown for each outcome of *age.* The tuples are shown partitioned accordingly.



youth:

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

middle_aged:

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

senior:

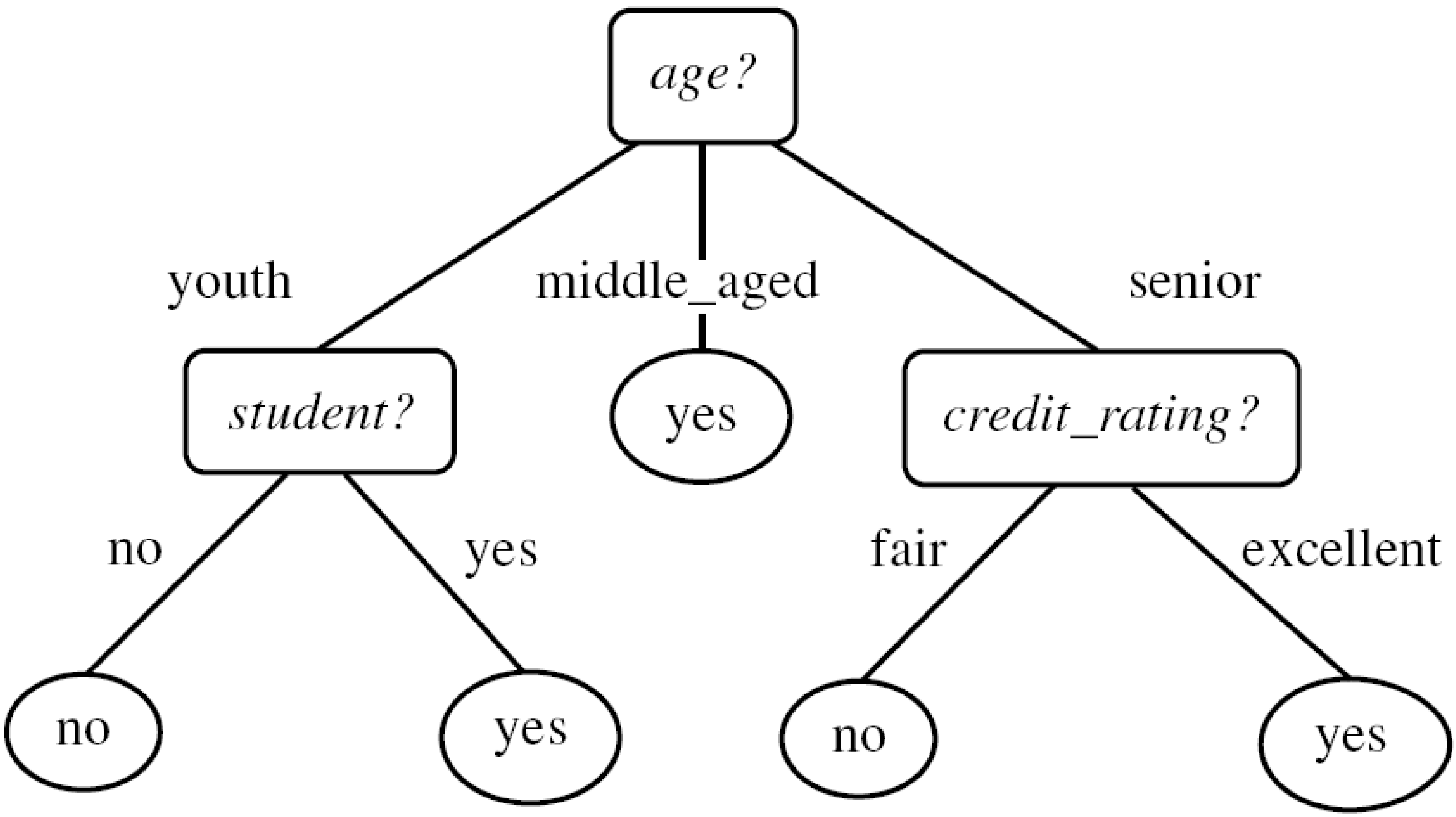
income	*student*	*credit_rating*	*class*
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

Example: *AllElectronics*

- Notice that the tuples falling into the partition for *age = middle_aged* all belong to the same class.
- Because they all belong to class *"yes," a* leaf should therefore be created at the end of this branch and labeled with *"yes."*

Example: *AllElectronics*

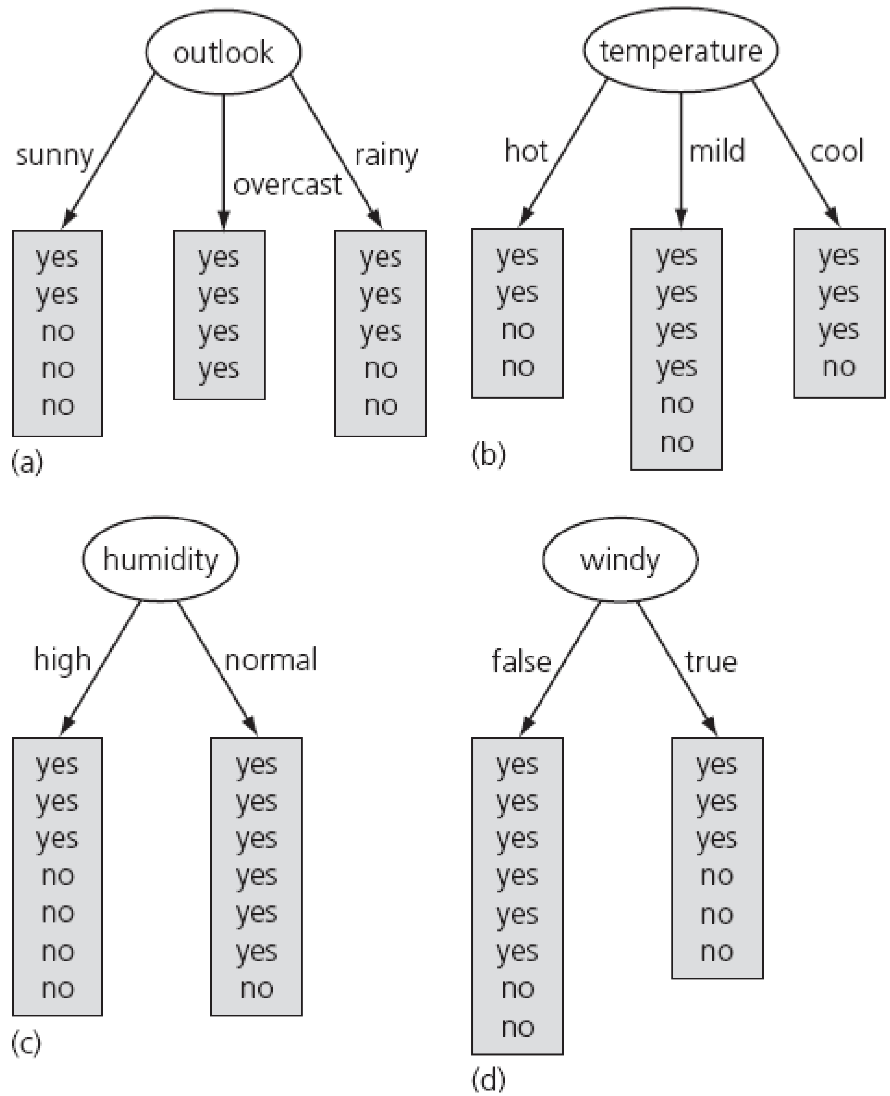
- The final decision tree returned by the algorithm



Example: Weather Problem

Outlook	Temperature	Humidity	Windy	Play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

Example: Weather Problem



Example: Weather Problem

- attribute *Outlook:*

$$Info_{outlook}(D) = \frac{5}{14} I(2,3) + \frac{4}{14} I(4,0) + \frac{5}{14} I(3,2) = 0.693$$

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

Example: Weather Problem

- Information gain: information before splitting – information after splitting:

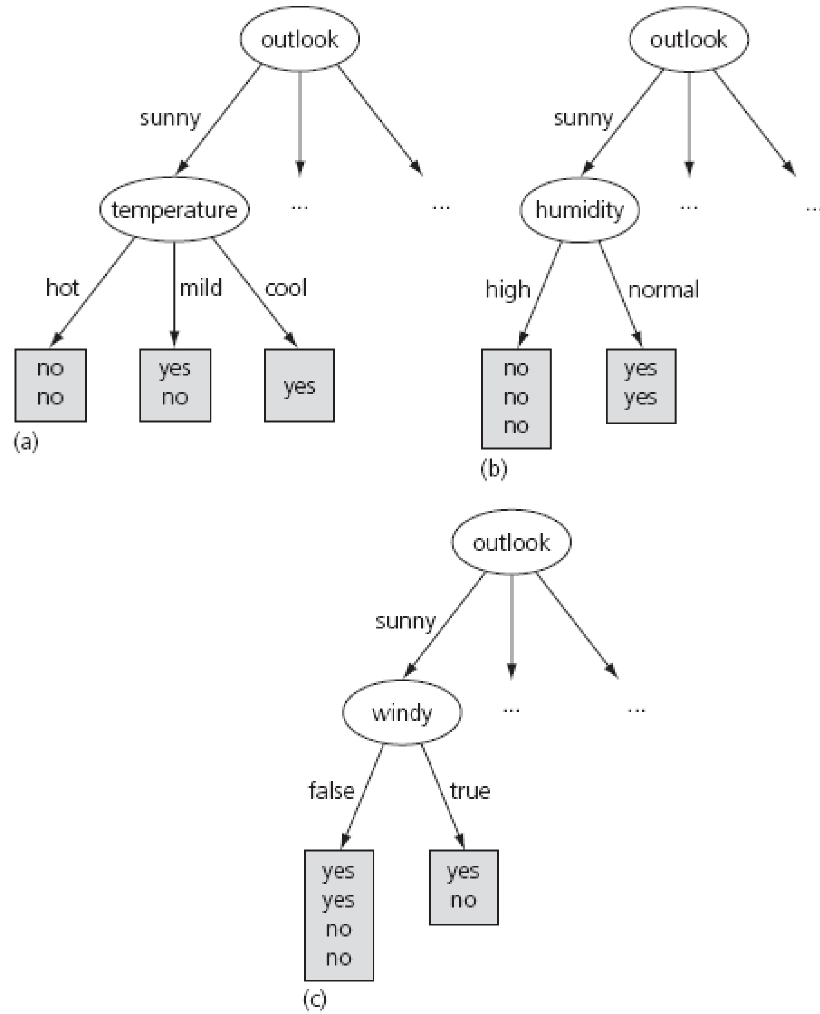
$$\text{gain}(\textit{Outlook}) = 0.940 - 0.693 = 0.247 \text{ bits}$$

- Information gain for attributes from weather data:

$$\begin{aligned} \text{gain}(\textit{Outlook}) &= 0.247 \text{ bits} \\ \text{gain}(\textit{Temperature}) &= 0.029 \text{ bits} \\ \text{gain}(\textit{Humidity}) &= 0.152 \text{ bits} \\ \text{gain}(\textit{Windy}) &= 0.048 \text{ bits} \end{aligned}$$

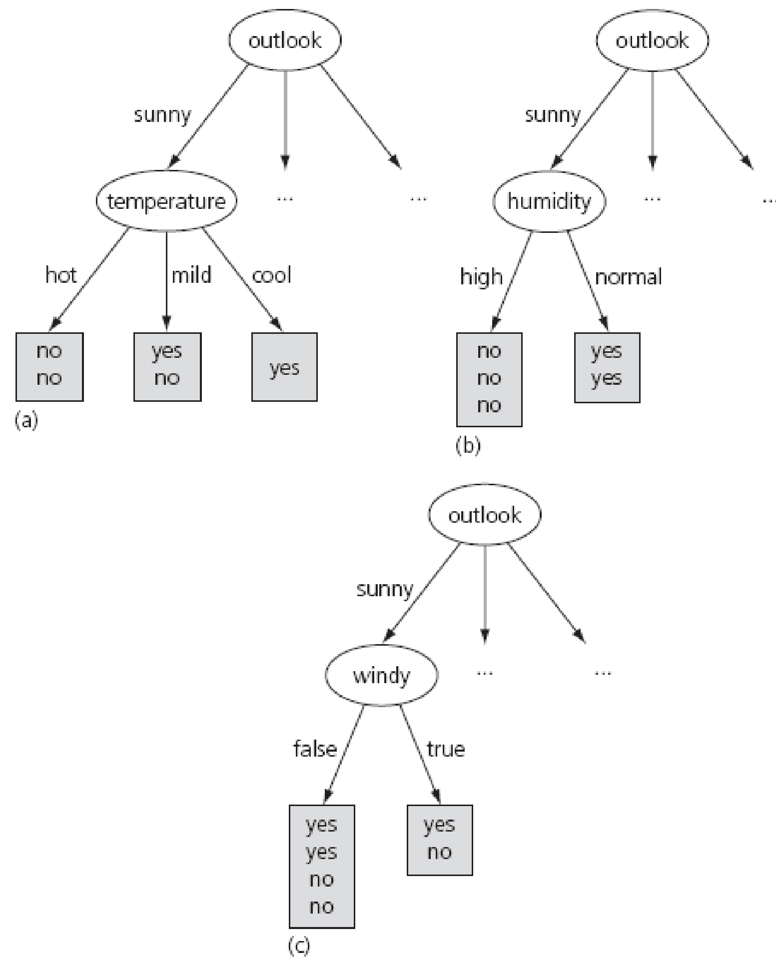
Example: Weather Problem

- Continuing to split



Example: Weather Problem

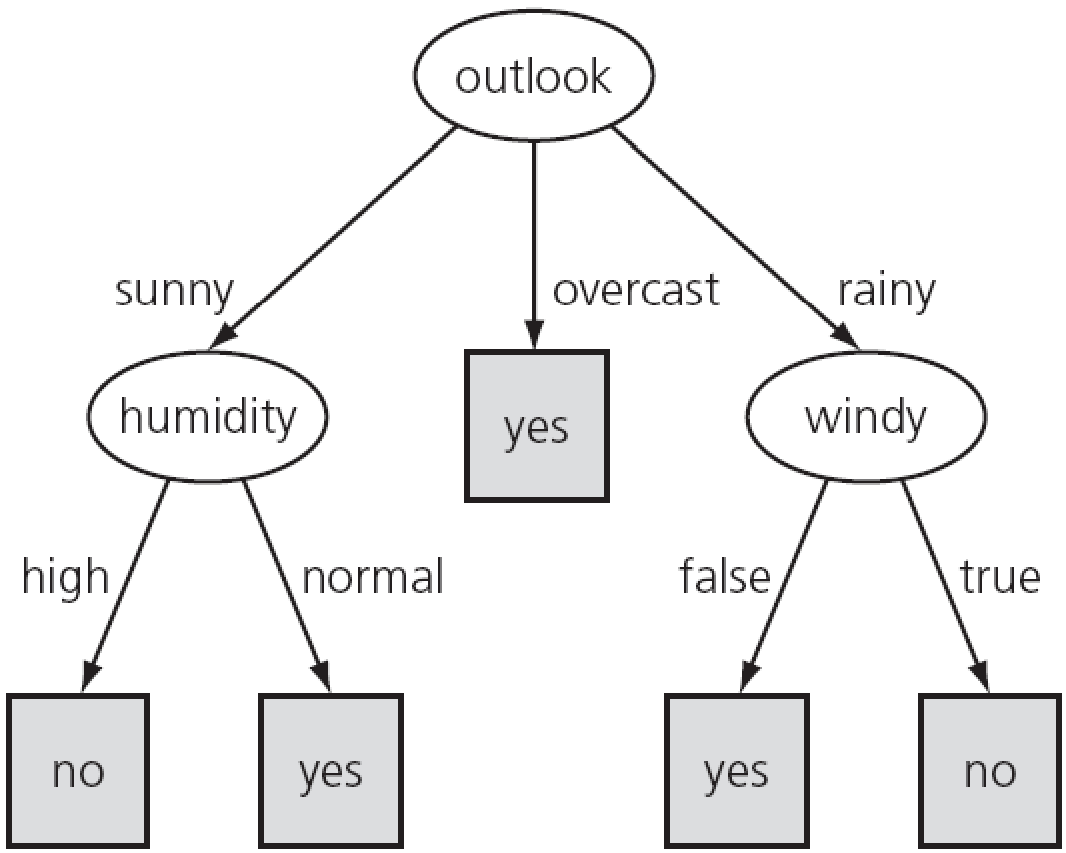
- Continuing to split



$$\text{gain}(\textit{temperature}) = 0.571 \text{ bits}$$
$$\text{gain}(\textit{humidity}) = 0.971 \text{ bits}$$
$$\text{gain}(\textit{windy}) = 0.020 \text{ bits}$$

Example: Weather Problem

- Final decision tree



Continuous-Value Attributes

- Let attribute A be a continuous-valued attribute
- Standard method: binary splits
- 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}
  - Therefore, given *v* values of *A,* then *v-1* possible splits are evaluated.
  - The point with the *minimum expected information requirement* for A is selected as the split-point for A

Continuous-Value Attributes

- Split:
  - 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
- Split on temperature attribute:

64	65	68	69	70	71	72	75	80	81	83	85
yes	no	yes	yes	yes	no	no yes	yes yes	no	yes	yes	no

  - E.g. temperature < 71.5: yes/4, no/2
    temperature ≥ 71.5: yes/5, no/3
  - Info = 6/14 info([4,2]) + 8/14 info([5,3])
    = 0.939 bits

Gain Ratio

Gain ratio

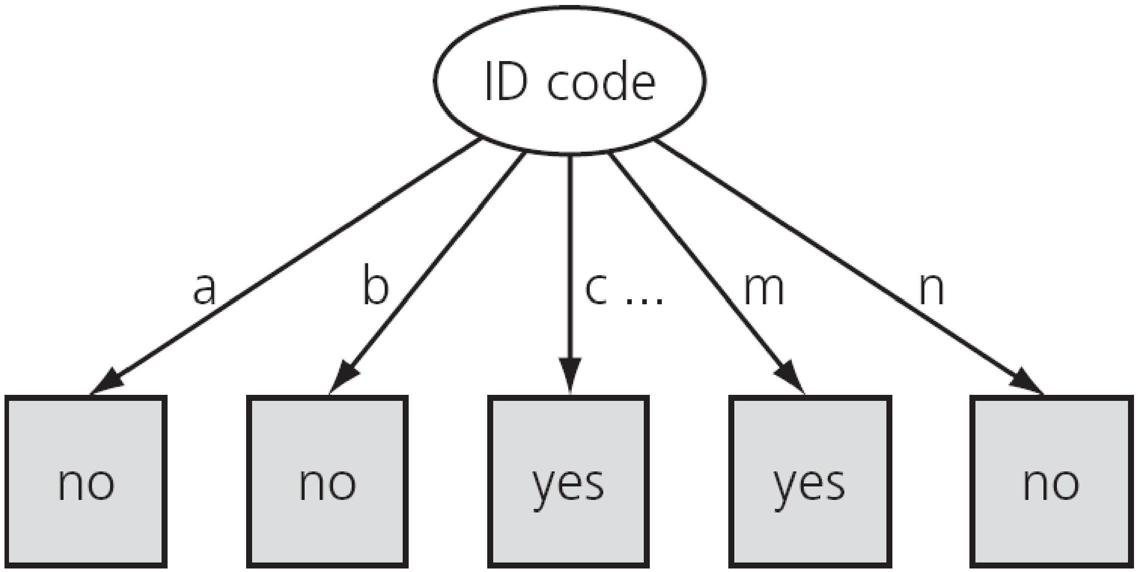
- Problem:
  - When there are attributes with a large number of values
  - Information gain measure is biased towards attributes with a large number of values
  - This may result in selection of an attribute that is non-optimal for prediction

Gain ratio

- Weather data with *ID code*

ID code	Outlook	Temperature	Humidity	Windy	Play
a	sunny	hot	high	false	no
b	sunny	hot	high	true	no
c	overcast	hot	high	false	yes
d	rainy	mild	high	false	yes
e	rainy	cool	normal	false	yes
f	rainy	cool	normal	true	no
g	overcast	cool	normal	true	yes
h	sunny	mild	high	false	no
i	sunny	cool	normal	false	yes
j	rainy	mild	normal	false	yes
k	sunny	mild	normal	true	yes
l	overcast	mild	high	true	yes
m	overcast	hot	normal	false	yes
n	rainy	mild	high	true	no

Gain ratio



- Information gain is maximal for ID code (namely 0.940 bits)

Gain ratio

- *Gain ratio*
  - a modification of the information gain
  - C4.5 uses gain ratio to overcome the problem
- Gain ratio applies a kind of normalization to information gain using a "split information"

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

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

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

Gain ratio

- Example
  - Computation of gain ratio for the attribute *income.*
  - A test on *income* splits the data into three partitions, namely *low, medium, and high,* containing four, six, and four tuples, respectively.
  - Computation of the gain ratio of *income:*

$$SplitInfo_A(D) = -\frac{4}{14}\times\log_2(\frac{4}{14}) - \frac{6}{14}\times\log_2(\frac{6}{14}) - \frac{4}{14}\times\log_2(\frac{4}{14}) = 0.926$$

  - Gain(income) = 0.029
  - GainRatio(income) = 0.029/0.926 = 0.031

Gain ratio

- Gain ratios for weather data

Outlook		Temperature		Humidity		Windy	
info:	0.693	info:	0.911	info:	0.788	info:	0.892
gain: 0.940– 0.693	0.247	gain: 0.940– 0.911	0.029	gain: 0.940– 0.788	0.152	gain: 0.940– 0.892	0.048
split info: info([5,4,5])	1.577	split info: info([4,6,4])	1.557	split info: info ([7,7])	1.000	split info: info([8,6])	0.985
gain ratio: 0.247/1.577	0.157	gain ratio: 0.029/1.557	0.019	gain ratio: 0.152/1	0.152	gain ratio: 0.048/0.985	0.049

Gini Index

Gini Index

- Gini index
  - The Gini index is used in CART.
  - The Gini index measures the impurity of D
  - The Gini index considers a binary split for each attribute.
- If a data set D contains examples from m classes, gini index, $gini(D)$ is defined as

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

  - where p_i is the relative frequency of class i in D

Gini Index

- When considering a binary split, we compute a weighted sum of the impurity of each resulting partition.
- 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

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

- The subset that gives the minimum gini index for that attribute is selected as its splitting subset.

Gini Index

- *To determine the best binary split on A, we examine all of the possible subsets that* can be formed using known values of *A.*
- *Given a tuple, this test is satisfied if the value* of *A for the tuple is among the values listed in* S_A.
- If *A* is a discrete-valued attribute having v distinct values, *then there* are 2^v *possible subsets.*
- For continuous-valued attributes, each possible split-point must be considered. The strategy is similar to that described for information gain.

Gini Index

- The reduction in impurity that would be incurred by a binary split on 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.

Gini Index

- Example:
  - *D* has 9 tuples in buys_computer = "yes" and 5 in "no"
  - The impurity of *D:*

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

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

$$\begin{aligned}
&Gini_{income \in \{low, medium\}}(D) \\
&= \frac{10}{14} Gini(D_1) + \frac{4}{14} Gini(D_2) \\
&= \frac{10}{14}\left(1 - \left(\frac{6}{10}\right)^2 - \left(\frac{4}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{1}{4}\right)^2 - \left(\frac{3}{4}\right)^2\right) \\
&= 0.450 \\
&= Gini_{income \in \{high\}}(D).
\end{aligned}$$

Gini Index

- The attribute *income* and splitting subset {*medium, high*} give the minimum gini index overall, with a reduction in impurity of 0.459 - 0.300 = 0.159.
- For continuous-valued attributes
  - May need other tools, e.g., clustering, to get the possible split values
  - Can be modified for categorical attributes

Other Attribute Selection Measures

- Other Attribute Selection Measures
  - CHAID
  - C-SEP
  - G-statistics
- Which attribute selection measure is the best?
  - Most give good results, none is significantly superior than others

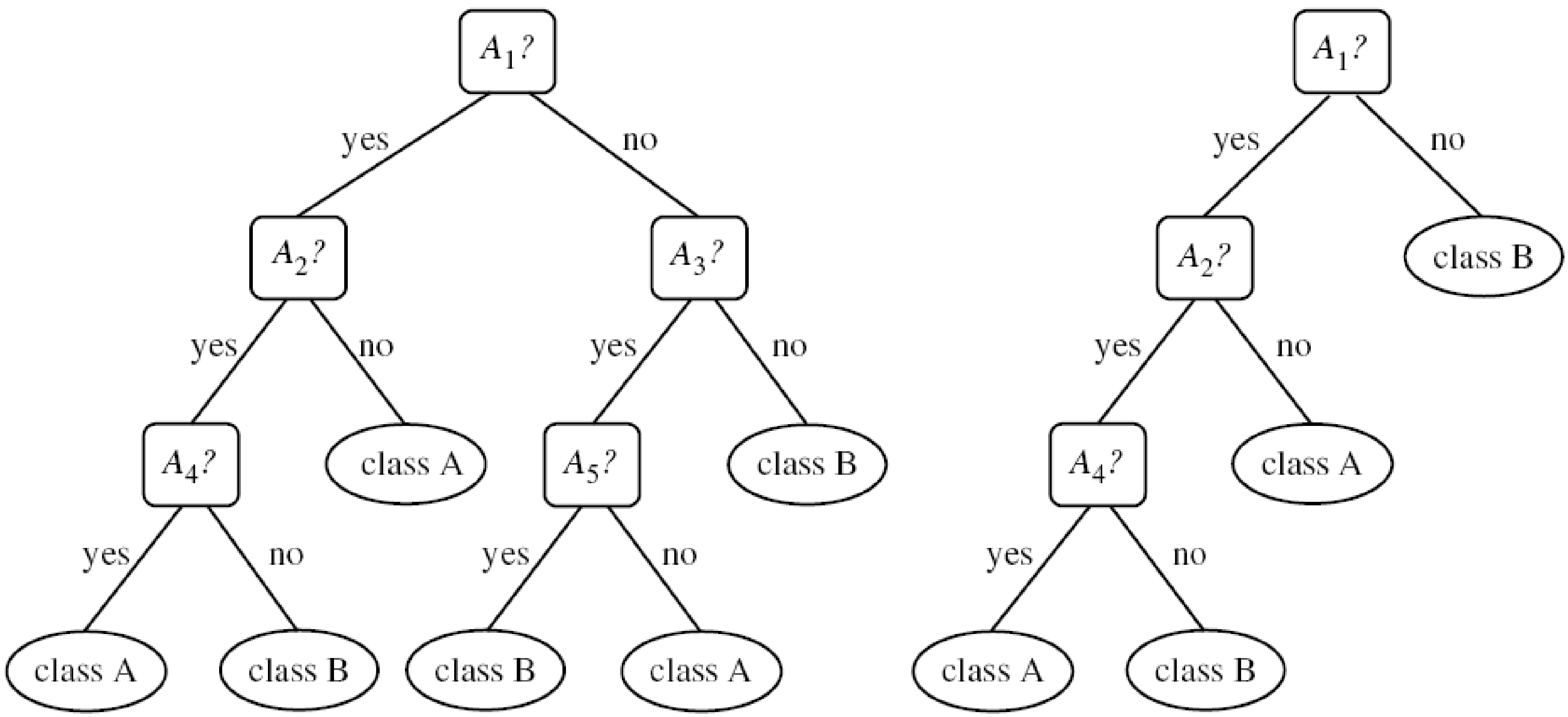
Tree Pruning

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
- Tree Pruning
  - To prevent overfitting to noise in the data
  - 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.

Tree Pruning

- An unpruned decision tree and a pruned version of it.



Tree Pruning

- Two approaches to avoid overfitting
  - Postpruning
    - take a fully-grown decision tree and remove unreliable branches
    - Postpruning preferred in practice
  - Prepruning
    - stop growing a branch when information becomes unreliable

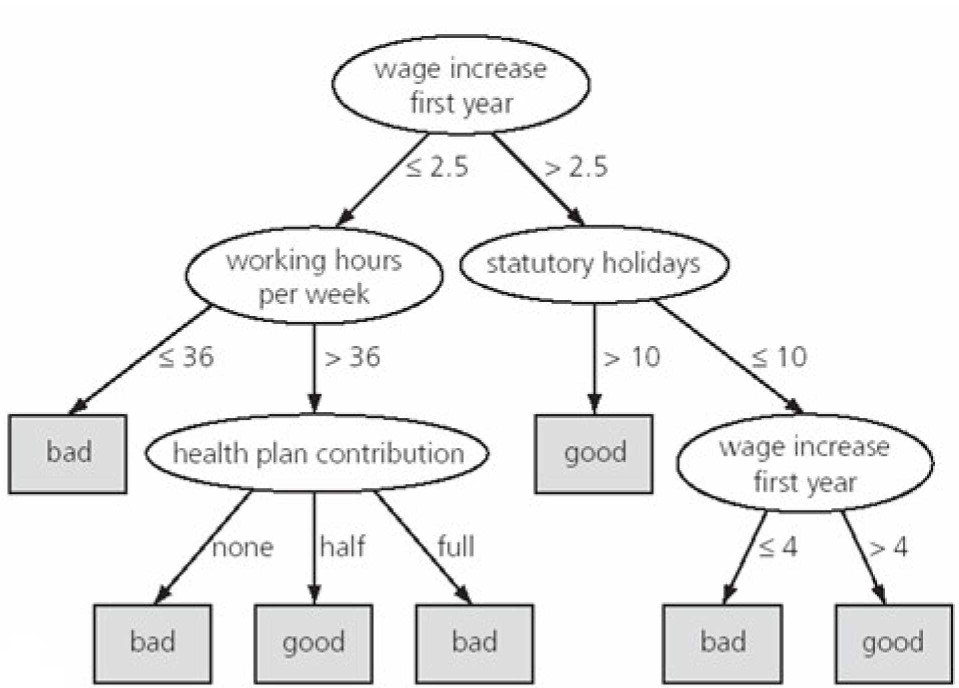
Prepruning

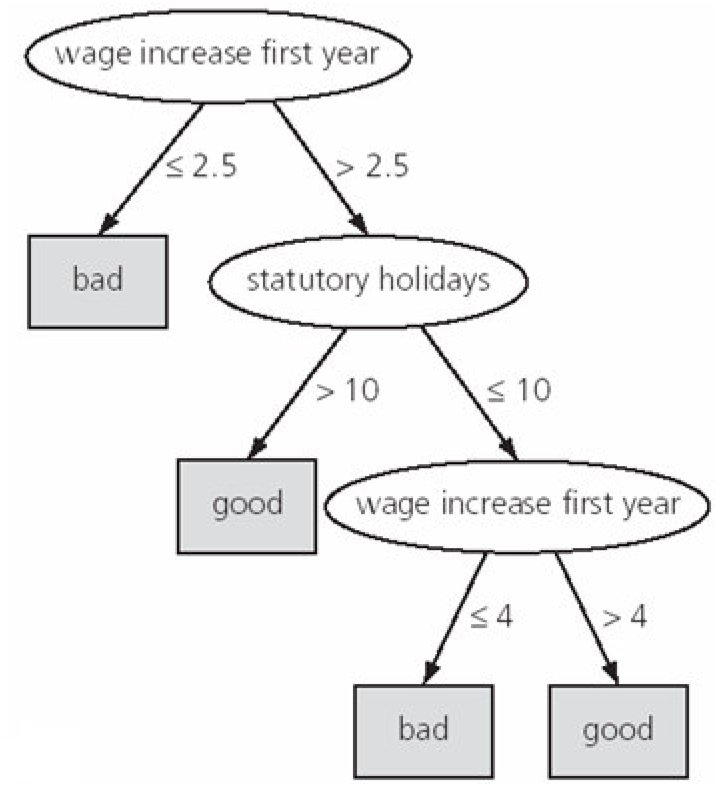
- Based on statistical significance test
  - Stop growing the tree when there is no *statistically significant* association between any attribute and the class at a particular node
- Most popular test: *chi-squared test*
- ID3 used chi-squared test in addition to information gain
  - Only statistically significant attributes were allowed to be selected by information gain procedure

Postpruning

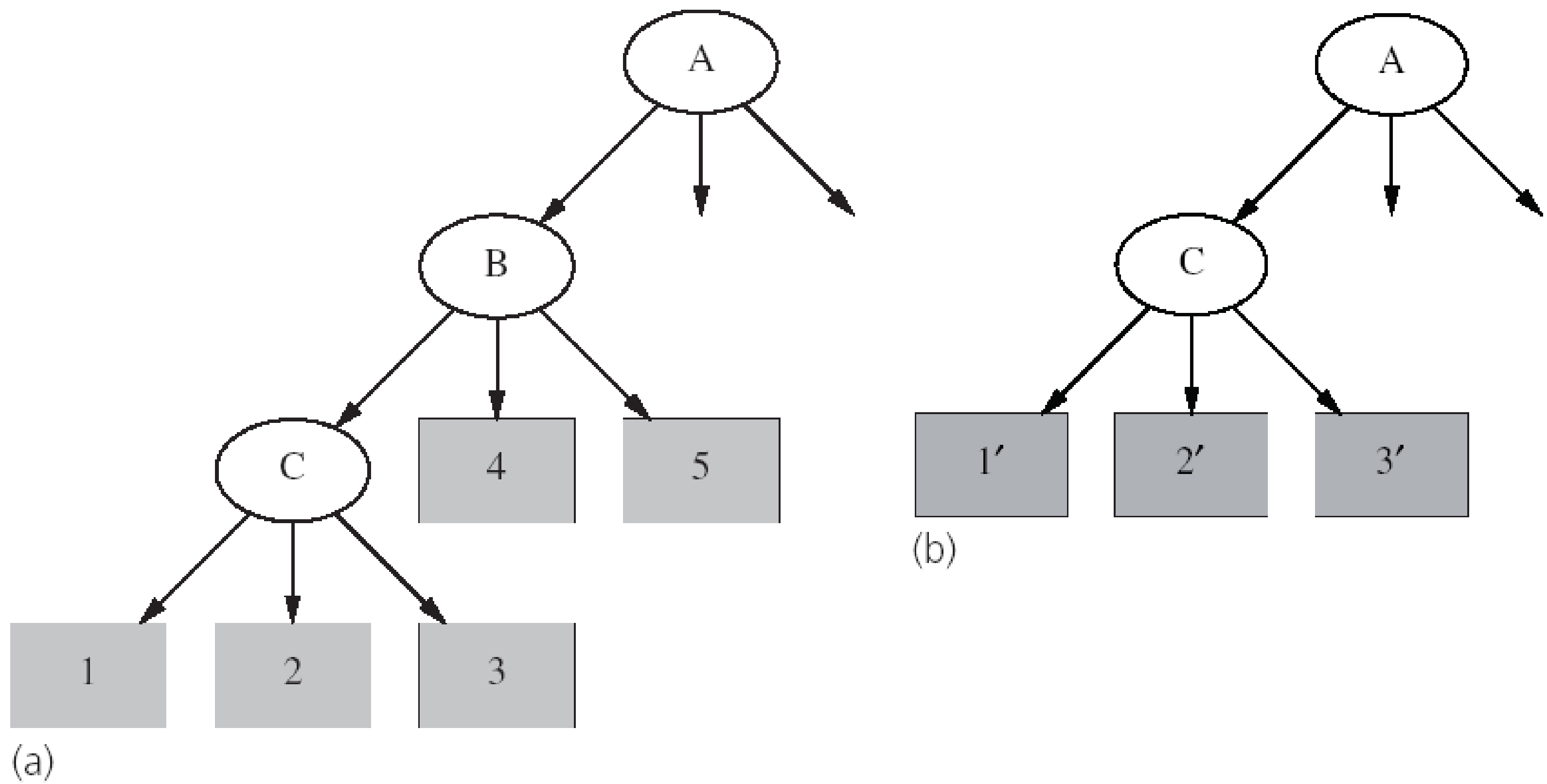
- Postpruning: First, build full tree & Then, prune it
- Two pruning operations:
  - *Subtree replacement:*
    - *Bottom-up*
    - To select some subtrees and replace them with single leaves
  - *Subtree raising*
    - Delete node, Redistribute instances
    - Slower than subtree replacement
- Possible strategies: error estimation, significance testing, …

Subtree replacement





Subtree raising



Scalable Decision Tree Induction Methods

Scalable Decision Tree Induction Methods

- The efficiency of existing decision tree algorithms, such as ID3, C4.5, and CART, has been well established for relatively small data sets.
- Efficiency becomes an issue of concern when these algorithms are applied to the mining of very large real-world databases.
- The pioneering decision tree algorithms that we have discussed so far have the restriction that the training tuples should reside *in memory.*

Scalable Decision Tree Induction Methods

- Algorithms for the induction of decision trees from very large training sets:
  - SLIQ (EDBT'96 — Mehta et al.)
    - Builds an index for each attribute and only class list and the current attribute list reside in memory
  - SPRINT (VLDB'96 — J. Shafer et al.)
    - Constructs an attribute list data structure
  - PUBLIC (VLDB'98 — Rastogi & Shim)
    - Integrates tree splitting and tree pruning: stop growing the tree earlier
  - RainForest (VLDB'98 — Gehrke, Ramakrishnan & Ganti)
    - Builds an AVC-list (attribute, value, class label)
  - BOAT (PODS'99 — Gehrke, Ganti, Ramakrishnan & Loh)
    - Uses bootstrapping to create several small samples

References

References

- J. Han, M. Kamber, **Data Mining: Concepts and Techniques**, Elsevier Inc. (2006). (Chapter 6)

- I. H. Witten and E. Frank, **Data Mining: Practical Machine Learning Tools and Techniques**, 2nd Edition, Elsevier Inc., 2005. (Chapter 6)

The end