# Module 3
# Classification

# Supervised vs. Unsupervised Learning

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

# Classification

- 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
- Typical applications
  - Credit/loan approval:
  - Medical diagnosis: if a tumor is cancerous or benign
  - Fraud detection: if a transaction is fraudulent
  - Web page categorization: which category it is

# 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.
- 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.
  - If the accuracy is acceptable, use the model to classify new data

# Process (1): Model Construction

Training Data

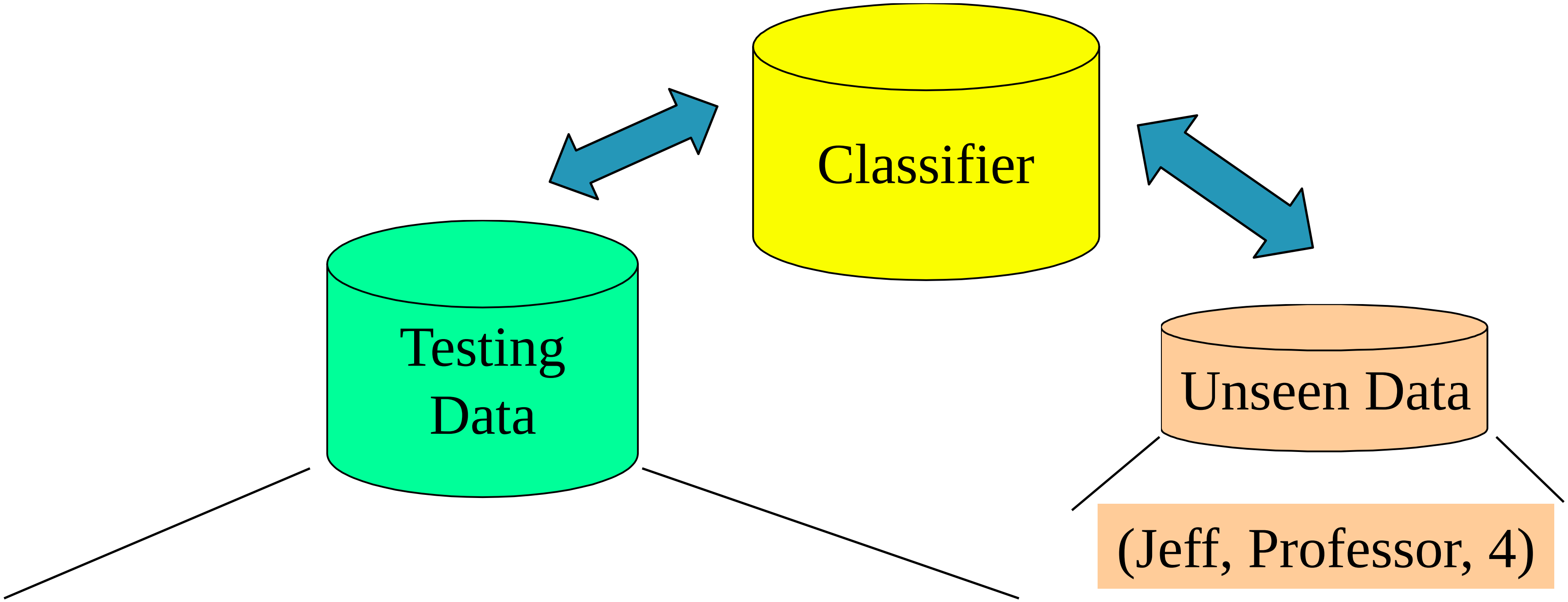
| NAME | RANK | YEARS | TENURED |
|---|---|---|---|
| Mike | Assistant Prof | 3 | no |
| Mary | Assistant Prof | 7 | yes |
| Bill | Professor | 2 | yes |
| Jim | Associate Prof | 7 | yes |
| Dave | Assistant Prof | 6 | no |
| Anne | Associate Prof | 3 | no |

Classification Algorithms

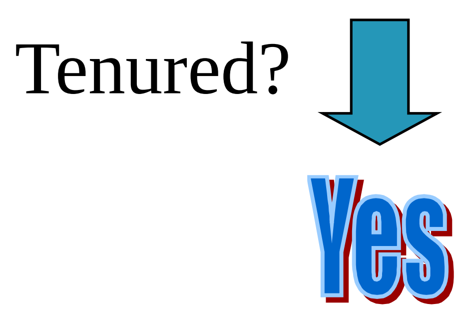
Classifier (Model)

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

# Process (2): Using the Model for Classification



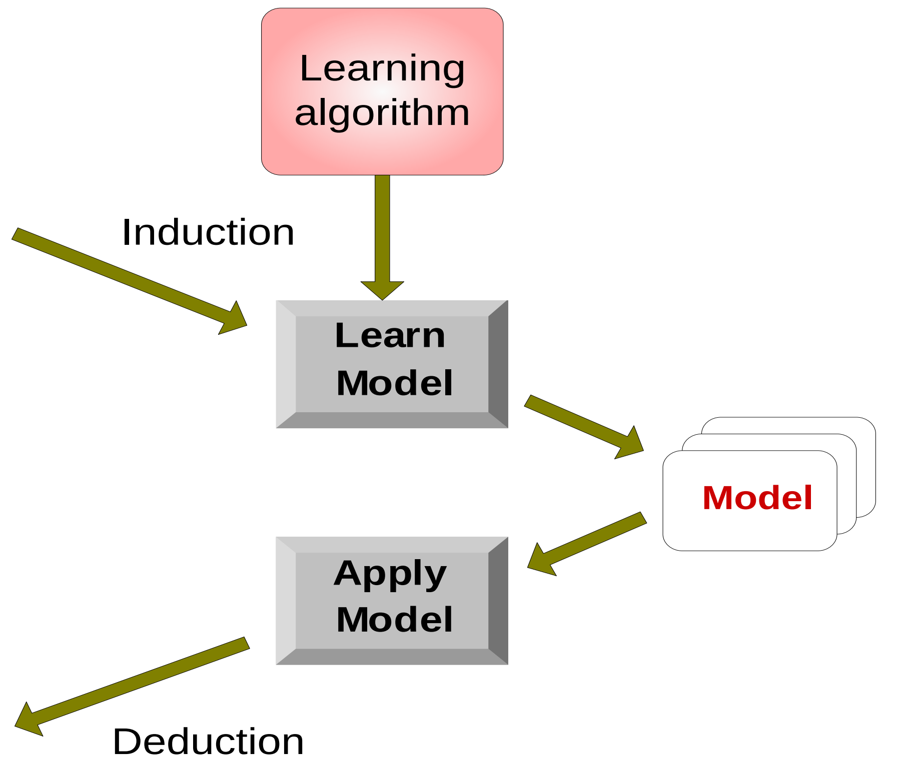

| NAME | RANK | YEARS | TENURED |
|---|---|---|---|
| Tom | Assistant Prof | 2 | no |
| Merlisa | Associate Prof | 7 | no |
| George | Professor | 5 | yes |
| Joseph | Assistant Prof | 7 | yes |



# Illustrating 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 | ? |

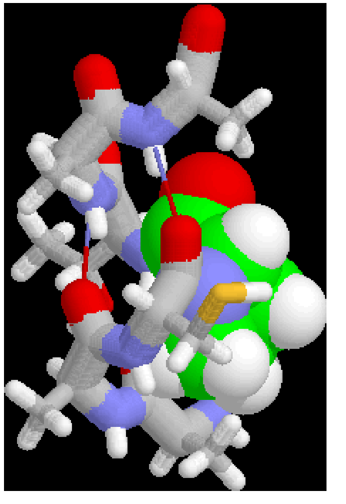
Test Set

# Examples of Classification

- Predicting tumor cells as benign or malignant

- 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

# Decision Tree Induction

# Decision Tree Induction

- J. Ross Quinlan
  - a researcher in machine learning
  - ID3 (Iterative Dichotomiser).
  - Late 1970 and early 1980
- E. B. Hunt, J. Marin , P. T. Stone. And Quinlan
  - presented C4.5 (a successor of ID3),
  - which became a benchmark to which newer supervised learning algorithms are often compared.
- L. Breiman, J. Friedman, R. Olshen, and C. Stone
  - *Classification and Regression Trees* (CART), which described the generation of binary decision trees.

# Decision Tree Induction

- ID3, C4.5, and CART adopt a greedy approach in which decision trees are constructed in a top-down recursive divide-and-conquer manner.
- The training set is recursively partitioned into smaller subsets as the tree is being built.

# 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.

# Algorithm : Generate 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*;

# Algorithm : Generate decision tree.

(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*;

# Algorithm : Generate decision tree.

(14) else attach the node returned by

Generate _decision _tree(*Dj*, *attribute list*) to node *N*;

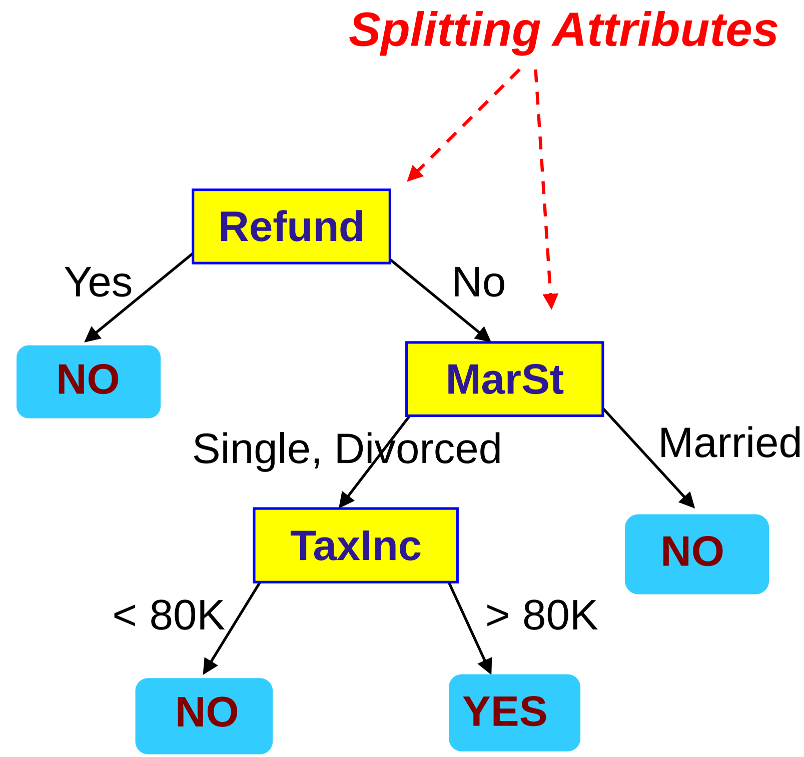
endfor

(15) return *N*;

# Example of a Decision Tree

categorical categorical continuous class

| *Tid* | Refund | Marital Status | Taxable Income | Cheat |
|---|---|---|---|---|
| 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 |

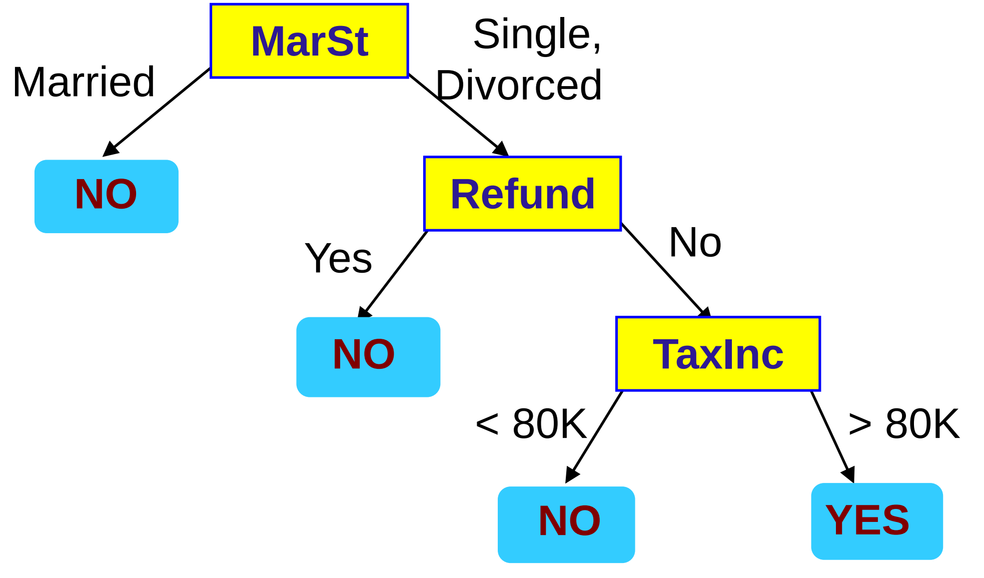



**Training Data**

**Model: Decision Tree**

# Another Example of Decision Tree

| | categorical | categorical | continuous | class |
|---|---|---|---|---|
| *Tid* | **Refund** | **Marital Status** | **Taxable Income** | **Cheat** |
| 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 |

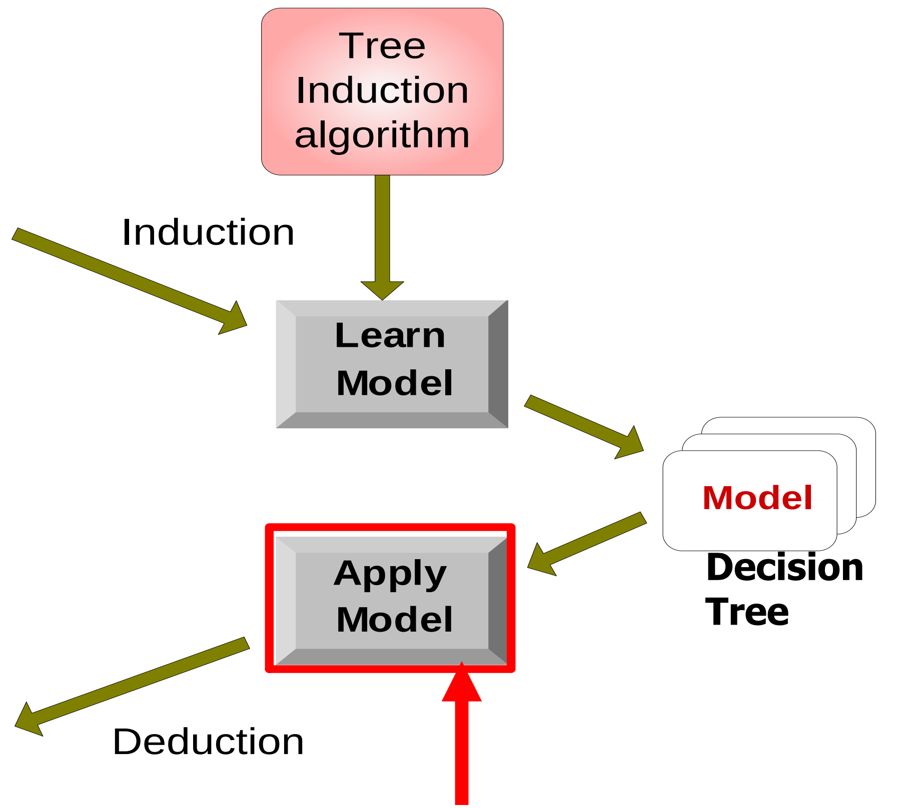



**There could be more than one tree that fits the same data!**

# 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

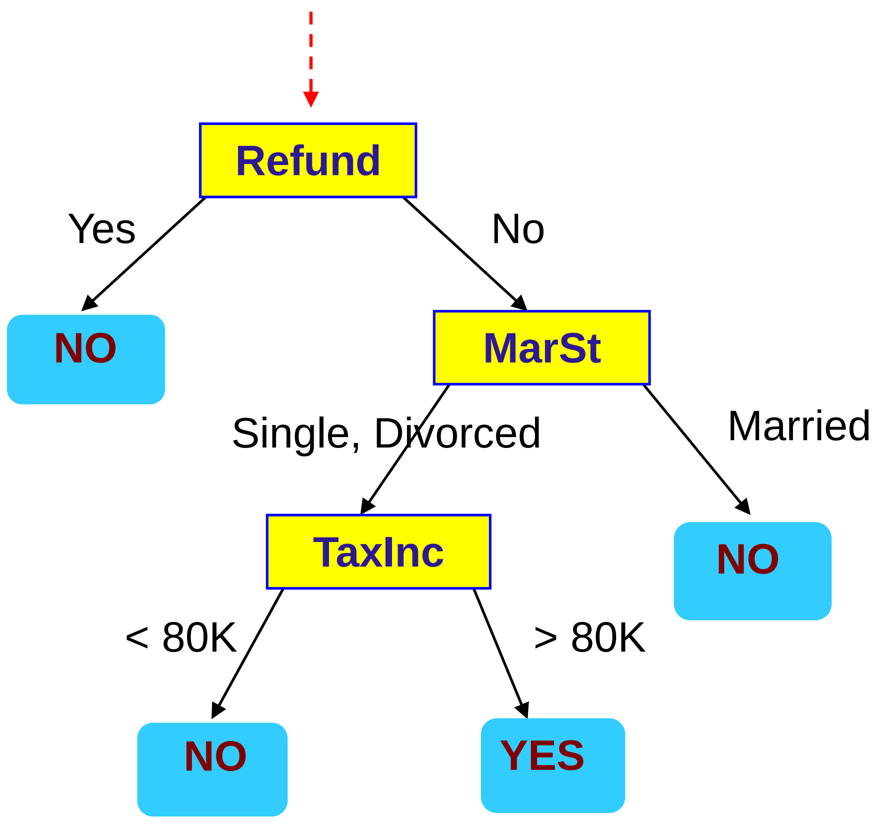



| *Tid* | Attrib1 | Attrib2 | Attrib3 | Class |
|---|---|---|---|---|
| 11 | No | Small | 55K | ? |
| 12 | Yes | Medium | 80K | ? |
| 13 | Yes | Large | 110K | ? |
| 14 | No | Small | 95K | ? |
| 15 | No | Large | 67K | ? |

Test Set

# Apply Model to Test Data

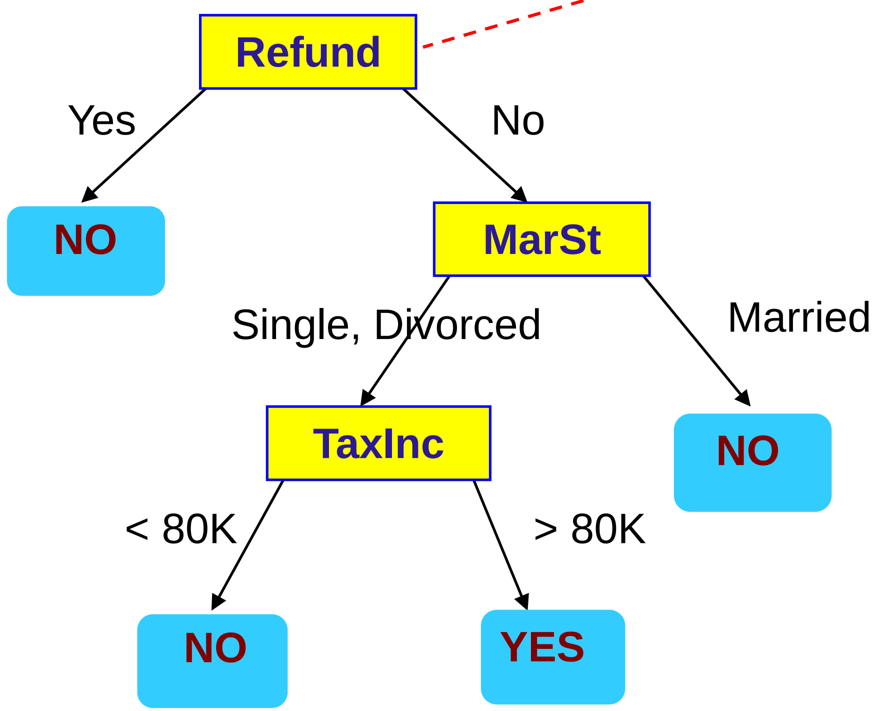
Start from the root of tree.




## Test Data

| Refund | Marital Status | Taxable Income | Cheat |
|---|---|---|---|
| No | Married | 80K | ? |

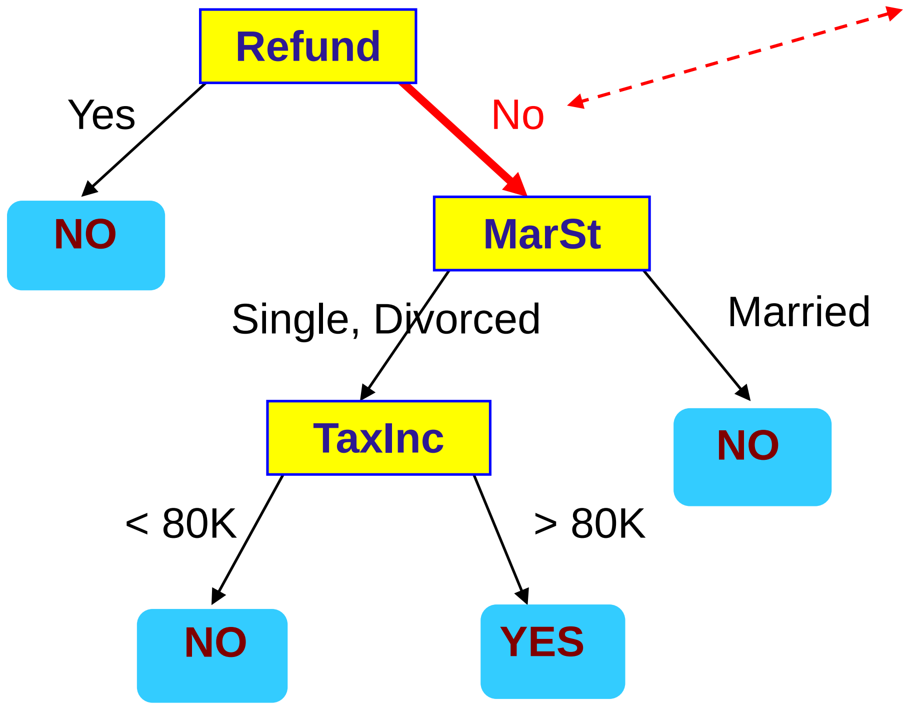
# Apply Model to Test Data




**Test Data**

| Refund | Marital Status | Taxable Income | Cheat |
|---|---|---|---|
| No | Married | 80K | **?** |

# Apply Model to Test Data



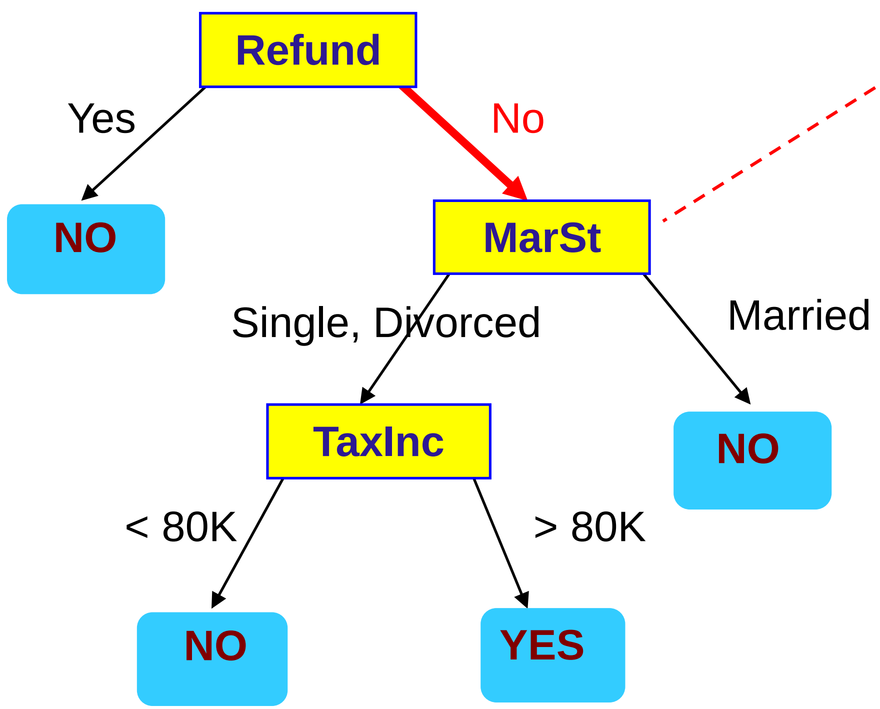

**Test Data**

| Refund | Marital Status | Taxable Income | Cheat |
|---|---|---|---|
| No | Married | 80K | ? |

# Apply Model to Test Data

## Test Data

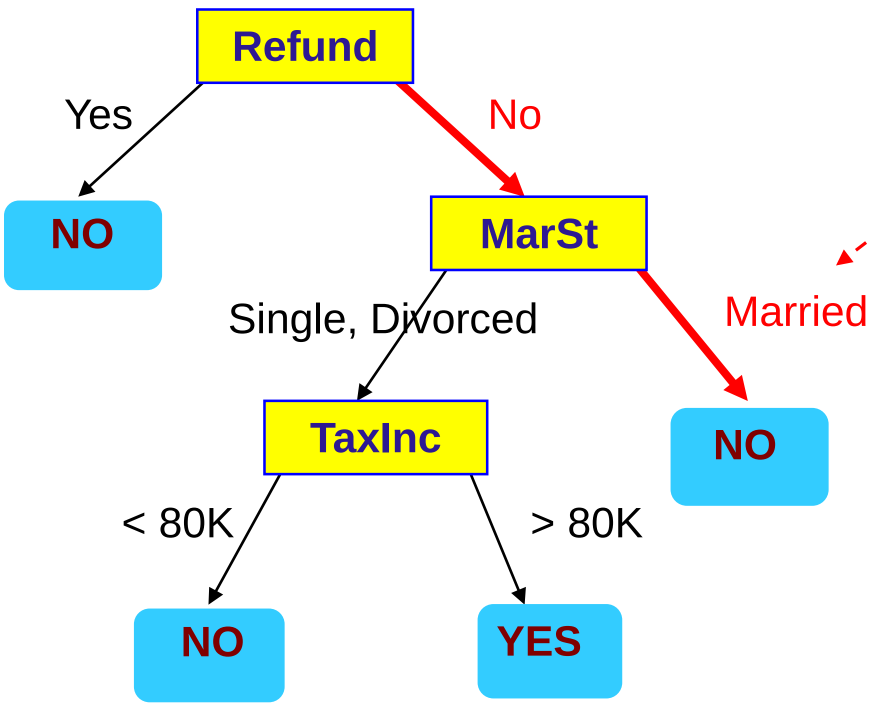
| Refund | Marital Status | Taxable Income | Cheat |
|---|---|---|---|
| No | Married | 80K | ? |



# Apply Model to Test Data

**Test Data**

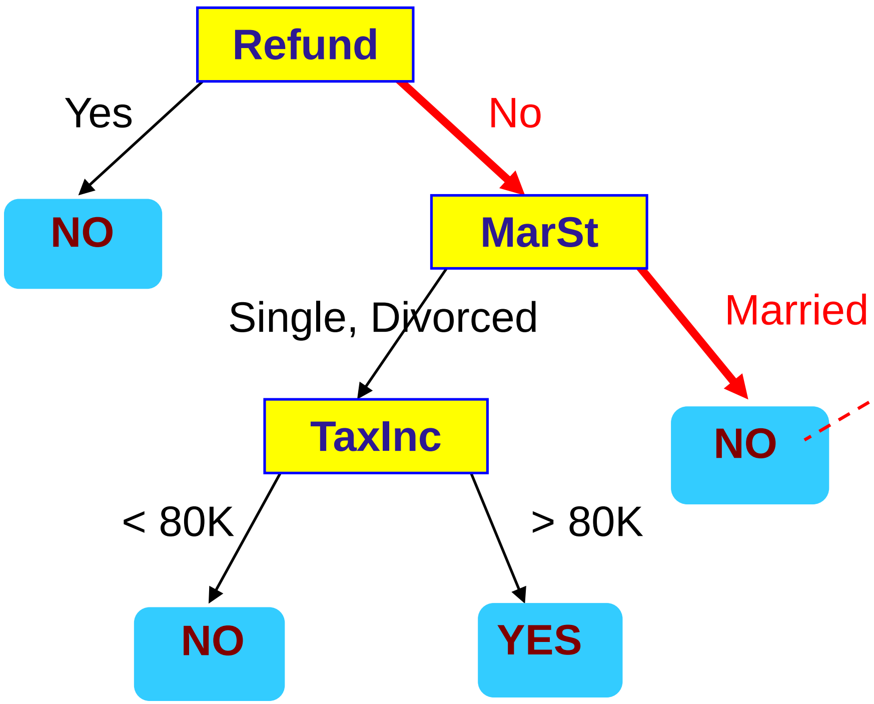
| Refund | Marital Status | Taxable Income | Cheat |
|---|---|---|---|
| No | Married | 80K | ? |



# Apply Model to Test Data

**Test Data**

| Refund | Marital Status | Taxable Income | Cheat |
|---|---|---|---|
| No | Married | 80K | ? |




Assign Cheat to "No"

# Algorithm for Decision Tree Induction

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

Table 1. A small training set

| No. | Attributes | | | | Class |
|---|---|---|---|---|---|
| | Outlook | Temperature | Humidity | Windy | |
| 1 | sunny | hot | high | false | N |
| 2 | sunny | hot | high | true | N |
| 3 | overcast | hot | high | false | P |
| 4 | rain | mild | high | false | P |
| 5 | rain | cool | normal | false | P |
| 6 | rain | cool | normal | true | N |
| 7 | overcast | cool | normal | true | P |
| 8 | sunny | mild | high | false | N |
| 9 | sunny | cool | normal | false | P |
| 10 | rain | mild | normal | false | P |
| 11 | sunny | mild | normal | true | P |
| 12 | overcast | mild | high | true | P |
| 13 | overcast | hot | normal | false | P |
| 14 | rain | mild | high | true | N |

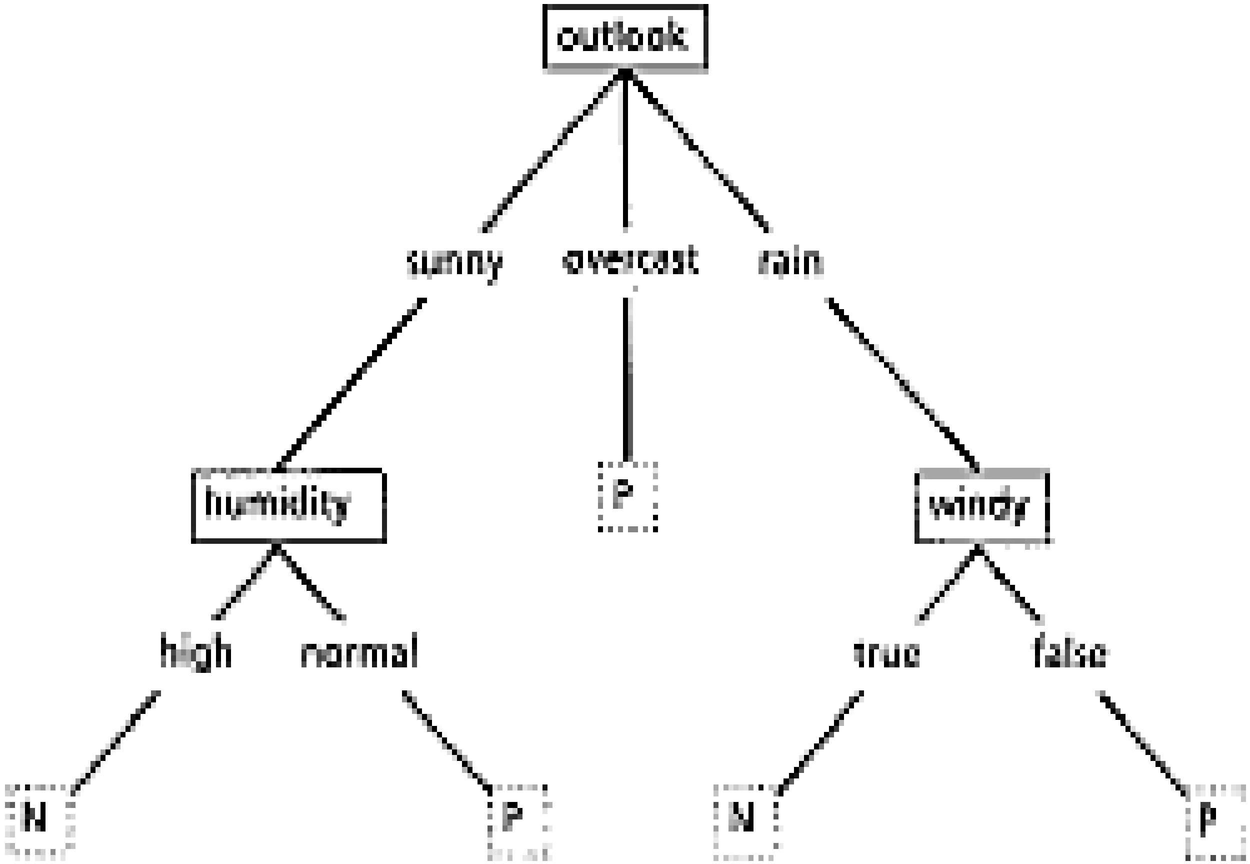



*Figure 2.* A simple decision tree

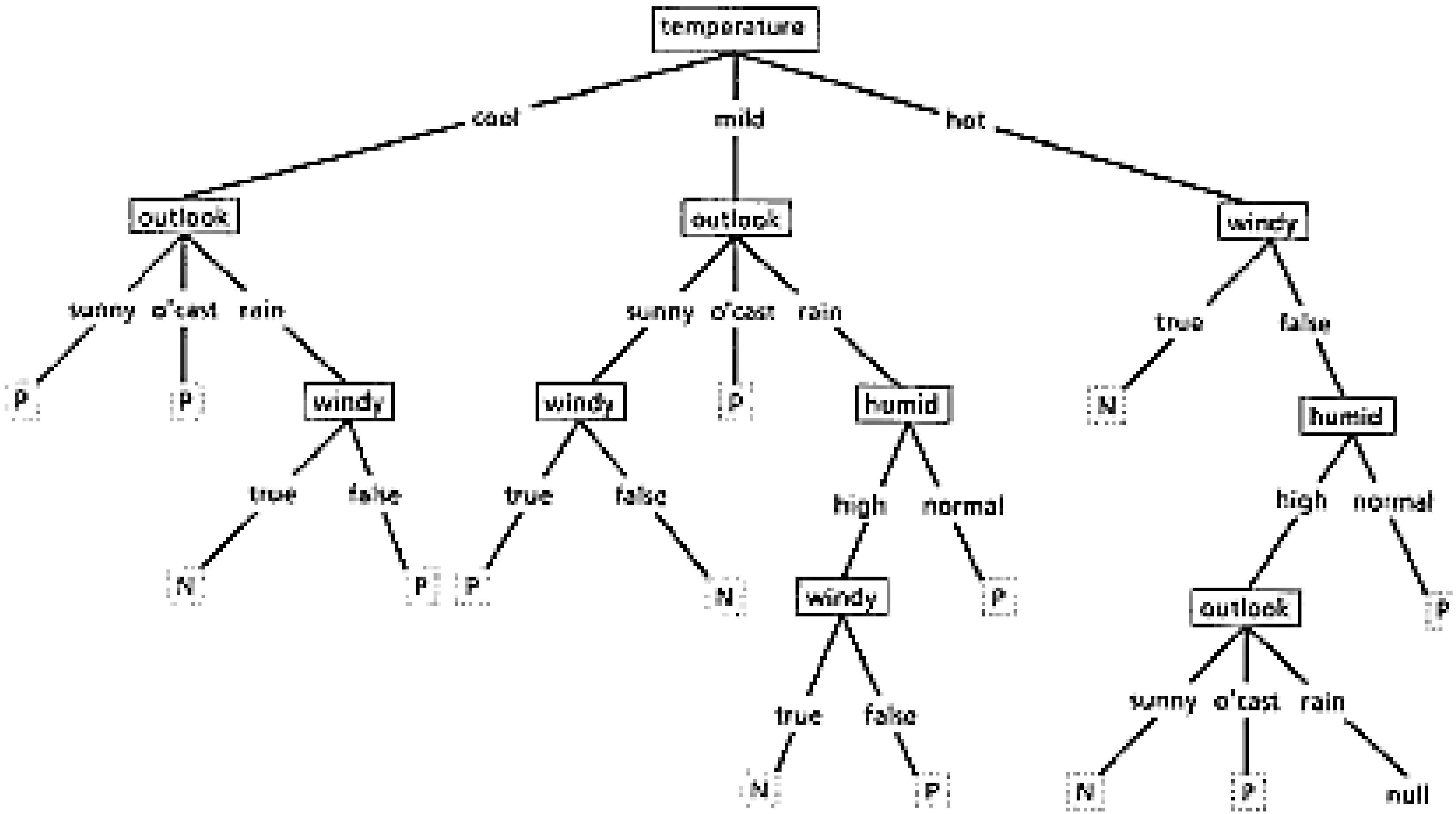



*Figure 3.* A complex decision tree.

# Attribute Selection Measure:

- Information Gain
- Gain Ratio
- Gini index

# Attribute Selection Measure: Information Gain

- **ID3** uses **information gain** as its attribute selection measure
- Let Node **N** holds tuples of partition D.
- Select the attribute with the highest information gain as the splitting attributes for Node **N**

# Attribute Selection Measure: Information Gain

- Expected information (entropy) needed to classify a tuple in D:

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

  - Let $p_i$ be the probability that an arbitrary tuple in D belongs to class $C_i$ (distinct class) estimated by $|C_{i,D}|/|D|$
  - $C_{i,D}$ be the set of tuples of class Ci in D.
  - $|D|$ and $|C_{i,D}|$ denote the number of tuples in D and $C_i$ respectively.
  - ***Info(D)*** is also known as **entropy of D**

# Attribute Selection Measure: Information Gain (ID3)

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

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

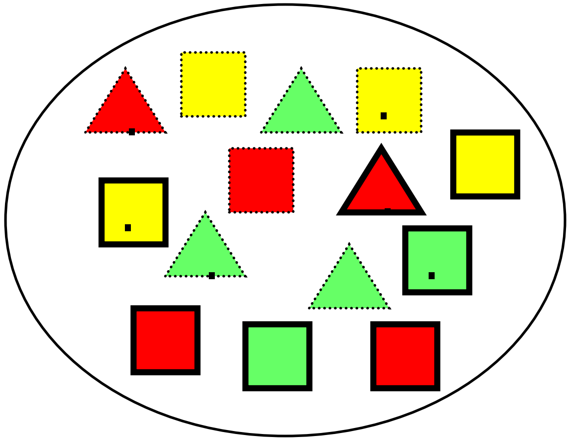
- Information gained by branching on attribute A

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

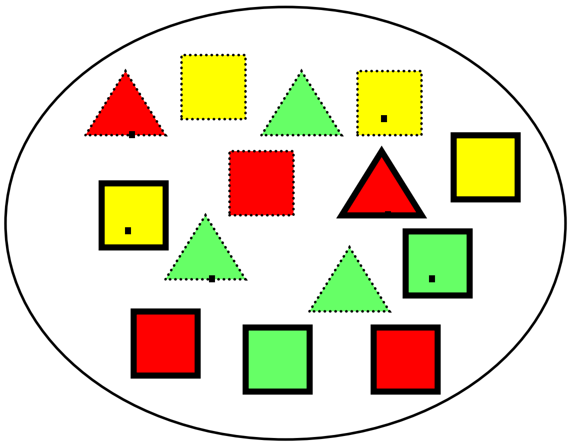
# Triangles and Squares

| # | Attribute | | | Shape |
|---|---|---|---|---|
| | Color | Outline | Dot | |
| 1 | green | dashed | no | triange |
| 2 | green | dashed | yes | triange |
| 3 | yellow | dashed | no | square |
| 4 | red | dashed | no | square |
| 5 | red | solid | no | square |
| 6 | red | solid | yes | triange |
| 7 | green | solid | no | square |
| 8 | green | dashed | no | triange |
| 9 | yellow | solid | yes | square |
| 10 | red | solid | no | square |
| 11 | green | solid | yes | square |
| 12 | yellow | dashed | yes | square |
| 13 | yellow | solid | no | square |
| 14 | red | dashed | yes | triange |

Data Set:
A set of classified objects



# Entropy



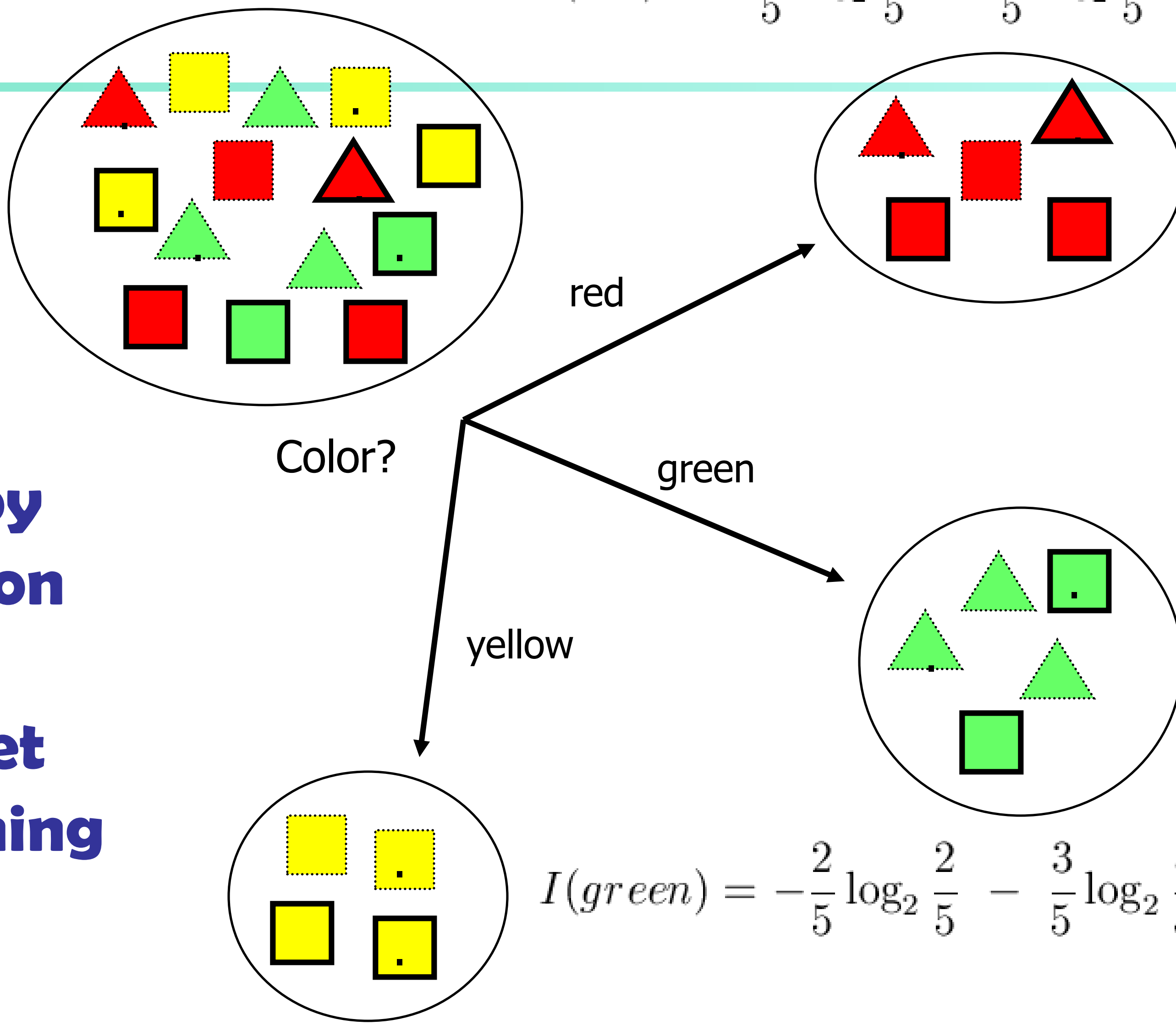
- 5 triangles
- 9 squares
- class probabilities

- entropy

$$I = -\frac{9}{14}\log_2\frac{9}{14} \; - \; \frac{5}{14}\log_2\frac{5}{14} = 0.940 \text{ bits}$$

# Entropy reduction by data set partitioning

$$I(red) = -\frac{3}{5}\log_2\frac{3}{5} \;-\; \frac{2}{5}\log_2\frac{2}{5} = 0.971 \text{ bits}$$

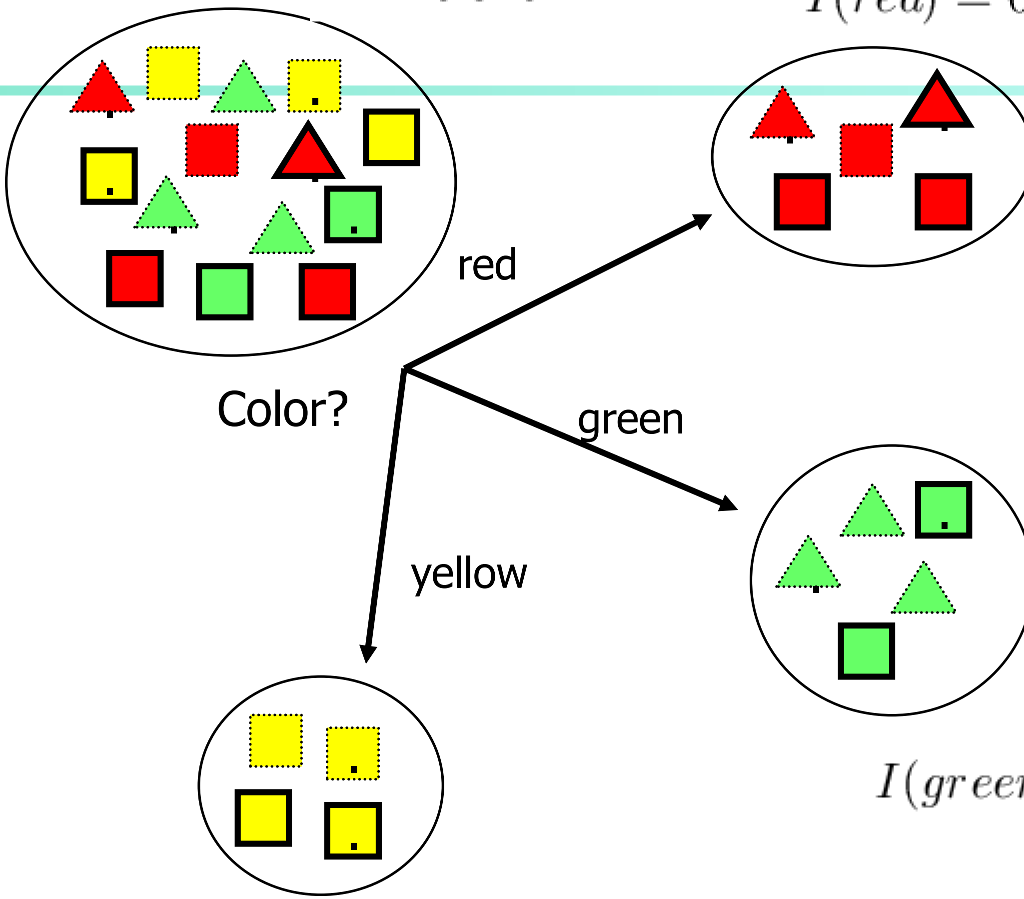



$$I(green) = -\frac{2}{5}\log_2\frac{2}{5} \;-\; \frac{3}{5}\log_2\frac{3}{5} = 0.971 \text{ bits}$$

$$I(yellow) = -\frac{4}{4}\log_2\frac{4}{4} \;-\; \frac{0}{4}\log_2\frac{0}{4} = 0.0 \text{ bits}$$

$I = 0.940$

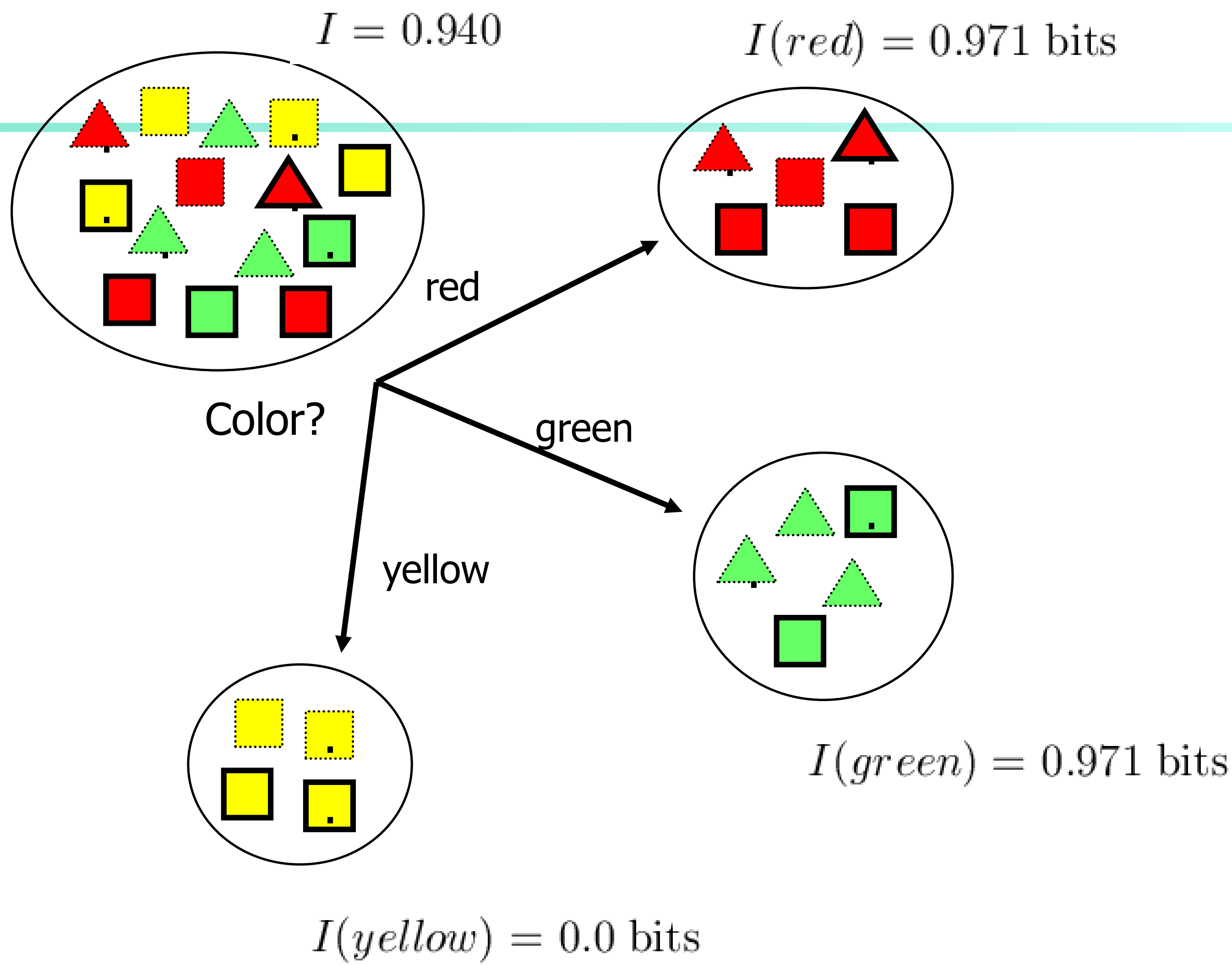
$I(red) = 0.971$ bits




$I(green) = 0.971$ bits

$I(yellow) = 0.0$ bits

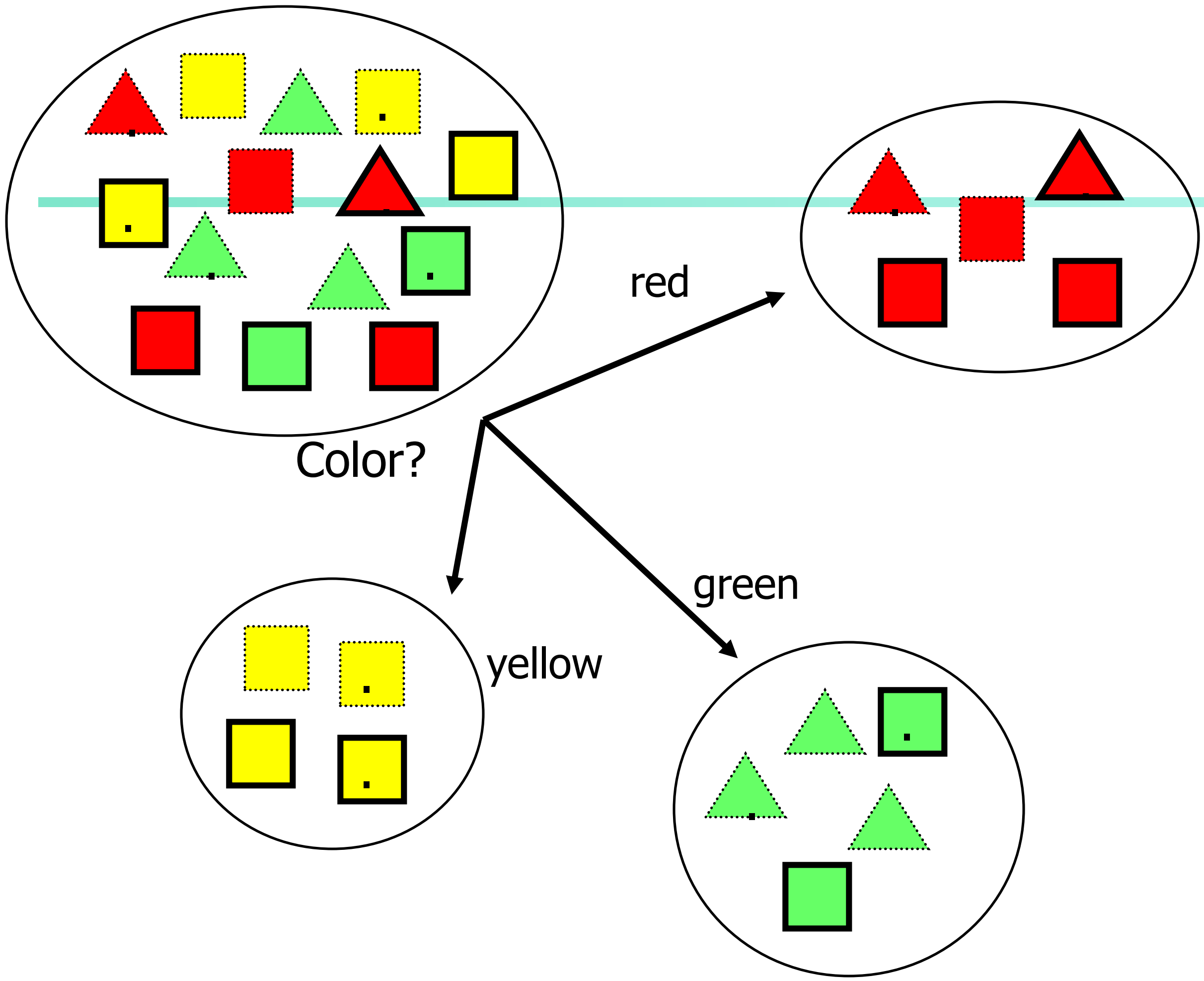
# Information Gain




$$\boxed{Gain(\text{Color}) = I - I_{res}(\text{Color}) = 0.940 - 0.694 = 0.246\ bits}$$

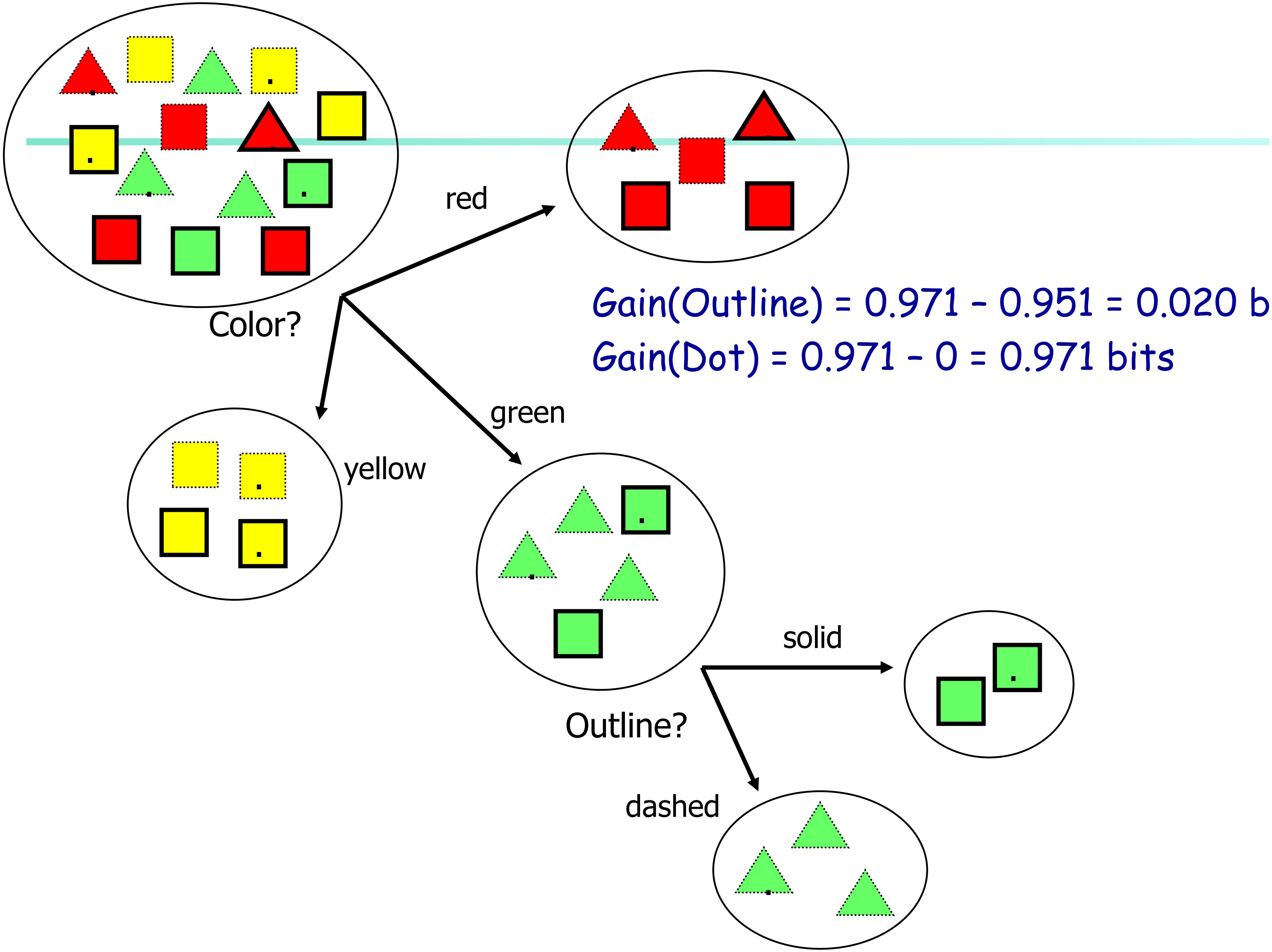
# Information Gain of The Attribute

- Attributes
  - Gain(Color) = 0.246
  - Gain(Outline) = 0.151
  - Gain(Dot) = 0.048
- Heuristics: attribute with the highest gain is chosen
- This heuristics is local (local minimization of impurity)



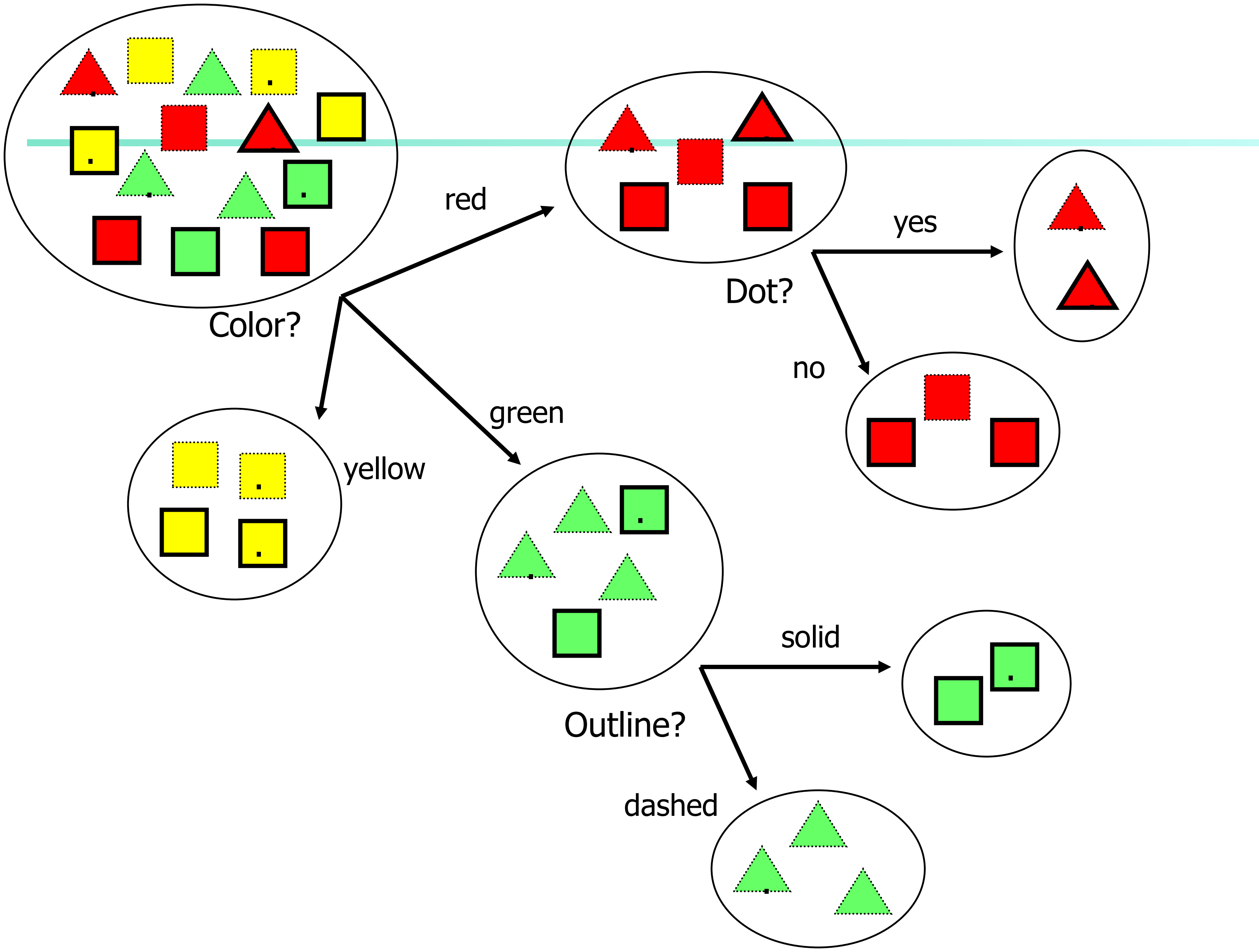

Gain(Outline) = 0.971 – 0 = 0.971 bits

Gain(Dot) = 0.971 – 0.951 = 0.020 bits

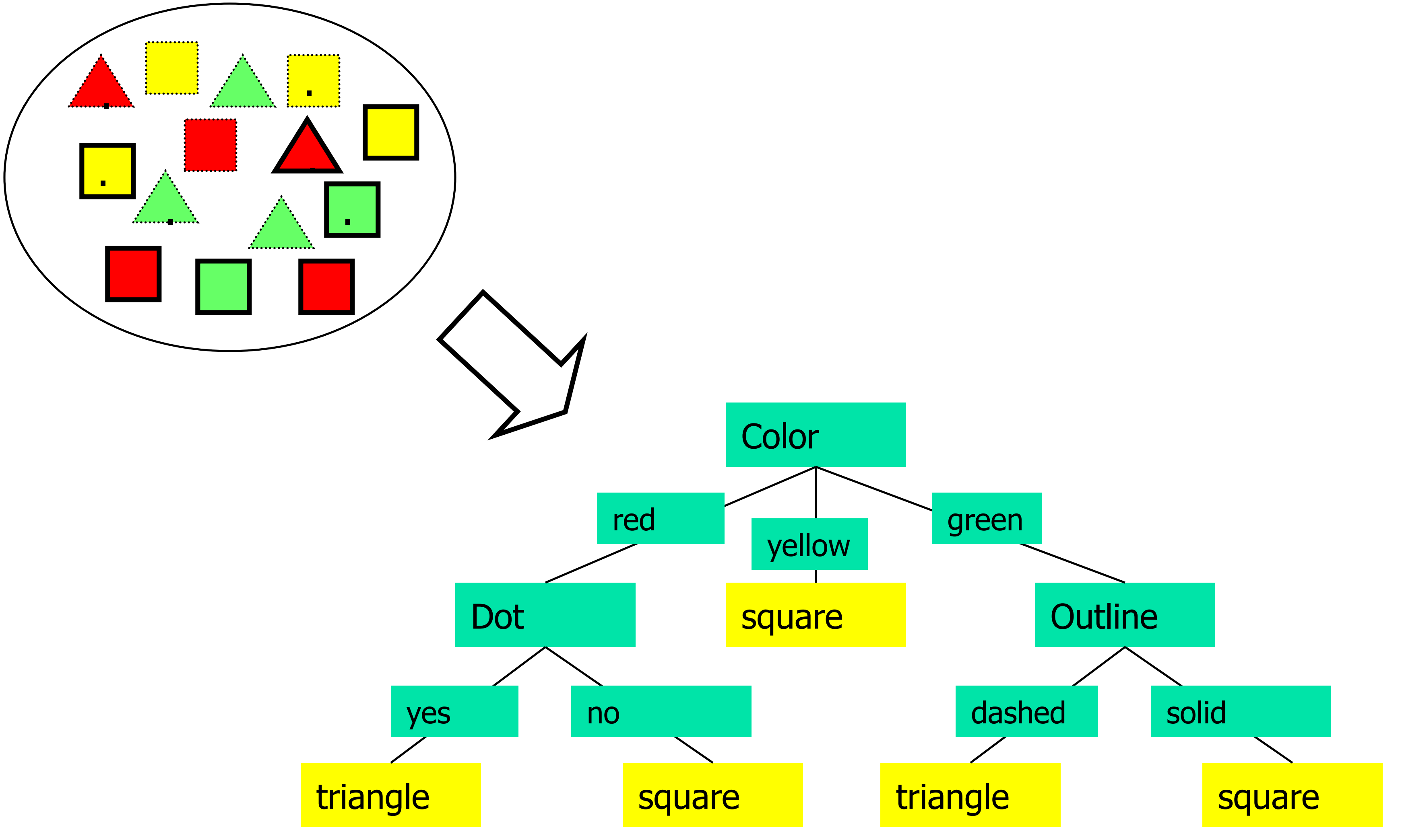



Gain(Outline) = 0.971 – 0.951 = 0.020 bits

Gain(Dot) = 0.971 – 0 = 0.971 bits


Color?
red
yellow
green
Dot?
yes
no
Outline?
solid
dashed

# Decision Tree



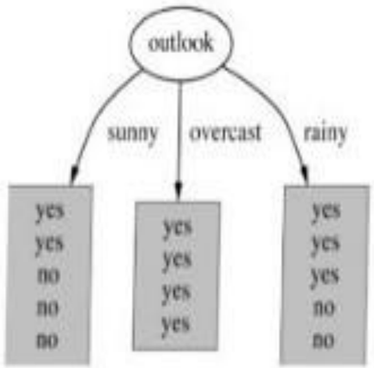
# Steps to solve ID3 Sum

- compute the entropy for data-set
- for every attribute/feature:

  1.calculate entropy for all categorical values

  2.take average information entropy for the current attribute

  3.calculate gain for the current attribute
- pick the highest gain attribute.
- Repeat until we get the tree we desired.

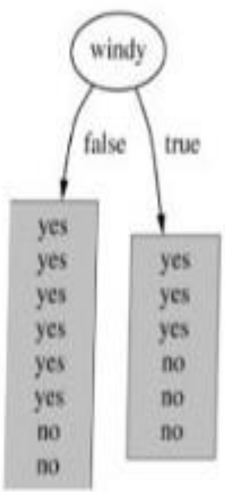



$$\left.\begin{array}{l}\text{E (Outlook=sunny)} = -\frac{2}{5}\log\left(\frac{2}{5}\right) - \frac{3}{5}\log\left(\frac{3}{5}\right) = 0.971 \\ \text{E (Outlook=overcast)} = -1\log(1) - 0\log(0) = 0 \\ \text{E (Outlook=rainy)} = -\frac{3}{5}\log\left(\frac{3}{5}\right) - \frac{2}{5}\log\left(\frac{2}{5}\right) = 0.971\end{array}\right\} \text{H(S,Outlook)}$$

Average Entropy information for Outlook

$$\left. I\text{ (Outlook)} = \frac{5}{14} * 0.971 + \frac{4}{14} * 0 + \frac{5}{14} * 0.971 = 0.693 \right\} \sum_{t \in T} p(t)H(t)$$

Gain (Outlook) = E(S) – I (outlook) = 0.94-.693 = 0.247 $\Rightarrow$ $IG(A,S) = H(S) - \sum_{t \in T} p(t)H(t)$




$$\text{E (Windy=false)} = -\frac{6}{8}\log\left(\frac{6}{8}\right) - \frac{2}{8}\log\left(\frac{2}{8}\right) = 0.811$$

$$\text{E (Windy=true)} = -\frac{3}{6}\log\left(\frac{3}{6}\right) - \frac{3}{6}\log\left(\frac{3}{6}\right) = 1$$

Average entropy information for Windy

$$I\text{ (Windy)} = \frac{8}{14} * 0.811 + \frac{6}{14} * 1 = 0.892$$

Gain (Windy) = E(S) – $I$ (Windy) = 0.94-0.892=0.048

# Attribute Selection: Information Gain

| 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 |

# Attribute Selection: Information Gain

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

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

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

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

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

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

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

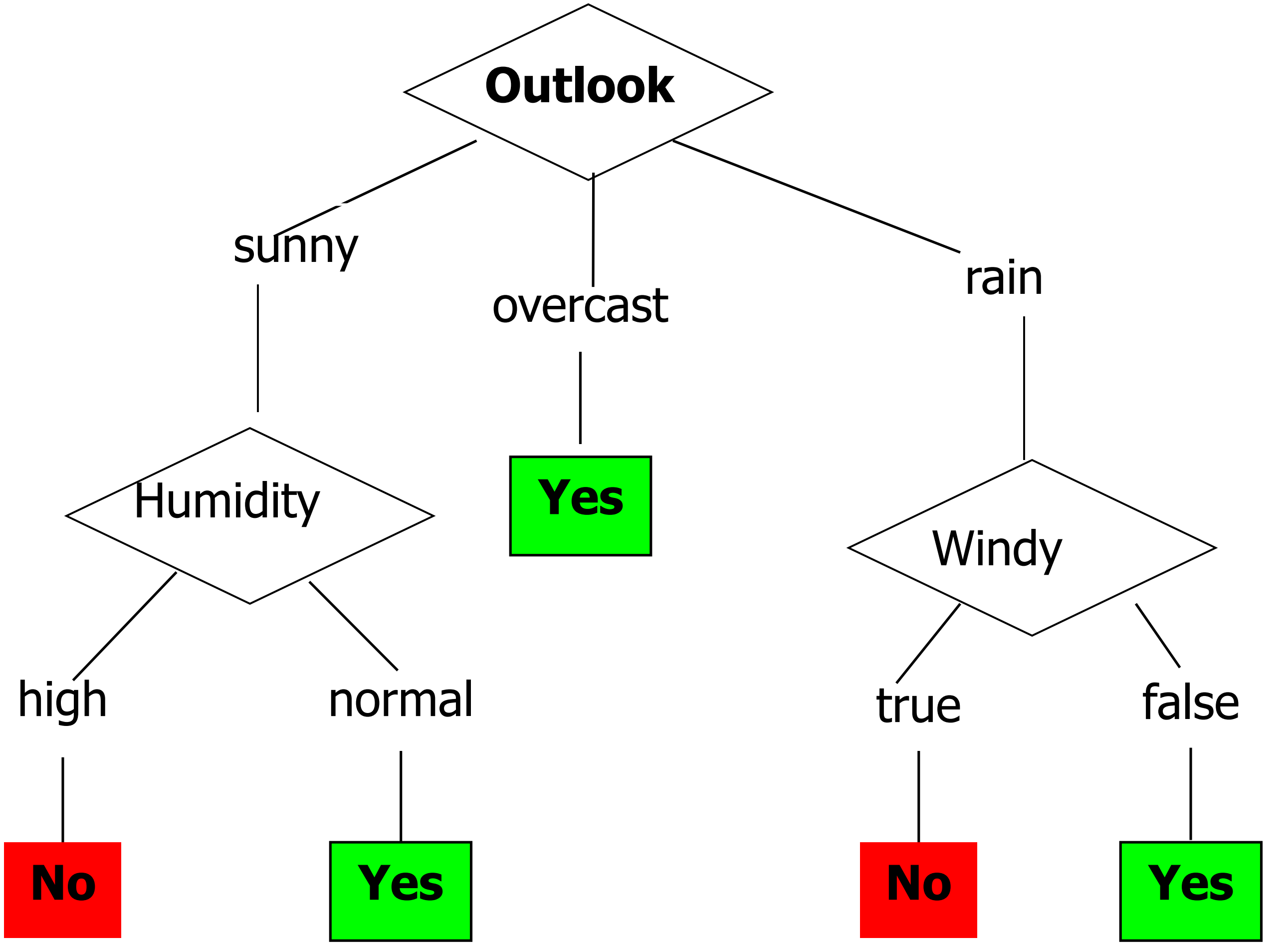
Similarly,

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

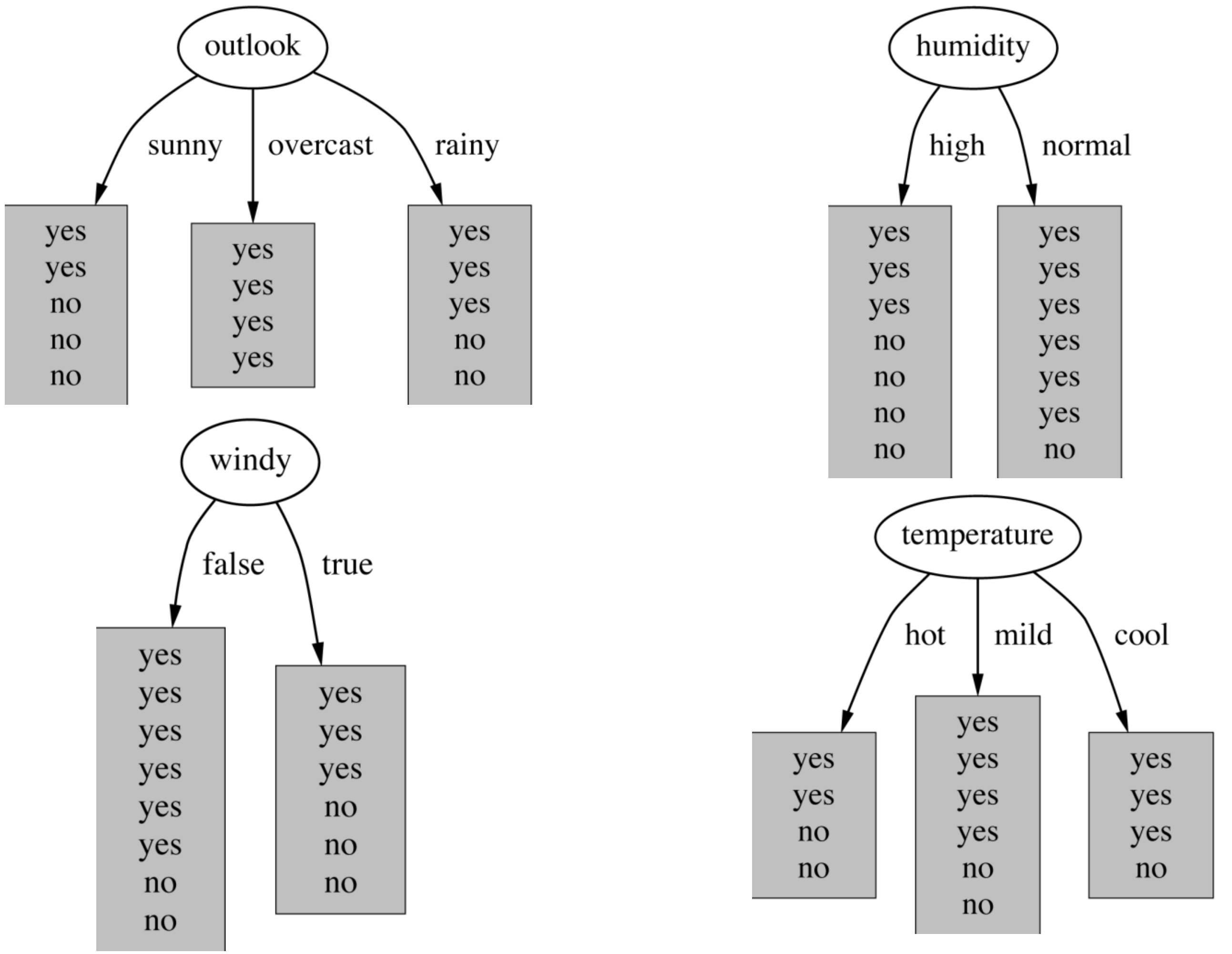
# Weather Data: Play or not Play?

| Outlook | Temperature | Humidity | Windy | Play? |
|---|---|---|---|---|
| sunny | hot | high | false | No |
| sunny | hot | high | true | No |
| overcast | hot | high | false | Yes |
| rain | mild | high | false | Yes |
| rain | cool | normal | false | Yes |
| rain | cool | normal | true | No |
| overcast | cool | normal | true | Yes |
| sunny | mild | high | false | No |
| sunny | cool | normal | false | Yes |
| rain | mild | normal | false | Yes |
| sunny | mild | normal | true | Yes |
| overcast | mild | high | true | Yes |
| overcast | hot | normal | false | Yes |
| rain | mild | high | true | No |

# Example Tree for "Play?"



# Which attribute to select?



# Example: attribute “Outlook”

- “Outlook” = “Sunny”:

$$\text{info}([2,3]) = \text{entropy}(2/5, 3/5) = -2/5\log(2/5) - 3/5\log(3/5) = 0.971\,\text{bits}$$

- “Outlook” = “Overcast”:

$$\text{info}([4,0]) = \text{entropy}(1,0) = -1\log(1) - 0\log(0) = 0\,\text{bits}$$

***Note: log(0) is not defined, but we evaluate 0*log(0) as zero***

- “Outlook” = “Rainy”:

$$\text{info}([3,2]) = \text{entropy}(3/5, 2/5) = -3/5\log(3/5) - 2/5\log(2/5) = 0.971\,\text{bits}$$

- Expected information for attribute:

$$\text{info}([3,2],[4,0],[3,2]) = (5/14)\times 0.971 + (4/14)\times 0 + (5/14)\times 0.971$$
$$= 0.693\,\text{bits}$$

# Computing the information gain

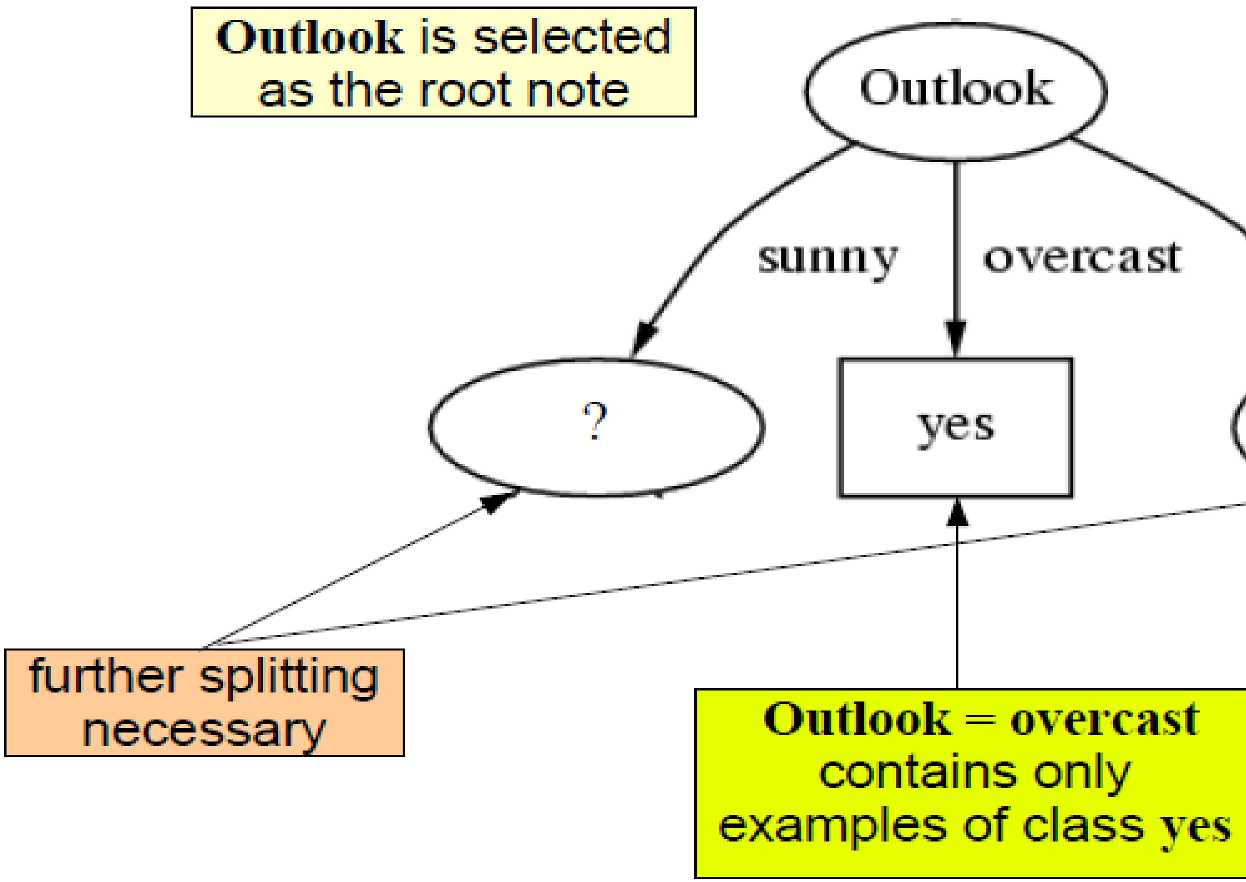
- Information gain:

(information before split) – (information after split)

$$\text{gain}(\text{"Outlook"}) = \text{info}([9,5]) - \text{info}([2,3],[4,0],[3,2]) = 0.940 - 0.693$$
$$= 0.247 \text{ bits}$$

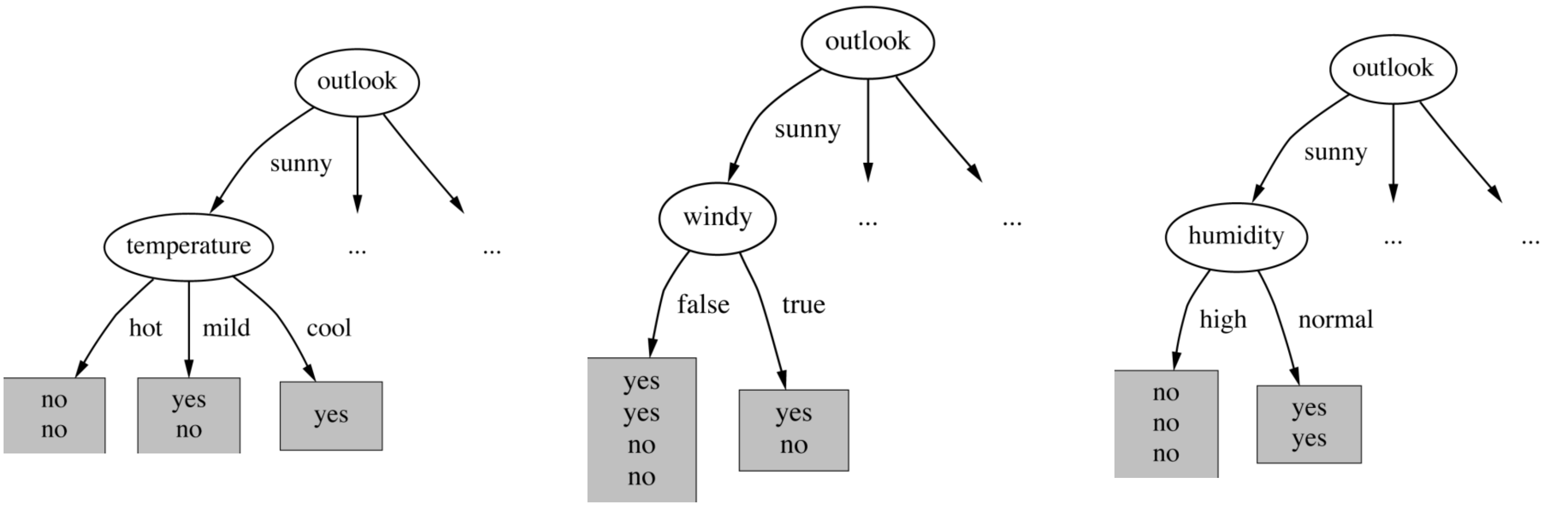
- Information gain for attributes from weather data:

$$\text{gain}(\text{"Outlook"}) = 0.247 \text{ bits}$$
$$\text{gain}(\text{"Temperature"}) = 0.029 \text{ bits}$$
$$\text{gain}(\text{"Humidity"}) = 0.152 \text{ bits}$$
$$\text{gain}(\text{"Windy"}) = 0.048 \text{ bits}$$



**Outlook** is selected as the root note
Outlook
sunny
overcast
rain
?
yes
?
further splitting necessary
**Outlook = overcast** contains only examples of class **yes**

# Continuing to split

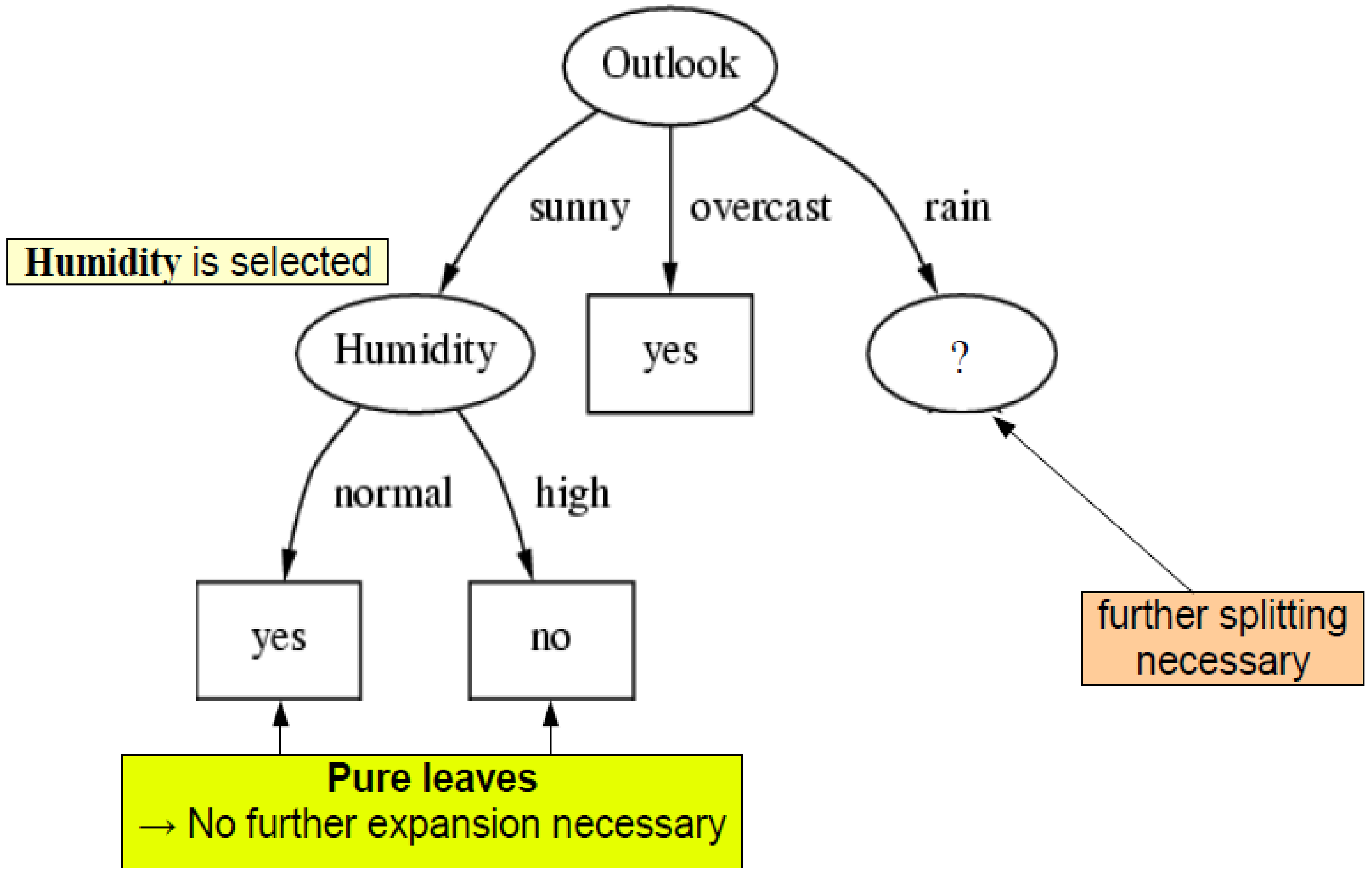



$$\text{gain}(\text{"Humidity"}) = 0.971\text{bits}$$

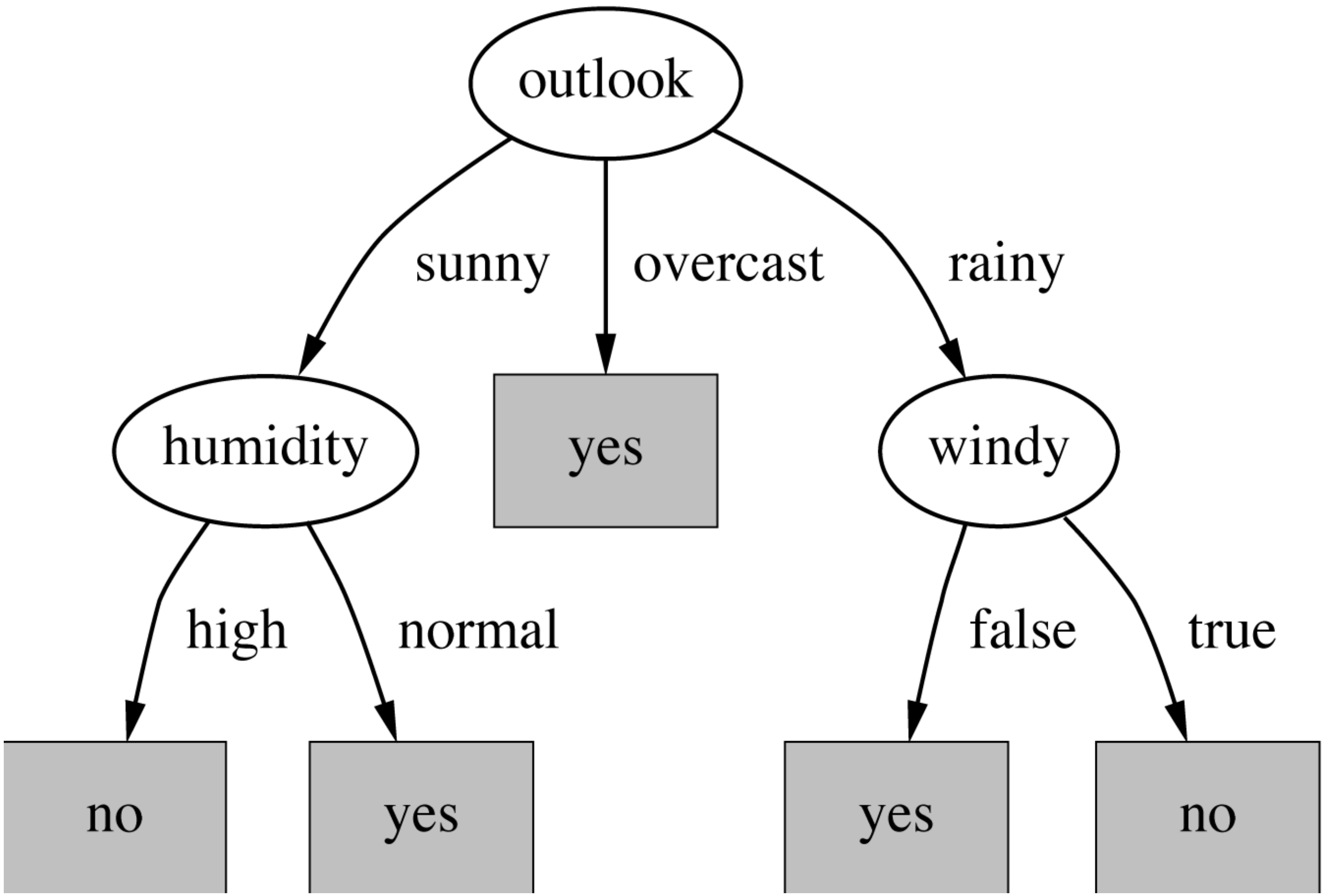
$$\text{gain}(\text{"Temperature"}) = 0.571\text{bits}$$

$$\text{gain}(\text{"Windy"}) = 0.020\text{bits}$$

**Humidity is selected**


Outlook
sunny
overcast
rain
Humidity is selected
Humidity
yes
?
normal
high
yes
no
further splitting necessary
Pure leaves
→ No further expansion necessary

# The final decision tree



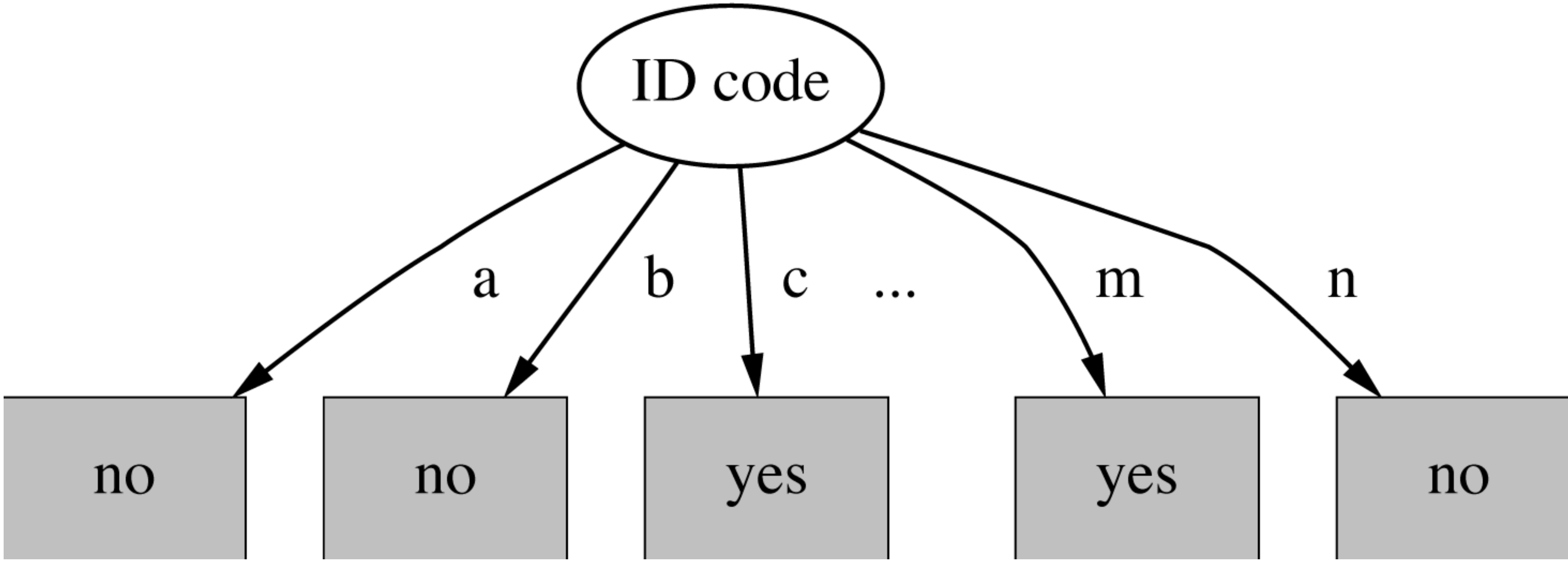
# Highly-branching attributes

- Problematic: attributes with a large number of values (extreme case: ID code)
- Subsets are more likely to be pure if there is a large number of values
  - ⇒ Information gain is biased towards choosing attributes with a large number of values

# Weather Data with ID code

| ID | Outlook | Temperature | Humidity | Windy | Play? |
|---|---|---|---|---|---|
| A | sunny | hot | high | false | No |
| B | sunny | hot | high | true | No |
| C | overcast | hot | high | false | Yes |
| D | rain | mild | high | false | Yes |
| E | rain | cool | normal | false | Yes |
| F | rain | cool | normal | true | No |
| G | overcast | cool | normal | true | Yes |
| H | sunny | mild | high | false | No |
| I | sunny | cool | normal | false | Yes |
| J | rain | mild | normal | false | Yes |
| K | sunny | mild | normal | true | Yes |
| L | overcast | mild | high | true | Yes |
| M | overcast | hot | normal | false | Yes |
| N | rain | mild | high | true | No |

# Split for ID Code Attribute




Entropy of split = 0 (since each leaf node is "pure", having only one case.

Information gain is maximal for ID code

# Gain ratio

- *Gain ratio*: a modification of the information gain that reduces its bias on high-branch attributes
- Gain ratio should be
  - Large when data is evenly spread
  - Small when all data belong to one branch
- Gain ratio takes number and size of branches into account when choosing an attribute
  - It corrects the information gain by taking the *intrinsic information* of a split into account (i.e. how much info do we need to tell which branch an instance belongs to)

# Gain Ratio and Intrinsic Info.

- Intrinsic information: entropy of distribution of instances into branches

$$IntrinsicInfo(S,A) \equiv -\sum \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}.$$

- *Gain ratio* (Quinlan'86) normalizes info gain by:

$$GainRatio(S,A) = \frac{Gain(S,A)}{IntrinsicInfo(S,A)}.$$

# Computing the gain ratio

- Example: intrinsic information for ID code

$$\text{info}([1,1,\ldots,1]) = 14 \times (-1/14 \times \log 1/14) = 3.807 \text{bits}$$

- **Importance of attribute decreases as intrinsic information gets larger**
- Example of gain ratio:

$$\text{gain\_ratio}(\text{"Attribute"}) = \frac{\text{gain}(\text{"Attribute"})}{\text{intrinsic\_info}(\text{"Attribute"})}$$

- Example:

$$\text{gain\_ratio}(\text{"ID\_code"}) = \frac{0.940 \text{bits}}{3.807 \text{bits}} = 0.246$$

# Gain ratios for weather data

| **Outlook** | | **Temperature** | |
|---|---|---|---|
| Info: | 0.693 | Info: | 0.911 |
| Gain: 0.940-0.693 | 0.247 | Gain: 0.940-0.911 | 0.029 |
| Split info: info([5,4,5]) | 1.577 | Split info: info([4,6,4]) | 1.362 |
| Gain ratio: 0.247/1.577 | 0.156 | Gain ratio: 0.029/1.362 | 0.021 |

| **Humidity** | | **Windy** | |
|---|---|---|---|
| Info: | 0.788 | Info: | 0.892 |
| Gain: 0.940-0.788 | 0.152 | Gain: 0.940-0.892 | 0.048 |
| Split info: info([7,7]) | 1.000 | Split info: info([8,6]) | 0.985 |
| Gain ratio: 0.152/1 | 0.152 | Gain ratio: 0.048/0.985 | 0.049 |

# More on the gain ratio

- "Outlook" still comes out top
- However: "ID code" has greater gain ratio
  - Standard fix: *ad hoc* test to prevent splitting on that type of attribute
- Problem with gain ratio: it may overcompensate
  - May choose an attribute just because its intrinsic information is very low
  - Standard fix:
    - First, only consider attributes with greater than average information gain
    - Then, compare them on gain ratio

# CART Splitting Criteria: Gini Index

- The Gini Index (used in CART) measures the impurity of a data partition **D**
- 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.

# Gini Index

- The Gini Index considers a **binary split** for each attribute **A** say T1 and T2.

  After splitting T into two subsets T1 and T2 with sizes N1 and N2, the gini index of the split data is defined as

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

**OR**

$$gini_{split}(D) = \frac{N_1}{N} gini(D1) + \frac{N_2}{N} gini(D2)$$

- The attribute providing smallest $gini_{split}(T)$ is chosen to split the node.

# Gini Index

- The reduction in impurity is given by

  $$\Delta\, Gini\,(A) = gini\,(T) - gini_{split}(T)$$

- The attribute that maximizes the reduction in impurity is chosen as the splitting attribute

# Weather Data with ID code

| ID | Outlook | Temperature | Humidity | Windy | Play? |
|---|---|---|---|---|---|
| A | sunny | hot | high | false | No |
| B | sunny | hot | high | true | No |
| C | overcast | hot | high | false | Yes |
| D | rain | mild | high | false | Yes |
| E | rain | cool | normal | false | Yes |
| F | rain | cool | normal | true | No |
| G | overcast | cool | normal | true | Yes |
| H | sunny | mild | high | false | No |
| I | sunny | cool | normal | false | Yes |
| J | rain | mild | normal | false | Yes |
| K | sunny | mild | normal | true | Yes |
| L | overcast | mild | high | true | Yes |
| M | overcast | hot | normal | false | Yes |
| N | rain | mild | high | true | No |

# Gini Index

- **Compute the Gini index of the training set D:** 9 tuples in class yes and 5 in class no

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

- **Using attribute temp:** there are three values: **cool, mild** and **hot**
- **Choosing the subset {cool, mild} results in two partions:**
- **D1 (temperature∈ {cool, mild} ):** 10 tuples
- **D2 (income ∈ {hot} ):** 4 tuples

# Gini Index

$\text{Gini}_{\text{temperature} \in \{\text{cool,mild}\} \text{ and } \{\text{hot}\}}$ (T)

$$= \frac{10}{14} Gini(D_1) + \frac{4}{14} Gini(D_2)$$

$$= \frac{10}{14}\left(1 - \left(\frac{6}{10}\right)^2 - \left(\frac{4}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{1}{4}\right)^2 - \left(\frac{3}{4}\right)^2\right)$$

$$= 0.450$$

- **The Gini Index measures of the remaining partitions are:**
- $\text{Gini}_{\text{temperature} \in \{\text{cool,hot}\} \text{ and} \{\text{mild}\}}$ (T)= 0.300
- $\text{Gini}_{\text{temperature} \in \{\text{mild,hot}\} \text{ and} \{\text{cool}\}}$ (T)= 0.315
- **Therefore, the best binary split for attribute temp is on { mild,hot} and {cool}**

$$\Delta\ Gini\ (A) = gini\ (T) - gini_{split}(T)$$

$$\text{Gini}_{\text{temperature} \in \{\text{cool,mild}\} \text{ and } \{\text{hot}\}} = 0.459 - 0.450$$

# Comparing Attribute Selection Measures

- The three measures, in general, return good results but
  - **Information gain**:
    - biased towards multivalued attributes
  - **Gini index**:
    - biased to multivalued attributes
    - has difficulty when # of classes is large

# Bayesian Classification

# Bayesian Classification

- The Bayesian Classification represents a supervised learning method as well as a statistical method for classification.
- It allows us to capture uncertainty about the model in a principled way by determining probabilities of the outcomes.
- It can solve diagnostic and predictive problems.
- This Classification is named after Thomas Bayes ( 1702-1761), who proposed the Bayes Theorem.
- Bayesian classification provides practical learning algorithms and prior knowledge and observed data can be combined.

# Bayes' Theorem

- Let ***X*** be a data tuple.
- In Bayesian terms, ***X*** is considered "evidence."
- Let *H* be some hypothesis, such as 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***

# Bayes' Theorem

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

P(H |**X**) is the posterior probability, or a posteriori probability, of H conditioned on **X**.
P(H) is the prior probability, or a priori probability, of H.
P(**X**|H) is the posterior probability of **X** conditioned on H.

# Naïve Bayesian Classification

- 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, $\boldsymbol{X} = (x1, x2, \ldots, xn)$, depicting $n$ measurements made on the tuple from $n$ attributes, respectively, $A1, A2, \ldots, An$.
- Suppose that there are $m$ classes, $C1, C2, \ldots, Cm$. 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

$$P(Ci|\boldsymbol{X}) > P(Cj|\boldsymbol{X})$$

assign to sample X the class label C such that

**P(C|X) is maximal**

# Naïve Bayesian Classification

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

- P(X) is constant for all classes
- P(C) = relative freq of class C samples
- C such that P(C|X) is maximum =

  C such that P(X|C)·P(C) is maximum

$$P(C_i|X) = P(X|C_i)P(C_i)$$

| Weather | Play |
|---|---|
| Sunny | No |
| Overcast | Yes |
| Rainy | Yes |
| Sunny | Yes |
| Sunny | Yes |
| Overcast | Yes |
| Rainy | No |
| Rainy | No |
| Sunny | Yes |
| Rainy | Yes |
| Sunny | No |
| Overcast | Yes |
| Overcast | Yes |
| Rainy | No |

| Frequency Table | | |
|---|---|---|
| Weather | No | Yes |
| Overcast | | 4 |
| Rainy | 3 | 2 |
| Sunny | 2 | 3 |
| Grand Total | 5 | 9 |

| Likelihood table | | | | |
|---|---|---|---|---|
| Weather | No | Yes | | |
| Overcast | | 4 | =4/14 | 0.29 |
| Rainy | 3 | 2 | =5/14 | 0.36 |
| Sunny | 2 | 3 | =5/14 | 0.36 |
| All | 5 | 9 | | |
| | =5/14 | =9/14 | | |
| | 0.36 | 0.64 | | |

# Naïve Bayesian Classification

- Class prior probabilities is given by

$$P(Ci)=|C_{i,D}|/|D|,$$

where )=$|C_{i,D}|$ is the number of training tuples of class Ci in D.

- To compute P(**X**|Ci).

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

  - We can easily estimate the probabilities *P*(*x*1|*Ci*), *P*(*x*2j*Ci*), : : : , *P*(*xn*|*Ci*) from the training tuples

# Naïve Bayesian Classification

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

$$P(X|Ci)P(Ci) > P(\boldsymbol{X}|Cj)P(Cj) \text{ for } 1 <= j < m, j \neq i.$$

- In other words, the predicted class label is the class $Ci$ for which $P(\boldsymbol{X}\text{j}Ci)P(Ci)$ is the maximum.

# Summary

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

$$X = (x_1, x_2, x_3, \ldots\ldots, x_n)$$

$$P(y|x_1, \ldots, x_n) = \frac{P(x_1|y)P(x_2|y)\ldots P(x_n|y)P(y)}{P(x_1)P(x_2)\ldots P(x_n)}$$

For all entries in the dataset, the denominator does not change, it remains static. Therefore, the denominator can be removed and proportionality can be injected.

$$P(y|x_1, \ldots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i|y)$$

$$y = argmax_y P(y) \prod_{i=1}^{n} P(x_i|y)$$

# Naïve Bayesian Classification

- Numerical Illustration
- The dataset :

| Example No. | Color | Type | Origin | Stolen? |
|---|---|---|---|---|
| 1 | Red | Sports | Domestic | Yes |
| 2 | Red | Sports | Domestic | No |
| 3 | Red | Sports | Domestic | Yes |
| 4 | Yellow | Sports | Domestic | No |
| 5 | Yellow | Sports | Imported | Yes |
| 6 | Yellow | SUV | Imported | No |
| 7 | Yellow | SUV | Imported | Yes |
| 8 | Yellow | SUV | Domestic | No |
| 9 | Red | SUV | Imported | No |
| 10 | Red | Sports | Imported | Yes |

- Classify a Red Domestic SUV is getting stolen or not.

- The posterior probability **P(y|X)** can be calculated
  - Create a **Frequency Table** for each attribute against the target.
  - Convert the frequency tables to **Likelihood Tables**
  - Use the Naïve Bayesian equation to calculate the posterior probability for each class.
  - The class with the highest posterior probability is the outcome of the prediction.

❖ Below are the Frequency and likelihood tables for all three predictors.

Frequency Table

| | | Stolen? | |
|---|---|---|---|
| | | Yes | No |
| Color | Red | 3 | 2 |
| | Yellow | 2 | 3 |

Likelihood Table

| | | Stolen? | |
|---|---|---|---|
| | | P(Yes) | P(No) |
| Color | Red | 3/5 | 2/5 |
| | Yellow | 2/5 | 3/5 |

Frequency Table

| | | Stolen? | |
|---|---|---|---|
| | | Yes | No |
| Type | Sports | 4 | 2 |
| | SUV | 1 | 3 |

Likelihood Table

| | | Stolen? | |
|---|---|---|---|
| | | P(Yes) | P(No) |
| Type | Sports | 4/5 | 2/5 |
| | SUV | 1/5 | 3/5 |

Frequency Table

| | | Stolen? | |
|---|---|---|---|
| | | Yes | No |
| Origin | Domestic | 2 | 3 |
| | Imported | 3 | 2 |

Likelihood Table

| | | Stolen? | |
|---|---|---|---|
| | | P(Yes) | P(No) |
| Origin | Domestic | 2/5 | 3/5 |
| | Imported | 3/5 | 2/5 |

- Calculate the posterior probability P(Yes | X) as

$$P(Yes \mid X) = P(Red \mid Yes) * P(SUV \mid Yes) * P(Domestic \mid Yes) * P(Yes)$$

$$= \tfrac{3}{5} * \tfrac{1}{5} * \tfrac{2}{5} * 1$$

$$= 0.048$$

and, P(No | X):

$$P(No \mid X) = P(Red \mid No) * P(SUV \mid No) * P(Domestic \mid No) * P(No)$$

$$= \tfrac{2}{5} * \tfrac{3}{5} * \tfrac{3}{5} * 1$$

$$= 0.144$$

- Since 0.144 > 0.048, Which means given the features RED SUV and Domestic, our example gets classified as 'NO' the car is not stolen.

# Play example: estimating $P(x_i|C)$

| Outlook | Temperature | Humidity | Windy | Class |
|---|---|---|---|---|
| sunny | hot | high | false | N |
| sunny | hot | high | true | N |
| overcast | hot | high | false | P |
| rain | mild | high | false | P |
| rain | cool | normal | false | P |
| rain | cool | normal | true | N |
| overcast | cool | normal | true | P |
| sunny | mild | high | false | N |
| sunny | cool | normal | false | P |
| rain | mild | normal | false | P |
| sunny | mild | normal | true | P |
| overcast | mild | high | true | P |
| overcast | hot | normal | false | P |
| rain | mild | high | true | N |

| |
|---|
| **P(p) = 9/14** |
| **P(n) = 5/14** |

| | |
|---|---|
| **outlook** | |
| **P(sunny\|p) = 2/9** | **P(sunny\|n) = 3/5** |
| **P(overcast\|p) = 4/9** | **P(overcast\|n) = 0** |
| **P(rain\|p) = 3/9** | **P(rain\|n) = 2/5** |
| **temperature** | |
| **P(hot\|p) = 2/9** | **P(hot\|n) = 2/5** |
| **P(mild\|p) = 4/9** | **P(mild\|n) = 2/5** |
| **P(cool\|p) = 3/9** | **P(cool\|n) = 1/5** |
| **humidity** | |
| **P(high\|p) = 3/9** | **P(high\|n) = 4/5** |
| **P(normal\|p) = 6/9** | **P(normal\|n) = 2/5** |
| **windy** | |
| **P(true\|p) = 3/9** | **P(true\|n) = 3/5** |
| **P(false\|p) = 6/9** | **P(false\|n) = 2/5** |

# Play example: classifying X

- An unseen sample X = <rain, hot, high, false>

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

$$P(C_i|X) = P(X|C_i)P(C_i)$$

- P(X|p)·P(p) = P(rain|p)·P(hot|p)·P(high|p)·P(false|p)·P(p) = 3/9·2/9·3/9·6/9·9/14 = 0.010582
- P(X|n)·P(n) = P(rain|n)·P(hot|n)·P(high|n)·P(false|n)·P(n) = 2/5·2/5·4/5·2/5·5/14 = 0.018286
- Sample X is classified in class n (don't play)

- today = (Sunny, Hot, Normal, False)

$$P(Yes|today) = \frac{P(SunnyOutlook|Yes)P(HotTemperature|Yes)P(NormalHumidity|Yes)P(NoWind|Yes)P(Yes)}{P(today)}$$

$$P(No|today) = \frac{P(SunnyOutlook|No)P(HotTemperature|No)P(NormalHumidity|No)P(NoWind|No)P(No)}{P(today)}$$

# Apply Baysian Classification algo and construct classifier for following training set

| # | Gender | Car Ownership | Travel Cost | Income level | Transportation |
|---|---|---|---|---|---|
| 1 | male | 0 | cheap | low | bus |
| 2 | male | 1 | cheap | medium | bus |
| 3 | female | 1 | cheap | medium | train |
| 4 | female | 0 | cheap | low | bus |
| 5 | male | 1 | cheap | medium | bus |
| 6 | male | 0 | standard | medium | train |
| 7 | female | 1 | standard | medium | train |
| 8 | female | 1 | expensive | high | car |
| 9 | male | 2 | expensive | medium | car |
| 10 | female | 2 | expensive | high | car |

**Classify following Samples to correct transportation class**

| Name | Gender | Car OwnerShip | Travel Cost | Income Level | Transportation |
|---|---|---|---|---|---|
| Alex | Male | 1 | Standard | High | |
| Buddy | Male | 0 | Cheap | Medium | |
| Cherry | Female | 1 | Cheap | High | |

$$P(Yes|today) = \frac{P(SunnyOutlook|Yes)P(HotTemperature|Yes)P(NormalHumidity|Yes)P(NoWind|Yes)P(Yes)}{P(today)}$$

| | | | | | |
|---|---|---|---|---|---|
| Alex | Male | 1 | Standard | High | |

**P(bus|x)=P(gender|bus)*P(1|bus)*P(standard|bus)*P(High|bus)*P(bus)**

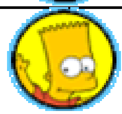
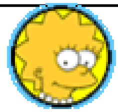
**=3/4*2/4*0/4*0/4*4/10**

**=0**

**P(bus|x)=P(gender|bus)*P(1|bus)*P(standard|bus)*P(High|bus)*P(bus)**

**=3/4*2/4*1/7*1/7*4/10**

**Laplacian correction or Laplace estimator**

| Person | Hair Length | Weight | Age | Class |
|---|---|---|---|---|
|  Homer | 0” | 250 | 36 | M |
|  Marge | 10” | 150 | 34 | F |
|  Bart | 2” | 90 | 10 | M |
|  Lisa | 6” | 78 | 8 | F |
|  Maggie | 4” | 20 | 1 | F |
|  Abe | 1” | 170 | 70 | M |
|  Selma | 8” | 160 | 41 | F |
|  Otto | 10” | 180 | 38 | M |
|  Krusty | 6” | 200 | 45 | M |
|  Comic | 8” | 290 | 38 | ? |

# Naïve Bayesian Classification

- 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 $D$.
- 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 $\mu$ and standard deviation s, defined by

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

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

# Overview of Naive Bayes

- The goal of Naive Bayes is to work out whether a new example is in a class given that it has a certain combination of attribute values. We work out the **likelihood of the example being in each class given the evidence (its attribute values)**, and take the highest likelihood as the classification.

- Bayes Rule: **E- Event has occurred**

$$P[H \mid E] = \frac{P[E \mid H].P[H]}{P[E]}$$

  - **P[H]** is called the **prior probability** (of the hypothesis).
    **P[H|E]** is called the **posterior probability** (of the hypothesis given the evidence)

# Overview of Naive Bayes

**For each class, *k*, work out:**

$$P[H_k \mid E] = \frac{P[E \mid H_k].P[H_k]}{P[E]}$$

- Our Hypotheses are:
  - **$H_1$: 'the example is in class A'**
  - **$H_2$: 'the example is in class B'** etc.
- Our Evidence is the attribute values of a particular new example that is presented:
  - **$E_1$=x : 'the example has value x for attribute $A_1$'**
  - **$E_2$=y : 'the example has value y for attribute $A_2$'**
  - **...**
  - **$E_n$=z : 'the example has value z for attribute $A_n$'**
  - Note that, *assuming the attributes are equally important* and *independent*, we estimate the **joint probability of that combination of attribute values** as:

$$P[E \mid H_k] = P[E_1 = x \mid H_k] \times P[E_2 = y \mid H_k] \times \ldots \times P[E_n = z \mid H_k]$$

- The goal is then to find the hypothesis **(i.e. the class k)** for which the value of **P[Hk|E]** is at a maximum.

# Overview of Naive Bayes

- For **categorical** variables we use simple proportions.

**$P[E_i=x|H_k]$ =**

$$\frac{\textbf{no. of training egs in class k having value x for attribute } A_i}{\textbf{number of training examples in class k}}$$

- For **continuous** variables we assume a normal (Gaussian) distribution, and use the mean ($\mu$) and standard deviation ($\sigma$) to compute the conditional probabilities.

$$P[E_i = x \mid H_k] = \frac{1}{\sqrt{2\prod \sigma_k}} e^{-\frac{x-\mu_k}{2\sigma_k^2}}$$

# Prediction

- Predicts unknown values, i.e., models continuous-valued functions
- Predicting the identity of one thing based purely on the description of another, related thing
- Based on the relationship between a thing that you can know and a thing you need to predict
- (Numerical) prediction is similar to classification
  - construct a model
  - use model to predict continuous or ordered value for a given input
- Prediction is different from classification
  - Classification refers to predict categorical class label
  - Prediction models continuous-valued functions

# Prediction

- Major method for prediction: regression
  - Regression analysis can be used to model the relationship between one or more *independent* or **predictor** variables and a *dependent* or **response** variable
- In data mining, the **predictor** variables are the attributes of interest describing the tuple.In general, the values of the predictor variables are known.
- The **response** variable is what we want to predict

# How Does it Differ From Classification?

- Predicted values are usually continuous whereas classifications are discreet.
- Predictions are often (but not always) about the future whereas classifications are about the present.
- Classification is more concerned with the input than the output.

# Prediction

- Regression analysis
  - Linear and multiple regression
  - Non-linear regression

predicts unknown or missing values, i.e., models continuous-valued functions

# Linear Regression

- **Linear regression :** Straight-line regression analysis involves a response variable, *y*, and a single predictor variable, *x*. It is the simplest form of regression, and models *y* as a linear function of *x*.

$$y = w_0 + w_1 x$$

where the variance of *y* is assumed to be constant, $w_0$ (y-intercept) and $w_1$ (slope) are regression coefficients

# Linear Regression

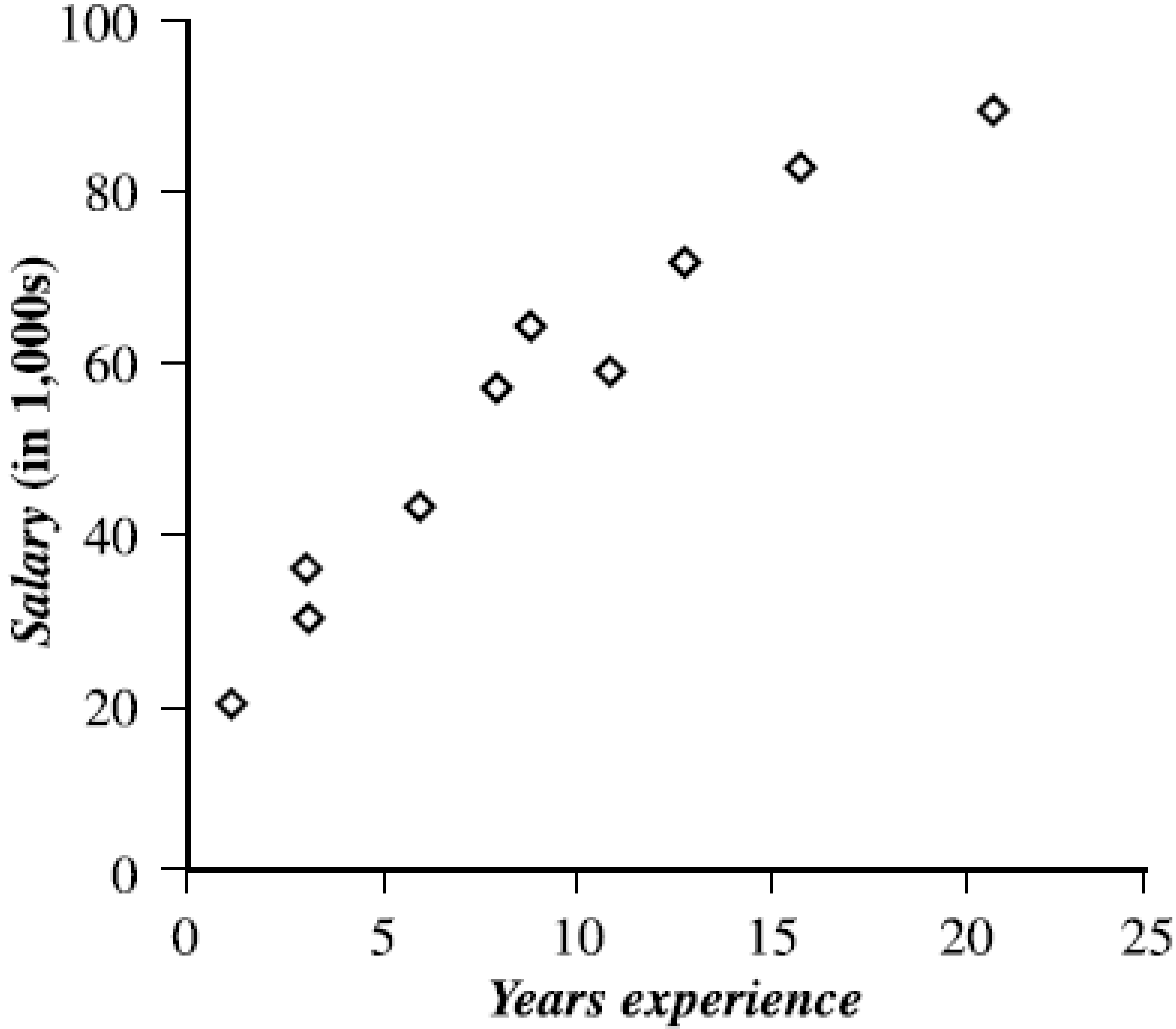
- **Method of least squares**:
- coefficients can be solved, which estimates the best-fitting straight line.
- Let *D* be a training set consisting of values of predictor variable, *x*, for some population and their associated values for response variable, *y*.
- The training set contains |*D*| data points of the form($x1$, $y1$), ($x2$, $y2$), : : : , ($x_{|D|}$ , $y_{|D|}$ ).
- The regression coefficients can be estimated using this method with the following equations:

$$w_0 = \bar{y} - w_1 \bar{x} \qquad w_1 = \frac{\sum_{i=1}^{|D|} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{|D|} (x_i - \bar{x})^2}$$

- where *x* is the mean value of $x1$, $x2$, : : : , $\mathbf{x}_{|D|}$ and *y* is the mean value of $y1$, $y2$, : : : , $\mathbf{y}_{|D|}$ .

# Example

| X(years of experience) | Y (salary) in thousands |
|---|---|
| 3 | 30 |
| 8 | 57 |
| 9 | 64 |
| 13 | 72 |
| 3 | 36 |
| 6 | 43 |
| 11 | 59 |
| 21 | 90 |
| 1 | 20 |
| 16 | 83 |


100
80
60
40
20
0
Salary (in 1,000s)
0
5
10
15
20
25
Years experience

| X(years of experience) | Y (salary) in thousands | xi-xmean | y-ymean | | |
|---|---|---|---|---|---|
| 3 | 30 | -6.1 | -25.4 | 154.94 | 37.21 |
| 8 | 57 | -1.1 | 1.6 | -1.76 | 1.21 |
| 9 | 64 | -0.1 | 8.6 | -0.86 | 0.01 |
| 13 | 72 | 3.9 | 16.6 | 64.74 | 15.21 |
| 3 | 36 | -6.1 | -19.4 | 118.34 | 37.21 |
| 6 | 43 | -3.1 | -12.4 | 38.44 | 9.61 |
| 11 | 59 | 1.9 | 3.6 | 6.84 | 3.61 |
| 21 | 90 | 11.9 | 34.6 | 411.74 | 141.61 |
| 1 | 20 | -8.1 | -35.4 | 286.74 | 65.61 |
| 16 | 83 | 6.9 | 27.6 | 190.44 | 47.61 |
| | | | | 1269.6 | 358.9 |
| | | | | | |
| | | | | w1 | 3.537476 |

# Example

- Xmean=9.1 Ymean=55.4

$$w_0 = \bar{y} - w_1 \bar{x} \qquad w_1 = \frac{\sum_{i=1}^{|D|} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{|D|} (x_i - \bar{x})^2}$$

$$w_1 = \frac{(3-9.1)(30-55.4)+(8-9.1)(57-55.4)+\cdots+(16-9.1)(83-55.4)}{(3-9.1)^2+(8-9.1)^2+\cdots+(16-9.1)^2} = 3.5$$

$$w_0 = 55.4 - (3.5)(9.1) = 23.6$$

Equation of line is: Y=23.6+3.5x

if x=10 years then Salary=?

# Linear Regression

- **Multiple linear regression**: involves more than one predictor variable
  - Training data is of the form ($\mathbf{X_1}$, $y_1$), ($\mathbf{X_2}$, $y_2$),…, ($\mathbf{X_{|D|}}$, $y_{|D|}$)
  - Ex. we may have: $y = w_0 + w_1 x_1 + w_2 x_2$
  - Solvable by extension of least square method or using SAS, S-Plus.
  - Many nonlinear functions can be transformed into the above

# Nonlinear Regression

- Some nonlinear models can be modeled by a polynomial function
- A polynomial regression model can be transformed into linear regression model. For example,

  $$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3$$

  convertible to linear with new variables: $x_2 = x^2$, $x_3 = x^3$

  $$y = w_0 + w_1 x + w_2 x_2 + w_3 x_3$$

- Other functions, such as power function, can also be transformed to linear model
- Some models are intractable nonlinear (e.g., sum of exponential terms)
  - possible to obtain least square estimates through extensive calculation on more complex formulae

# Other Regression-Based Models

- Generalized linear model:
  - Logistic regression: models the prob. of some event occurring as a linear function of a set of predictor variables
  - Poisson regression: models the data that exhibit a Poisson distribution
- Log-linear models: (for categorical data)
  - Approximate discrete multidimensional prob. distributions
  - Also useful for data compression and smoothing
- Regression trees and model trees
  - Trees to predict continuous values rather than class labels

# Accuracy and error measures

# Classifier Evaluation Metrics: Confusion Matrix

**Confusion Matrix:**

| Actual class\Predicted class | $C_1$ | $\neg C_1$ |
|---|---|---|
| $C_1$ | **True Positives (TP)** | **False Negatives (FN)** |
| $\neg C_1$ | **False Positives (FP)** | **True Negatives (TN)** |

- Given *m* classes, an entry, $\boldsymbol{CM_{i,j}}$ in a **confusion matrix** indicates # of tuples in class *i* that were labeled by the classifier as class *j*
- May have extra rows/columns to provide totals

| A\P | C | ¬C | |
|---|---|---|---|
| C | **TP** | **FN** | **P** |
| ¬C | **FP** | **TN** | **N** |
| | **P'** | **N'** | **All** |

- **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* D *no* for which the classifier predicted*buys computer* D *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* D *yes* for which the classifier predicted *buys computer* D *no*). Let *FN* be the number of false negatives.

# Classifier Evaluation Metrics: Confusion Matrix

**Example of Confusion Matrix:**

| Actual class\Predicted class | Play= yes | Play = no | Total |
|---|---|---|---|
| Play= yes | **6954** | **46** | 7000 |
| Play = no | **412** | **2588** | 3000 |
| Total | 7366 | 2634 | 10000 |

**Classifier Accuracy,** or recognition rate: percentage of test set tuples that are correctly classified

**Error rate:** *1 – accuracy*

# Classifier Evaluation Metrics: Accuracy, Error Rate, Sensitivity and Specificity

| A\P | C | ¬C | |
|---|---|---|---|
| C | **TP** | **FN** | **P** |
| ¬C | **FP** | **TN** | **N** |
| | **P'** | **N'** | **All** |

- **Class Imbalance Problem**:
  - One class may be *rare*, e.g. fraud, or HIV-positive
  - Significant *majority of the negative class* and minority of the positive class
- **Sensitivity**: (True Positive recognition rate) proportion of actual positives which are predicted positive
  - **Sensitivity = TP/P**
- **Specificity**: (True Negative recognition rate) proportion of actual negative which are predicted negative
  - **Specificity = TN/N**
  - **Accuracy = [Sensitivity * P /(P+ N)] + [Specificity * N /(P+ N)]**
  - **Accuracy = (TP + TN)/All**
  - **Error rate = (FP + FN)/All**

- **Accuracy** : the proportion of the total number of predictions that were correct.
- **Positive Predictive Value or Precision** : the proportion of positive cases that were correctly identified.
- **Negative Predictive Value** : the proportion of negative cases that were correctly identified.
- **Sensitivity or Recall** : the proportion of actual positive cases which are correctly identified.
- **Specificity** : the proportion of actual negative cases which are correctly identified.

# Classifier Evaluation Metrics: Precision and Recall, and F-measures

- **Precision**: exactness – what % of tuples that the classifier labeled as positive are actually positive

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

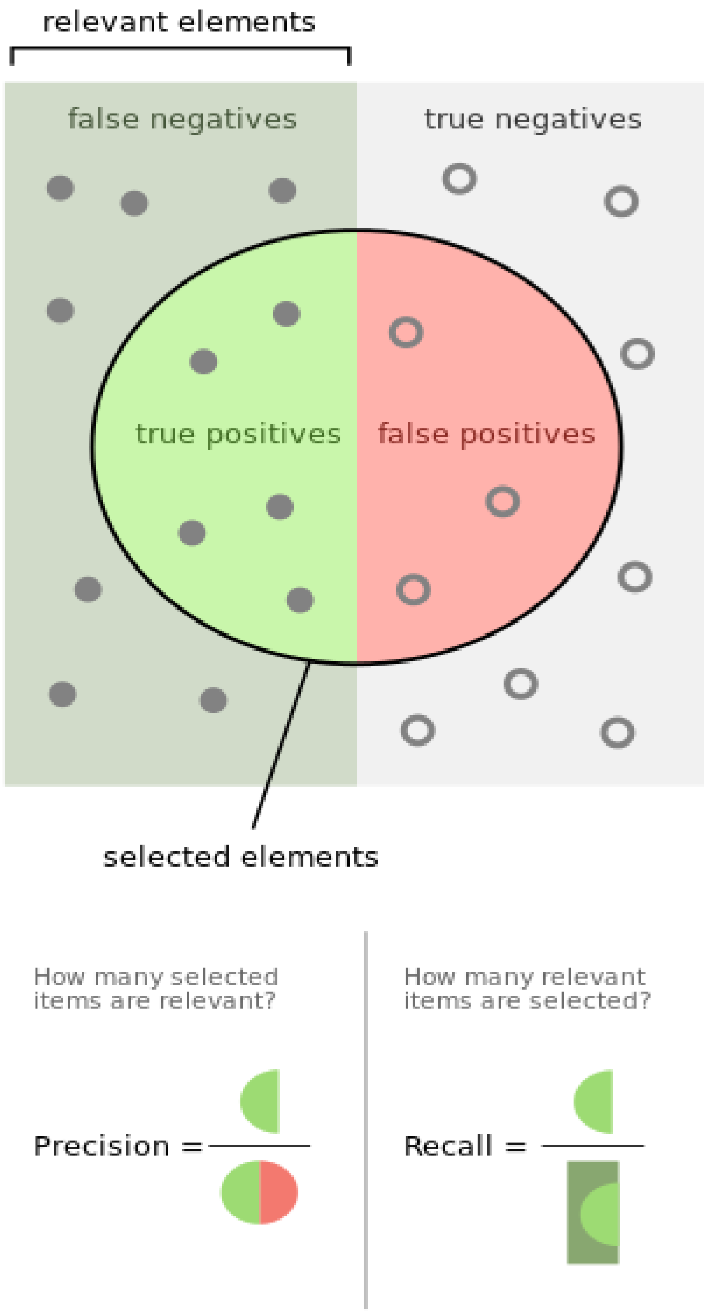
- **Recall:** completeness – what % of positive tuples did the classifier label as positive?

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

| A\P | C | ¬C | |
|---|---|---|---|
| C | **TP** | **FN** | **P** |
| ¬C | **FP** | **TN** | **N** |
| | **P'** | **N'** | **All** |

| Confusion Matrix | | **Target** | | | |
|---|---|---|---|---|---|
| | | Positive | Negative | | |
| **Model** | Positive | a | b | *Positive Predictive Value* | a/(a+b) |
| | Negative | c | d | *Negative Predictive Value* | d/(c+d) |
| | | *Sensitivity* | *Specificity* | **Accuracy** = (a+d)/(a+b+c+d) | |
| | | a/(a+c) | d/(b+d) | | |

https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c


relevant elements
false negatives
true negatives
true positives
false positives
selected elements
How many selected items are relevant?
How many relevant items are selected?
Precision =
Recall =

# Classifier Evaluation Metrics: Example

| Actual Class\Predicted class | cancer = yes | cancer = no | Total | Recognition(%) |
|---|---|---|---|---|
| cancer = yes | **90** | **210** | 300 | 30.00 (*sensitivity* |
| cancer = no | **140** | **9560** | 9700 | 98.56 (*specificity)* |
| Total | 230 | 9770 | 10000 | 96.40 (*accuracy*) |

- *Precision* = 90/230 = 39.13%  *Recall* = 90/300 = 30.00%

# Evaluating Classifier Accuracy: Holdout & Cross-Validation Methods

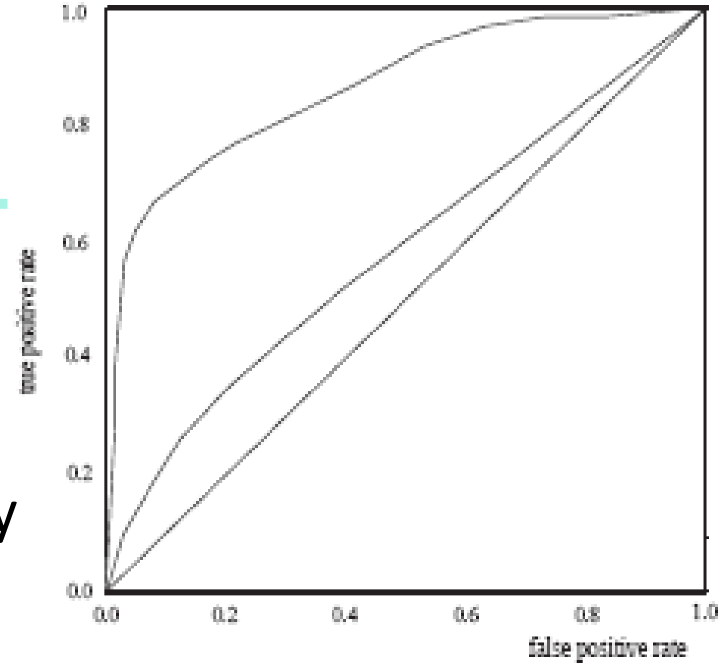
- **Holdout method**
  - Given data is randomly partitioned into two independent sets
    - Training set (e.g., 2/3) for model construction
    - Test set (e.g., 1/3) for accuracy estimation
  - Random sampling: a variation of holdout
    - Repeat holdout k times, accuracy = avg. of the accuracies obtained
- **Cross-validation** (*k*-fold, where k = 10 is most popular)
  - Randomly partition the data into *k mutually exclusive* subsets, each approximately equal size
  - At *i*-th iteration, use $D_i$ as test set and others as training set
  - **Leave-one-out:** *k* folds where *k* = # of tuples, for small sized data
  - **Stratified cross-validation**: folds are stratified so that class distribution of tuples in each fold is approx. the same as that in the initial data

# Evaluating Classifier Accuracy: Bootstrap

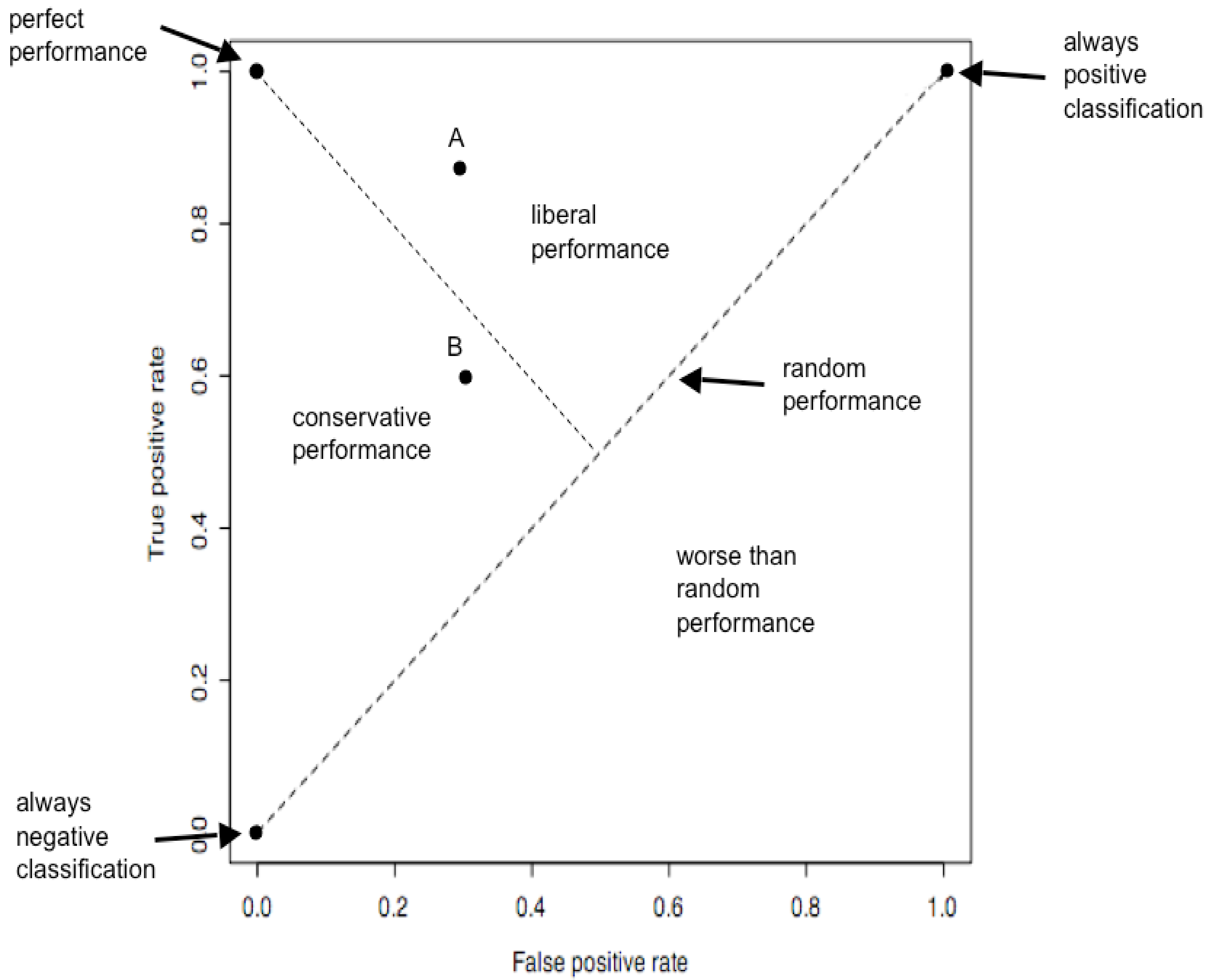
- **Bootstrap**
  - Works well with small data sets
  - Samples the given training tuples uniformly *with replacement*
    - i.e., each time a tuple is selected, it is equally likely to be selected again and re-added to the training set

# Model Selection: ROC Curves

- **ROC** (Receiver Operating Characteristics) curves: for visual comparison of classification models
- Originated from signal detection theory
- Shows the trade-off between the true positive rate and the false positive rate
- The area under the ROC curve is a measure of the accuracy of the model
- The closer to the diagonal line (i.e., the closer the area is to 0.5), the less accurate is the model

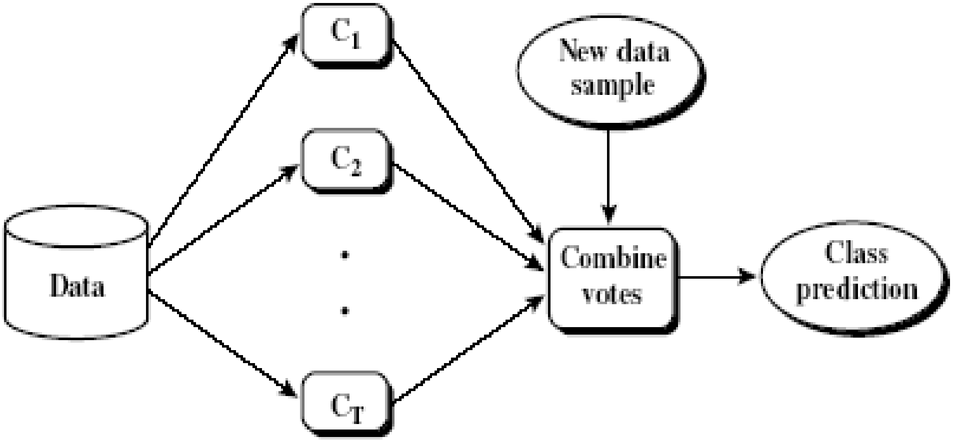



- Vertical axis represents the true positive rate
- Horizontal axis rep. the false positive rate
- The plot also shows a diagonal line
- A model with perfect accuracy will have an area of 1.0


perfect performance
always positive classification
A
B
liberal performance
conservative performance
random performance
worse than random performance
always negative classification
True positive rate
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0

# Increasing the Accuracy

# Ensemble Methods: Increasing the Accuracy




- Ensemble methods
  - Use a combination of models to increase accuracy
  - Combine a series of k learned models, $M_1$, $M_2$, …, $M_k$, with the aim of creating an improved model M*
- Popular ensemble methods
  - Bagging: averaging the prediction over a collection of classifiers
  - Boosting: weighted vote with a collection of classifiers
  - Ensemble: combining a set of heterogeneous classifiers

# Bagging: Boostrap Aggregation

- Algorithm: Bagging. The bagging algorithm—create an ensemble of models (classifiers or predictors) 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 learning scheme (e.g., decision tree algorithm etc.)
- Output: A composite model, *M*.

# Bagging: Boostrap Aggregation

Method:

(1) for $i$ = 1 to $k$ do // create $k$ models:

(2) create bootstrap sample, *Di*, by sampling *D* with replacement;

(3) use *Di* to derive a model, *Mi*;

(4) endfor

- To use the composite model on a tuple, ***X***:

(1) if classification then

(2) let each of the $k$ models classify ***X*** and return the majority vote;

(3) if prediction then

(4) let each of the $k$ models predict a value for ***X*** and return the average predicted value;

# Bagging: Boostrap Aggregation

- **Training**
  - Given a set D of *d* tuples, at each iteration *i*, a training set $D_i$ of *d* tuples is sampled with replacement from D (i.e., bootstrap)
  - A classifier model $M_i$ is learned for each training set $D_i$
- **Classification**: classify an unknown sample **X**
  - Each classifier $M_i$ returns its class prediction
  - The bagged classifier M* counts the votes and assigns the class with the most votes to **X**
- **Prediction:** can be applied to the prediction of continuous values by taking the average value of each prediction for a given test tuple
- **Accuracy**
  - Often significantly better than a single classifier derived from D
  - For noise data: not considerably worse, more robust
  - Proved improved accuracy in prediction

# Boosting

- How boosting works?
  - **Weights** are assigned to each training tuple
  - A series of k classifiers is iteratively learned
  - After a classifier $M_i$ is learned, the weights are updated to allow the subsequent classifier, $M_{i+1}$, to **pay more attention to the training tuples that were misclassified** by $M_i$
  - The final **M* combines the votes** of each individual classifier, where the weight of each classifier's vote is a function of its accuracy
- Boosting algorithm can be extended for numeric prediction
- Comparing with bagging: Boosting tends to have greater accuracy, but it also risks overfitting the model to misclassified data

# Adaboost

- Given a set of *d* class-labeled tuples, ($\mathbf{X_1}$, $y_1$), …, ($\mathbf{X_d}$, $y_d$)
- Initially, all the weights of tuples are set the same (1/d)
- Generate k classifiers in k rounds. At round i,
  - Tuples from D are sampled (with replacement) to form a training set $D_i$ of the same size
  - Each tuple's chance of being selected is based on its weight
  - A classification model $M_i$ is derived from $D_i$
  - Its error rate is calculated using $D_i$ as a test set
  - If a tuple is misclassified, its weight is increased, o.w. it is decreased
- Error rate: err($\mathbf{X_j}$) is the misclassification error of tuple $\mathbf{X_j}$. Classifier $M_i$ error rate is the sum of the weights of the misclassified tuples:

$$error(M_i) = \sum_{j}^{d} w_j \times err(\mathbf{X_j})$$

- If error(Mi)>0.5 //generate a new *Di* training set, from whichn derive a new *Mi*.
- The weight of classifier $M_i$'s vote is $\log \frac{1 - error(M_i)}{error(M_i)}$

# Random Forest (Breiman 2001)

- Random Forest:
  - Each classifier in the ensemble is a *decision tree* classifier and is generated using a random selection of attributes at each node to determine the split
  - During classification, each tree votes and the most popular class is returned
- Two Methods to construct Random Forest:
  - Forest-RI (*random input selection*): Randomly select, at each node, F attributes as candidates for the split at the node. The CART methodology is used to grow the trees to maximum size
  - Forest-RC (*random linear combinations*): Creates new attributes (or features) that are a linear combination of the existing attributes (reduces the correlation between individual classifiers)
- Comparable in accuracy to Adaboost, but more robust to errors and outliers
- Insensitive to the number of attributes selected for consideration at each split, and faster than bagging or boosting

# Classification of Class-Imbalanced Data Sets

- Class-imbalance problem: Rare positive example but numerous negative ones, e.g., medical diagnosis, fraud, oil-spill, fault, etc.
- Traditional methods assume a balanced distribution of classes and equal error costs: not suitable for class-imbalanced data
- Typical methods for imbalance data in 2-class classification:
  - **Oversampling**: re-sampling of data from positive class
  - **Under-sampling**: randomly eliminate tuples from negative class
  - **Threshold-moving**: moves the decision threshold, t, so that the rare class tuples are easier to classify, and hence, less chance of costly false negative errors
  - Ensemble techniques: Ensemble multiple classifiers introduced above
- Still difficult for class imbalance problem on multiclass tasks

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

e 1

$$P(C_i|X) = P(X|C_i)P(C_i)$$

Take the following training data, from bank loan applicants:

| ApplicantID | City | Children | Income | Status |
|---|---|---|---|---|
| 1 | Delhi | Many | Medium | DEFAULTS |
| 2 | Delhi | Many | Low | DEFAULTS |
| 3 | Delhi | Few | Medium | PAYS |
| 4 | Delhi | Few | High | PAYS |

- **P[City=Delhi | Status = DEFAULTS] = 2/2 = 1**
- **P[City=Delhi | Status = PAYS] = 2/2 = 1**
- **P[Children=Many | Status = DEFAULTS] = 2/2 = 1**
- **P[Children=Few | Status = DEFAULTS] = 0/2 = 0**

# Worked Example 1

Summarizing, we have the following probabilities:

| Probability of... | ... given DEFAULTS | ... given PAYS |
|---|---|---|
| City=Delhi | 2/2 = 1 | 2/2 = 1 |
| Children=Few | 0/2 = 0 | 2/2 = 1 |
| Children=Many | 2/2 = 1 | 0/2 = 0 |
| Income=Low | 1/2 = 0.5 | 0/2 = 0 |
| Income=Medium | 1/2 = 0.5 | 1/2 = 0.5 |
| Income=High | 0/2 = 0 | 1/2 = 0.5 |

**and P[Status = DEFAULTS] = 2/4 = 0.5**
**P[Status = PAYS] = 2/4 = 0.5**

The probability of Income=Medium given the applicant DEFAULTs =
the number of applicants with Income=Medium who DEFAULT
divided by the number of applicants who DEFAULT
= 1/2 = 0.5

# Worked Example 1

Now, assume a new example is presented where

**City=Delhi, Children=Many, and Income=Medium:**

First, we estimate the likelihood that the example is a defaulter, given its attribute values

**P[Status = DEFAULTS | Delhi,Many,Medium] =**

**P[Delhi|DEFAULTS] *
P[Many|DEFAULTS] *
P[Medium|DEFAULTS]*
P[DEFAULTS]**

**= 1 *1 * 0.5 * 0.5**

**= 0.25**

Then we estimate the likelihood that the example is a payer, given its attributes

**P[Status = PAYS | Delhi,Many,Medium] =**

**P[Delhi|PAYS] x
P[Many|PAYS] x
P[Medium|PAYS] x
P[PAYS]**

**= 1 x 0 x 0.5 x 0.5 = 0**

As the conditional likelihood of being a defaulter is higher (because 0.25 > 0), we conclude that the new example is a defaulter.

# Worked Example 1

Now, assume a new example is presented where

**City=Delhi, Children=Many, and Income=High:**

First, we estimate the likelihood that the example is a defaulter, given its attribute values:

**P[Status = DEFAULTS | Delhi,Many,High] =**
**P[Delhi|DEFAULTS] x P[Many|DEFAULTS] x P[High|DEFAULTS] x P[DEFAULTS]**

**= 1 x 1 x 0 x 0.5 = 0**

Then we estimate the likelihood that the example is a payer, given its attributes:

**P[Status = PAYS | Delhi,Many,High] =**
**P[Delhi|PAYS] x P[Many|PAYS] x P[High|PAYS] x P[PAYS]**

**= 1 x 0 x 0.5 x 0.5 = 0**

As the conditional likelihood of being a defaulter is the same as that for being a payer, we can come to no conclusion for this example.

# Worked Example 2

Take the following training data, for credit card authorizations:

| TransactionID | Income | Credit | Decision |
|---|---|---|---|
| 1 | Very High | Excellent | AUTHORIZE |
| 2 | High | Good | AUTHORIZE |
| 3 | Medium | Excellent | AUTHORIZE |
| 4 | High | Good | AUTHORIZE |
| 5 | Very High | Good | AUTHORIZE |
| 6 | Medium | Excellent | AUTHORIZE |
| 7 | High | Bad | REQUEST ID |
| 8 | Medium | Bad | REQUEST ID |
| 9 | High | Bad | REJECT |
| 10 | Low | Bad | CALL POLICE |

Assume we'd like to determine how to classify a new transaction, with `Income = Medium` and `Credit=Good`.

# Worked Example 2

Our conditional probabilities are:

| Probability of... | ... given `AUTHORIZE` | ... given `REQUEST ID` | ... given `REJECT` | ... given `CALL POLICE` |
|---|---|---|---|---|
| `Income=Very High` | 2/6 | 0/2 | 0/1 | 0/1 |
| `Income=High` | 2/6 | 1/2 | 1/1 | 0/1 |
| `Income=Medium` | 2/6 | 1/2 | 0/1 | 0/1 |
| `Income=Low` | 0/6 | 0/2 | 0/1 | 1/1 |
| `Credit=Excellent` | 3/6 | 0/2 | 0/1 | 0/1 |
| `Credit=Good` | 3/6 | 0/2 | 0/1 | 0/1 |
| `Credit=Bad` | 0/6 | 2/2 | 1/1 | 1/1 |

Our class probabilities are:

P[`Decision = AUTHORIZE`] = 6/10

P[`Decision = REQUEST ID`] = 2/10

P[`Decision = REJECT`] = 1/10

P[`Decision = CALL POLICE`] = 1/10

# Weaknesses

- Naive Bayes assumes that variables are **equally important** and that they are **independent** which is often not the case in practice.
- Naive Bayes is **damaged by the inclusion of redundant (strongly dependent) attributes**. e.g. if people with high income have expensive houses, then including both income and house-price in the model would unfairly multiply the effect of having low income.
- Sparse data: If some attribute values are not present in the data, then a **zero probability** for P[X|C] might exist. This would lead P[C|X] to be zero no matter how high P[X|C] is for other attribute values.