UEM
श्रद्धावान लभते ज्ञानम्
UNIVERSITY OF ENGINEERING & MANAGEMENT
Good Education, Good Jobs

UNIVERSITY OF ENGINEERING & MANAGEMENT, KOLKATA

Course Name : AI & ML

Concept of Decision Tree

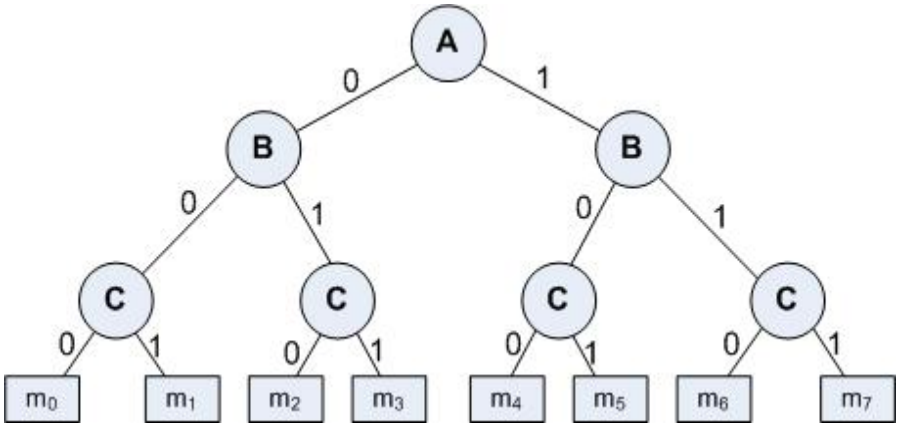
A Decision Tree is an important data structure known to solve many computational problems.

It may be binary (if based on single yes or no)

It may be m-array

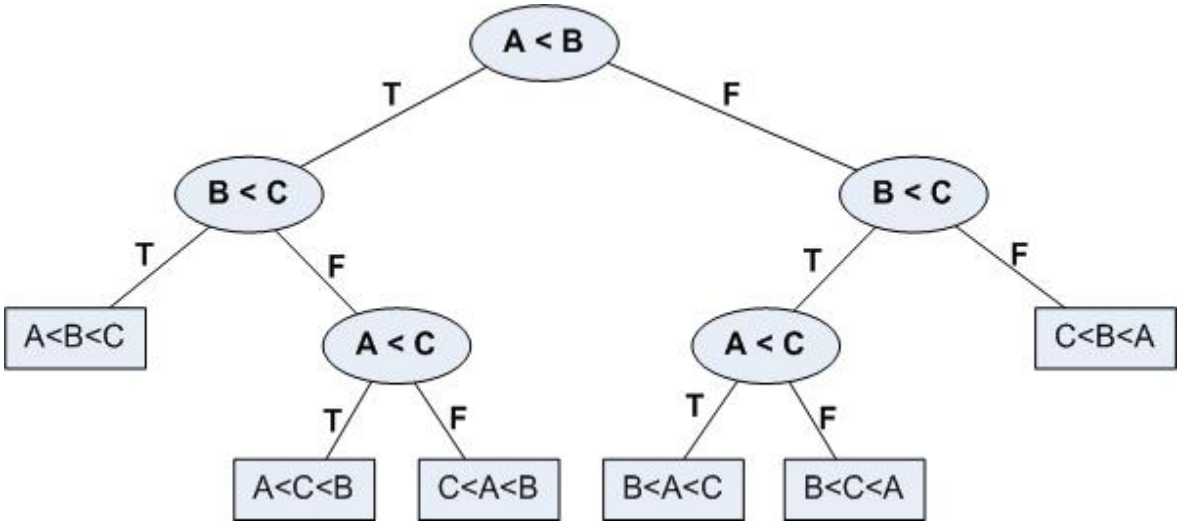
Binary Decision Tree

A	*B*	*C*	*f*
0	0	0	m_0
0	0	1	m_1
0	1	0	m_2
0	1	1	m_3
1	0	0	m_4
1	0	1	m_5
1	1	0	m_6
1	1	1	m_7



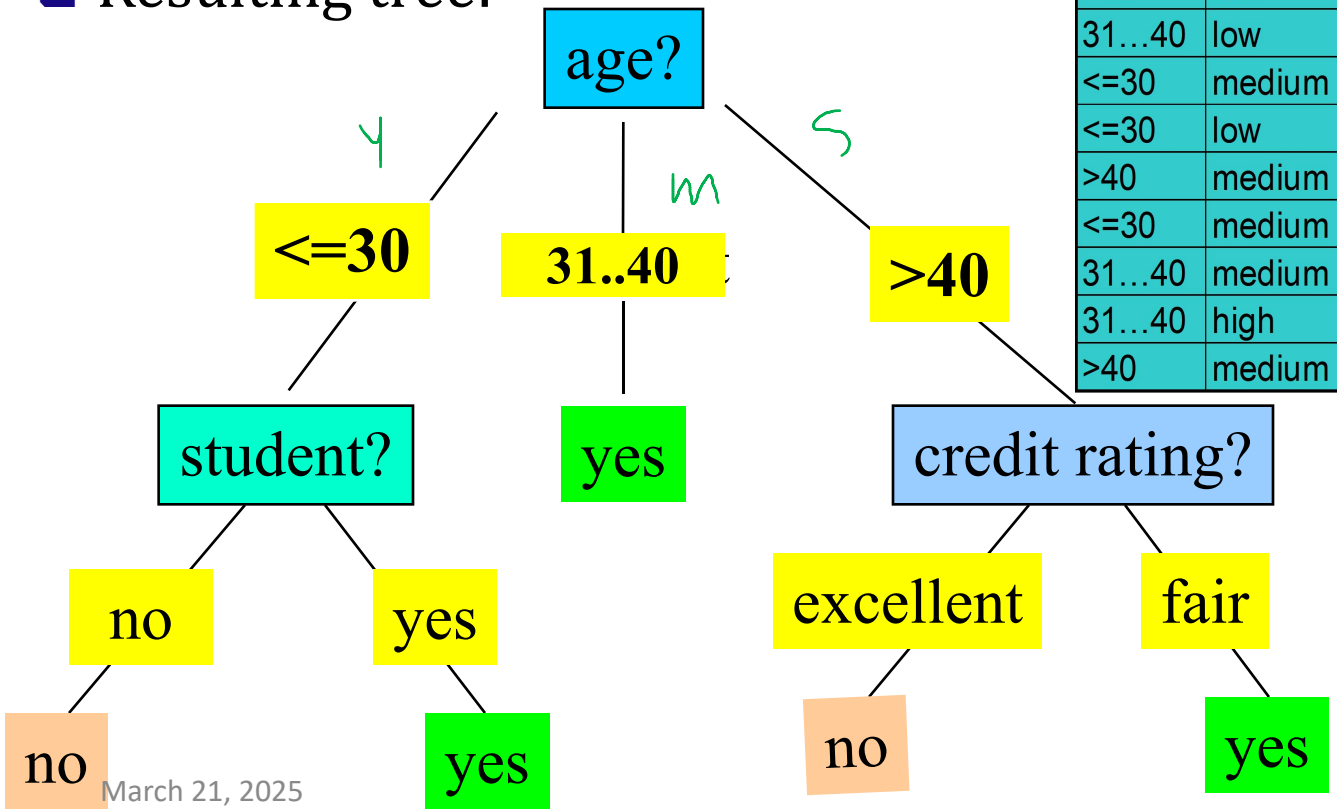
- In the last slide we have considered a decision tree where values of any attribute if binary only. Decision tree is also possible where attributes are of continuous data type

Decision Tree with numeric data



- 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



Some Characteristics

- Decision tree may be *n*-ary, $n \geq 2$.
- There is a special node called root node.
- All nodes drawn with circle (ellipse) are called internal nodes.
- All nodes drawn with rectangle boxes are called terminal nodes or leaf nodes.
- Edges of a node represent the outcome for a value of the node.
- In a path, a node with same label is never repeated.
- Decision tree is not unique, as different ordering of internal nodes can give different decision tree.

Decision tree helps us to classify data.

- Internal nodes are some attribute
- Edges are the values of attributes
- External nodes are the outcome of classification

- Such a classification is, in fact, made by posing questions starting from the root node to each terminal node.

How are decision tree 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.

Why are decision tree classifiers so popular?

- Construction of decision tree does not require any domain knowledge.
- It can deal with high dimensional data.
- Decision tree classifiers have good accuracy.

Building Decision Tree

In principle, there are exponentially many decision tree that can be constructed from a given database (also called training data).

- Two approaches are known
  - **Greedy strategy**
    - A top-down recursive divide-and-conquer
  - **Modification of greedy strategy**
    - ID3
    - C4.5
    - CART, etc.

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: 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;

Discussions on Algorithm:

Called with three parameters: D, attribute_list, attribute_selection_method

D: data partition

attribute_list: list of attributes describing the tuple

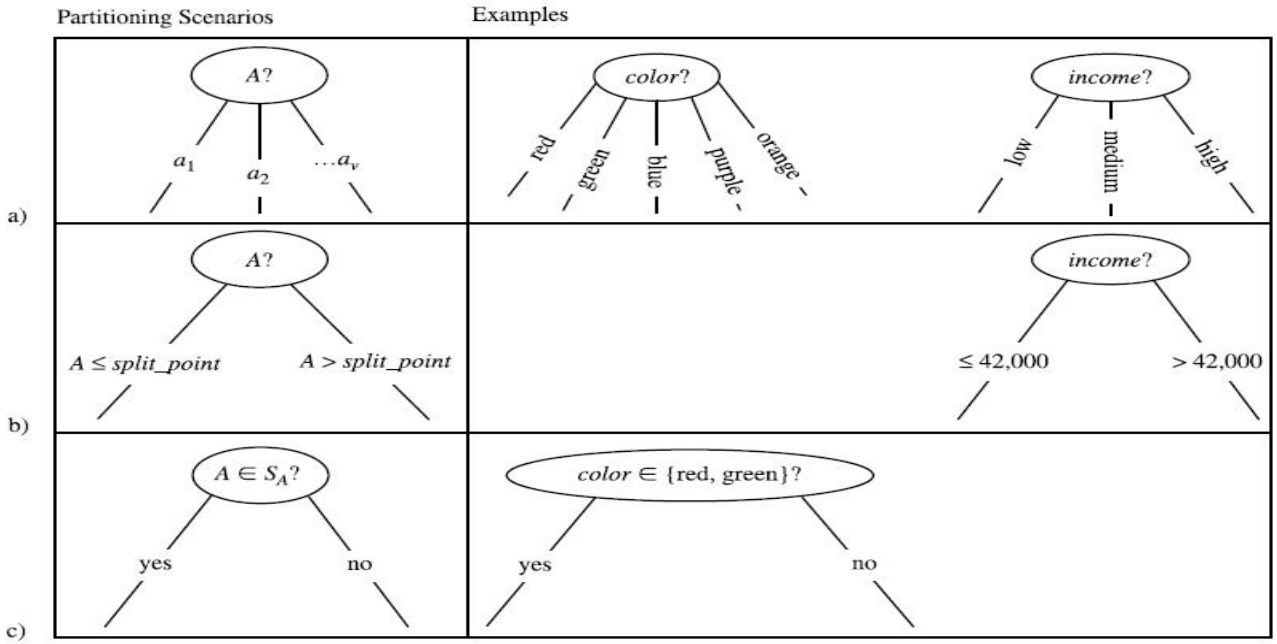
attribute_selection_method: heuristic method for selecting best attribute.

- The tree starts as a single node, *N, representing the training tuples in D*
- 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.
- Otherwise, the algorithm calls *Attribute selection method to determine the splitting* criterion.

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 of the tuples in it belong to the same class. In other words, if we were to 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.

- Let *A be the splitting attribute. A has v* distinct values, *a1, a2, : : : , based on the training data.*
- *A is discrete-valued: In this case, the outcomes of the test at node N correspond* directly to the known values of *A.*
- *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.*
- *A is discrete-valued and a binary tree*

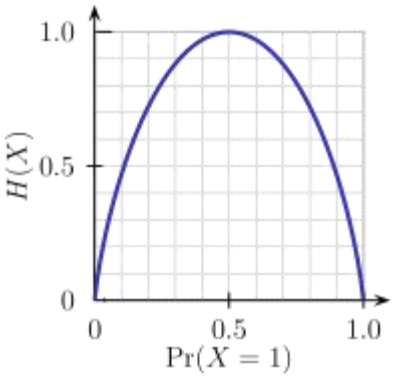


Three possibilities for partitioning tuples based on the splitting criterion, shown with examples. Let A be the splitting attribute. (a) If A is discrete-valued, then one branch is grown for each known value of A. (b) If A is continuous-valued, then two branches are grown, corresponding to $A \leq split\_point$ and $A > split\_point$. (c) If A is discrete-valued and a binary tree must be produced, then the test is of the form $A \in S_A$, where S_A is the splitting subset for A.

- 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—*information gain, gain ratio, and 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, *Ci (for i = 1, : : : , m). Let Ci,D be the set of tuples of class Ci in D. Let |D|*and *|Ci,D| denote the number of tuples in D and |Ci,D| respectively.*

- Entropy (Information Theory)
  - A measure of uncertainty associated with a random variable
  - Calculation: For a discrete random variable Y taking m distinct values $\{y_1, \dots, y_m\}$,
    - $H(Y) = -\sum_{i=1}^{m} p_i \log(p_i)$, where $p_i = P(Y = y_i)$
  - Interpretation:
    - Higher entropy => higher uncertainty
    - Lower entropy => lower uncertainty
- Conditional Entropy
  - $H(Y|X) = \sum_x p(x) H(Y|X = x)$



m = 2

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

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

The expected information needed to classify a tuple in *D is given by Info(D)*

- *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.*

The expected information needed to classify a tuple in *D is given by Info(D)*

- *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.*

Class-labeled training tuples from the *AllElectronics* customer database.

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

- In other words, *Gain(A) tells us how much would be gained by branching on A. It is the* expected reduction in the information requirement caused by knowing the value of *A.*
- The attribute *A with the highest information gain, (Gain(A)), is chosen as the splitting* attribute at node *N. This is equivalent to saying that we want to partition on the attribute A that would do the "best classification," so that the amount of information still required* to finish classifying the tuples is minimal (i.e., minimum *InfoA(D)).*

Discussions

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

$= 0.940$ bits

$Info_{age}(D) = \frac{5}{14} \times \left(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2^{3/5}\right) +$

$\frac{4}{14} \times \left(-\frac{4}{4}\log_2^{4/4} - \frac{0}{4}\log_2^{0/4}\right) +$

$\frac{5}{14} + \left(-\frac{3}{5}\log_2^{3/5} - \frac{2}{5}\log_2^{2/5}\right)$

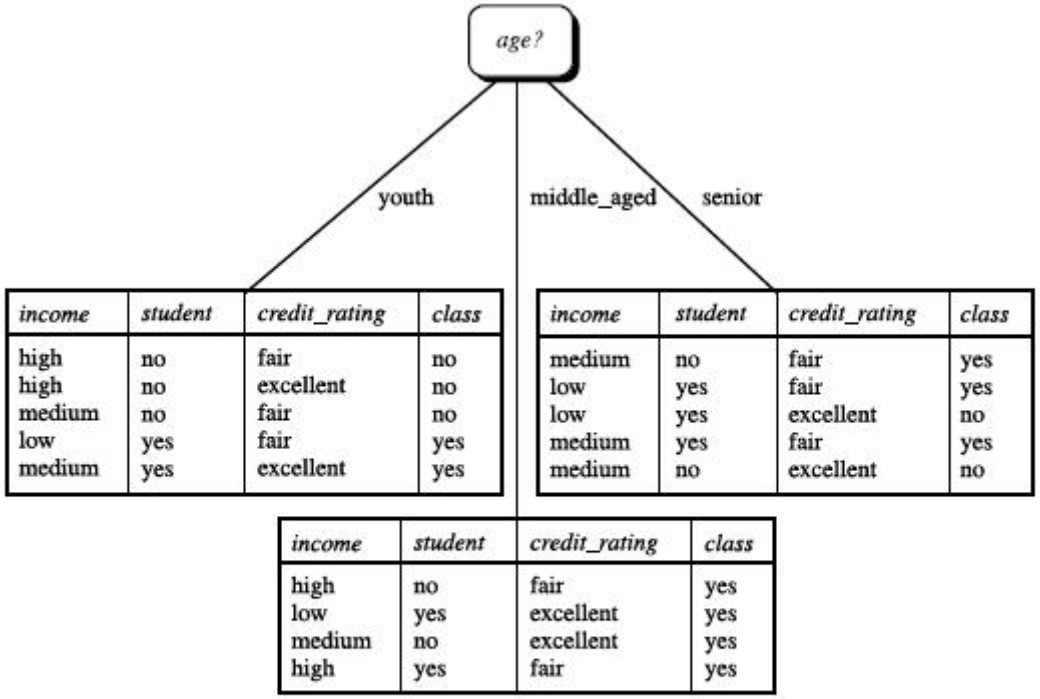
$= 0.694$ bits

Gain age

$= 0.940 - 0.694$

$= 0.246$

Tree(its not final) after selecting age as splitting attribute



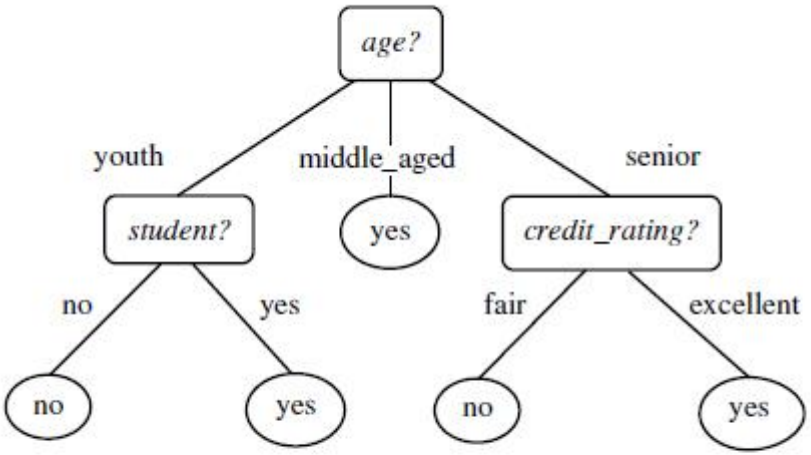
income	*student*	*credit_rating*	*class*
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

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

The attribute *age* has the highest information gain and therefore becomes the splitting attribute at the root node of the decision tree. Branches are grown for each outcome of *age*. The tuples are shown partitioned accordingly.

Final Tree



A decision tree for the concept *buys_computer*, indicating whether a customer at *AllElectronics* is likely to purchase a computer. Each internal (nonleaf) node represents a test on an attribute. Each leaf node represents a class (either *buys_computer* = *yes* or *buys_computer* = *no*).

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:
  - 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

Gain Ratio for Attribute Selection (C4.5)

- Information gain measure is biased towards attributes with a large number of values
- C4.5 (a successor of ID3) uses gain ratio to overcome the problem (normalization to information gain)

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

  - 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)

- 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*)

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.

- 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

- 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?

Using IF-THEN Rules for Classification

- Represent the knowledge in the form of IF-THEN rules

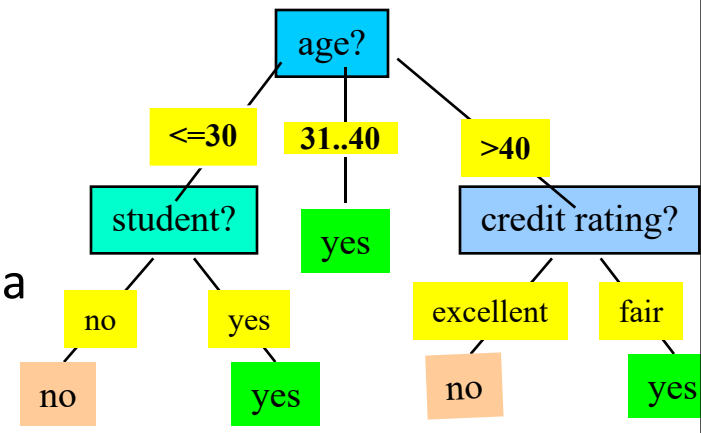
  R: IF *age* = youth AND *student* = yes THEN *buys_computer* = yes

  – Rule antecedent/precondition vs. rule consequent
- Assessment of a rule: *coverage* and *accuracy*
  – n_{covers} = # of tuples covered by R
  – $n_{correct}$ = # of tuples correctly classified by R

  coverage(R) = n_{covers} /|D| /* D: training data set */

  accuracy(R) = $n_{correct}$ / n_{covers}
- If more than one rule are triggered, need **conflict resolution**
  – Size ordering: assign the highest priority to the triggering rules that has the "toughest" requirement (i.e., with the *most attribute test*)
  – Class-based ordering: decreasing order of *prevalence or misclassification cost per class*
  – Rule-based ordering (**decision list**): rules are organized into one long priority list, according to some measure of rule quality or by experts

Rule Extraction from a Decision Tree



- Rules are easier to understand than large trees
- One rule is created for each path from the root to a leaf
- Each attribute-value pair along a path forms a conjunction: the leaf holds the class prediction
- Rules are mutually exclusive and exhaustive

- Example: Rule extraction from our *buys_computer* decision-tree

IF *age* = young AND *student* = *no* THEN *buys_computer* = *no*

IF *age* = young AND *student* = *yes* THEN *buys_computer* = *yes*

IF *age* = mid-age THEN *buys_computer* = *yes*

IF *age* = old AND *credit_rating* = *excellent* THEN *buys_computer* = *yes*

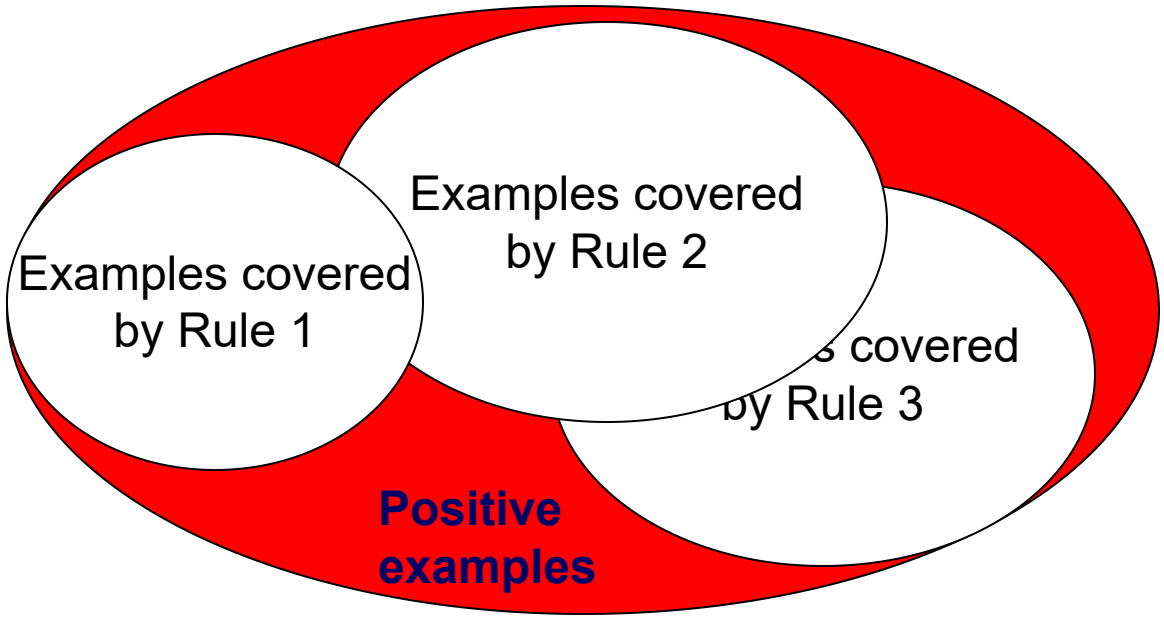
IF *age* = young AND *credit_rating* = *fair* THEN *buys_computer* = *no*

Rule Extraction from the Training Data

- Sequential covering algorithm: Extracts rules directly from training data
- Typical sequential covering algorithms: FOIL, AQ, CN2, RIPPER
- Rules are learned *sequentially*, each for a given class C_i will cover many tuples of C_i but none (or few) of the tuples of other classes
- Steps:
  - Rules are learned one at a time
  - Each time a rule is learned, the tuples covered by the rules are removed
  - The process repeats on the remaining tuples unless *termination condition*, e.g., when no more training examples or when the quality of a rule returned is below a user-specified threshold
- Comp. w. decision-tree induction: learning a set of rules *simultaneously*

Sequential Covering Algorithm

while (enough target tuples left)
    generate a rule
    remove positive target tuples satisfying this rule



How to Learn-One-Rule?

- Star with the most general rule possible: condition = empty
- Adding new attributes by adopting a greedy depth-first strategy
  - Picks the one that most improves the rule quality
- Rule-Quality measures: consider both coverage and accuracy
  - Foil-gain (in FOIL & RIPPER): assesses info_gain by extending condition

$$FOIL\_Gain = pos' \times (\log_2 \frac{pos'}{pos'+neg'} - \log_2 \frac{pos}{pos+neg})$$

It favors rules that have high accuracy and cover many positive tuples

- Rule pruning based on an independent set of test tuples

$$FOIL\_Prune(R) = \frac{pos - neg}{pos + neg}$$

Pos/neg are # of positive/negative tuples covered by R.

If *FOIL_Prune* is higher for the pruned version of R, prune R

Rule Generation

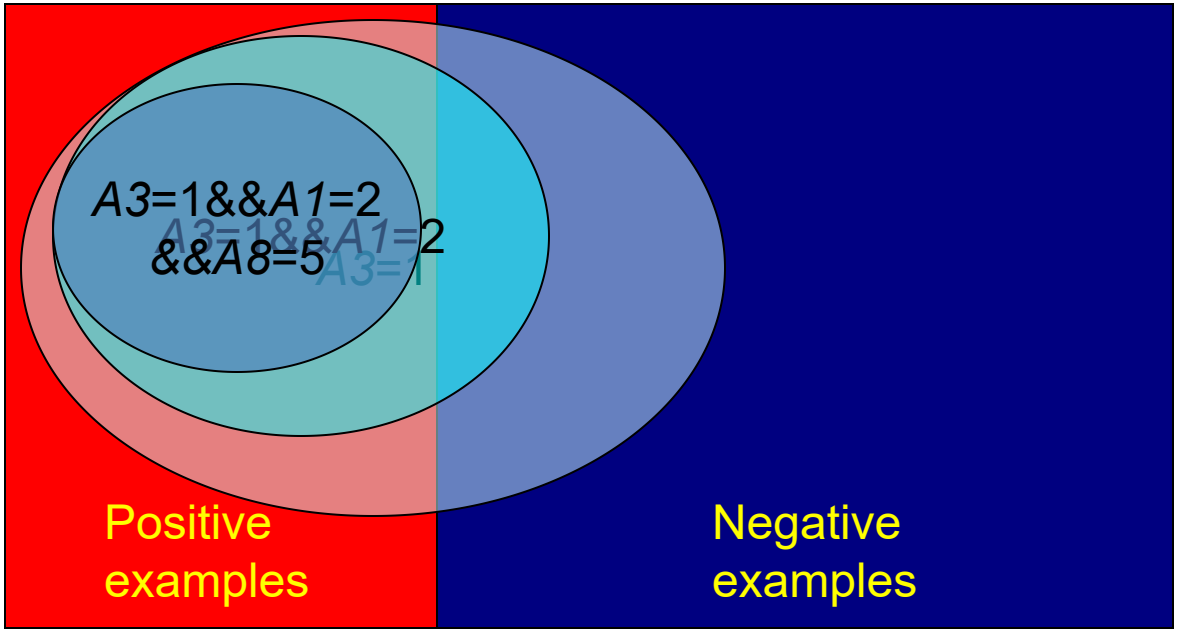
- To generate a rule

  while(true)

    find the best predicate *p*

    if foil-gain(*p*) > threshold **then** add *p* to current rule

    else break



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
- 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"

Enhancements to Basic Decision Tree Induction

- Allow for **continuous-valued attributes**
  - Dynamically define new discrete-valued attributes that partition the continuous attribute value into a discrete set of intervals
- Handle **missing attribute values**
  - Assign the most common value of the attribute
  - Assign probability to each of the possible values
- **Attribute construction**
  - Create new attributes based on existing ones that are sparsely represented
  - This reduces fragmentation, repetition, and replication

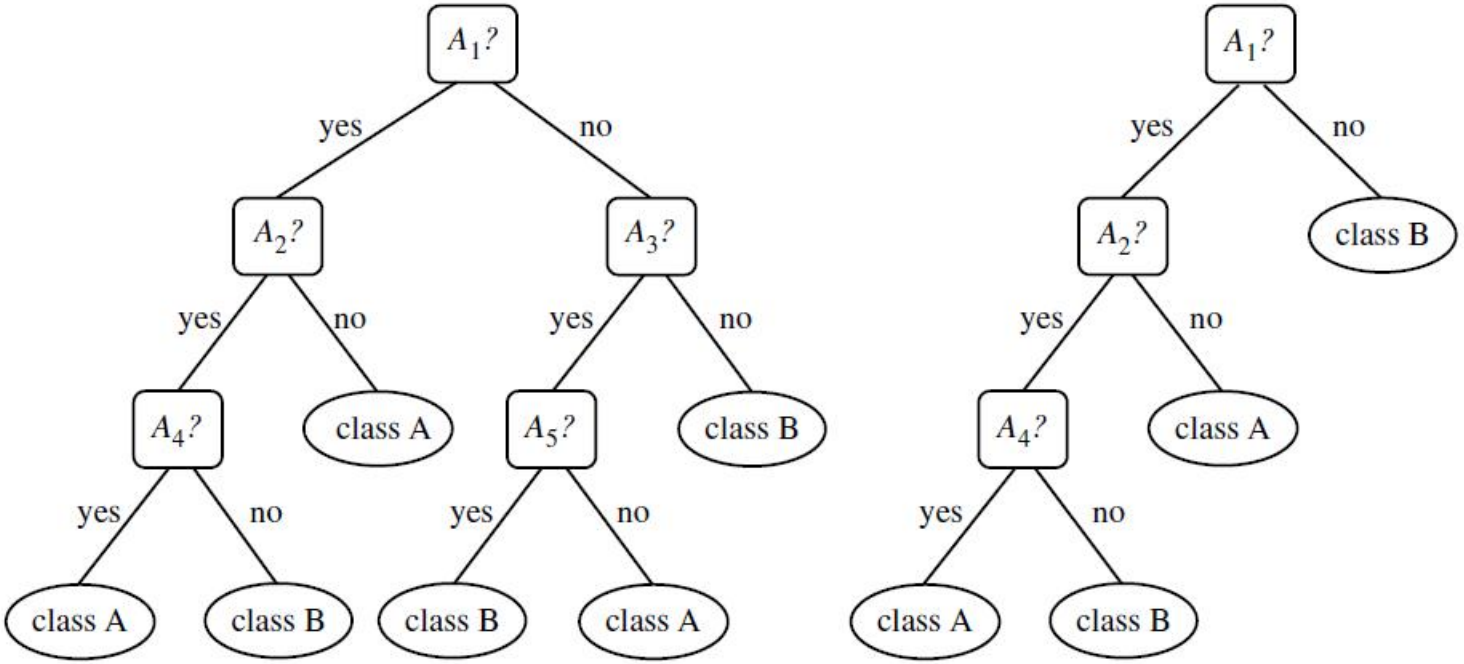
When a decision tree is built, many of the branches will reflect anomalies in the training data due to noise or outliers. Tree pruning methods address this problem of *overfitting* the data.

- Pruned trees tend to be smaller and less complex and, thus, easier to comprehend. Theyare usually faster and better at correctly classifying independent test data (i.e., of previously unseen tuples) than unpruned trees.

How does tree pruning work?" There are two common approaches to tree pruning:

- prepruning and postpruning.
- In the prepruning approach, a tree is "pruned" by halting its construction early (e.g.,by deciding not to further split or partition the subset of training tuples at a given node). Upon halting, the node becomes a leaf. The leaf may hold the most frequent class among the subset tuples or the probability distribution of those tuples.

- 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 pre-specified threshold, then further partitioning of the given subset is halted.
- There are difficulties, however, in choosing an appropriate threshold. High thresholds could result in oversimplified trees, whereas low thresholds could result in very little simplification.



An unpruned decision tree and a pruned version of it.

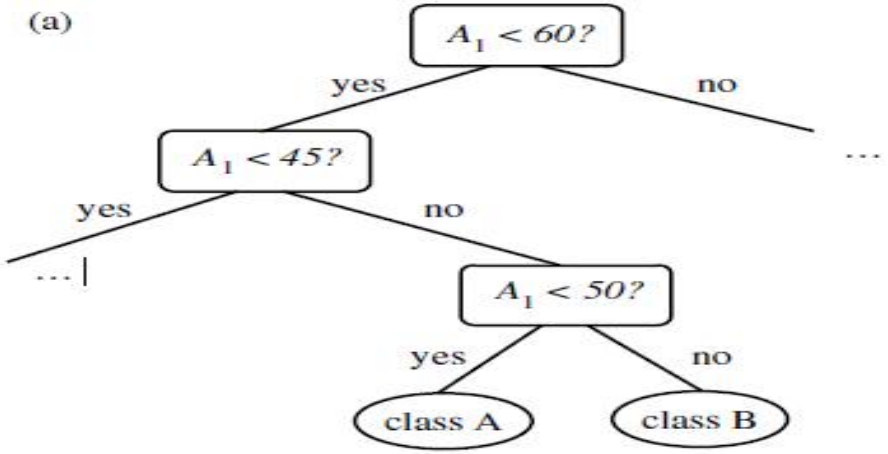
The second and more common approach is postpruning, which removes subtrees from a "fully grown" tree. A subtree at a given node is pruned by removing its branches and replacing it with a leaf. The leaf is labeled with the most frequent class among the subtree being replaced. For example, notice the subtree at node "*A3?" in the unpruned* tree of the given Figure. Suppose that the most common class within this subtree is "*class B.*" In the pruned version of the tree, the subtree in question is pruned by replacing it withthe leaf "*class B.*"

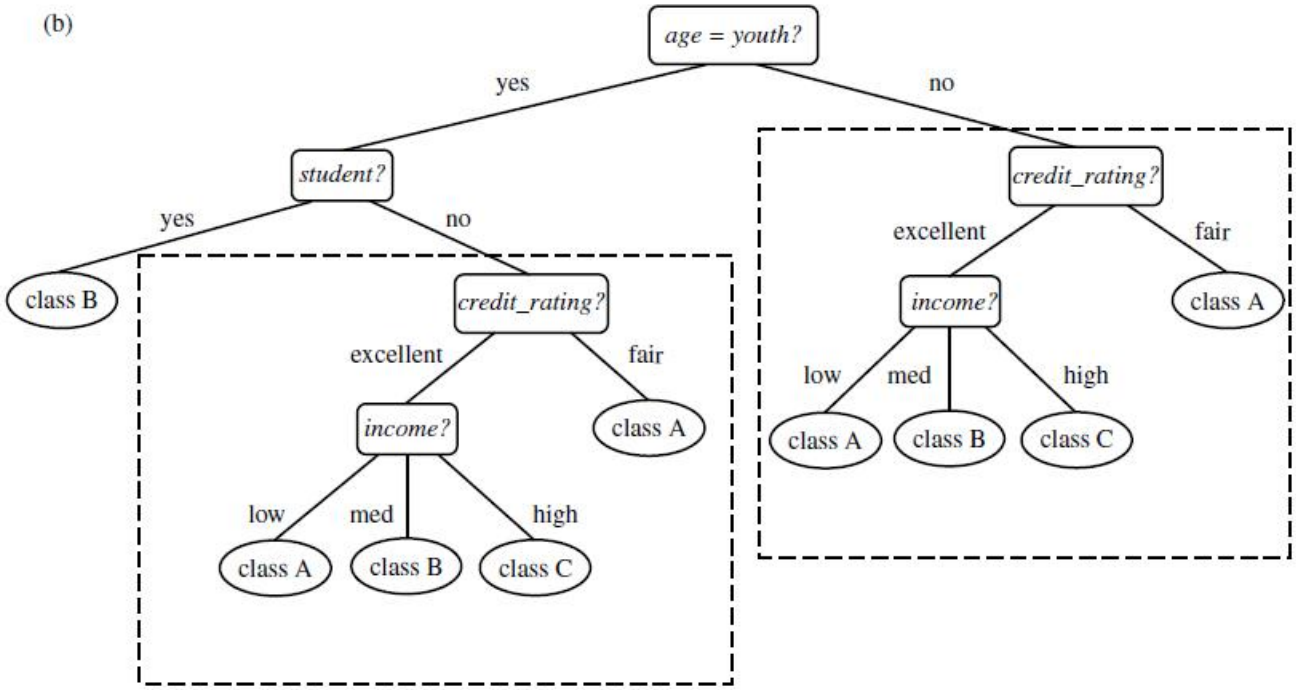
- The cost complexity pruning algorithm used in CART is an example of the postpruning approach.
- This approach considers the cost complexity of a tree to be a function of the number of leaves in the tree and the error rate of the tree (where the error rate is the percentage of tuples misclassified by the tree).
- It starts from the bottom of the tree. For each internal node, *N, it computes the cost complexity of the sub-tree at N, and* the cost complexity of the sub-tree at *N if it were to be pruned (i.e., replaced by a leaf* node). The two values are compared. If pruning the sub-tree at node *N would result in a* smaller cost complexity, then the sub-tree is pruned. Otherwise, it is kept.

- C4.5 uses a method called pessimistic pruning, which is similar to the cost complexity method in that it also uses error rate estimates to make decisions regarding subtree pruning.
- Rather than pruning trees based on estimated error rates, we can prune trees based on the number of bits required to encode them. The "best" pruned tree is the one that minimizes the number of encoding bits. This method adopts theMinimum Description Length (MDL) principle.

- Alternatively, prepruning and postpruning may be interleaved for a combined approach. Postpruning requires more computation than prepruning, yet generally leads to a more reliable tree. No single pruning method has been found to be superior over all others.

- Although pruned trees tend to be more compact than their unpruned counterparts, they may still be rather large and complex.
- Decision trees can suffer from *repetition and replication , making them overwhelming to interpret.*
- *Repetition occurs when* an attribute is repeatedly tested along a given branch of the tree (such as *"age < 60?",* followed by *"age < 45"?, and so on). In replication, duplicate subtrees exist within the* tree.
- These situations can impede the accuracy and comprehensibility of a decision tree.
- The use of multivariate splits (splits based on a combination of attributes) can prevent these problems.
- Another approach is to use a different form of knowledge representation, such as rules, instead of decision trees.


(a)
$A_1 < 60?$
yes
no
$A_1 < 45?$
...
yes
no
...
$A_1 < 50?$
yes
no
class A
class B



An example of subtree (a) repetition (where an attribute is repeatedly tested along a given branch of the tree, e.g., *age*) and (b) replication (where duplicate subtrees exist within a tree, such as the subtree headed by the node "*credit_rating?*").

- Scalability:-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.*
- *In data mining applications, very large training sets of millions* of tuples are common. Most often, the training data will not fit in memory. Decision tree construction therefore becomes inefficient due to swapping of the training tuples in and out of main and cache memories.

- More recent decision tree algorithms that address the scalability issue have been proposed. Algorithms for the induction of decision trees from very large training sets include SLIQ and SPRINT, both of which can handle categorical and continuous valued attributes. Both algorithms propose presorting techniques on disk-resident data sets that are too large to fit in memory.

Thank You

Thank You

Information Sources: NPTEL, HanKamber.