



**ADITYA**  
ENGINEERING COLLEGE (A)

## UNIT III

# Classification: Basic Concepts and Techniques

# Classification vs. Prediction

- **Classification:**
  - predicts categorical class labels (discrete or nominal)
  - classifies data (constructs a model) based on the training set and the values (**class labels**) in a classifying attribute and uses it in classifying new data
- **Prediction:**
  - models continuous-valued functions, i.e., predicts unknown or missing values
- **Typical Applications**
  - credit approval
  - target marketing
  - medical diagnosis
  - treatment effectiveness analysis



# Classification: Definition

- Given a collection of records (*training set*)
  - Each record contains a set of *attributes*, one of the attributes is the *class*.
- Find a *model* for class attribute as a function of the values of other attributes.

# Classification

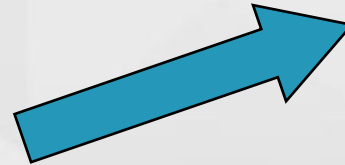
- Goal: previously unseen records should be assigned a class as accurately as possible.
  - A *test set* is used to determine the accuracy of the model. Usually, the given data set is divided into training and test sets, with training set used to build the model and test set used to validate it.



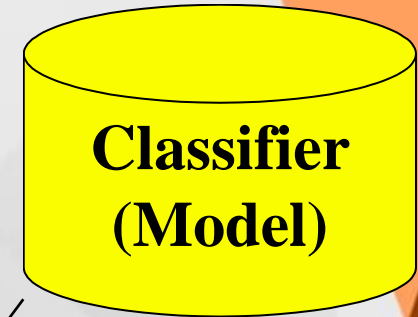
# Classification—A Two-Step Process

- **Model construction:** describing a set of predetermined classes
  - Each tuple/sample is assumed to belong to a predefined class, as determined by the **class label attribute**
  - The set of tuples used for model construction is **training set**
  - The model is represented as classification rules, decision trees, or mathematical formulae
- **Model usage:** for classifying future or unknown objects
  - Estimate accuracy of the model
    - The known label of test sample is compared with the classified result from the model
    - Accuracy rate is the percentage of test set samples that are correctly classified by the model
    - Test set is independent of training set, otherwise over-fitting will occur
  - If the accuracy is acceptable, use the model to classify data tuples whose class labels are not known

# Classification Process (1): Model Construction



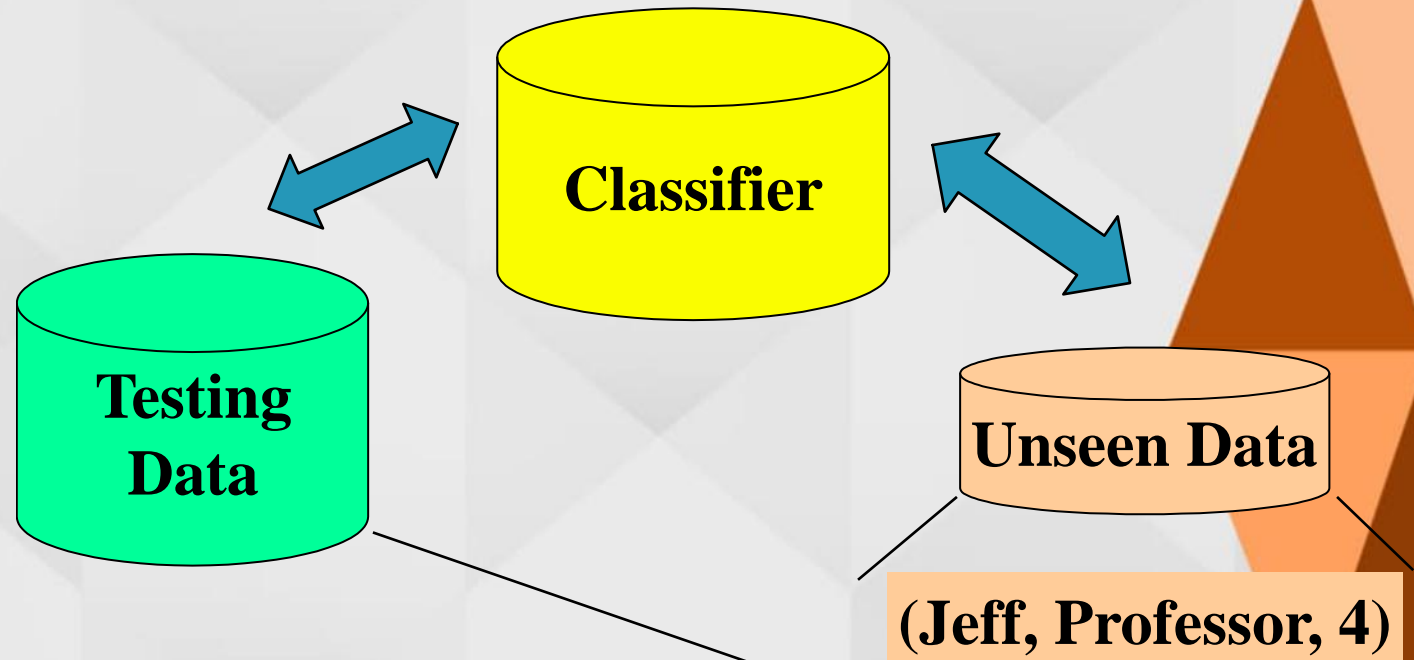
**Classification Algorithms**



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

**IF rank = 'professor'  
OR years > 6  
THEN tenured = 'yes'**

## Classification Process (2): Use the Model in Prediction



| NAME    | RANK           | YEARS | TENURED |
|---------|----------------|-------|---------|
| Tom     | Assistant Prof | 2     | no      |
| Merlisa | Associate Prof | 3     | no      |
| George  | Professor      | 5     | yes     |
| Joseph  | Assistant Prof | 7     | yes     |

Tenured? ↓  
**Yes**



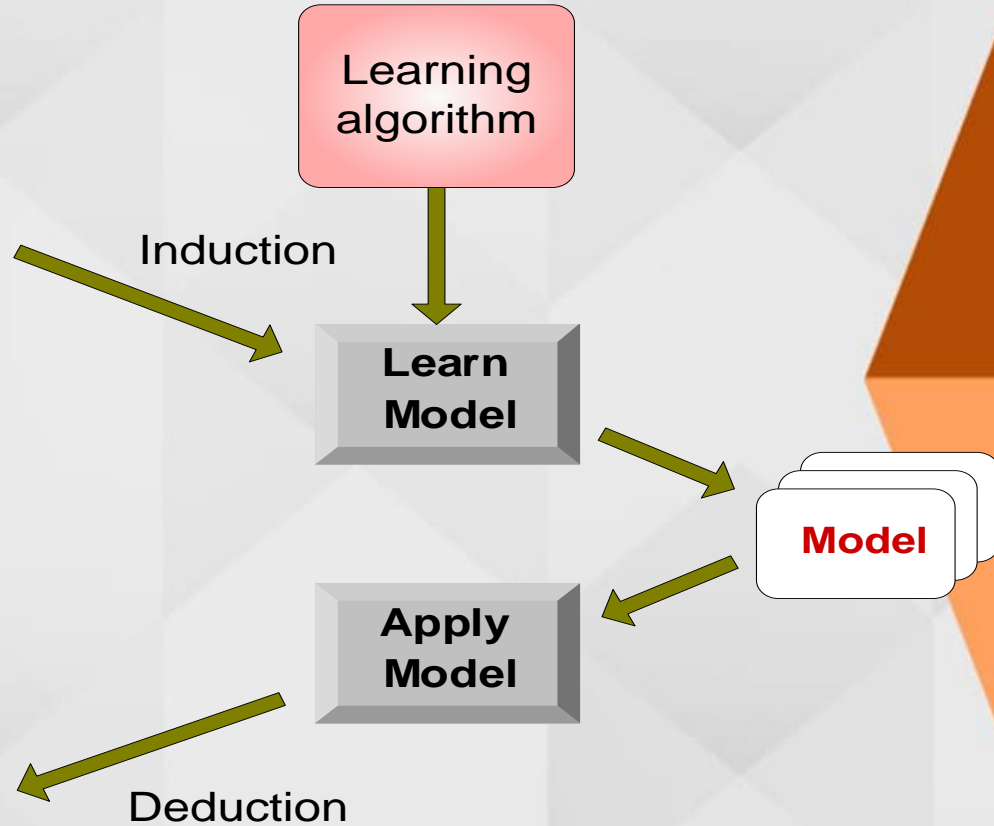
# Classification Task

| Tid | Attrib1 | Attrib2 | Attrib3 | Class |
|-----|---------|---------|---------|-------|
| 1   | Yes     | Large   | 125K    | No    |
| 2   | No      | Medium  | 100K    | No    |
| 3   | No      | Small   | 70K     | No    |
| 4   | Yes     | Medium  | 120K    | No    |
| 5   | No      | Large   | 95K     | Yes   |
| 6   | No      | Medium  | 60K     | No    |
| 7   | Yes     | Large   | 220K    | No    |
| 8   | No      | Small   | 85K     | Yes   |
| 9   | No      | Medium  | 75K     | No    |
| 10  | No      | Small   | 90K     | Yes   |

Training Set

| Tid | Attrib1 | Attrib2 | Attrib3 | Class |
|-----|---------|---------|---------|-------|
| 11  | No      | Small   | 55K     | ?     |
| 12  | Yes     | Medium  | 80K     | ?     |
| 13  | Yes     | Large   | 110K    | ?     |
| 14  | No      | Small   | 95K     | ?     |
| 15  | No      | Large   | 67K     | ?     |

Test Set



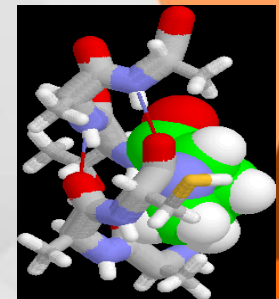


# Supervised vs. Unsupervised Learning

- **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

# Examples of Classification Task

- Classifying credit card transactions as legitimate or fraudulent
- Classifying secondary structures of protein as alpha-helix, beta-sheet, or random coil
- Categorizing news stories as finance, weather, entertainment, sports, etc





# Issues Regarding Classification and Prediction (1): Data Preparation

- Data cleaning
  - Preprocess data in order to reduce noise and handle missing values
- Relevance analysis (feature selection)
  - Remove the irrelevant or redundant attributes
- Data transformation
  - Generalize and/or normalize data



## Issues regarding classification and prediction (2): Evaluating Classification Methods

- Predictive accuracy
- Speed and scalability
  - time to construct the model
  - time to use the model
- Robustness
  - handling noise and missing values
- Scalability
  - efficiency in disk-resident databases
- Interpretability:
  - understanding and insight provided by the model
- Goodness of rules
  - decision tree size
  - compactness of classification rules



## Classification Techniques

- Base Classifiers
  - Decision Tree based Methods
  - Nearest-neighbor
  - Naïve Bayes and Bayesian Belief Networks
  - Support Vector Machines
  - Neural Networks



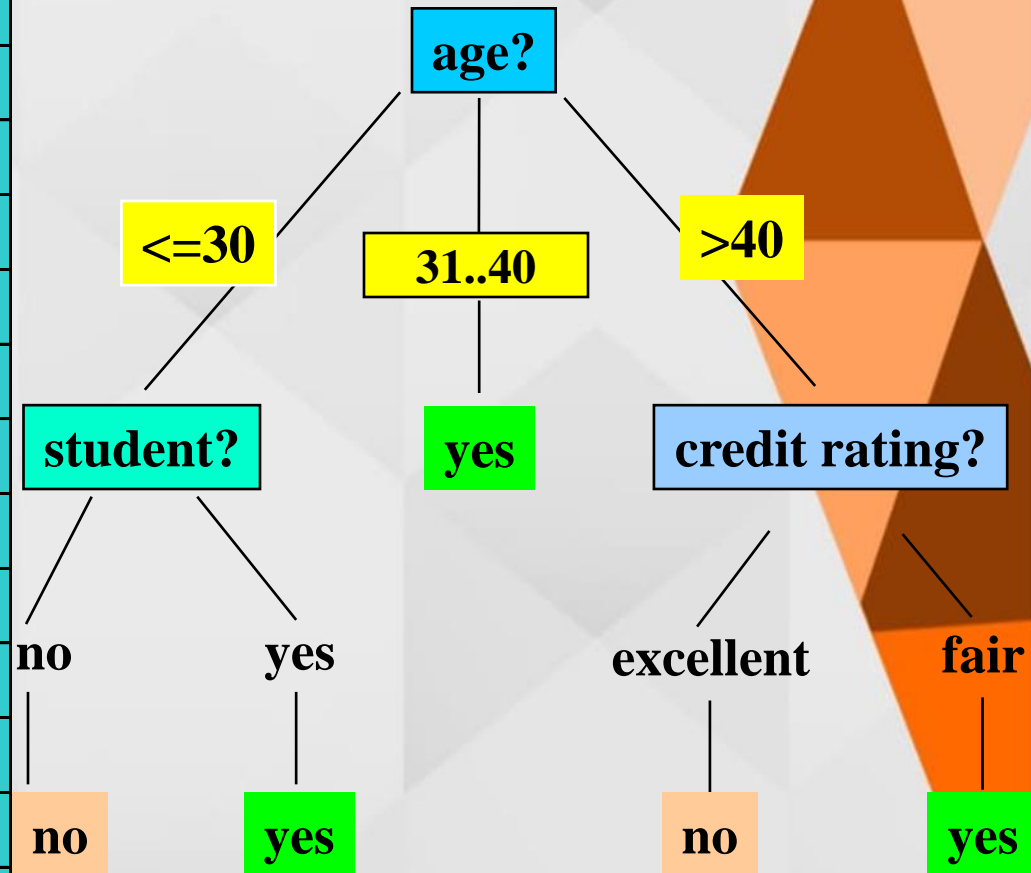
# Classification by Decision Tree Induction

- A **decision tree** is a flowchart-like tree structure, where
  - Each internal node (non-leaf node) denotes a test on an attribute
  - Each branch represents an outcome of the test
  - Each leaf node (or terminal node) holds a class label
- Can be represented by logical formulas

## Training Dataset

| 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            |

Output: A Decision Tree for  
*"buys\_computer"*



# Extracting Classification Rules from Trees

- Represent the knowledge in the form of **IF-THEN** rules
- 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 node holds the class prediction
- Rules are easier for humans to understand
- Example

IF *age* = " $\leq 30$ " AND *student* = "*no*" THEN *buys\_computer* = "*no*"

IF *age* = " $\leq 30$ " AND *student* = "*yes*" THEN *buys\_computer* = "*yes*"

IF *age* = " $31 \dots 40$ " THEN *buys\_computer* = "*yes*"

IF *age* = " $> 40$ " AND *credit\_rating* = "*excellent*" THEN *buys\_computer* = "*yes*"

IF *age* = " $\leq 30$ " AND *credit\_rating* = "*fair*" THEN *buys\_computer* = "*no*"

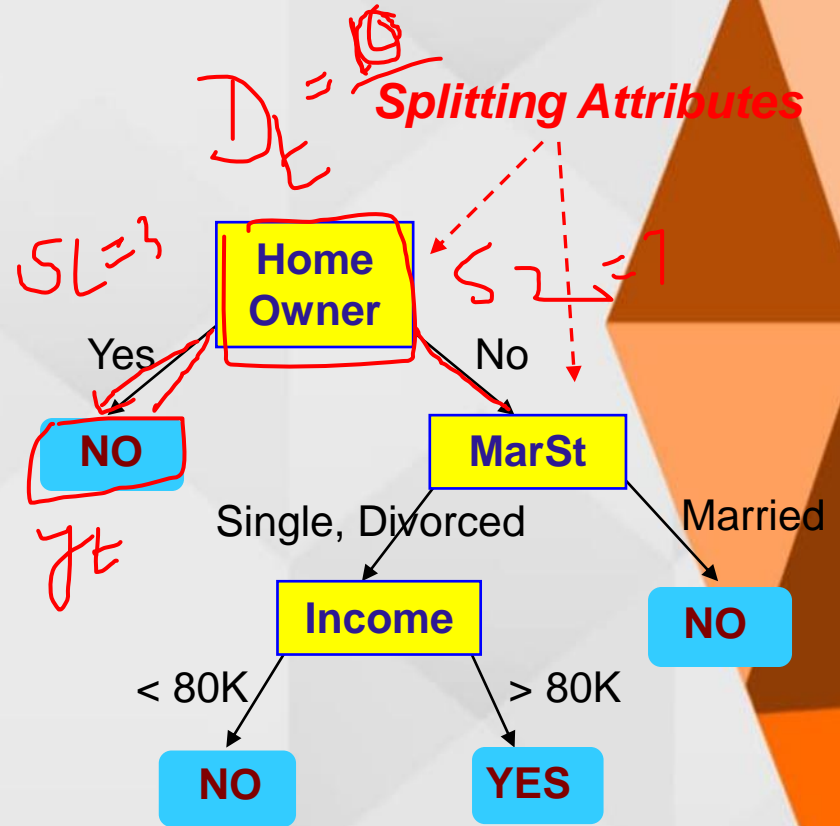


# Example of a Decision Tree

categorical  
categorical  
continuous  
class

| ID | Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|----|------------|----------------|---------------|--------------------|
| 1  | Yes        | Single         | 125K          | No ✓               |
| 2  | No         | Married        | 100K          | No                 |
| 3  | No         | Single         | 70K           | No                 |
| 4  | Yes        | Married        | 120K          | No ✓               |
| 5  | No         | Divorced       | 95K           | Yes                |
| 6  | No         | Married        | 60K           | No                 |
| 7  | Yes        | Divorced       | 220K          | No ✓               |
| 8  | No         | Single         | 85K           | Yes                |
| 9  | No         | Married        | 75K           | No                 |
| 10 | No         | Single         | 90K           | Yes                |

Training Data

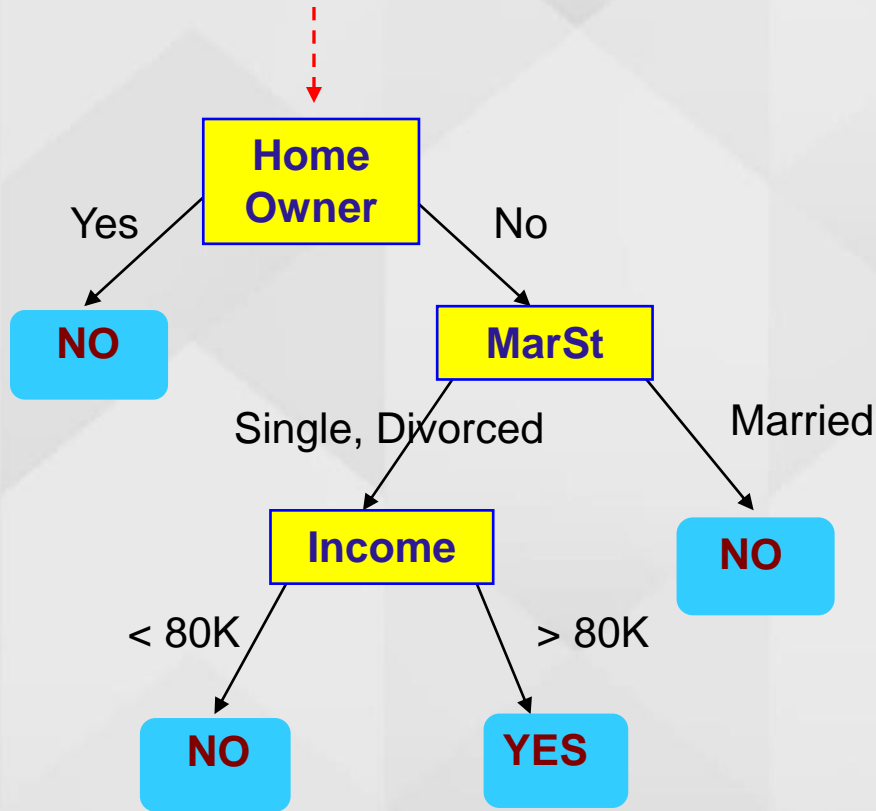


Model: Decision Tree

# Apply Model to Test Data

Test Data

Start from the root of tree.

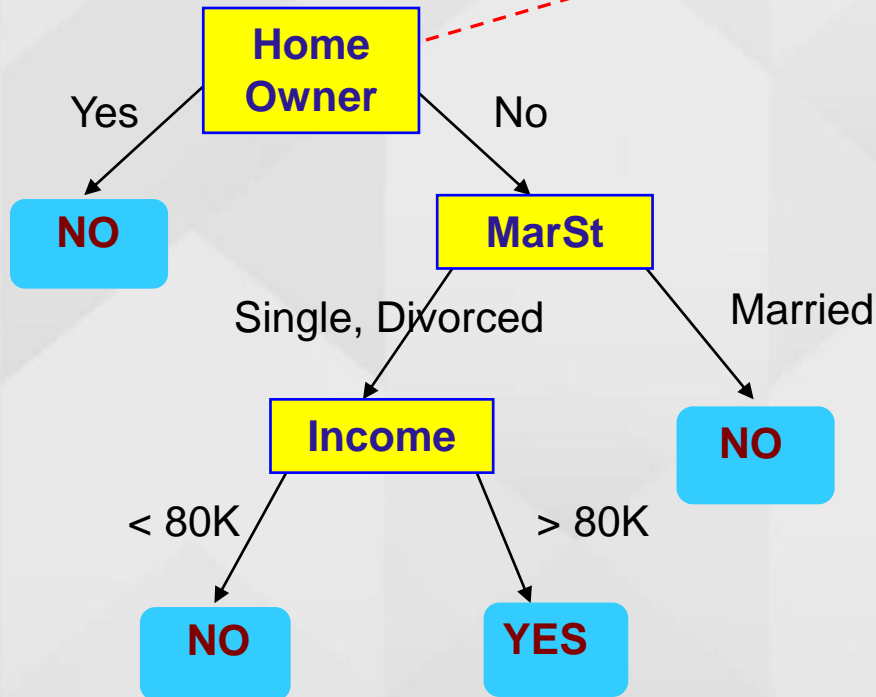


| Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|------------|----------------|---------------|--------------------|
| No         | Married        | 80K           | ?                  |

# Apply Model to Test Data

Test Data

| Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|------------|----------------|---------------|--------------------|
| No         | Married        | 80K           | ?                  |

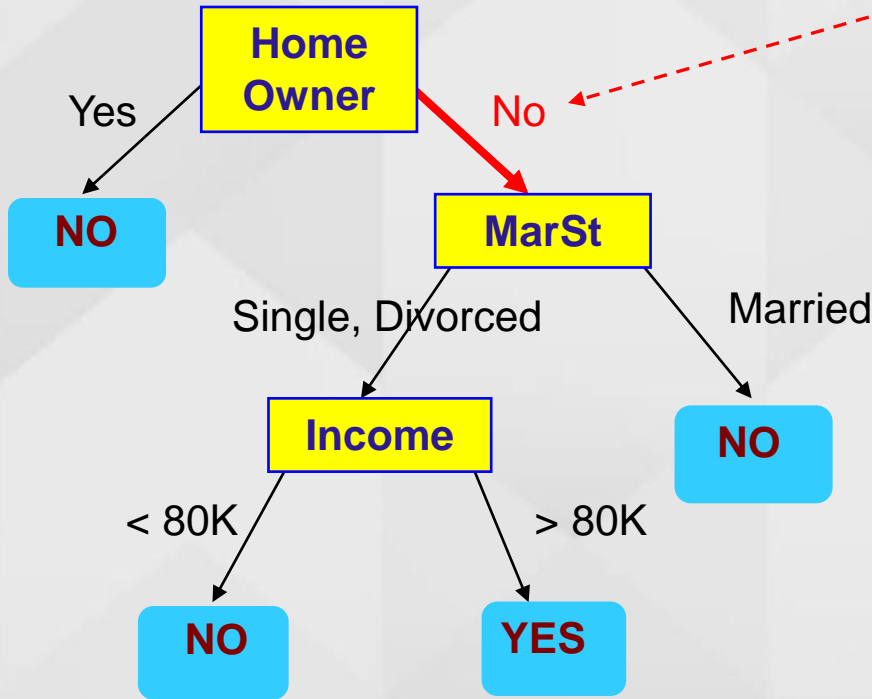




# Apply Model to Test Data

## Test Data

| Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|------------|----------------|---------------|--------------------|
| No         | Married        | 80K           | ?                  |

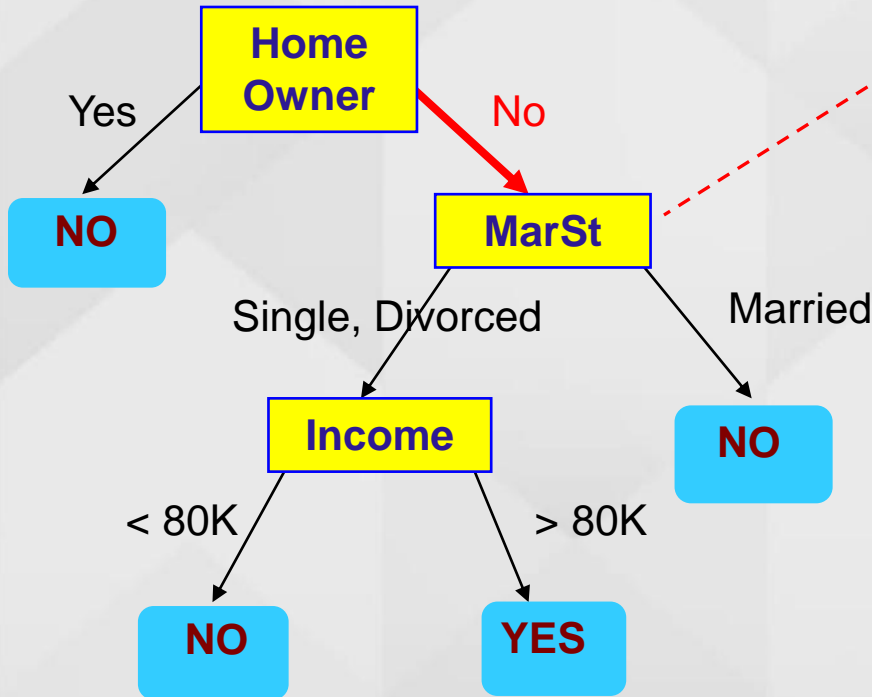




# Apply Model to Test Data

## Test Data

| Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|------------|----------------|---------------|--------------------|
| No         | Married        | 80K           | ?                  |

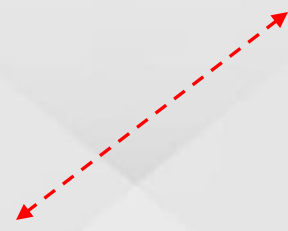
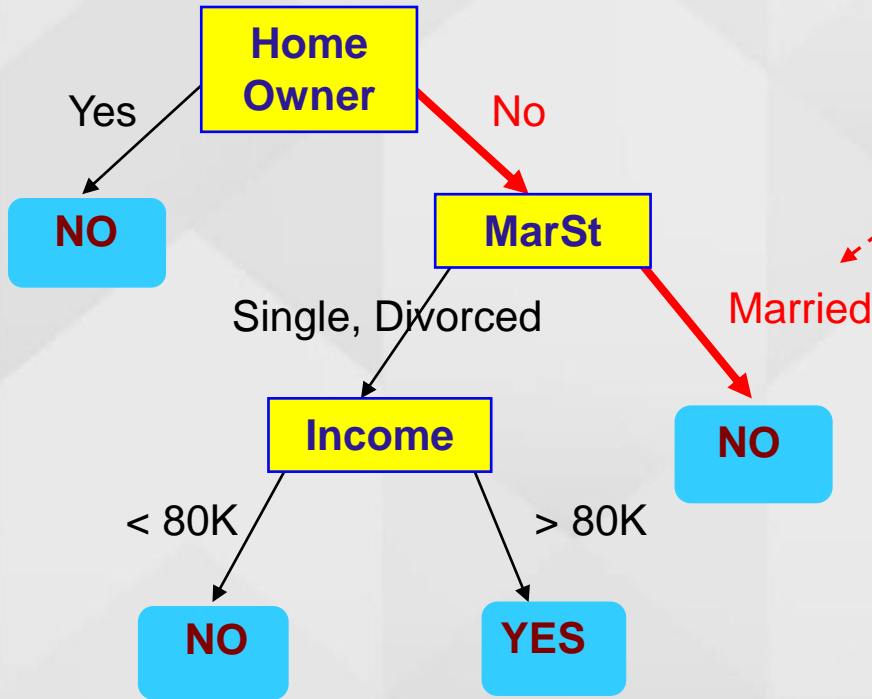




# Apply Model to Test Data

## Test Data

| Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|------------|----------------|---------------|--------------------|
| No         | Married        | 80K           | ?                  |

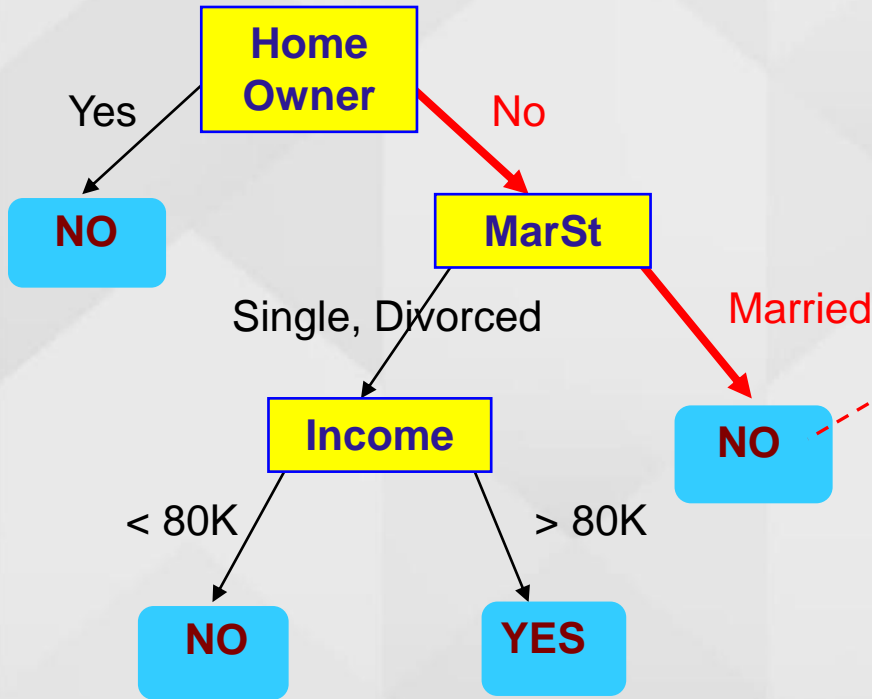




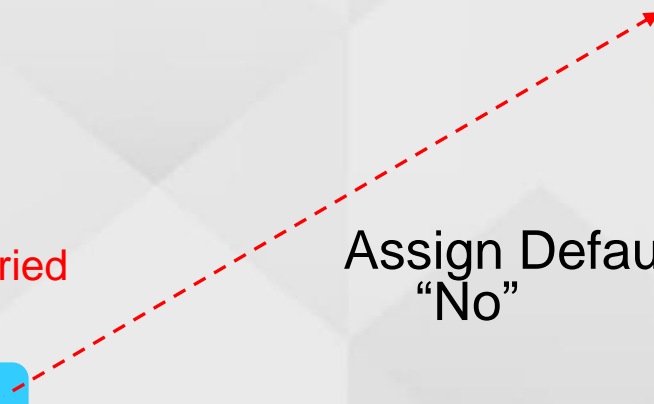
# Apply Model to Test Data

## Test Data

| Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|------------|----------------|---------------|--------------------|
| No         | Married        | 80K           | ?                  |



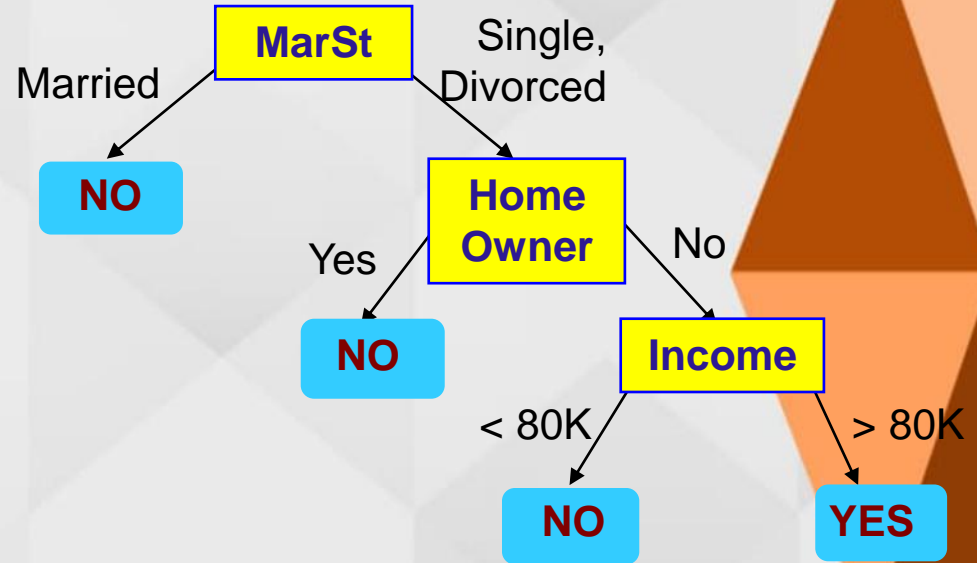
Assign Defaulted to "No"



# Another Example of Decision Tree

*categorical*  
*categorical*  
*continuous*  
*class*

| ID | Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|----|------------|----------------|---------------|--------------------|
| 1  | Yes        | Single         | 125K          | No                 |
| 2  | No         | Married        | 100K          | No                 |
| 3  | No         | Single         | 70K           | No                 |
| 4  | Yes        | Married        | 120K          | No                 |
| 5  | No         | Divorced       | 95K           | Yes                |
| 6  | No         | Married        | 60K           | No                 |
| 7  | Yes        | Divorced       | 220K          | No                 |
| 8  | No         | Single         | 85K           | Yes                |
| 9  | No         | Married        | 75K           | No                 |
| 10 | No         | Single         | 90K           | Yes                |



**There could be more than one tree that fits the same data!**





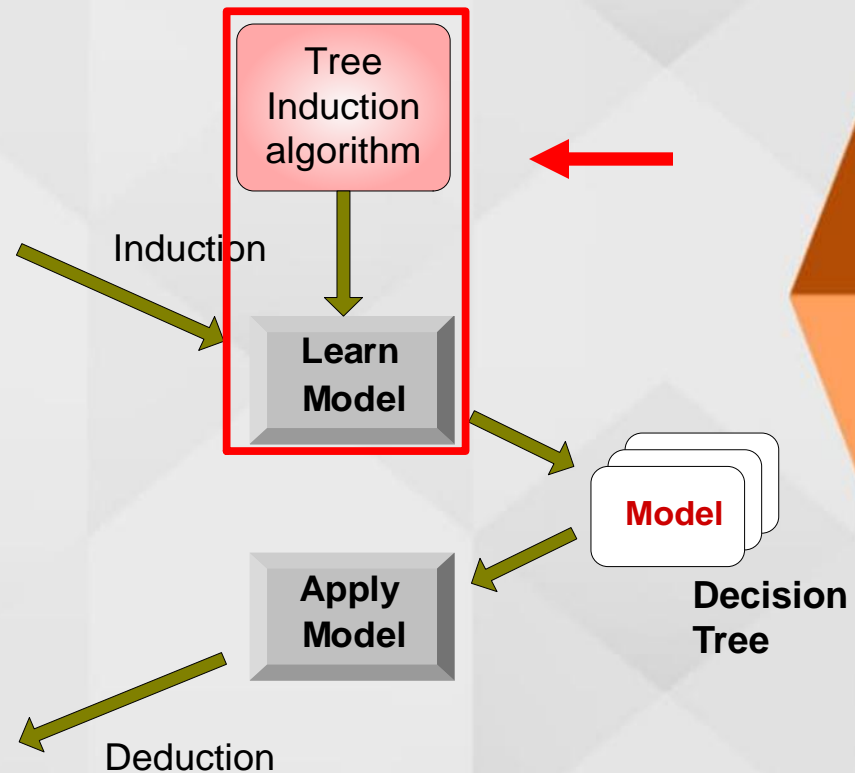
# Decision Tree Classification Task

| Tid | Attrib1 | Attrib2 | Attrib3 | Class |
|-----|---------|---------|---------|-------|
| 1   | Yes     | Large   | 125K    | No    |
| 2   | No      | Medium  | 100K    | No    |
| 3   | No      | Small   | 70K     | No    |
| 4   | Yes     | Medium  | 120K    | No    |
| 5   | No      | Large   | 95K     | Yes   |
| 6   | No      | Medium  | 60K     | No    |
| 7   | Yes     | Large   | 220K    | No    |
| 8   | No      | Small   | 85K     | Yes   |
| 9   | No      | Medium  | 75K     | No    |
| 10  | No      | Small   | 90K     | Yes   |

Training Set

| Tid | Attrib1 | Attrib2 | Attrib3 | Class |
|-----|---------|---------|---------|-------|
| 11  | No      | Small   | 55K     | ?     |
| 12  | Yes     | Medium  | 80K     | ?     |
| 13  | Yes     | Large   | 110K    | ?     |
| 14  | No      | Small   | 95K     | ?     |
| 15  | No      | Large   | 67K     | ?     |

Test Set





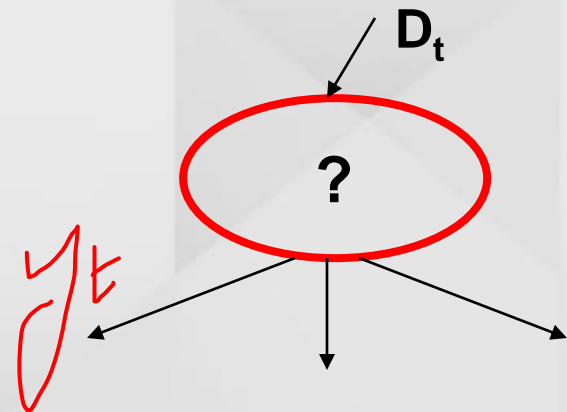
# Decision Tree Induction

- Many Algorithms:
  - Hunt's Algorithm (one of the earliest)
  - CART
  - ID3, C4.5
  - SLIQ, SPRINT

# General Structure of Hunt's Algorithm

- Let  $D_t$  be the set of training records that reach a node  $t$
- General Procedure:
  - If  $D_t$  contains records that belong to the same class  $y_t$ , then  $t$  is a leaf node labeled as  $y_t$
  - If  $D_t$  contains records that belong to more than one class, use an attribute test to split the data into smaller subsets. Recursively apply the procedure to each subset.

| ID | Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|----|------------|----------------|---------------|--------------------|
| 1  | Yes        | Single         | 125K          | No ✓               |
| 2  | No         | Married        | 100K          | No ✓               |
| 3  | No         | Single         | 70K           | No ✓               |
| 4  | Yes        | Married        | 120K          | No ✓               |
| 5  | No         | Divorced       | 95K           | Yes ✓              |
| 6  | No         | Married        | 60K           | No ✓               |
| 7  | Yes        | Divorced       | 220K          | No ✓               |
| 8  | No         | Single         | 85K           | Yes ✓              |
| 9  | No         | Married        | 75K           | No ✓               |
| 10 | No         | Single         | 90K           | Yes ✓              |



# Hunt's Algorithm

Defaulted = No

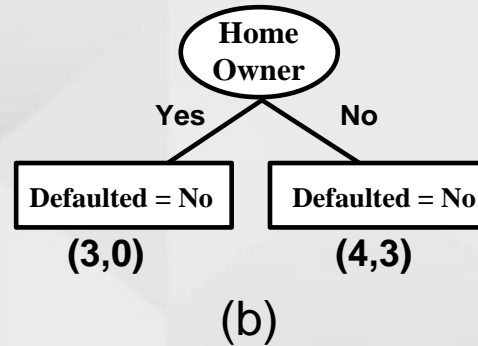
(7,3)

(a)

| ID | Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|----|------------|----------------|---------------|--------------------|
| 1  | Yes        | Single         | 125K          | No                 |
| 2  | No         | Married        | 100K          | No                 |
| 3  | No         | Single         | 70K           | No                 |
| 4  | Yes        | Married        | 120K          | No                 |
| 5  | No         | Divorced       | 95K           | Yes                |
| 6  | No         | Married        | 60K           | No                 |
| 7  | Yes        | Divorced       | 220K          | No                 |
| 8  | No         | Single         | 85K           | Yes                |
| 9  | No         | Married        | 75K           | No                 |
| 10 | No         | Single         | 90K           | Yes                |

# Hunt's Algorithm

| ID | Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|----|------------|----------------|---------------|--------------------|
| 1  | Yes        | Single         | 125K          | No                 |
| 2  | No         | Married        | 100K          | No                 |
| 3  | No         | Single         | 70K           | No                 |
| 4  | Yes        | Married        | 120K          | No                 |
| 5  | No         | Divorced       | 95K           | Yes                |
| 6  | No         | Married        | 60K           | No                 |
| 7  | Yes        | Divorced       | 220K          | No                 |
| 8  | No         | Single         | 85K           | Yes                |
| 9  | No         | Married        | 75K           | No                 |
| 10 | No         | Single         | 90K           | Yes                |



Defaulted = No

(7,3)

(a)

(b)

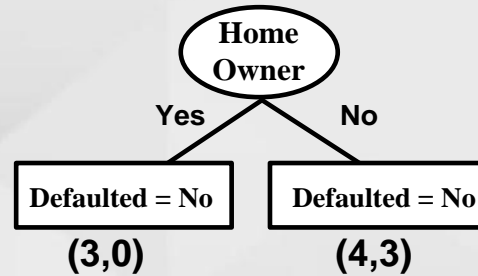
# Hunt's Algorithm

| ID | Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|----|------------|----------------|---------------|--------------------|
| 1  | Yes        | Single         | 125K          | No                 |
| 2  | No         | Married        | 100K          | No                 |
| 3  | No         | Single         | 70K           | No                 |
| 4  | Yes        | Married        | 120K          | No                 |
| 5  | No         | Divorced       | 95K           | Yes                |
| 6  | No         | Married        | 60K           | No                 |
| 7  | Yes        | Divorced       | 220K          | No                 |
| 8  | No         | Single         | 85K           | Yes                |
| 9  | No         | Married        | 75K           | No                 |
| 10 | No         | Single         | 90K           | Yes                |

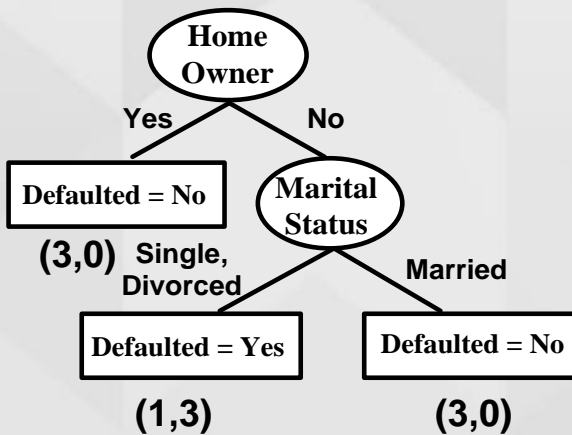
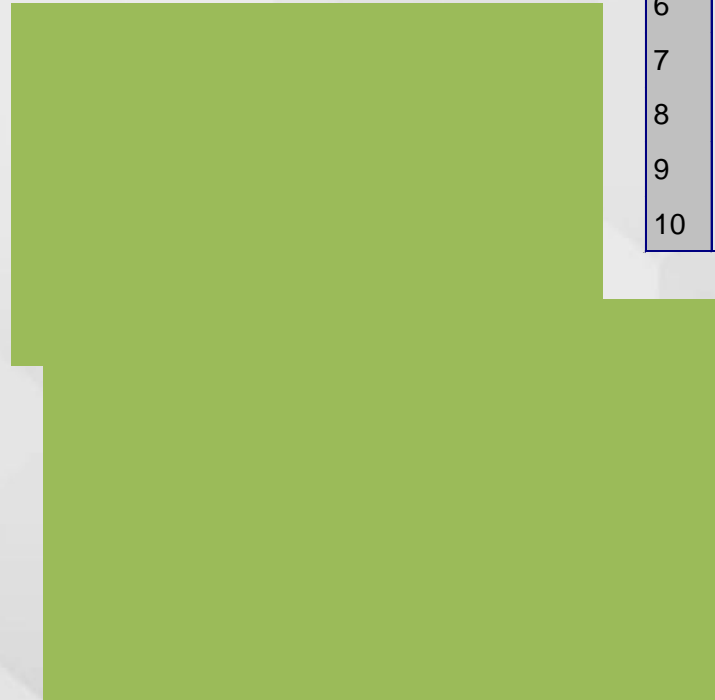
Defaulted = No

(7,3)

(a)



(b)



(c)

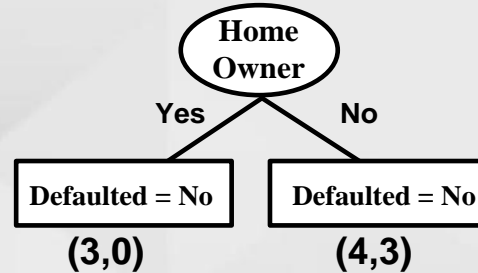
# Hunt's Algorithm

| ID | Home Owner | Marital Status | Annual Income | Defaulted Borrower |
|----|------------|----------------|---------------|--------------------|
| 1  | Yes        | Single         | 125K          | No                 |
| 2  | No         | Married        | 100K          | No                 |
| 3  | No         | Single         | 70K           | No                 |
| 4  | Yes        | Married        | 120K          | No                 |
| 5  | No         | Divorced       | 95K           | Yes                |
| 6  | No         | Married        | 60K           | No                 |
| 7  | Yes        | Divorced       | 220K          | No                 |
| 8  | No         | Single         | 85K           | Yes                |
| 9  | No         | Married        | 75K           | No                 |
| 10 | No         | Single         | 90K           | Yes                |

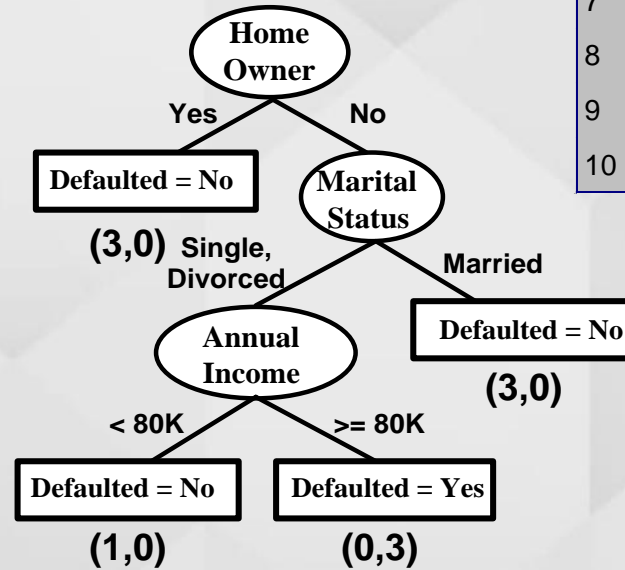
Defaulted = No

(7,3)

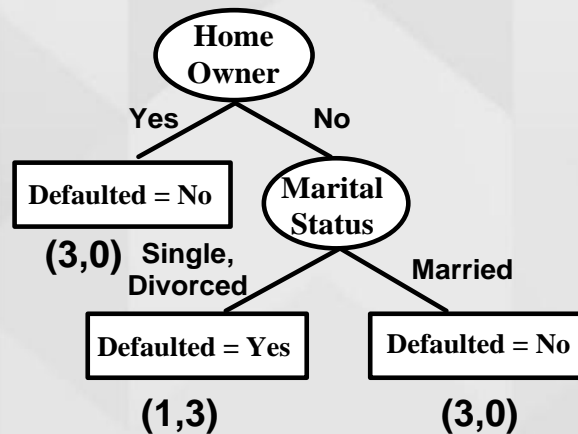
(a)



(b)



(d)



(c)

# Design Issues of Decision Tree Induction

- How should training records be split?
  - Method for expressing test condition
    - ◆ depending on attribute types
  - Measure for evaluating the goodness of a test condition
- **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



# Methods for Expressing Test Conditions

□ Depends on attribute types

– Binary

– Nominal

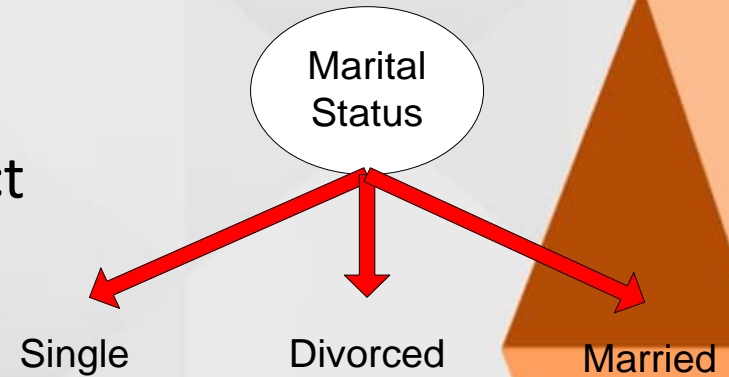
– Ordinal

– Continuous

# Test Condition for Nominal Attributes

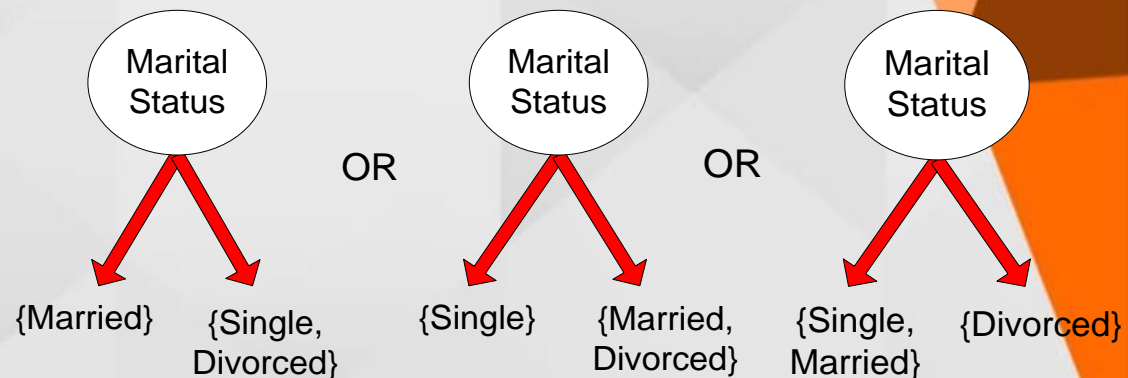
## □ Multi-way split:

- Use as many partitions as distinct values.



## □ Binary split:

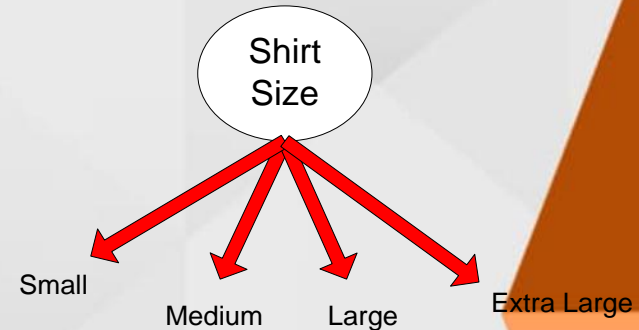
- Divides values into two subsets



## Test Condition for Ordinal Attributes

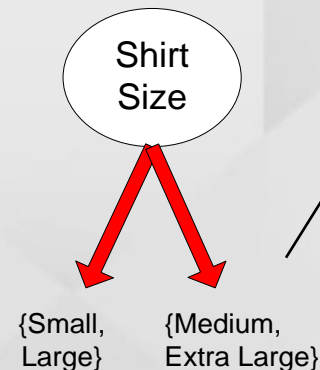
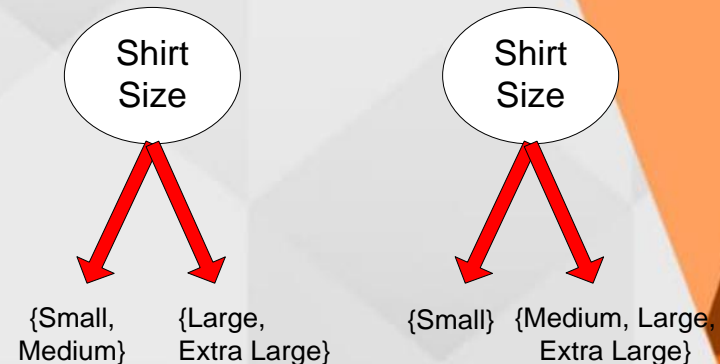
### □ Multi-way split:

- Use as many partitions as distinct values



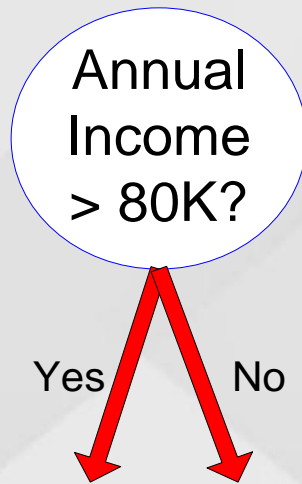
### □ Binary split:

- Divides values into two subsets
- Preserve order property among attribute values

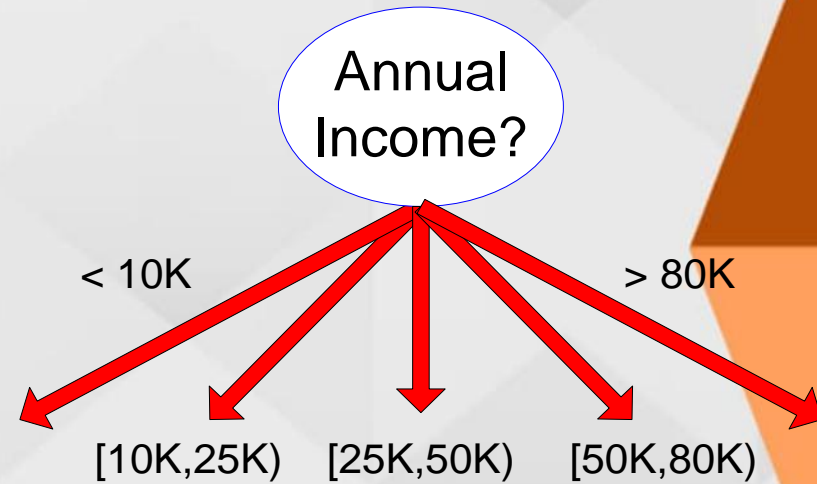


**This grouping  
violates order  
property**

# Test Condition for Continuous Attributes



(i) Binary split



(ii) Multi-way split

## Splitting Based on Continuous Attributes

- Different ways of handling
  - **Discretization** to form an ordinal categorical attribute

Ranges can be found by equal interval bucketing, equal frequency bucketing (percentiles), or clustering.

    - Static – discretize once at the beginning
    - Dynamic – repeat at each node
  - **Binary Decision**:  $(A < v)$  or  $(A \geq v)$ 
    - consider all possible splits and finds the best cut
    - can be more compute intensive

# Attribute Selection Measure:(Measures for selecting Best Split)

## Information Gain (ID3/C4.5)

---

- ID3 uses **information gain** as its attribute selection measure.
- Select the attribute with the highest information gain
- **information** measures info required to classify any arbitrary tuple

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

where  $p_i$  is the nonzero probability that an arbitrary tuple in  $D$  belongs to class  $C_i$  and is estimated by  $|C_{i,D}|/|D|$ .

- **entropy** of attribute  $A$  with values  $\{a_1, a_2, \dots, a_v\}$

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

- **information gained** by branching on attribute  $A$

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

# Attribute Selection by Information Gain Computation

- Class P: buys\_computer = "yes"
- Class N: buys\_computer = "no"
- $I(p, n) = I(9, 5) = 0.940$
- Compute the entropy for *age*:

| age       | $p_i$ | $n_i$ | $I(p_i, n_i)$ |
|-----------|-------|-------|---------------|
| $\leq 30$ | 2     | 3     | 0.971         |
| 30...40   | 4     | 0     | 0             |
| $> 40$    | 3     | 2     | 0.971         |

| age       | income | student | credit_rating | buys_computer |
|-----------|--------|---------|---------------|---------------|
| $\leq 30$ | high   | no      | fair          | no            |
| $\leq 30$ | 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           |
| $\leq 30$ | medium | no      | fair          | no            |
| $\leq 30$ | low    | yes     | fair          | yes           |
| $> 40$    | medium | yes     | fair          | yes           |
| $\leq 30$ | medium | yes     | excellent     | yes           |
| 31...40   | medium | no      | excellent     | yes           |
| 31...40   | high   | yes     | fair          | yes           |
| $> 40$    | medium | no      | excellent     | no            |

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

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

$$\text{Gain}(\text{age}) = I(p, n) - E(\text{age}) = 0.246$$

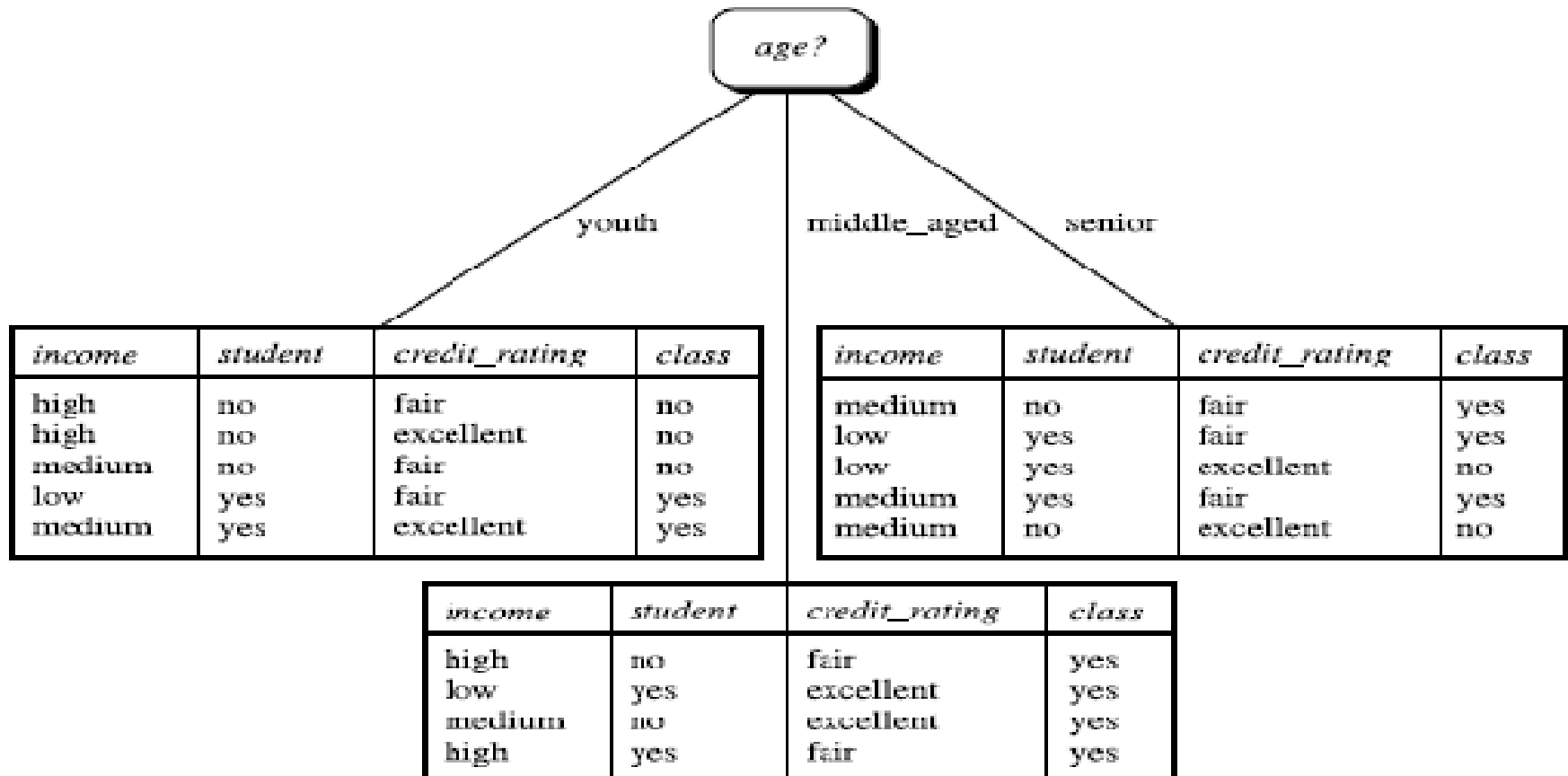
Similarly,

$$\text{Gain}(\text{income}) = 0.029$$

$$\text{Gain}(\text{student}) = 0.151$$

$$\text{Gain}(\text{credit\_rating}) = 0.048$$

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



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.



# GAIN RATIO

C4.5, a successor of ID3, uses an extension to information gain known as *gain ratio*

$$\text{Gain ratio (Att)} = \frac{\text{Gain (Att)}}{\text{Split Information (Att)}}$$

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

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

**Computation of gain ratio for the attribute *income*.** A test on *income* splits the data of Table 8.1 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 Eq. (8.5) to obtain

$$\begin{aligned} \text{SplitInfo}_{\text{income}}(D) &= -\frac{4}{14} \times \log_2 \left( \frac{4}{14} \right) - \frac{6}{14} \times \log_2 \left( \frac{6}{14} \right) - \frac{4}{14} \times \log_2 \left( \frac{4}{14} \right) \\ &= 1.557. \end{aligned}$$

From Example 8.1, we have  $\text{Gain}(\text{income}) = 0.029$ . Therefore,  $\text{GainRatio}(\text{income}) = 0.029/1.557 = 0.019$ . ■

# Other Attribute Selection Measures

---

- **Gini index** (CART, IBM IntelligentMiner)
  - The Gini index is used in CART
  - All attributes are assumed continuous-valued
  - Assume there exist several possible split values for each attribute
  - May need other tools, such as clustering, to get the possible split values
  - Can be modified for categorical attributes

# *Gini* Index (IBM IntelligentMiner)

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

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

where  $p_j$  is the relative frequency of class  $j$  in  $T$ .

- If a data set  $T$  is split into two subsets  $T_1$  and  $T_2$  with sizes  $N_1$  and  $N_2$  respectively, the *gini* index of the split data contains examples from  $n$  classes, the *gini* index  $gini(T)$  is defined as

$$gini_{split}(T) = \frac{N_1}{N} gini(T_1) + \frac{N_2}{N} gini(T_2)$$

- The attribute provides the smallest  $gini_{split}(T)$  is chosen to split the node (*need to enumerate all possible splitting points for each attribute*).

# *Gini* Index (IBM IntelligentMiner)

---

**Induction of a decision tree using the Gini index.** Let  $D$  be the training data shown earlier in Table 8.1, 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 Eq. (8.7) for the Gini index to compute the impurity of  $D$ :

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

# Measure of Impurity: GINI

- Gini Index for a given node  $t$  :

$$\text{Gini Index} = 1 - \sum_{i=0}^{c-1} p_i(t)^2$$

- For 2-class problem  $(p, 1 - p)$ :

- $\text{GINI} = 1 - p^2 - (1 - p)^2 = 2p(1-p)$

|                   |          |
|-------------------|----------|
| C1                | <b>0</b> |
| C2                | <b>6</b> |
| <b>Gini=0.000</b> |          |

|                   |          |
|-------------------|----------|
| C1                | <b>1</b> |
| C2                | <b>5</b> |
| <b>Gini=0.278</b> |          |

|                   |          |
|-------------------|----------|
| C1                | <b>2</b> |
| C2                | <b>4</b> |
| <b>Gini=0.444</b> |          |

|                   |          |
|-------------------|----------|
| C1                | <b>3</b> |
| C2                | <b>3</b> |
| <b>Gini=0.500</b> |          |

# Computing Gini Index of a Single Node

$$Gini\ Index = 1 - \sum_{i=0}^{c-1} p_i(t)^2$$

|    |          |
|----|----------|
| C1 | <b>0</b> |
| C2 | <b>6</b> |

$$P(C1) = 0/6 = 0 \quad P(C2) = 6/6 = 1$$

$$Gini = 1 - P(C1)^2 - P(C2)^2 = 1 - 0 - 1 = 0$$

|    |          |
|----|----------|
| C1 | <b>1</b> |
| C2 | <b>5</b> |

$$P(C1) = 1/6 \quad P(C2) = 5/6$$

$$Gini = 1 - (1/6)^2 - (5/6)^2 = 0.278$$

|    |          |
|----|----------|
| C1 | <b>2</b> |
| C2 | <b>4</b> |

$$P(C1) = 2/6 \quad P(C2) = 4/6$$

$$Gini = 1 - (2/6)^2 - (4/6)^2 = 0.444$$



# Computing Gini Index for a Collection of Nodes

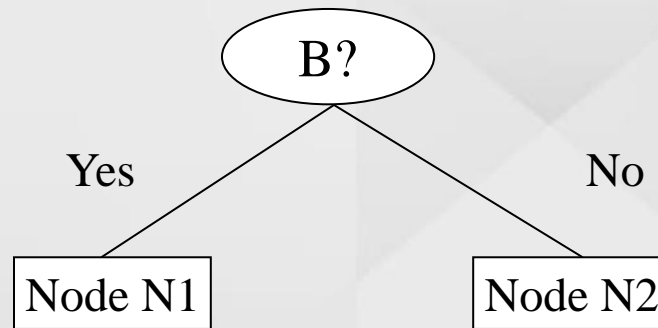
- | When a node  $p$  is split into  $k$  partitions (children)

$$GINI_{split} = \sum_{i=1}^k \frac{n_i}{n} GINI(i)$$

where,  $n_i$  = number of records at child  $i$ ,  
 $n$  = number of records at parent node  $p$ .

# Binary Attributes: Computing GINI Index

- Splits into two partitions (child nodes)
- Effect of Weighing partitions:
  - Larger and purer partitions are sought



|                     | Parent |
|---------------------|--------|
| C1                  | 7      |
| C2                  | 5      |
| <b>Gini = 0.486</b> |        |

**Gini(N1)**

$$= 1 - (5/6)^2 - (1/6)^2$$

$$= 0.278$$

**Gini(N2)**

$$= 1 - (2/6)^2 - (4/6)^2$$

$$= 0.444$$

|                   | N1 | N2 |
|-------------------|----|----|
| C1                | 5  | 2  |
| C2                | 1  | 4  |
| <b>Gini=0.361</b> |    |    |

**Weighted Gini of N1 N2**

$$= 6/12 * 0.278 +$$

$$6/12 * 0.444$$

$$= 0.361$$

**Gain = 0.486 - 0.361 = 0.125**



# Categorical Attributes: Computing Gini Index

- For each distinct value, gather counts for each class in the dataset
- Use the count matrix to make decisions

Multi-way split

|      | CarType      |        |        |
|------|--------------|--------|--------|
|      | Family       | Sports | Luxury |
| C1   | 1            | 8      | 1      |
| C2   | 3            | 0      | 7      |
| Gini | <b>0.163</b> |        |        |

Two-way split  
(find best partition of values)

|      | CarType          |          |
|------|------------------|----------|
|      | {Sports, Luxury} | {Family} |
| C1   | 9                | 1        |
| C2   | 7                | 3        |
| Gini | <b>0.468</b>     |          |

|      | CarType      |                  |
|------|--------------|------------------|
|      | {Sports}     | {Family, Luxury} |
| C1   | 8            | 2                |
| C2   | 0            | 10               |
| Gini | <b>0.167</b> |                  |


Which of these is the best?

# Continuous Attributes: Computing Gini Index

- Use Binary Decisions based on one value
- Several Choices for the splitting value
  - Number of possible splitting values = Number of distinct values
- Each splitting value has a count matrix associated with it
  - Class counts in each of the partitions,  $A \leq v$  and  $A > v$
- Simple method to choose best  $v$ 
  - For each  $v$ , scan the database to gather count matrix and compute its Gini index
  - Computationally Inefficient! Repetition of work.

| ID | Home Owner | Marital Status | Annual Income | Defaulted |
|----|------------|----------------|---------------|-----------|
| 1  | Yes        | Single         | 125K          | No        |
| 2  | No         | Married        | 100K          | No        |
| 3  | No         | Single         | 70K           | No        |
| 4  | Yes        | Married        | 120K          | No        |
| 5  | No         | Divorced       | 95K           | Yes       |
| 6  | No         | Married        | 60K           | No        |
| 7  | Yes        | Divorced       | 220K          | No        |
| 8  | No         | Single         | 85K           | Yes       |
| 9  | No         | Married        | 75K           | No        |
| 10 | No         | Single         | 90K           | Yes       |

**Annual Income ?**



$\leq 80$        $> 80$

|               |   |   |
|---------------|---|---|
| Defaulted Yes | 0 | 3 |
| Defaulted No  | 3 | 4 |

# Continuous Attributes: Computing Gini Index...

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index

|                        |                      |    |    |     |     |     |     |     |     |     |
|------------------------|----------------------|----|----|-----|-----|-----|-----|-----|-----|-----|
| <b>Cheat</b>           | No                   | No | No | Yes | Yes | Yes | No  | No  | No  | No  |
|                        | <b>Annual Income</b> |    |    |     |     |     |     |     |     |     |
| <b>Sorted Values</b> → | 60                   | 70 | 75 | 85  | 90  | 95  | 100 | 120 | 125 | 220 |



# Continuous Attributes: Computing Gini Index...

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index

↓

|                          |                      |      |      |       |      |      |      |      |      |      |      |
|--------------------------|----------------------|------|------|-------|------|------|------|------|------|------|------|
| <b>Cheat</b>             | No                   | No   | No   | Yes   | Yes  | Yes  | No   | No   | No   | No   |      |
|                          | <b>Annual Income</b> |      |      |       |      |      |      |      |      |      |      |
| <b>Sorted Values</b> →   | 60                   | 70   | 75   | 85    | 90   | 95   | 100  | 120  | 125  | 220  |      |
| <b>Split Positions</b> → | 55                   | 65   | 72   | 80    | 87   | 92   | 97   | 110  | 122  | 172  | 230  |
|                          | <= >                 | <= > | <= > | <= >  | <= > | <= > | <= > | <= > | <= > | <= > | <= > |
| <b>Yes</b>               |                      |      |      | 0     | 3    |      |      |      |      |      |      |
| <b>No</b>                |                      |      |      | 3     | 4    |      |      |      |      |      |      |
| <b>Gini</b>              |                      |      |      | 0.343 |      |      |      |      |      |      |      |

# Continuous Attributes: Computing Gini Index...

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index

↓

|                          |                      |    |    |       |       |     |     |     |     |     |     |   |
|--------------------------|----------------------|----|----|-------|-------|-----|-----|-----|-----|-----|-----|---|
| <b>Cheat</b>             | No                   | No | No | Yes   | Yes   | Yes | No  | No  | No  | No  |     |   |
|                          | <b>Annual Income</b> |    |    |       |       |     |     |     |     |     |     |   |
| <b>Sorted Values</b> →   | 60                   | 70 | 75 | 85    | 90    | 95  | 100 | 120 | 125 | 220 |     |   |
| <b>Split Positions</b> → | 55                   | 65 | 72 | 80    | 87    | 92  | 97  | 110 | 122 | 172 | 230 |   |
|                          | <=                   | >  | <= | >     | <=    | >   | <=  | >   | <=  | >   | <=  | > |
| <b>Yes</b>               |                      |    |    | 0     | 3     | 1   | 2   |     |     |     |     |   |
| <b>No</b>                |                      |    |    | 3     | 4     | 3   | 4   |     |     |     |     |   |
| <b>Gini</b>              |                      |    |    | 0.343 | 0.417 |     |     |     |     |     |     |   |

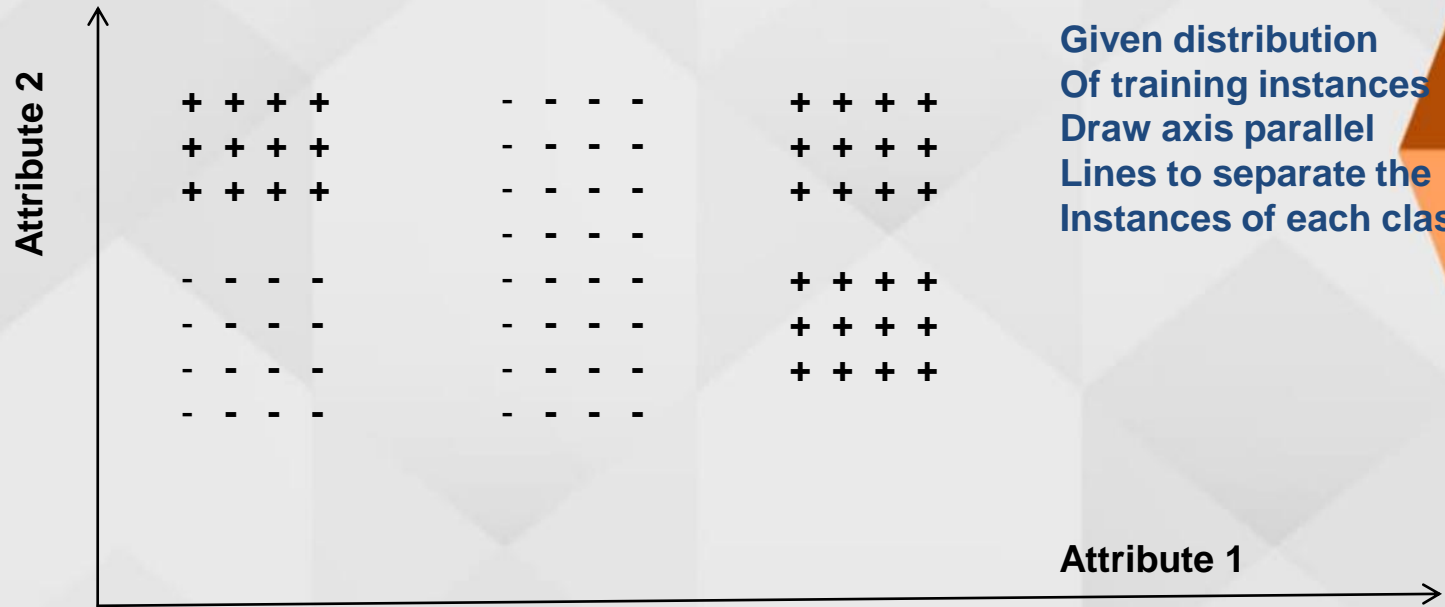
# Continuous Attributes: Computing Gini Index...

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index

| Cheat           | No           | No            | No           | Yes          | Yes          | Yes          | No                  | No           | No           | No           |              |   |   |   |   |   |   |   |   |   |
|-----------------|--------------|---------------|--------------|--------------|--------------|--------------|---------------------|--------------|--------------|--------------|--------------|---|---|---|---|---|---|---|---|---|
| Sorted Values   |              | Annual Income |              |              |              |              |                     |              |              |              |              |   |   |   |   |   |   |   |   |   |
| Split Positions | 60           | 70            | 75           | 85           | 90           | 95           | 100                 | 120          | 125          | 220          |              |   |   |   |   |   |   |   |   |   |
|                 | 55           | 65            | 72           | 80           | 87           | 92           | 97                  | 110          | 122          | 172          | 230          |   |   |   |   |   |   |   |   |   |
|                 | <= >         | <= >          | <= >         | <= >         | <= >         | <= >         | <= >                | <= >         | <= >         | <= >         | <= >         |   |   |   |   |   |   |   |   |   |
| Yes             | 0            | 3             | 0            | 3            | 0            | 3            | 0                   | 3            | 0            | 3            | 0            |   |   |   |   |   |   |   |   |   |
| No              | 0            | 7             | 1            | 6            | 2            | 5            | 3                   | 4            | 3            | 4            | 3            | 4 | 4 | 3 | 5 | 2 | 6 | 1 | 7 | 0 |
| Gini            | <b>0.420</b> | <b>0.400</b>  | <b>0.375</b> | <b>0.343</b> | <b>0.417</b> | <b>0.400</b> | <b><u>0.300</u></b> | <b>0.343</b> | <b>0.375</b> | <b>0.400</b> | <b>0.420</b> |   |   |   |   |   |   |   |   |   |



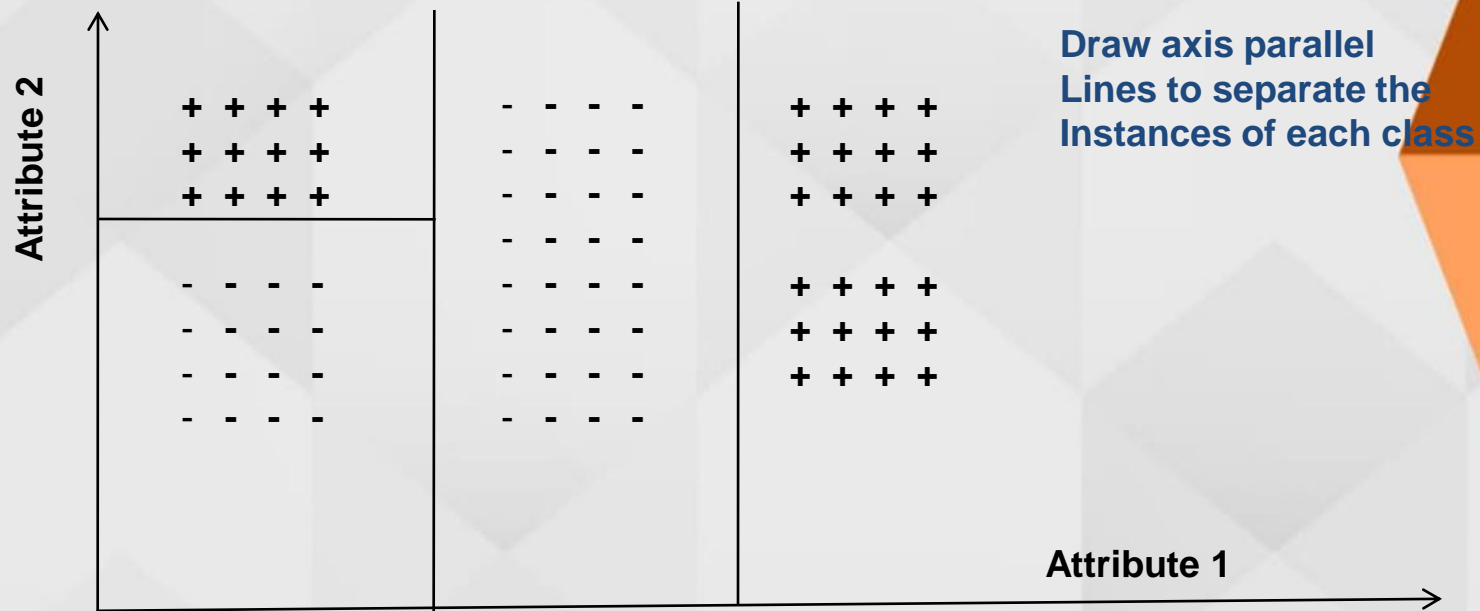
# Decision Trees



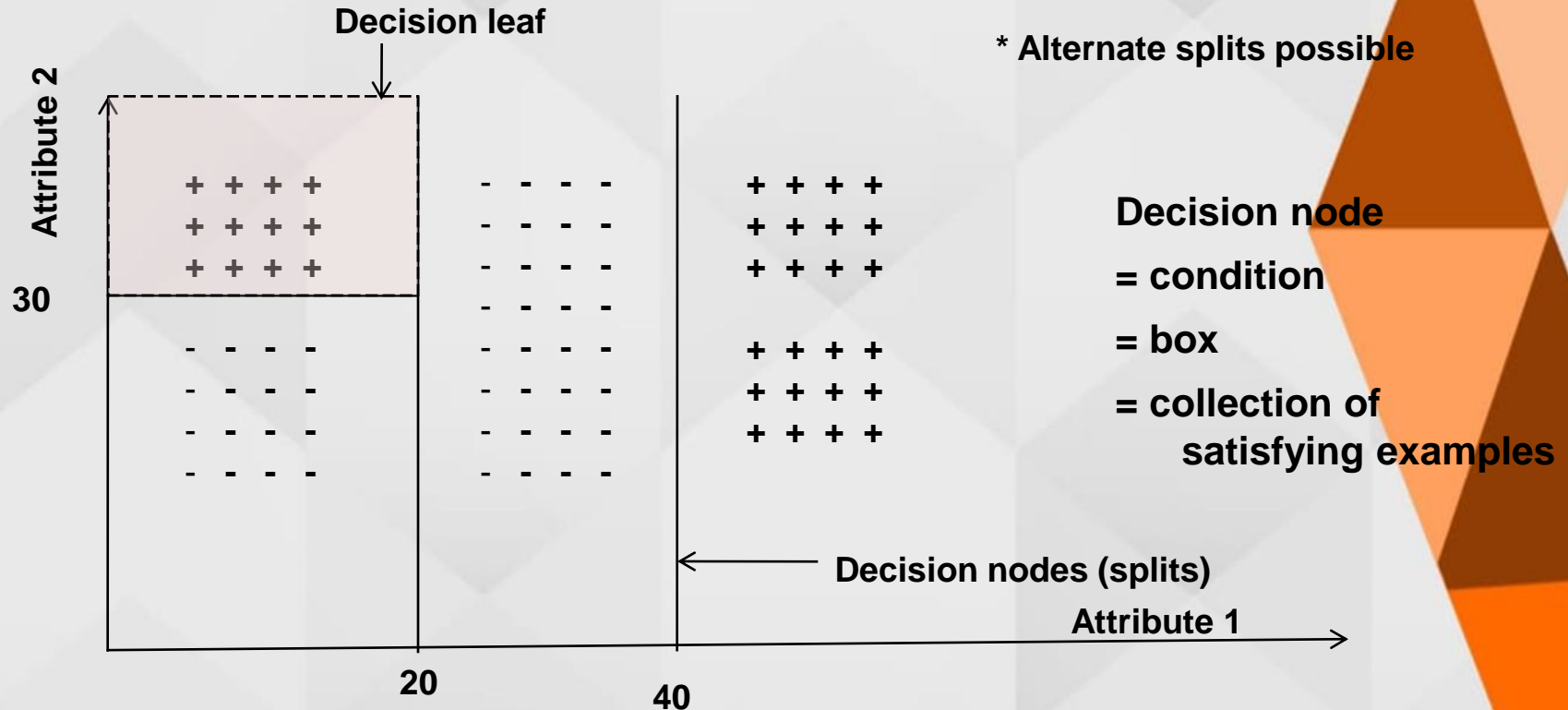




# Decision Tree Structure



# Decision Tree Structure



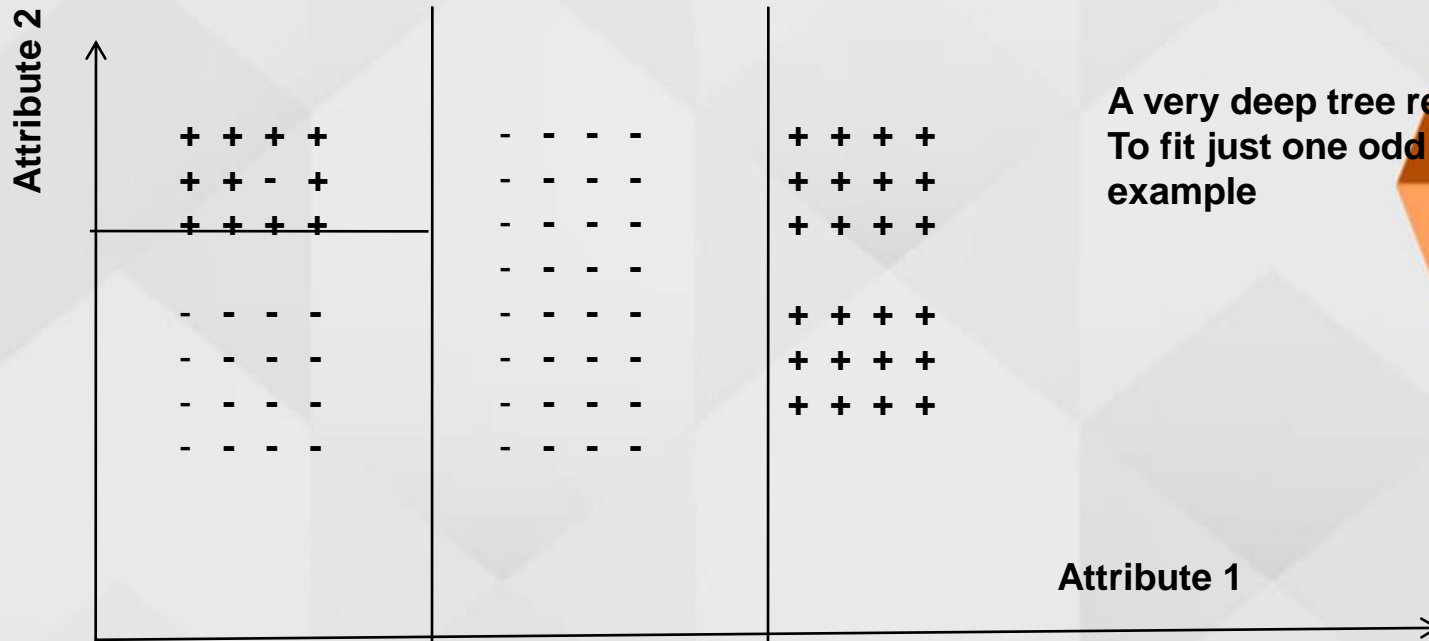
# Overfitting the Data

- Learning a tree that classifies the training data perfectly may not lead to the tree with the **best generalization performance**.
  - There may be noise in the training data the tree is fitting
  - The algorithm might be making decisions based on very little data
- A hypothesis  $h$  is said to **overfit the training data** if there is another hypothesis,  $h'$ , such that  $h$  has smaller error than  $h'$  on the training data but  $h$  has larger error on the test data than  $h'$ .





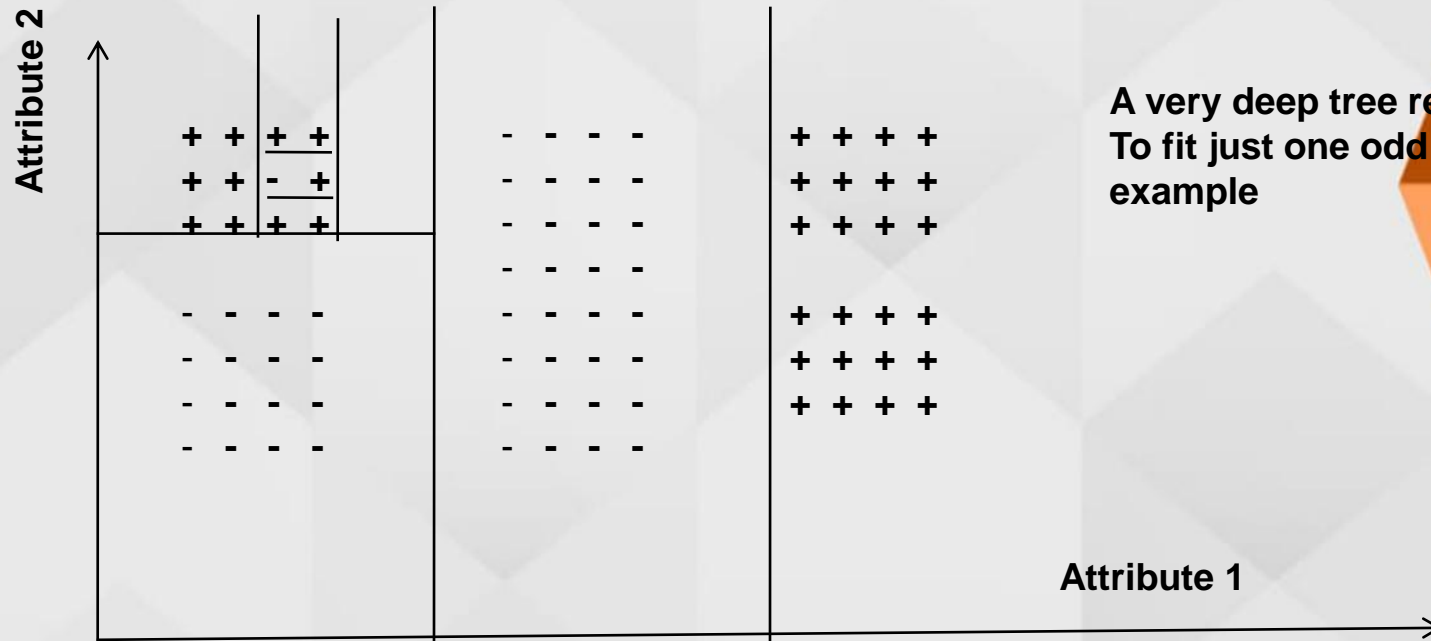
# Overfitting



**A very deep tree required  
To fit just one odd training  
example**



## When to stop splitting further?

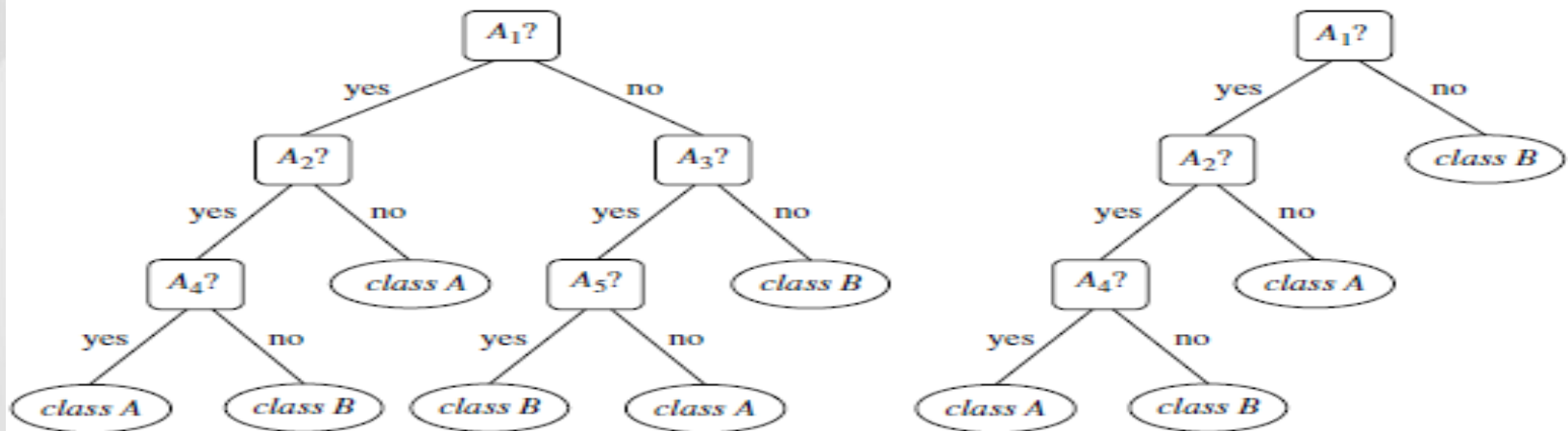


**A very deep tree required  
To fit just one odd training  
example**

**Attribute 1**

## Avoiding Overfitting

- Two basic approaches
  - **Prepruning:** Stop growing the tree at some point during construction when it is determined that there is not enough data to make reliable choices.
  - **Postpruning:** Grow the full tree and then remove nodes that seem not to have sufficient evidence. (more popular)



An unpruned decision tree and a pruned version of it.

# Approaches to Determine the Final Tree Size

---

- Use hold out, Separate training (2/3) and testing (1/3) sets
- Use cross validation, e.g., 10-fold cross validation
- Use all the data for training
  - but apply a **statistical test** (e.g., chi-square) to estimate whether expanding or pruning a node may improve the entire distribution
- Use Bootstrap, the given training tuples uniformly *with replacement*.

# Classification in Large Databases

---

- Classification—a classical problem extensively studied by statisticians and machine learning researchers
- Scalability: Classifying data sets with millions of examples and hundreds of attributes with reasonable speed
- Why decision tree induction in data mining?
  - relatively faster learning speed (than other classification methods)
  - convertible to simple and easy to understand classification rules
  - can use SQL queries for accessing databases
  - comparable classification accuracy with other methods



# Scalable Decision Tree Induction Methods in Data Mining

---

- **SLIQ** (EDBT'96 — Mehta et al.)
  - builds an index for each attribute and only class list and the current attribute list reside in memory
- **SPRINT** (VLDB'96 — J. Shafer et al.)
  - constructs an attribute list data structure
- **PUBLIC** (VLDB'98 — Rastogi & Shim)
  - integrates tree splitting and tree pruning: stop growing the tree earlier
- **RainForest** (VLDB'98 — Gehrke, Ramakrishnan & Ganti)
  - separates the scalability aspects from the criteria that determine the quality of the tree
  - builds an AVC-list (attribute, value, class label)
- **BOAT** (PODS'99 — Gehrke, Ganti, Ramakrishnan & Loh)
  - Uses bootstrapping to create several small samples.

## Scalability Framework for RainForest

- Separates the scalability aspects from the criteria that determine the quality of the tree
- Builds an AVC-list: **AVC (Attribute, Value, Class\_label)**
- **AVC-set** (of an attribute  $X$ )
  - Projection of training dataset onto the attribute  $X$  and class label where counts of individual class label are aggregated
- **AVC-group** (of a node  $n$ )
  - Set of AVC-sets of all predictor attributes at the node  $n$

| 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            |

| age         | <i>buys_computer</i> |    |
|-------------|----------------------|----|
|             | yes                  | no |
| youth       | 2                    | 3  |
| middle_aged | 4                    | 0  |
| senior      | 3                    | 2  |

| income | <i>buys_computer</i> |    |
|--------|----------------------|----|
|        | yes                  | no |
| low    | 3                    | 1  |
| medium | 4                    | 2  |
| high   | 2                    | 2  |

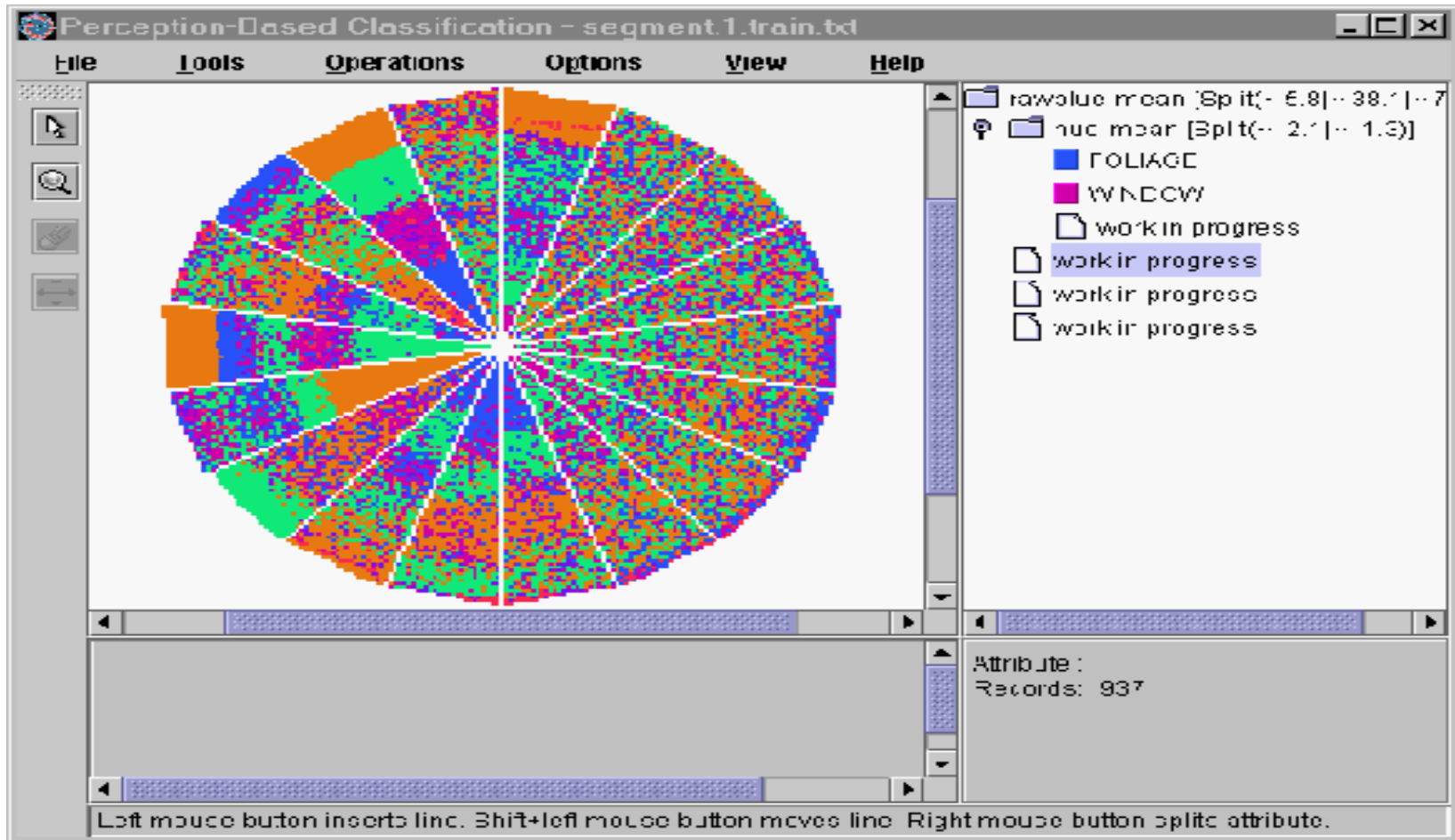
| student | <i>buys_computer</i> |    |
|---------|----------------------|----|
|         | yes                  | no |
| yes     | 6                    | 1  |
| no      | 3                    | 4  |

| credit_rating | <i>buys_computer</i> |    |
|---------------|----------------------|----|
|               | yes                  | no |
| fair          | 6                    | 2  |
| excellent     | 3                    | 3  |

---

The use of data structures to hold aggregate information regarding the training data (e.g., these AVC-sets describing Table 8.1's data) are one approach to improving the scalability of decision tree induction.

# Interactive Visual Mining by Perception-Based Classification (PBC)



A screenshot of PBC, a system for interactive decision tree construction. Multidimensional training data are viewed as circle segments in the Data Interaction window (*left*). The Knowledge Interaction window (*right*) displays the current decision tree.

# Perception-Based Classification (PBC)

---

- **PBC uses a pixel-oriented approach to view multidimensional data with its class label information.**
- **The PBC system displays a split screen, consisting of a Data Interaction window and a Knowledge Interaction window.**
- **The Data Interaction window displays the circle segments of the data under examination, while the Knowledge Interaction window displays the decision tree constructed so far.**
- **Initially, the complete training set is visualized in the Data Interaction window, while the Knowledge Interaction window displays an empty decision tree.**
- **A tree is interactively constructed as follows.**
- **The user visualizes the multidimensional data in the Data Interaction window and selects a splitting attribute and one or more split-points.**
- **The current decision tree in the Knowledge Interaction window is expanded.**
- **The user selects a node of the decision tree. The user may either assign a class label to the node (which makes the node a leaf) or request the visualization of the training data corresponding to the node.**
- **This leads to a new visualization of every attribute except the ones used for splitting criteria on the same path from the root.**
- **The interactive process continues until a class has been assigned to each leaf of the decision tree.**