SRM UNIVERSITY
(Under section 3 of UGC Act 1956)

Classification

UNIT-3

Classification - Basic Concepts

- Classification is a form of data analysis that extracts models describing important data classes.
- Eg: classification model to categorize bank loan applications as either safe or risky
- Classification and prediction are two forms of data analysis that can be used to extract models describing important data classes or to predict future data trends.
- *Classification*- predicts categorical (discrete, unordered) labels.
- *prediction* - models continuous valued functions.

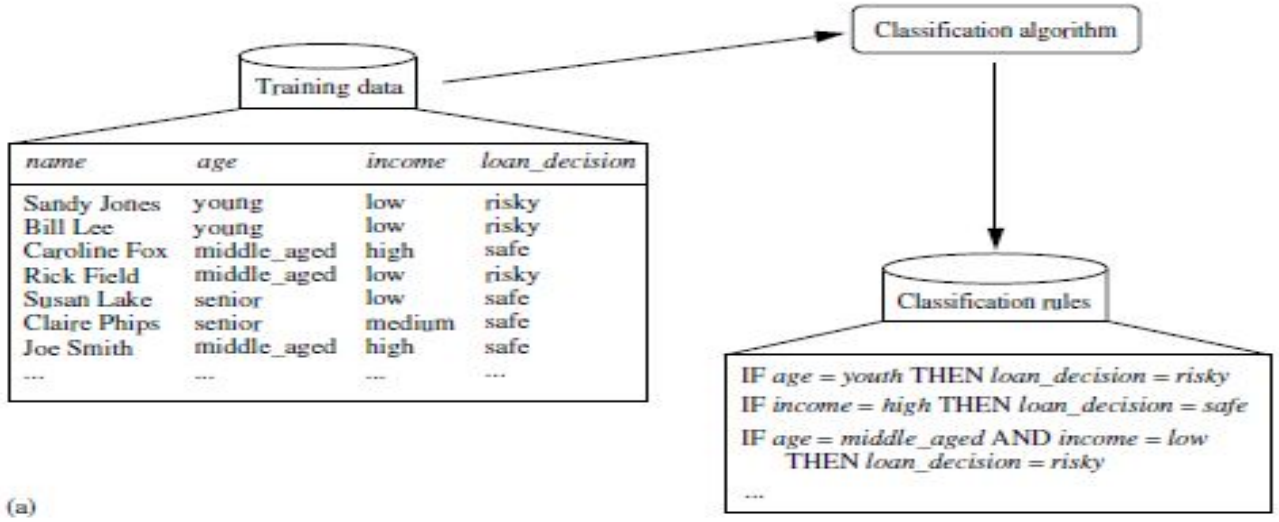
What Is Classification? What Is Prediction?

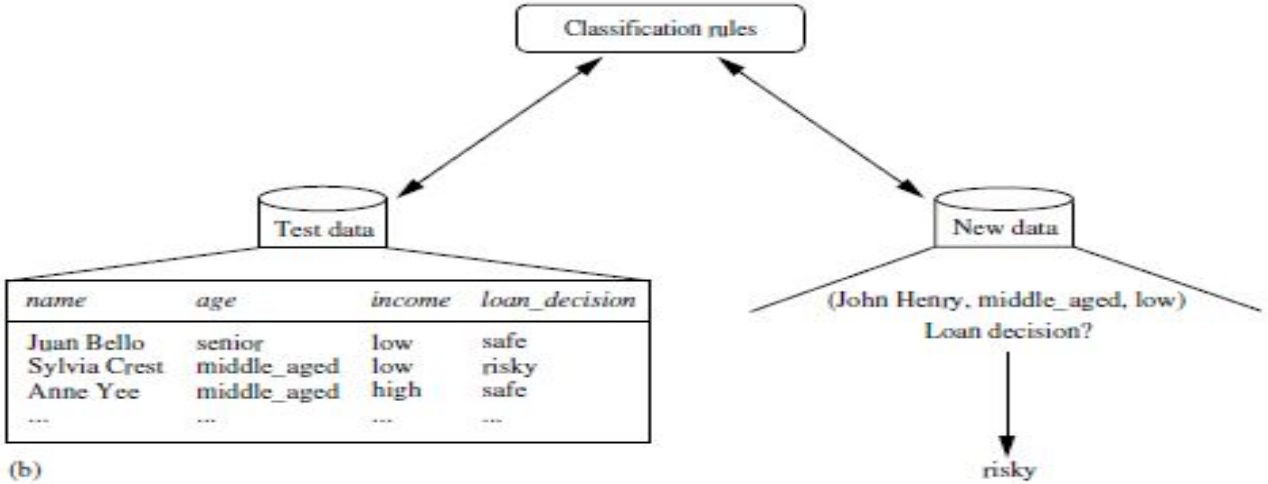
- A bank loans officer needs analysis of her data in order to learn which loan applicants are “safe” and which are “risky” for the bank.
- A marketing manager at *AllElectronics* needs data analysis to help guess whether a customer with a given profile will buy a new computer.
- The data analysis task is **classification**, where a model or classifier is constructed to predict categorical labels, such as **“safe” or “risky”** for the loan application data; **“yes” or “no”** for the marketing data.

General Approach to Classification

How does classification work?

- Data classification is a two-step process, consisting of a
  - *-learning step (where a classification model is constructed)*
  - *-classification step (where* the model is used to predict class labels for given data).


Classification algorithm
Training data
name age income loan_decision
Sandy Jones young low risky
Bill Lee young low risky
Caroline Fox middle_aged high safe
Rick Field middle_aged low risky
Susan Lake senior low safe
Claire Phips senior medium safe
Joe Smith middle_aged high safe
... ...
Classification rules
IF age = youth THEN loan_decision = risky
IF income = high THEN loan_decision = safe
IF age = middle_aged AND income = low
THEN loan_decision = risky
...
(a)


Classification rules
Test data
New data
name age income loan_decision
Juan Bello senior low safe
Sylvia Crest middle_aged low risky
Anne Yee middle_aged high safe
... ...
(John Henry, middle_aged, low)
Loan decision?
risky
(b)

•In the first step, a classifier is built describing a predetermined set of data classes or concepts.

•This is the learning step (or training phase), where a classification algorithm builds the classifier by analyzing or "learning from" a training set made up of database tuples and their associated class labels.

• A tuple, *X, is represented by an n-dimensional* attribute vector, *X = (x1, x2, ... , xn), depicting n measurements made on the tuple* from *n database attributes, respectively, A1, A2,.... , An.*

• *Each tuple, X, is assumed to* belong to a predefined class as determined by another database attribute called the class label attribute.

-The class label attribute is discrete-valued and unordered.

-It is *categorical* (or nominal) in that each value serves as a category or class.

•The individual tuples making up the training set are referred to as training tuples and are randomly sampled from the database under analysis.

•The data tuples can be referred to as *samples, examples, instances, data*

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

"What about classification accuracy?"

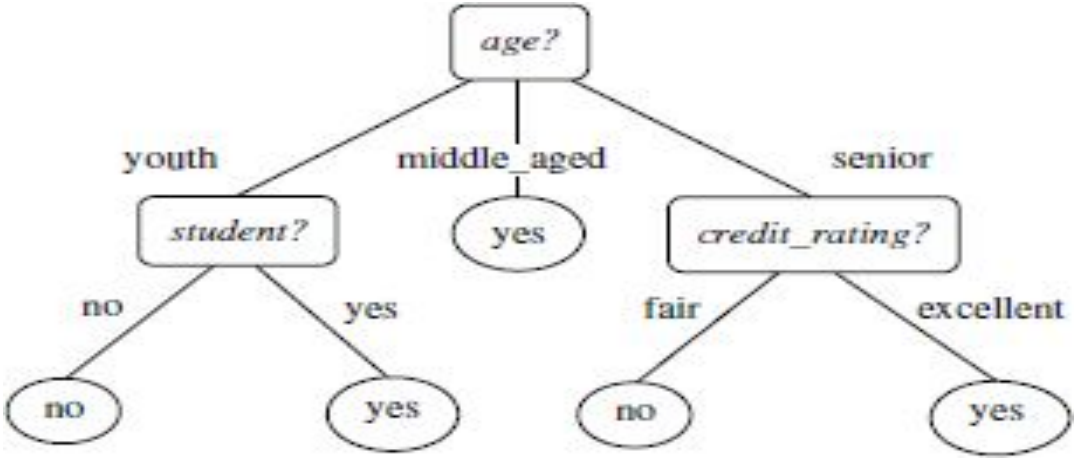
SRM UNIVERSITY
(Under section 3 of UGC Act 1956)

- First, the predictive accuracy of the classifier is estimated.
- If we were to use the training set to measure the classifier's accuracy, this estimate would likely be optimistic
  - -The classifier tends to **overfit the data (i.e., during learning it may** incorporate some particular anomalies of the training data that are not present in the general data set overall).
- A **test set is used, made up of test tuples and their** associated class labels.
- They are independent of the training tuples,
  - -They were not used to construct the classifier.

Decision Tree Induction

A **decision tree** is a flowchart-like tree structure.

- Internal node (non leaf node) - test on an attribute.
- Branch - outcome of the test.
- Leaf node (or *terminal node*) - holds a class label.
- The topmost node in a tree is the **root** node.



"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 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.
- Decision trees can handle high dimensional data.
- The learning and classification steps of decision tree induction are simple and fast.
- Decision tree classifiers have good accuracy.

Decision Tree Induction

- During the late 1970s and early 1980s, J. Ross Quinlan, a researcher in machine learning, developed a decision tree algorithm known as **ID3** (Iterative Dichotomiser).
- P. T. Stone. Quinlan later presented **C4.5** (a successor of ID3).
- In 1984, a group of statisticians (L. Breiman, J. Friedman, R. Olshen, and C. Stone) published the book ***Classification and Regression Trees*** **(CART)**, which described the generation of binary decision trees.
- ID3, C4.5, and CART adopt a **greedy** (i.e., non backtracking) approach in which decision trees are constructed in a **top-down** recursive divide-and-conquer manner.

- The algorithm is called with three parameters:
  - ***D***
  - ***attribute list***, and
  - ***Attribute selection method***.
- *D* - data partition
- *Attribute list* - list of attributes describing the tuples.
- *Attribute selection method*
  - *p*rocedure for selecting the attribute.
  - *It specifies a heuristic* procedure for selecting the attribute that "best" discriminates the given tuples according to class.
  - Eg. information gain or the Gini index.

ALGORITHM FOR DECISION TREE INDUCTION

SRM UNIVERSITY
(Under section 3 of UGC Act 1956)

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;
```

Basic algorithm for inducing a decision tree from training tuples.

1. The tree starts as a single node, *N* representing the training tuples in *D.*

2. 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 terminating conditions are explained at the end of the algorithm.

3.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).

4.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.

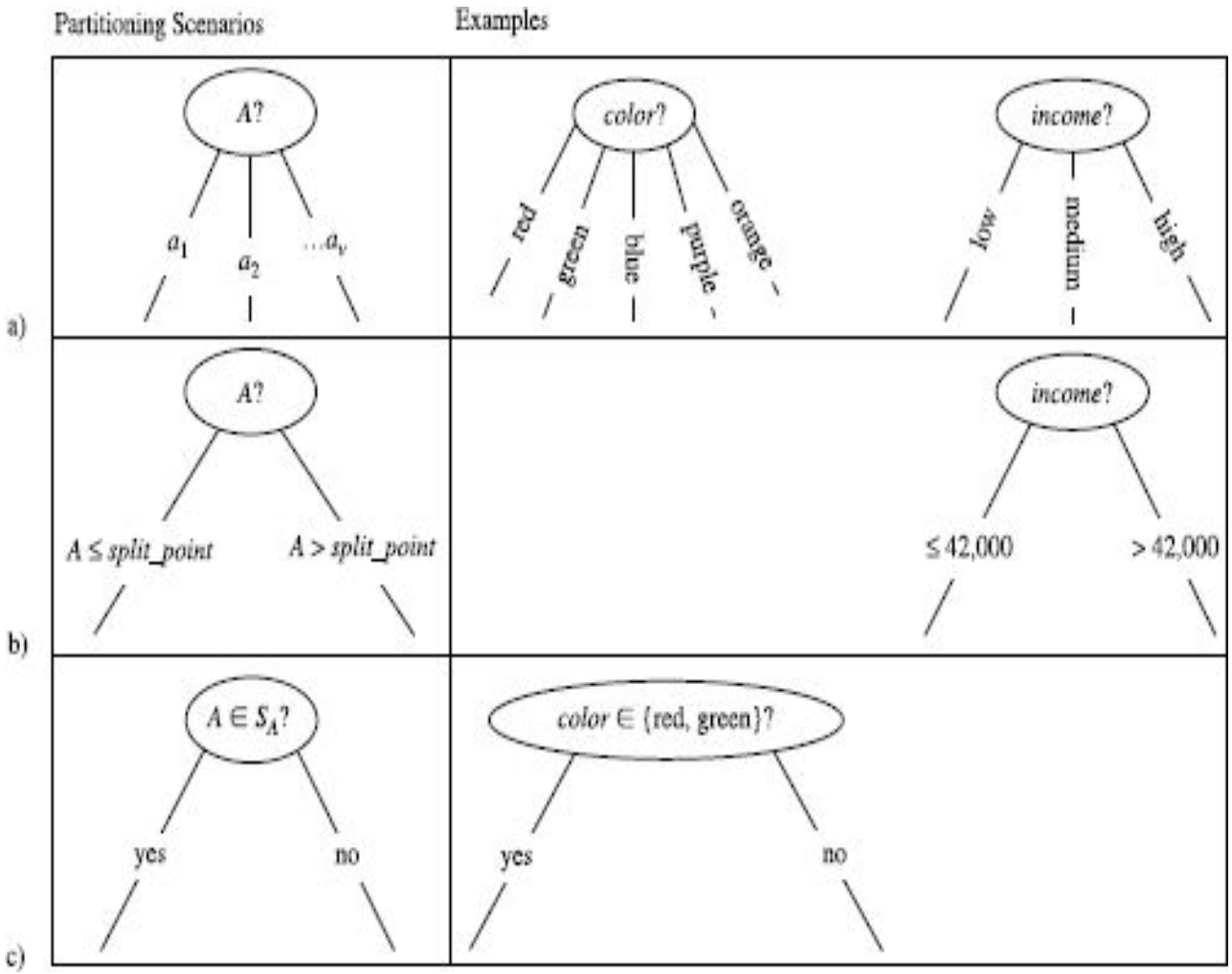
-A partition is **pure if all the tuples** in it belong to the same class.

- 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.

5. 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,
  - *A be the splitting attribute.*
  - *A is continuous-valued*
  - *A is discrete-valued and a binary tree*

There are three possible scenarios, as illustrated in Figure 8.4. Let *A be the splitting attribute. A has v* distinct values, f*a1, a2, : : : , avg, based on the training data.*



***A is discrete-valued*:**

- The outcomes of the test at node *N* correspond directly to the known values of *A*.
- A branch is created for each known value, *aj*, of *A* and labeled with that value.
- Partition *Dj* is the subset of class-labeled tuples in *D* having value *aj* 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*

***A is continuous-valued*:**

- 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.

***A is discrete-valued* and a *binary tree* must be produced**

- The test at node *N* is of the form "$A \in SA$?"
- *SA* is the splitting subset for *A*, returned by *Attribute selection method* as part of the splitting criterion.
- 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, *Dj , 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 D_j is empty (step 12). In this case, a leaf is created with the majority class in D (step 13).

The resulting decision tree is returned (step 15).

Attribute Selection Measures

SRM UNIVERSITY
(Under section 3 of UGC Act 1956)

•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 the tuples that fall into a given partition would belong to the same class).

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

- Three popular attribute selection **measures**
  - ***information gain***
  - ***gain ratio*, and**
  - ***gini index*.**

Information gain

- **ID3** uses information gain as its attribute selection measure.
- The attribute with the **highest information gain** is chosen as the splitting attribute for node *N*.
- Where
  - p_i is the probability that an arbitrary tuple in D belongs to class Ci and is estimated by |Ci,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.
- *Info(D)* is also known as the **entropy of *D*.**
- The expected info$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i),$$ssify a tuple in *D is given by*

•How much more information would we still need (after the partitioning) in order to arrive at an exact classification?
This amount is measured by

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

- The term |*Dj* |/|*D*| acts as the weight of the *j*th partition.
- *InfoA*(*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*).

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

Induction of a decision tree using information gain

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

- The class label attribute, *buys computer*, has two distinct values (namely, {*yes, no*}).
- There are two distinct classes (that is, $m = 2$).
- Let class ***C*1** correspond to ***yes*** and class ***C*2** correspond to ***no***.
- There are **nine** tuples of class ***yes*** and **five** tuples of class ***no***.
- A (root) node *N* is created for the tuples in *D*.

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

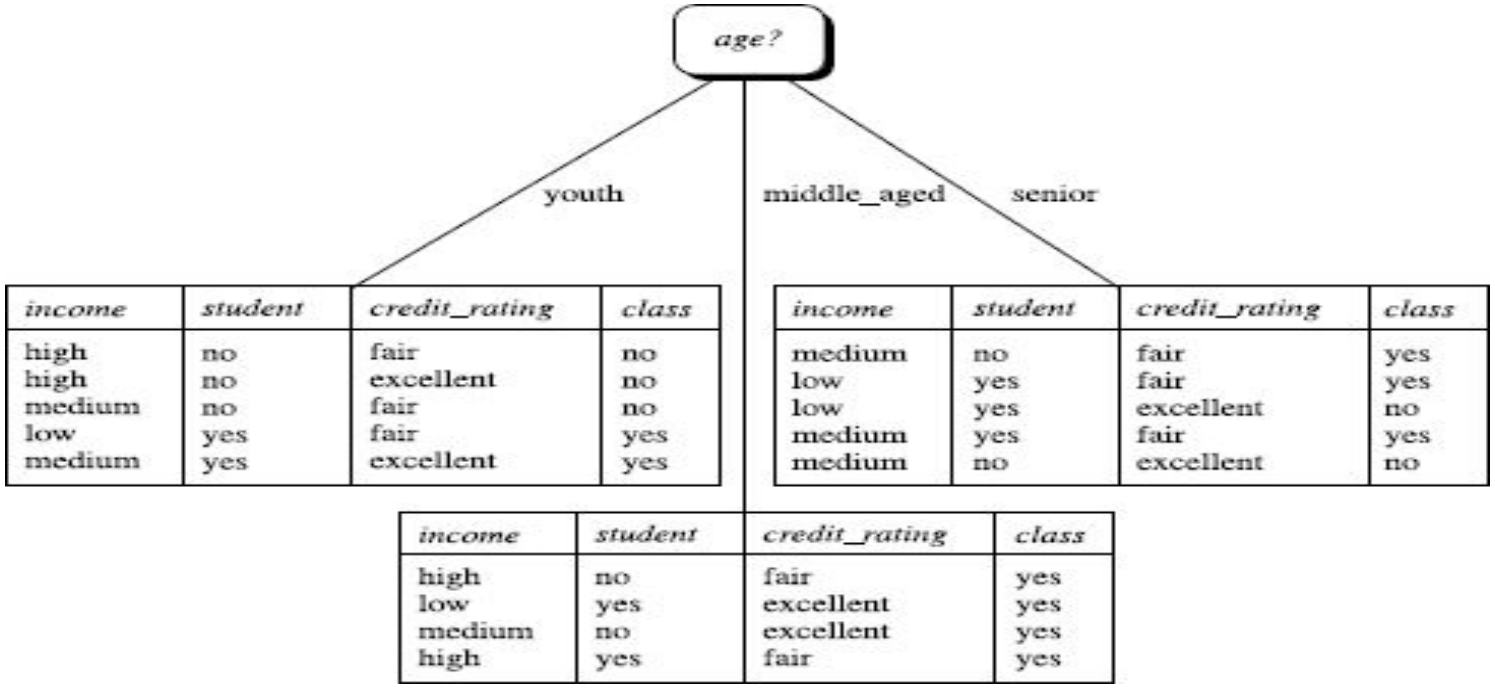
- Compute the expected information requirement for each attribute.
- *Age* category *youth – 2 yes &3 no, Middle aged – 4 yes & 0 no, Senior – 3 yes & 2 no.*

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

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

- Compute *Gain*(*income*) = 0.029 bits, *Gain*(*student*) = 0.151 bits, and *Gain*(*credit rating*) = 0.048 bits.
- Age has the highest information gain among the attributes, it is selected as the splitting attribute.

- 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."*



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

Information gain of attribute – continuous valued

- 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.

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

Predict if John will play tennis

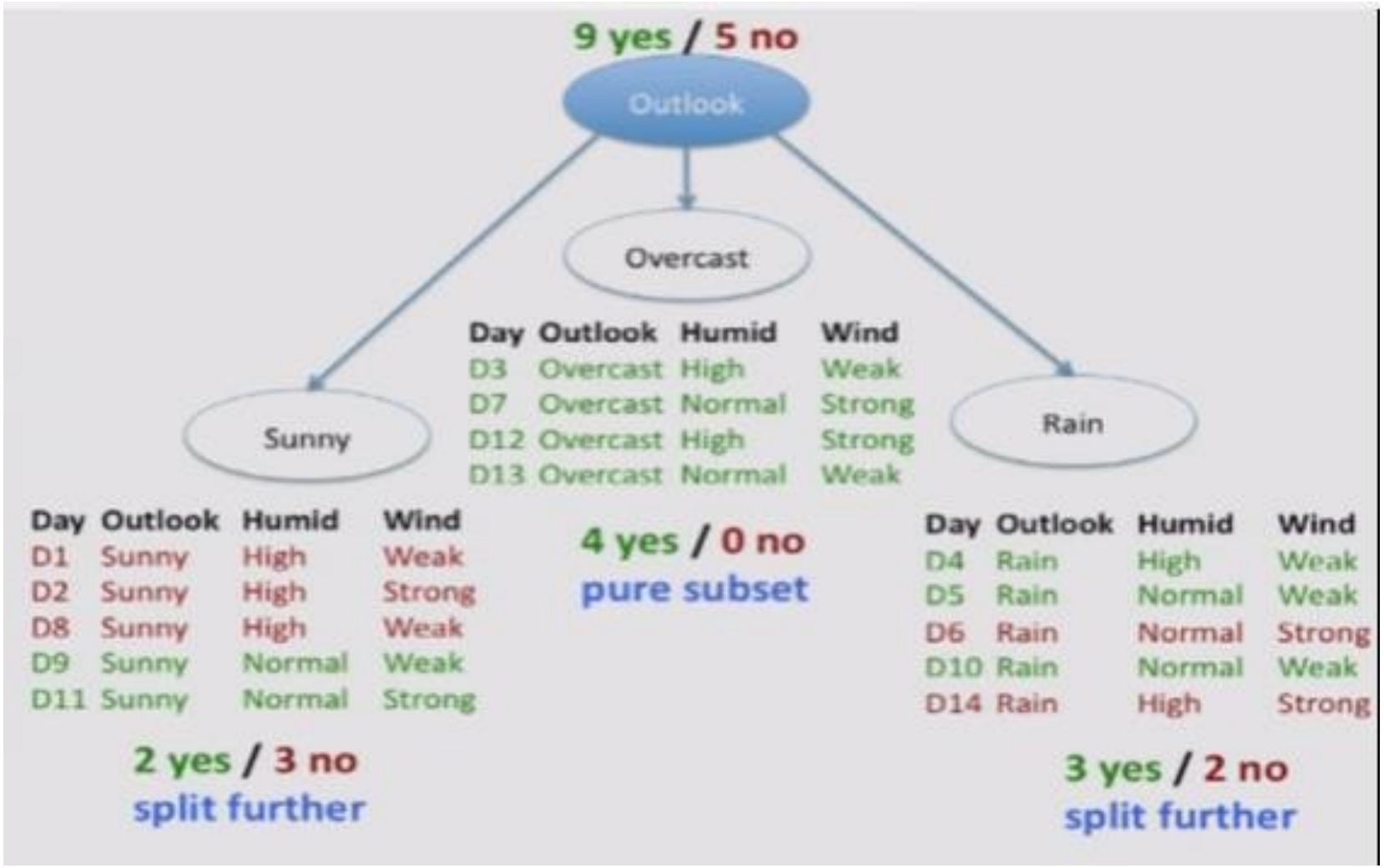
- Hard to guess
- Try to *understand* when John plays
- Divide & conquer:
  - split into subsets
  - are they pure? (all yes or all no)
  - if yes: stop
  - if not: repeat
- See which subset new data falls into

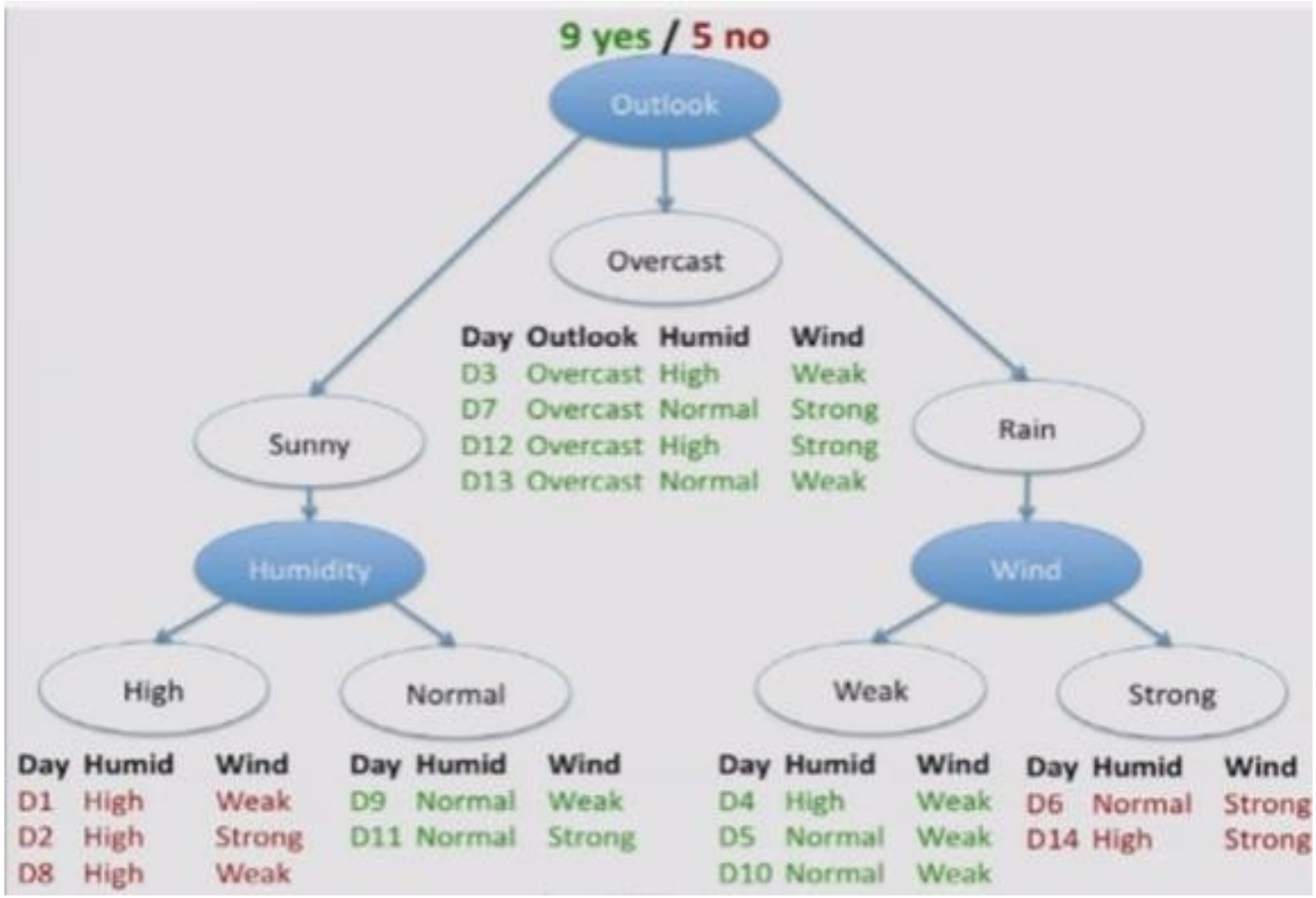
Training examples: **9 yes / 5 no**

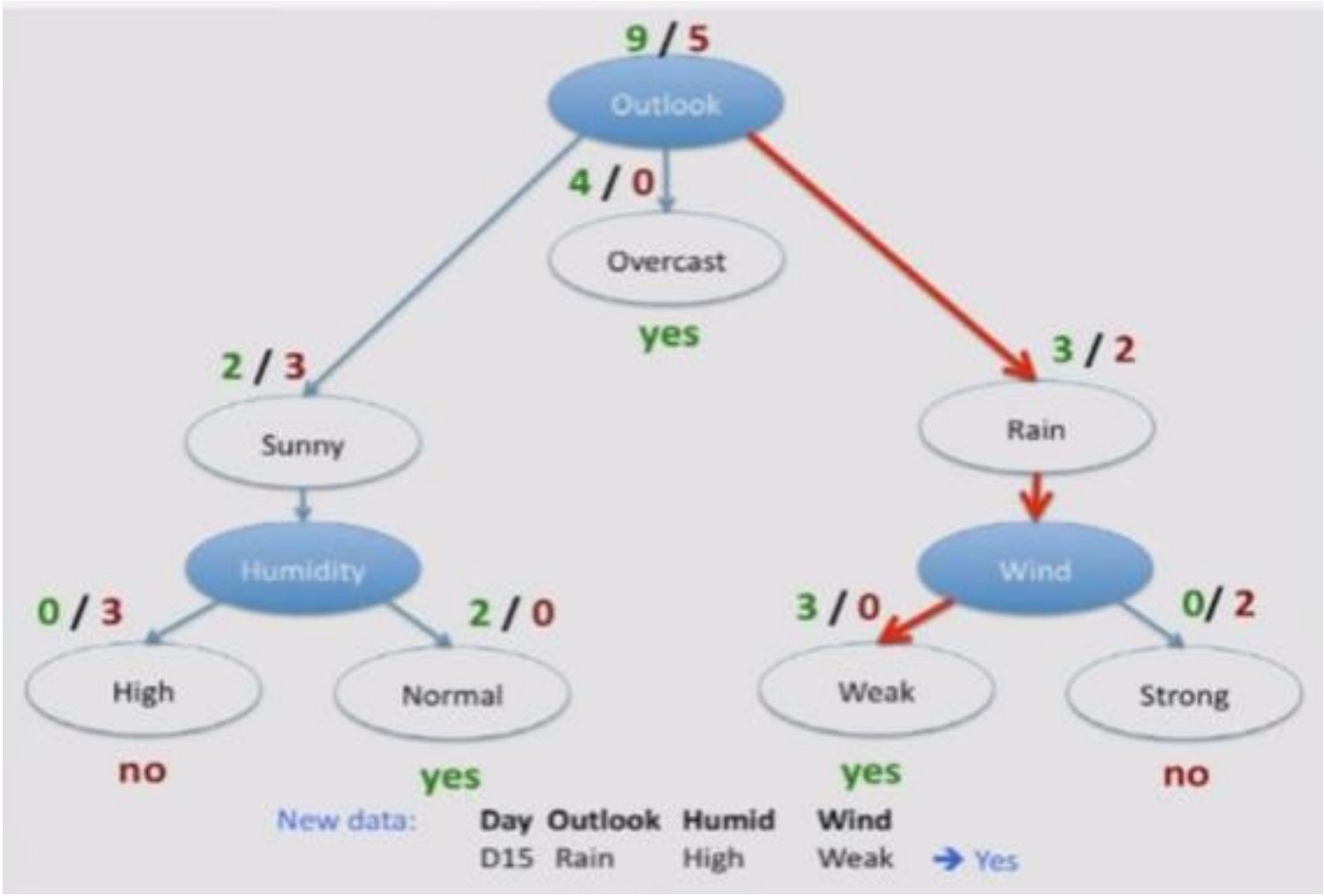
Day	Outlook	Humidity	Wind	Play
D1	Sunny	High	Weak	No
D2	Sunny	High	Strong	No
D3	Overcast	High	Weak	Yes
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No
D7	Overcast	Normal	Strong	Yes
D8	Sunny	High	Weak	No
D9	Sunny	Normal	Weak	Yes
D10	Rain	Normal	Weak	Yes
D11	Sunny	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes
D14	Rain	High	Strong	No

New data:

D15	Rain	High	Weak	?


9 yes / 5 no
Outlook
Overcast
Sunny
Rain
Day Outlook Humid Wind
D3 Overcast High Weak
D7 Overcast Normal Strong
D12 Overcast High Strong
D13 Overcast Normal Weak
4 yes / 0 no
pure subset
Day Outlook Humid Wind
D1 Sunny High Weak
D2 Sunny High Strong
D8 Sunny High Weak
D9 Sunny Normal Weak
D11 Sunny Normal Strong
2 yes / 3 no
split further
Day Outlook Humid Wind
D4 Rain High Weak
D5 Rain Normal Weak
D6 Rain Normal Strong
D10 Rain Normal Weak
D14 Rain High Strong
3 yes / 2 no
split further


9 yes / 5 no
Outlook
Overcast
Sunny
Rain
Humidity
Wind
High
Normal
Weak
Strong
Day Outlook Humid Wind
D3 Overcast High Weak
D7 Overcast Normal Strong
D12 Overcast High Strong
D13 Overcast Normal Weak
Day Humid Wind
D1 High Weak
D2 High Strong
D8 High Weak
Day Humid Wind
D9 Normal Weak
D11 Normal Strong
Day Humid Wind
D4 High Weak
D5 Normal Weak
D10 Normal Weak
Day Humid Wind
D6 Normal Strong
D14 High Strong


9 / 5
Outlook
4 / 0
Overcast
yes
2 / 3
Sunny
3 / 2
Rain
Humidity
Wind
0 / 3
2 / 0
3 / 0
0 / 2
High
Normal
Weak
Strong
no
yes
yes
no
New data:
Day Outlook Humid Wind
D15 Rain High Weak
Yes

Gain ratio

- It prefersto select attributes having a large number of values.
- C4.5, a successor of ID3, uses an extension to information gain known as *gain ratio.*
- It applies a kind of normalization to information gain using a "split information" value defined analogously with *Info*(*D*) as

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

- 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

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

Computation of gain ratio for the attribute *income*. A test on *income* splits the data of Table 6.1 into three partitions, namely *low*, *medium*, and *high*, containing four, six, and four tuples, respectively. To compute the gain ratio of *income*, we first use Equation (6.5) to obtain

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

$$= 0.926.$$

$$Gain(income) = 0.029. \qquad GainRatio(income) = 0.029/0.926 = 0.031.$$

Gini index

- The Gini index is used in CART. $Gini(D) = 1 - \sum_{i=1}^{m} p_i^2,$

where p_i is the probability that a tuple in D belongs to class C_i 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.
- where *A* is a discrete-valued attribute having *v* distinct values, {*a*1, *a*2, … , *av*}, occurring in *D*.
- Determine the best binary split on *A*, we examine all of the possible subsets that can be formed using known values of *A*.
- If *A* has *v* possible values, then there are 2*v* possible subsets.

Eg: *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 {}.

- Exclude the power set,{*low, medium, high*}, and the empty - they do not represent a split.
- 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 a discrete-valued attribute, the subset that gives the minimum gini index for that attribute is selected as its splitting subset.

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.

Induction of a decision tree using gini index:

Let *D* be the training data where there are nine tuples belonging to the class *buys computer = yes* and the remaining five tuples belong to the class *buys computer = no*. A (root) node *N* is created for the tuples in *D*. We first use Equation for Gini index to compute the impurity of *D*

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

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* $\in$ {*low, medium*}." The remaining four tuples of *D* would be assigned to partition *D*2.

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

The Gini index values for splits on the remaining subsets are:

- ({*low, high*} and {*medium*}) = 0.315
- ({*medium, high*} and {*low*}) = 0.300

- Therefore, the best binary split for attribute *income* is on {*medium, high*} (or {*low*})because it minimizes the gini index.

- {*youth, senior}* (or {*middle aged*}) best split for *age* with a Gini index of 0.375.

- {*student*} = 0.367
- {*credit rating*} = 0.429

- The attribute *income* and splitting subset {*medium, high*} therefore give the minimum gini index overall, with a reduction in impurity of 0.459-0.300 = 0.159.

- The binary split "*income* $\in$ {*medium, high*}" results in the maximum reduction in impurity of the tuples in *D* and is returned as the splitting criterion.

Tree Pruning

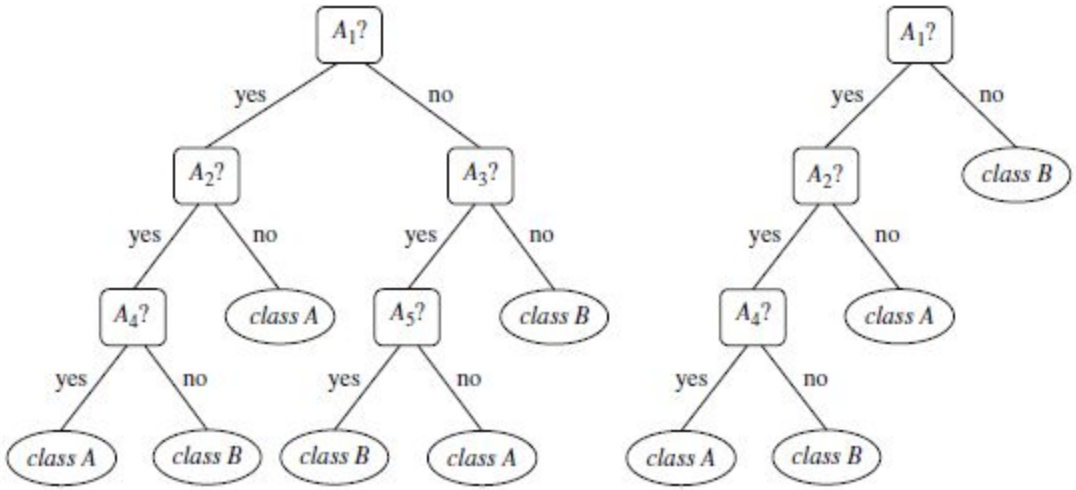
•When a decision tree is built, many of the branches will reflect anomalies in the training data due to noise or outliers.

•Tree pruning methods address this problem of *overfitting* the data.

•Such methods typically use statistical measures to remove the least-reliable branches.

•Pruned trees tend to be smaller and less complex and, thus, easier to comprehend.

•An unpruned tree and a pruned version of it are shown in Figure.



"How does tree pruning work?"

SRM UNIVERSITY
(Under section 3 of UGC Act 1956)

There are two common approaches to tree pruning:

-prepruning
-Postpruning

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 prespecified 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.

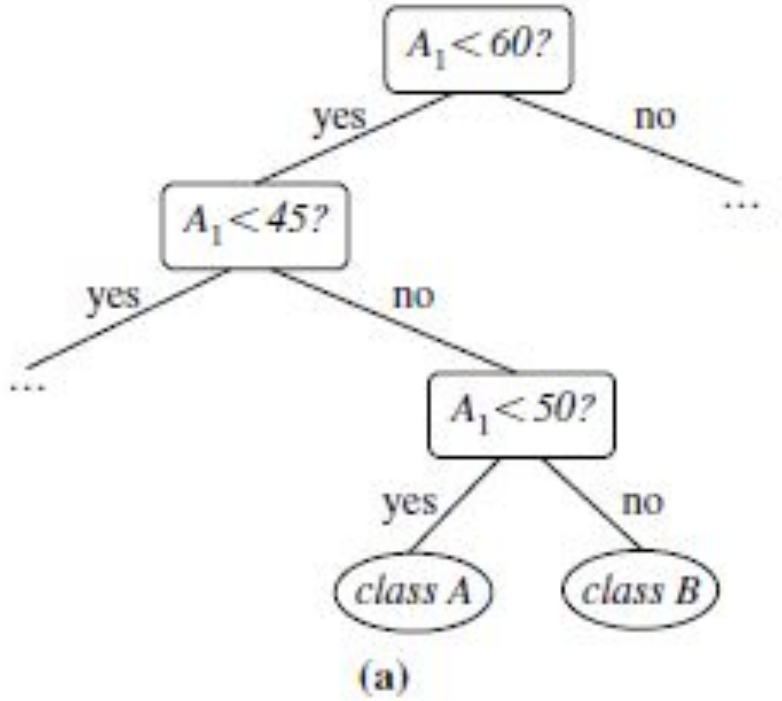
Postpruning Approach

SRM UNIVERSITY
(Under section 3 of UGC Act 1956)

- The second and more common approach is **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*

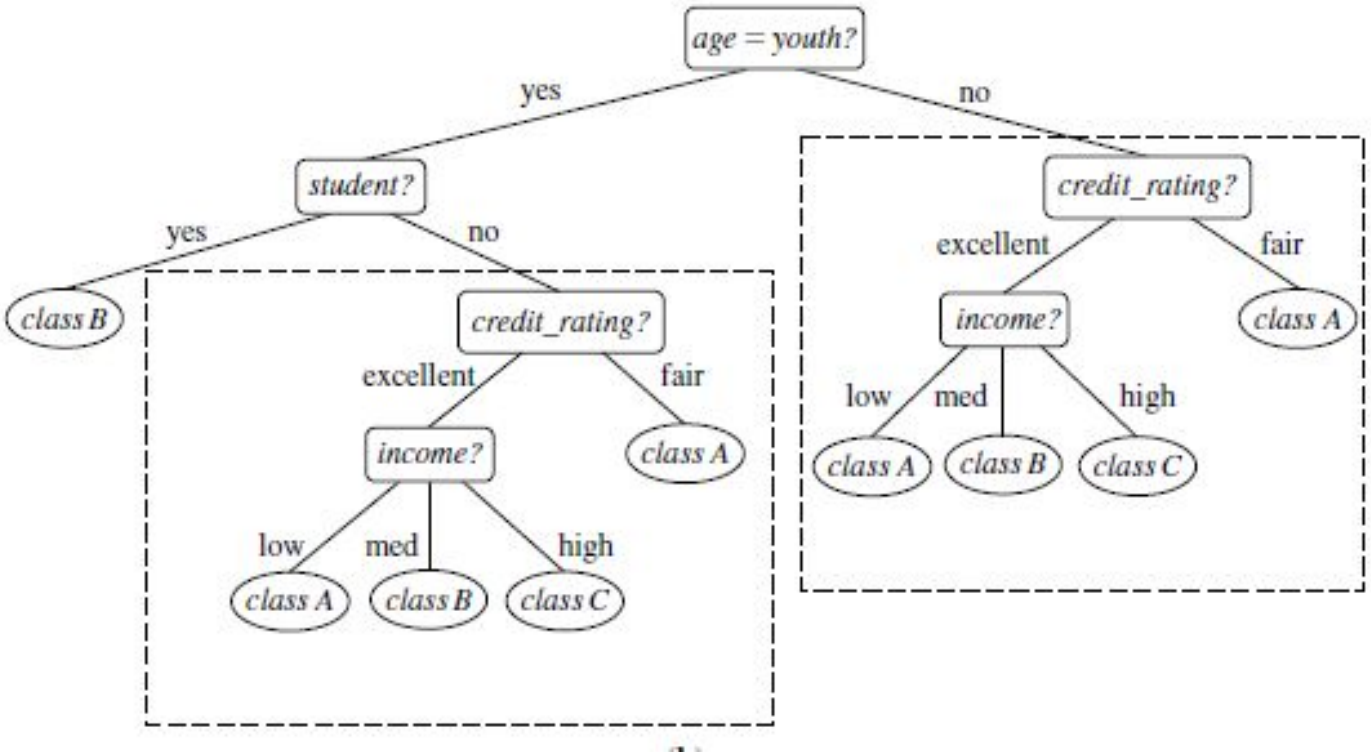
subtree repetition, where an attribute is repeatedly tested along a given branch of the tree (e.g., *age)*



(a)

subtree replication, where duplicate subtrees exist within a tree

-e.g., the subtree headed by the node "*credit rating?*"



Scalability and Decision Tree Induction

SRM UNIVERSITY
(Under section 3 of UGC Act 1956)

- In data mining applications, very large training sets of millions of tuples are common.
- Most often, the training data will not fit in memory.
- So the decision tree construction becomes inefficient due to swapping of the training tuples in and out of main and cache memories.
- More scalable approaches, capable of handling training data that are too large to fit in memory, are required.
- Earlier strategies to "save space" included discretizing continuous-valued attributes and sampling data at each node.
- These techniques, however, still assume that the training set can fit in memory.

"Attribute-Value, Classlabel"

- One of the scalable decision tree induction method
- It is an **AVC-set**
  - "AVC" stands for *"Attribute-Value, Classlabel") for each attribute, at each tree node,* describing the training tuples at the node.
- The AVC-set of an attribute *A at node N* gives the class label counts for each value of *A for the tuples at N.*

age	*buys_computer* yes	no
youth	2	3
middle_aged	4	0
senior	3	2

income	*buys_computer* yes	no
low	3	1
medium	4	2
high	2	2

student	*buys_computer* yes	no
yes	6	1
no	3	4

credit_ratting	*buys_computer* yes	no
fair	6	2
excellent	3	3

- The set of all AVC-sets at a node *N is the* ***AVC-group*** of *N.*
- *The size of an AVC-set for attribute A at node N depends only on the number of* distinct values of *A and the number of classes in the set of tuples at N.*
- *Typically, this size* should fit in memory, even for real-world data.

BOAT - Another scalable decision tree induction method

BOAT usually requires only two scans of *D.*

- *This is quite an improvement, even* in comparison to traditional decision tree algorithms (which require one scan per tree level)
- BOAT was found to be two to three times faster than RainForest, while constructing exactly the same tree.
- An additional advantage of BOAT is that it can be used for incremental updates.
- That is, BOAT can take new insertions and deletions for the training data and update the decision tree to reflect these changes, without having to reconstruct the tree from scratch.

Bayes Classification Methods

- *Bayesian classifiers are statistical classifiers*
- *They can* predict class membership probabilities such as the probability that a given tuple belongs to a particular class.
- Bayesian classifiers have also exhibited high accuracy and speed when applied to large databases
- Bayesian classification is based on Bayes' theorem.
- Studies comparing classification algorithms have found a simple Bayesian classifier known as the *naïve Bayesian classifier.*
- Naïve Bayesian classifiers assume that the effect of an attribute value on a given class is independent of the values of the other attributes.
  - This assumption is called *class conditional independence*

Bayes' Theorem

- Let ***X*** be a data tuple.
- In Bayesian terms, ***X*** is considered "evidence."
- Let *H* be some hypothesis, such as that the data tuple ***X*** belongs to a specified class *C*.
- For classification problems, we want to
- determine $P(H|\boldsymbol{X})$, the probability that the hypothesis *H* holds given the "evidence" or observed data tuple ***X***.
- P(H|X) is the posterior probability, or a posteriori probability, of H conditioned on X.
- The attributes age and income, respectively, and that X is a 35-year-old customer with an income of $40,000.
- Suppose that H is the hypothesis that our customer will buy a computer. Then P(H|X) reflects the probability that customer X will buy a computer given that we know the customer's age and income.

- $P(H)$ is the prior probability, or *a priori probability,* of H.
- $P(\boldsymbol{X}|H)$ is the posterior probability of $\boldsymbol{X}$ conditioned on H.
- it is the probability that a customer, $\boldsymbol{X}$, is 35 years old and earns $40,000, given that we know the customer will buy a computer.
- $P(\boldsymbol{X})$ is the prior probability of $\boldsymbol{X}$. Using our example, it is the probability that a person from our set of customers is 35 years old and earns $40,000.
- calculating the posterior probability, P(H|X).
- **Bayesian classifiers** have the **minimum error rate** in comparison to all other classifiers.

$$P(H|\boldsymbol{X}) = \frac{P(\boldsymbol{X}|H)P(H)}{P(\boldsymbol{X})}.$$

Naïve Bayesian Classification

The naïve Bayesian classifier, or simple Bayesian classifier, works as follows:

1. Let D be a training set of tuples and their associated class labels. As usual, each tuple is represented by an n-dimensional attribute vector, $\boldsymbol{X} = (x_1, x_2, \ldots, x_n)$, depicting n measurements made on the tuple from n attributes, respectively, $A_1, A_2, \ldots, A_n$.

2. Suppose that there are m classes, $C_1, C_2, \ldots, C_m$. Given a tuple, $\boldsymbol{X}$, the classifier will predict that $\boldsymbol{X}$ belongs to the class having the highest posterior probability, conditioned on $\boldsymbol{X}$. That is, the naïve Bayesian classifier predicts that tuple $\boldsymbol{X}$ belongs to the class C_i if and only if

$$P(C_i|\boldsymbol{X}) > P(C_j|\boldsymbol{X}) \quad \text{for } 1 \leq j \leq m, j \neq i.$$

Thus we maximize $P(C_i|\boldsymbol{X})$. The class C_i for which $P(C_i|\boldsymbol{X})$ is maximized is called the *maximum posteriori hypothesis*. By Bayes' theorem (Equation (6.10)),

$$P(C_i|\boldsymbol{X}) = \frac{P(\boldsymbol{X}|C_i)P(C_i)}{P(\boldsymbol{X})}. \tag{6.11}$$

3. As $P(\boldsymbol{X})$ is constant for all classes, only $P(\boldsymbol{X}|C_i)P(C_i)$ need be maximized. If the class prior probabilities are not known, then it is commonly assumed that the classes are

equally likely, that is, $P(C_1) = P(C_2) = \cdots = P(C_m)$, and we would therefore maximize $P(\boldsymbol{X}|C_i)$. Otherwise, we maximize $P(\boldsymbol{X}|C_i)P(C_i)$. Note that the class prior probabilities may be estimated by $P(C_i) = |C_{i,D}|/|D|$, where $|C_{i,D}|$ is the number of training tuples of class C_i in D.

4. In order to reduce computation in evaluating $P(\boldsymbol{X}|Ci)$, the naïve assumption of class conditional independence is made.

$$P(\boldsymbol{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i)$$
$$= P(x_1|C_i) \times P(x_2|C_i) \times \cdots \times P(x_n|C_i).$$

(a) If A_k is categorical, then $P(x_k|C_i)$ is the number of tuples of class C_i in D having the value x_k for A_k, divided by $|C_{i,D}|$, the number of tuples of class C_i in D.

(b) If A_k is continuous-valued, then we need to do a bit more work, but the calculation is pretty straightforward. A continuous-valued attribute is typically assumed to have a Gaussian distribution with a mean μ and standard deviation σ, defined by

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{6.13}$$

so that

$$P(x_k|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i}). \tag{6.14}$$

5. In order to predict the class label of $\boldsymbol{X}$, $P(\boldsymbol{X}|C_i)P(C_i)$ is evaluated for each class C_i. The classifier predicts that the class label of tuple $\boldsymbol{X}$ is the class C_i if and only if

$$P(\boldsymbol{X}|C_i)P(C_i) > P(\boldsymbol{X}|C_j)P(C_j) \quad \text{for } 1 \leq j \leq m, j \neq i. \tag{6.15}$$

In other words, the predicted class label is the class C_i for which $P(\boldsymbol{X}|C_i)P(C_i)$ is the maximum.

Predicting a class label using naïve Bayesian classification

- The data tuples are described by the attributes *age*, *income*, *student*, and *credit rating*.
- The class label attribute, *buys computer*, has two distinct values (namely, {*yes, no*}).
- Let *C*1 correspond to the class *buys computer = yes* and

  *C*2 correspond to *buys computer = no.*
- The tuple we wish to classify is
- ***X*** = (*age = youth, income = medium, student = yes, credit rating = fair*)
- We need to maximize $P(\boldsymbol{X}|Ci)P(Ci)$, for $i = 1, 2$. $P(Ci)$, the prior probability of each class, can be computed based on the training tuples

$$P(buys\_computer = yes) = 9/14 = 0.643$$
$$P(buys\_computer = no) = 5/14 = 0.357$$

To compute $PX|C_i)$, for $i = 1, 2$, we compute the following conditional probabilities:

$$P(age = youth \mid buys\_computer = yes) = 2/9 = 0.222$$
$$P(age = youth \mid buys\_computer = no) = 3/5 = 0.600$$
$$P(income = medium \mid buys\_computer = yes) = 4/9 = 0.444$$
$$P(income = medium \mid buys\_computer = no) = 2/5 = 0.400$$
$$P(student = yes \mid buys\_computer = yes) = 6/9 = 0.667$$
$$P(student = yes \mid buys\_computer = no) = 1/5 = 0.200$$
$$P(credit\_rating = fair \mid buys\_computer = yes) = 6/9 = 0.667$$
$$P(credit\_rating = fair \mid buys\_computer = no) = 2/5 = 0.400$$

Using the above probabilities, we obtain

$$P(\mathbf{X}|buys\_computer = yes) = P(age = youth \mid buys\_computer = yes) \times$$
$$P(income = medium \mid buys\_computer = yes) \times$$
$$P(student = yes \mid buys\_computer = yes) \times$$
$$P(credit\_rating = fair \mid buys\_computer = yes)$$
$$= 0.222 \times 0.444 \times 0.667 \times 0.667 = 0.044.$$

Similarly,

$$P(\mathbf{X}|buys\_computer = no) = 0.600 \times 0.400 \times 0.200 \times 0.400 = 0.019.$$

To find the class, C_i, that maximizes $P(\mathbf{X}|C_i)P(C_i)$, we compute

$$P(\mathbf{X}|buys\_computer = yes)P(buys\_computer = yes) = 0.044 \times 0.643 = 0.028$$
$$P(\mathbf{X}|buys\_computer = no)P(buys\_computer = no) = 0.019 \times 0.357 = 0.007$$

Therefore, the naïve Bayesian classifier predicts $buys\_computer = yes$ for tuple $\mathbf{X}$.

Using the Laplacian correction to avoid computing probability values of zero:

- Suppose that for the class ***buys computer = yes*** in some training database, *D*, containing **1,000 tuples**, we have **0 tuples** with ***income = low*****, 990 tuples with *income = medium*****, and 10 tuples with *income = high*****.**

- The probabilities of these events, **without** the **Laplacian correction**,

are 0, 0.990 (from 999/1000), and 0.010 (from 10/1,000), respectively.

- Using the Laplacian correction for the three quantities, we pretend that we have 1 more tuple for each income-value pair.

$$\frac{1}{1,003} = 0.001, \frac{991}{1,003} = 0.988, \text{ and } \frac{11}{1,003} = 0.011,$$

The “corrected” probability estimates are close to their “uncorrected” counterparts, yet the zero probability value is avoided.

Rule-Based Classification

Using IF-THEN Rules for Classification

- A rule-based classifier uses a set of IF-THEN rules for classification.
- An IF-THEN rule is an expression of the form

•IF *condition* THEN *conclusion*.

An example is rule *R*1,

- R1: IF *age* = *youth* AND *student* = *yes* THEN *buys computer* = *yes*.
- The "IF"-part (or left-hand side)of a rule - rule antecedent or precondition.
- The "THEN"-part (or right-hand side) of a rule - rule consequent.

- The rule antecedent, the condition consists of one or more *attribute tests* that are logically ANDed.
- The rule's consequent contains a class prediction.

R1: (*age* = *youth*) ^ (*student* = *yes*))(*buys computer* = *yes*).

- If the condition in a rule antecedent holds true for a given tuple, we say that the rule antecedent is satisfied and that the rule covers the tuple.
- A rule *R* can be assessed by its coverage and accuracy
- let n_{covers} be the number of tuples covered by *R.*
- $n_{correct}$ be the number of tuples correctly classified by *R.*
- $|D|$ be the number of tuples in *D.*
- **Rule's Coverage** - percentage of tuples that are covered by the rule.
- **Rule's Accuracy**- percentage of them the rulecan correctly classify.

Rule accuracy and coverage

$$coverage(R) = \frac{n_{covers}}{|D|}$$

$$accuracy(R) = \frac{n_{correct}}{n_{covers}}.$$

- The class-labeled tuples from the *AllElectronics* customer database is taken. Our task is to predict whether a customer will buy a computer.
- Consider rule *R*1 above, which covers 2 of the 14 tuples.
- It can correctly classify both tuples.
- Therefore,
- *coverage*(*R*1) = 2=14 = 14:28%
- *Accuracy* (*R*1) = 2=2 = 100%.

- If a rule is satisfied by ***X***, the rule is said to be triggered.
- If more than one rule is triggered, we need a **conflict resolution strategy** to figure out which rule gets to fire and assign its class prediction to ***X***.
- ***size ordering, rule ordering***
- **size ordering** - assigns the **highest priority** to the triggering rule that has the "**toughest**" requirements, where toughness is measured by the rule antecedent *size*.
- **Rule ordering** - prioritizes the rules beforehand. The ordering may be class based or rule-based.
- **class-based ordering** - classes are sorted in order of decreasing "importance," such as by decreasing order of prevalence.

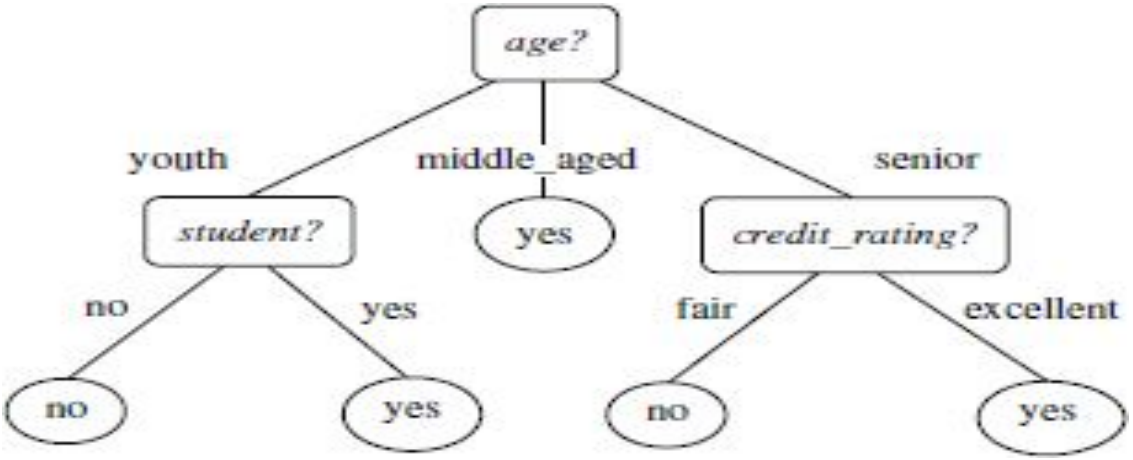
.

- **Rule-based ordering** - the rules are organized into one long priority list, according to some measure of rule quality such as accuracy, coverage, or size.
- When rule ordering is used, the rule set is known as a **decision list**.
- When there is **no rule satisfied** by X.
- A **fallback or default rule** can be set up to specify a default class, based on a training set.
- The **default rule is evaluated at the end**, if and only if no other rule covers X. The condition in the default rule is empty.

Rule Extraction from a Decision Tree

- Decision tree classifiers are a popular method of classification.
- They are known for their **accuracy**.
- Decision trees can become large and difficult to interpret.
- In comparison with a decision tree, the IF-THEN rules may be easier for humans to understand.
- To extract rules from a decision tree, **one rule** is created for **each path** from the root to a leaf node.
- Each **splitting criterion** along a given path is **logically ANDed** to form the rule antecedent.
- The **leaf node** holds the **class prediction**, forming the rule Consequent.

Extracting classification rules from a decision tree



R1: IF *age* = *youth* AND *student* = *no* THEN *buys_computer* = *no*
R2: IF *age* = *youth* AND *student* = *yes* THEN *buys_computer* = *yes*
R3: IF *age* = *middle_aged* THEN *buys_computer* = *yes*
R4: IF *age* = *senior* AND *credit_rating* = *excellent* THEN *buys_computer* = *yes*
R5: IF *age* = *senior* AND *credit_rating* = *fair* THEN *buys_computer* = *no*

- A **disjunction (logical OR)** is implied between each of the extracted rules.
- Because the rules are extracted directly from the tree, they are **mutually exclusive and exhaustive**.
- ***Mutually exclusive*** - cannot have rule conflicts here because no two rules will be triggered for the same tuple.
- ***Exhaustive*** - there is one rule for each possible attribute-value combination, so that this set of rules does not require a default rule.
- We may end up in **repetition** and **replication** of some rules.
- we may need to do some more work by **pruning** the resulting rule set.

Model Evaluation and Selection

SRM UNIVERSITY
(Under section 3 of UGC Act 1956)

- Data from previous sales to build a classifier to predict customer purchasing behavior.
- An estimation of how accurately the classifier can predict the purchasing behavior of future customers, that is, future customer data on which the classifier has not been trained.
- Different methods to build more than one classifier and now wish to compare their accuracy.

1. **Metrics for Evaluating Classifier Performance**
2. **Holdout Method and Random Subsampling**
3. **Cross-Validation**
4. **Bootstrap**
5. **Model Selection Using Statistical Tests of Significance**

Metrics for Evaluating classifier Performance

- It presents measures for assessing how good or how "accurate" your classifier is at predicting the class label of tuples
- Consider the case of where the class tuples are more or less evenly distributed, as well as the case where classes are unbalanced
  - e.g., where an important class of interest is rare such as in medical tests
- They include accuracy (also known as recognition rate), sensitivity (or recall), specificity, precision, *F1, and F.*
- *Note that although accuracy is a specific measure, the word "accuracy" is* also used as a general term to refer to a classifier's predictive abilities.
- Using training data to derive a classifier and then estimate the accuracy of the resulting learned model can result in misleading overoptimistic estimates due to overspecialization of the learning algorithm to the data.

Measure	Formula
accuracy, recognition rate	$\frac{TP+TN}{P+N}$
error rate, misclassification rate	$\frac{FP+FN}{P+N}$
sensitivity, true positive rate, recall	$\frac{TP}{P}$
specificity, true negative rate	$\frac{TN}{N}$
precision	$\frac{TP}{TP+FP}$
F, F_1, F-score, harmonic mean of precision and recall	$\frac{2 \times precision \times recall}{precision + recall}$
F_β, where β is a non-negative real number	$\frac{(1+\beta^2) \times precision \times recall}{\beta^2 \times precision + recall}$

Meaning of the various measures

- **True positives** (*TP*): These refer to the positive tuples that were correctly labeled by the classifier. Let *TP* be the number of true positives.
- **True negatives** (*TN*): These are the negative tuples that were correctly labeled by the classifier. Let *TN* be the number of true negatives.
- **False positives** (*FP*): These are the negative tuples that were incorrectly labeled as positive (e.g., tuples of class *buys_computer* = *no* for which the classifier predicted *buys_computer* = *yes*). Let *FP* be the number of false positives.
- **False negatives** (*FN*): These are the positive tuples that were mislabeled as negative (e.g., tuples of class *buys_computer* = *yes* for which the classifier predicted *buys_computer* = *no*). Let *FN* be the number of false negatives.

Confusion Matrix

- The confusion matrix is a useful tool for analyzing how well your classifier can recognize tuples of different classes
- *TP and TN tell us when the classifier is getting* things right, while *FP and FN tell us when the classifier is getting things wrong*

		Predicted class		
		yes	*no*	Total
Actual class	*yes*	*TP*	*FN*	*P*
	no	*FP*	*TN*	*N*
	Total	*P′*	*N′*	*P* + *N*

Confusion matrix for the classes *buys computer D yes and buys computer D no, where an* entry in row *i and column j shows the number of tuples of class i that were labeled by the* classifier as class *j. Ideally, the nondiagonal entries should be zero or close to zero.*

Classes	*buys_computer = yes*	*buys_computer = no*	*Total*	*Recognition (%)*
buys_computer = yes	**6954**	**46**	7000	99.34
buys_computer = no	**412**	**2588**	3000	86.27
Total	7366	2634	10,000	95.42

We can also speak of the **error rate** or **misclassification rate** of a classifier, M, which is simply $1 - accuracy(M)$, where $accuracy(M)$ is the accuracy of M. This also can be computed as

$$error\ rate = \frac{FP + FN}{P + N}.$$

• The sensitivity and specificity measures can be used, respectively, for this purpose.

•Sensitivity is also referred to as the *true positive (recognition) rate (i.e., the proportion* of positive tuples that are correctly identified)

•Specificity is the *true negative rate* (i.e., the proportion of negative tuples that are correctly identified).

•These measures are

$$sensitivity = \frac{TP}{P}$$

$$specificity = \frac{TN}{N}.$$

It can be shown that accuracy is a function of sensitivity and specificity:

$$accuracy = sensitivity\frac{P}{(P+N)} + specificity\frac{N}{(P+N)}.$$

Example

•confusion matrix for medical data where the class values are *yes and no for a class label attribute, cancer.*

Classes	*yes*	*no*	*Total*	*Recognition (%)*
yes	**90**	**210**	300	30.00
no	**140**	**9560**	9700	98.56
Total	230	9770	10,000	96.40

Confusion matrix for the classes *cancer* = *yes* and *cancer* = *no*.

the classifier is $\frac{90}{300} = 30.00\%$. The specificity is $\frac{9560}{9700} = 98.56\%$. The classifier's overall accuracy is $\frac{9650}{10{,}000} = 96.50\%$. Thus, we note that although the classifier has a high accuracy, it's ability to correctly label the positive (rare) class is poor given its low sensitivity.

Precision and Recall

- The *precision and recall measures are also widely used in classification.*
- ***Precision* –** can be thought of as a measure of *exactness (i.e., what percentage of tuples labeled as* positive are actually such),
- **Recall is a measure of *completeness*** *(what percentage* of positive tuples are labeled as such).
- If recall seems familiar, that's because it is the same as sensitivity (or the *true positive rate).*
- *These measures can be computed as*

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN} = \frac{TP}{P}.$$

Precision and recall. The precision of the classifier for the *yes* class is $\frac{90}{230} = 39.13\%$. The recall is $\frac{90}{300} = 30.00\%$, which is the same calculation for sensitivity

F measure

- An alternative way to use precision and recall is to combine them into a single measure.
- This is the approach of the *F measure (also known as the F1 score or F-score) and* the *F measure.*
- *They are defined as*

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

$$F_\beta = \frac{(1+\beta^2) \times precision \times recall}{\beta^2 \times precision + recall},$$

- where is a non-negative real number.

- The *F measure is the harmonic mean of precision* and recall.
- It gives equal weight to precision and recall.
- The *F measure is a weighted measure of precision and recall.*
- *It assigns times* as much weight to recall as to precision.
- Commonly used *F measures are F2 (which* weights recall twice as much as precision) and *F0.5 (which weights precision twice as* much as recall).

In addition to accuracy-based measures, classifiers can also be compared with respect to the following additional aspects:

1. Speed: This refers to the computational costs involved in generating and using the given classifier.

2. Robustness: This is the ability of the classifier to make correct predictions given noisy data or data with missing values. Robustness is typically assessed with a series of synthetic data sets representing increasing degrees of noise and missing values.

3. Scalability: This refers to the ability to construct the classifier efficiently given large amounts of data. Scalability is typically assessed with a series of data sets of increasing size.

4. Interpretability: This refers to the level of understanding and insight that is provided by the classifier or predictor. Interpretability is subjective and therefore more difficult to assess.

Cross Validation

- In ***k-fold cross-validation, the initial data are randomly partitioned into k mutually*** exclusive subsets or "folds," D_1, D_2, ….., D_k, *each of approximately equal size.*
- *Training* and testing is performed *k times.*
- *In iteration i, partition Di is reserved as the test set,* and the remaining partitions are collectively used to train the model.
- That is, in the first iteration, subsets D_2,…, D_k *collectively serve as the training set to obtain a first* model, which is tested on D_1;
- *The second iteration is trained on subsets* D_1, *D3,* …., D_k and tested on D_2, *and so on.*
- *Unlike the holdout and random subsampling methods,* here each sample is used the same number of times for training and once for testing.
- For classification, the accuracy estimate is the overall number of correct classifications from the *k iterations, divided by the total number of tuples in the initial data.*

Cross Validation

- **Leave-one-out is a special case of *k-fold cross-validation where k is set to the number*** of initial tuples.
- That is, only one sample is “left out” at a time for the test set
- In **stratified cross-validation, the folds are stratified so that the class distribution of the tuples** in each fold is approximately the same as that in the initial data.
- In general, stratified 10-fold cross-validation is recommended for estimating accuracy (even if computation power allows using more folds) due to its relatively low bias and variance.

Bootstrap

- The **bootstrap method samples** the given training tuples uniformly *with replacement.*
- *Each time a tuple is* selected, it is equally likely to be selected again and re-added to the training set.
- For instance, imagine a machine that randomly selects tuples for our training set.
- In *sampling with replacement, the machine is allowed to select the same tuple more than once.*
- A commonly used one is the **.632 bootstrap**
- Repeat the sampling procedure *k times, where in each iteration, use the* current test set to obtain an accuracy estimate of the model obtained from the current bootstrap sample.
- The overall accuracy of the model, *M, is then estimated as*

$$Acc(M) = \frac{1}{k}\sum_{i=1}^{k}(0.632 \times Acc(M_i)_{test\_set} + 0.368 \times Acc(M_i)_{train\_set}),$$

- where *Acc(M0 test set is the accuracy of the model obtained with bootstrap sample i when* it is applied to test set *i.*

Techniques to Improve Classification Accuracy

1. **Introducing Ensemble Methods**
2. **Bagging**
3. **Boosting and AdaBoost**
4. **Random Forests**

•Ensemble for classification is a composite model, made up of a combination of classifiers.

•The individual classifiers vote, and a class label prediction is returned by the ensemble based on the collection of votes.

•Ensembles tend to be more accurate than their component classifiers.

Ensemble methods Introduction

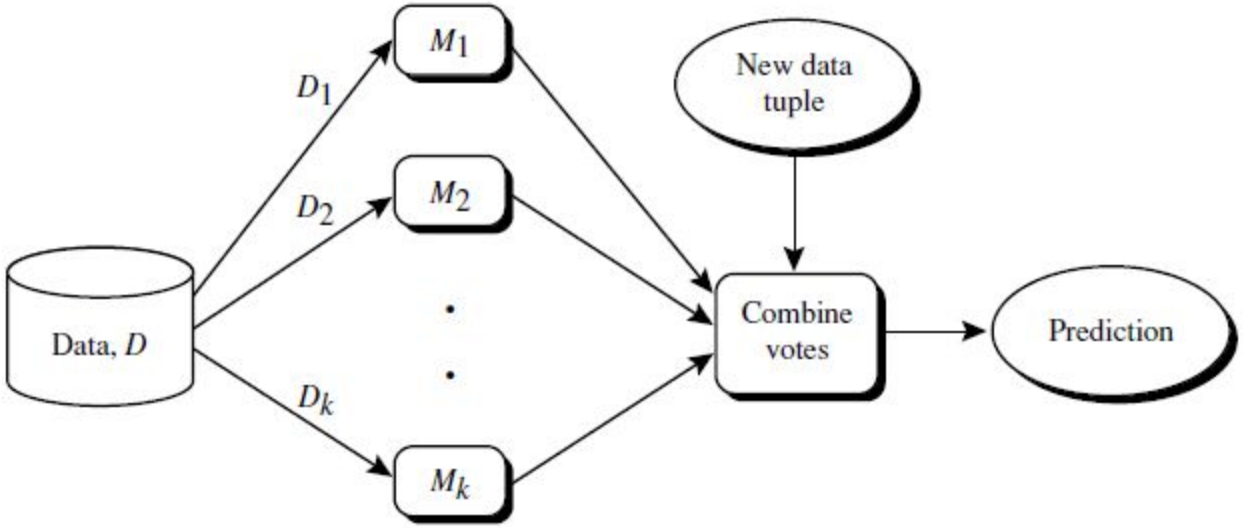
• *Bagging, boosting, and random forests are examples of* ***ensemble methods***

•An ensemble combines a series of *k learned models (or base classifiers),M1, M2, ... , Mk,* with the aim of creating an improved composite classification model, *M*.*

• *A given data* set, *D, is used to create k training sets, D1, D2,.... , Dk, where Di (1 ≤i≤ k-1/)is used* to generate classifier *Mi .*

• *Given a new data tuple to classify, the base classifiers each vote* by returning a class prediction.

•The ensemble returns a class prediction based on the votes of the base classifiers.



- An ensemble tends to be more accurate than its base classifiers.
- For example, consider an ensemble that performs majority voting.
- That is, given a tuple ***X to classify, it*** collects the class label predictions returned fromthe base classifiers and outputs the class in majority.
- The base classifiers may make mistakes, but the ensemble will misclassify ***X*** only if over half of the base classifiers are in error.
- Ensembles yield better results when there is significant diversity among the models.
- That is, ideally, there is little correlation among classifiers. The classifiers should also perform better than random guessing.
- Each base classifier can be allocated to a different CPU and so ensemble methods are parallelizable.

Bagging

- Given a set, *D, of d tuples,* ***bagging works as follows.***
- ***For iteration i (i = 1, 2, … , k)*** a training set, *Di , of d tuples is sampled with replacement from the original set of* tuples, *D.*
- *The term bagging stands for bootstrap aggregation.*
- *Each training* set is a bootstrap sample and sampling with replacement is used, some of the original tuples of *D may not be included in Di , whereas others* may occur more than once.
- A classifier model, *Mi , is learned for each training set, Di .*
- To classify an unknown tuple, ***X, each classifier, Mi , returns its class prediction, which*** counts as one vote.
- The bagged classifier, *M, counts the votes and assigns the class* with the most votes to ***X.***
- ***Bagging can be applied to the prediction of continuous values*** by taking the average value of each prediction for a given test tuple.

Algorithm: Bagging. The bagging algorithm—create an ensemble of classification models for a learning scheme where each model gives an equally weighted prediction.

Input:

- D, a set of d training tuples;
- k, the number of models in the ensemble;
- a classification learning scheme (decision tree algorithm, naïve Bayesian, etc.).

Output: The ensemble—a composite model, $M*$.

Method:

(1) **for** $i = 1$ to k **do** // create k models:
(2) create bootstrap sample, D_i, by sampling D with replacement;
(3) use D_i and the learning scheme to derive a model, M_i;
(4) **endfor**

To use the ensemble to classify a tuple, X:

let each of the k models classify X and return the majority vote;

BOOSTING

- A patient have certain symptoms.
- Instead of consulting one doctor, assign weights to the value or worth of each doctor's diagnosis, based on the accuracies of previous diagnoses they have made.
- The final diagnosis is then a combination of the weighted diagnoses.
- This is the essence behind boosting.
- In **boosting, weights are also assigned to each training tuple.**
- **A series of *k classifiers is*** iteratively learned.
- *The final boosted classifier, combines the votes of each individual*

classifier, where the weight of each classifier's vote is a function of its accuracy.

BOOSTING

Algorithm: AdaBoost. A boosting algorithm—create an ensemble of classifiers. Each one gives a weighted vote.

Input:

- D, a set of d class-labeled training tuples;
- k, the number of rounds (one classifier is generated per round);
- a classification learning scheme.

Output: A composite model.

Method:

(1) initialize the weight of each tuple in D to $1/d$;
(2) **for** $i = 1$ to k **do** // for each round:
(3) sample D with replacement according to the tuple weights to obtain D_i;
(4) use training set D_i to derive a model, M_i;
(5) compute $error(M_i)$, the error rate of M_i (Eq. 8.34)
(6) **if** $error(M_i) > 0.5$ **then**
(7) go back to step 3 and try again;
(8) **endif**
(9) **for** each tuple in D_i that was correctly classified **do**
(10) multiply the weight of the tuple by $error(M_i)/(1 - error(M_i))$; // update weights
(11) normalize the weight of each tuple;
(12) **endfor**

To use the ensemble to classify tuple, X:

(1) initialize weight of each class to 0;
(2) **for** $i = 1$ to k **do** // for each classifier:
(3) $w_i = \log \frac{1 - error(M_i)}{error(M_i)}$; // weight of the classifier's vote
(4) $c = M_i(X)$; // get class prediction for X from M_i
(5) add w_i to weight for class c
(6) **endfor**
(7) return the class with the largest weight;

Random Forests Introduction

•Given two-class data, the data are class-imbalanced if the main class of interest (the positive class) is represented by only a few tuples, while the majority of tuples represent the negative class.

•The class imbalance problem is closely related to cost-sensitive learning, wherein the costs of errors, per class, are not equal.

• In medical diagnosis, for example, it is much more costly to falsely diagnose a cancerous patient as healthy (a false negative) than to misdiagnose a healthy patient as having cancer (a false positive).

•A false negative error could lead to the loss of life and therefore is much more expensive than a false positive error.

•Other applications involving class-imbalanced data include fraud detection, the detection of oil spills from satellite radar images, and fault monitoring.

•Traditional classification algorithms aim to minimize the number of errors made during classification.

•They assume that the costs of false positive and false negative errors are equal.

•By assuming a balanced distribution of classes and equal error costs, they are therefore not suitable for class-imbalanced data.

•Although the accuracy measure assumes that the cost of classes are equal, alternative evaluation metrics can be used that consider the different types of classifications.

•For example, presented *sensitivity* or recall (the true positive rate) and *specificity (the true negative rate), which help* to assess how well a classifier can predict the class label of imbalanced data.

Approaches for *improving the classification accuracy* of class-imbalanced data:

1) oversampling,
2) undersampling,
3) threshold moving, and
4) ensemble techniques.

- The first three do not involve any changes to the construction of the classification model.
- The oversampling and undersampling change the distribution of tuples in the training set
- Threshold moving affects how the model makes decisions when classifying new data.

•Both oversampling and undersampling change the training data distribution so that the rare (positive) class is well represented.

•**Oversampling works by resampling the positive** tuples so that the resulting training set contains an equal number of positive and negative tuples.

•**Undersampling works by decreasing the number of negative tuples.**

• **It** randomly eliminates tuples from the majority (negative) class until there are an equal number of positive and negative tuples.