
Unit–IV Classification

- **Basic Concepts**
- **Decision Tree Induction**
- **Bayes Classification Methods**
- **Rule-Based Classification**
- **Classification by Back-propagation**
- **Support Vector Machine**
- **Lazy Learners**
- **Other Classification Methods.**

Prediction Problems: Classification vs. Numeric 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
- **Numeric Prediction**
 - models continuous-valued functions, i.e., predicts unknown or missing values
- 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

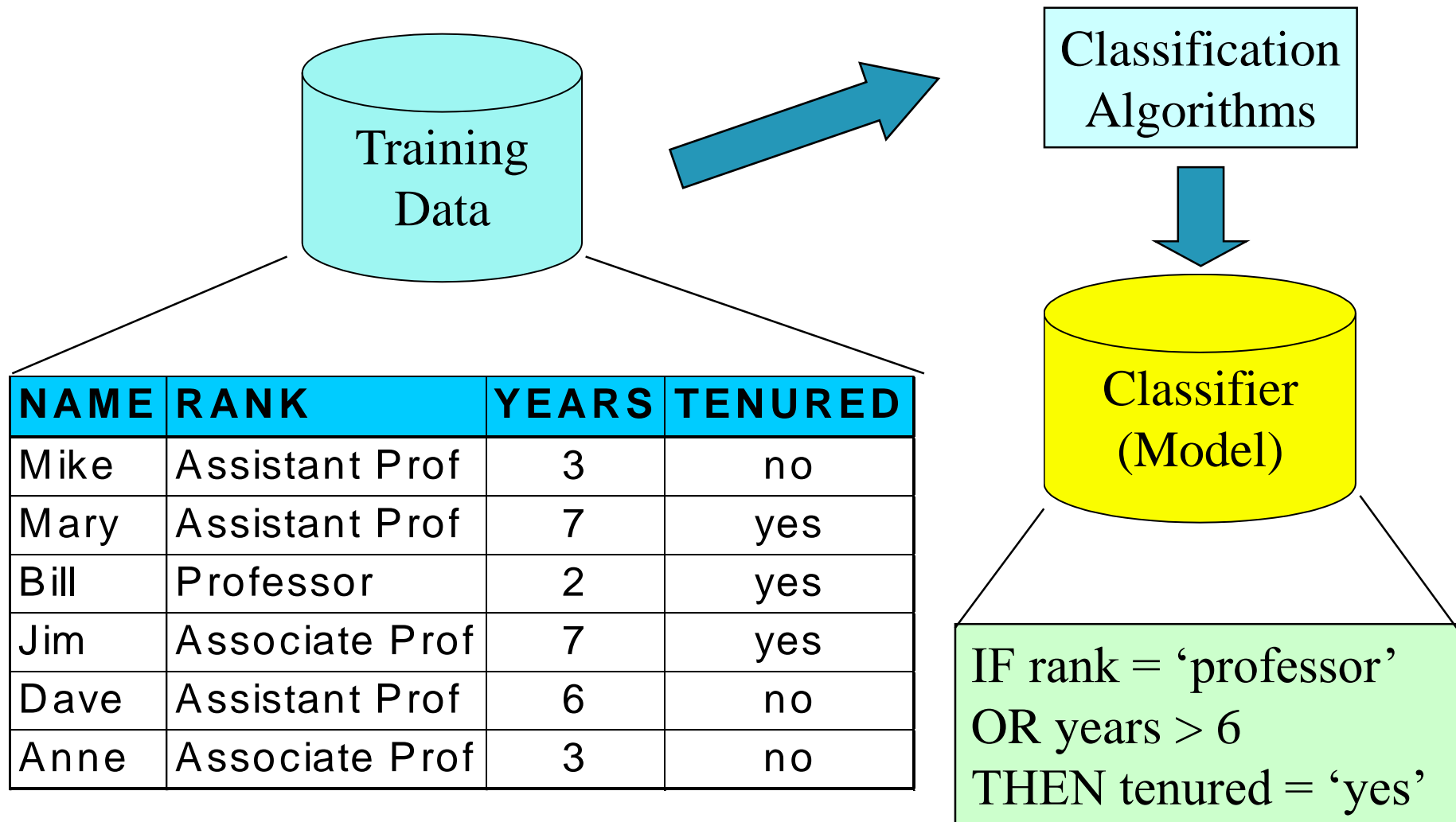
Supervised vs. Unsupervised Learning

- 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

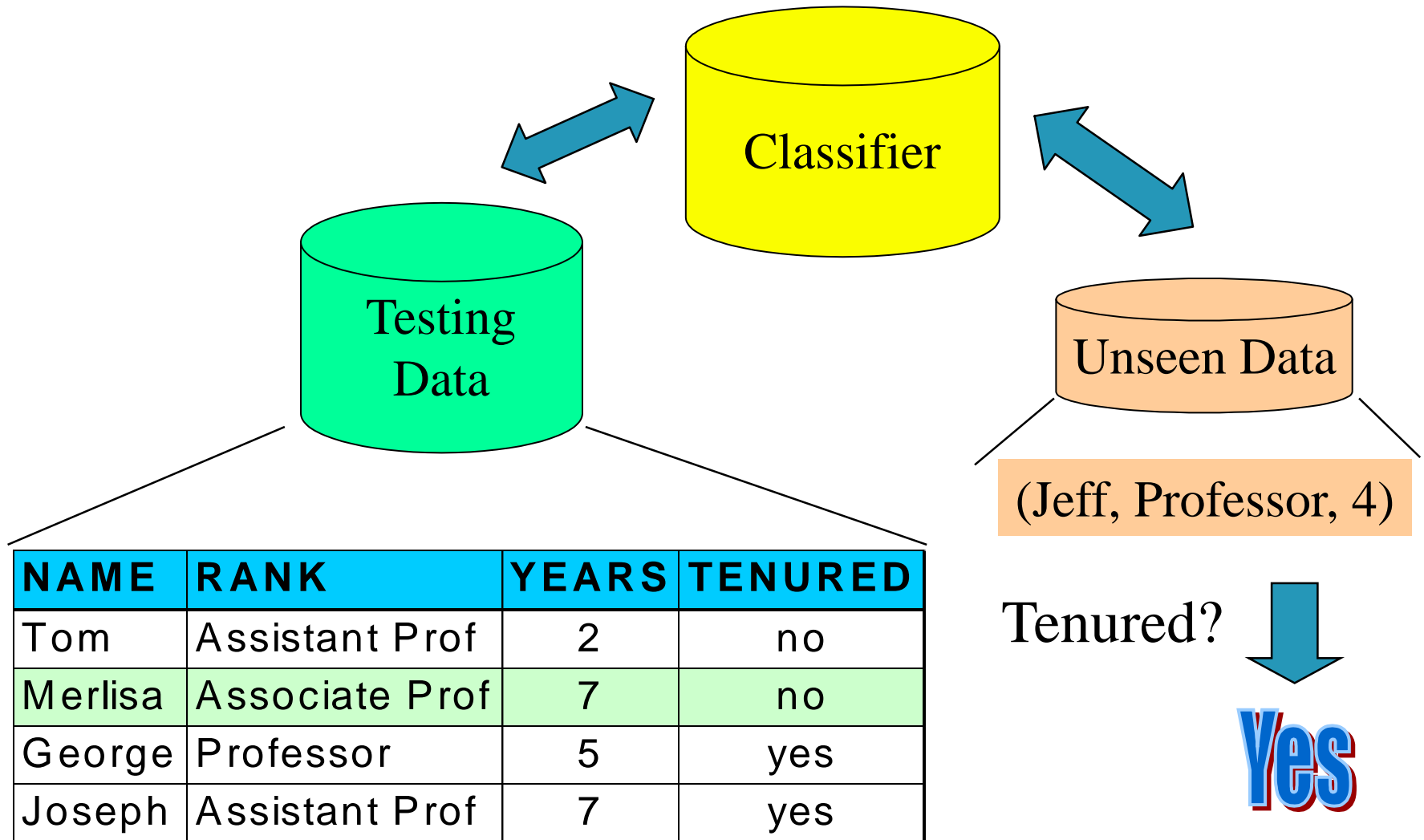
Classification—A Two-Step Process

- **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 overfitting)
 - If the accuracy is acceptable, use the model to **classify new data**
- Note: If *the test set* is used to select models, it is called **validation (test) set**

Process (1): Model Construction



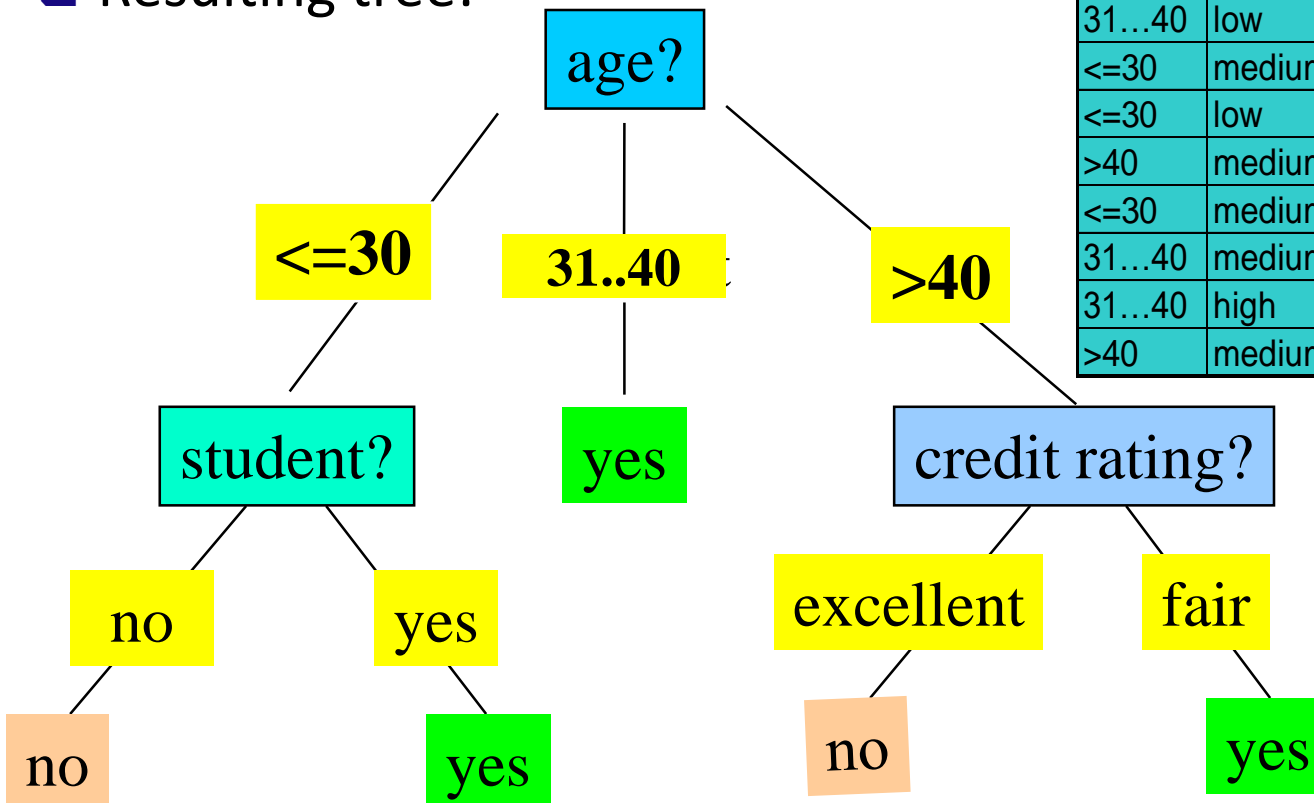
Process (2): Using the Model in Prediction



Decision Tree Induction: An Example

- ❑ Training data set: Buys_computer
- ❑ The data set follows an example of Quinlan's ID3 (Playing Tennis)
- ❑ Resulting tree:

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



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

Brief Review of Entropy

■ Entropy (Information Theory)

- A measure of uncertainty associated with a random variable

- Calculation: For a discrete random variable Y taking m distinct values $\{y_1, \dots, y_m\}$,

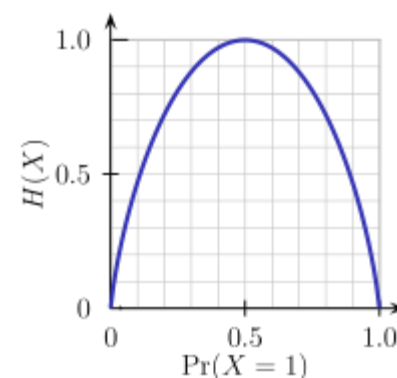
- $H(Y) = -\sum_{i=1}^m p_i \log(p_i)$, where $p_i = P(Y = y_i)$

- Interpretation:

- Higher entropy => higher uncertainty
 - Lower entropy => lower uncertainty

■ Conditional Entropy

- $H(Y|X) = \sum_x p(x)H(Y|X = x)$



m = 2

Attribute Selection Measure: Information Gain (ID3/C4.5)

- Select the attribute with the highest information gain
- Let p_i be the probability that an arbitrary tuple in D belongs to class C_i , estimated by $|C_{i,D}|/|D|$
- **Expected information** (entropy) needed to classify a tuple in D :

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

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

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

- **Information gained** by branching on attribute A

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

Attribute Selection: Information Gain

■ Class P: buys_computer = “yes”

■ Class N: buys_computer = “no”

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

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

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

$\frac{5}{14} I(2,3)$ 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,

$$Gain(income) = 0.029$$

$$Gain(student) = 0.151$$

$$Gain(credit_rating) = 0.048$$

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

Gain Ratio for Attribute Selection (C4.5)

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

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

- $GainRatio(A) = Gain(A)/SplitInfo(A)$
- Ex. $SplitInfo_{income}(D) = -\frac{4}{14} \times \log_2 \left(\frac{4}{14} \right) - \frac{6}{14} \times \log_2 \left(\frac{6}{14} \right) - \frac{4}{14} \times \log_2 \left(\frac{4}{14} \right) = 1.557$
 - $gain_ratio(income) = 0.029/1.557 = 0.019$
- The attribute with the maximum gain ratio is selected as the splitting attribute

Gini Index (CART, IBM IntelligentMiner)

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

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

where p_j is the relative frequency of class j in D

- If a data set D is split on A into two subsets D_1 and D_2 , the $gini$ index $gini_A(D)$ is defined as

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

- Reduction in Impurity:

$$\Delta gini(A) = gini(D) - gini_A(D)$$

- The attribute provides the smallest $gini_{split}(D)$ (or the largest reduction in impurity) is chosen to split the node (*need to enumerate all the possible splitting points for each attribute*)

Computation of Gini Index

- Ex. D has 9 tuples in buys_computer = “yes” and 5 in “no”

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

- Suppose the attribute income partitions D into 10 in D_1 : {low, medium} and 4 in D_2 $gini_{income \in \{low, medium\}}(D) = \left(\frac{10}{14}\right)Gini(D_1) + \left(\frac{4}{14}\right)Gini(D_2)$

$$= \frac{10}{14} \left(1 - \left(\frac{7}{10}\right)^2 - \left(\frac{3}{10}\right)^2\right) + \frac{4}{14} \left(1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2\right)$$

$$= 0.443$$

$$= Gini_{income \in \{high\}}(D).$$

$Gini_{\{low, high\}}$ is 0.458; $Gini_{\{medium, high\}}$ is 0.450. Thus, split on the {low, medium} (and {high}) since it has the lowest Gini index

- All attributes are assumed continuous-valued
- May need other tools, e.g., clustering, to get the possible split values
- Can be modified for categorical attributes

Comparing Attribute Selection Measures

- The three measures, in general, return good results but
 - **Information gain:**
 - biased towards multivalued attributes
 - **Gain ratio:**
 - tends to prefer unbalanced splits in which one partition is much smaller than the others
 - **Gini index:**
 - biased to multivalued attributes
 - has difficulty when # of classes is large
 - tends to favor tests that result in equal-sized partitions and purity in both partitions

Bayesian Classification: Why?

- A statistical classifier: performs *probabilistic prediction*, i.e., predicts class membership probabilities
- Foundation: Based on Bayes' Theorem.
- Performance: A simple Bayesian classifier, *naïve Bayesian classifier*, has comparable performance with decision tree and selected neural network classifiers
- Incremental: Each training example can incrementally increase/decrease the probability that a hypothesis is correct — prior knowledge can be combined with observed data
- Standard: Even when Bayesian methods are computationally intractable, they can provide a standard of optimal decision making against which other methods can be measured

Bayes' Theorem: Basics

- Total probability Theorem:
$$P(B) = \sum_{i=1}^M P(B|A_i)P(A_i)$$
- Bayes' Theorem:
$$P(H | \mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})} = P(\mathbf{X}|H) \times P(H) / P(\mathbf{X})$$
 - Let \mathbf{X} be a data sample (“evidence”): class label is unknown
 - Let H be a *hypothesis* that X belongs to class C
 - Classification is to determine $P(H|\mathbf{X})$, (i.e., *posteriori probability*): the probability that the hypothesis holds given the observed data sample \mathbf{X}
 - $P(H)$ (*prior probability*): the initial probability
 - E.g., \mathbf{X} will buy computer, regardless of age, income, ...
 - $P(\mathbf{X})$: probability that sample data is observed
 - $P(\mathbf{X}|H)$ (likelihood): the probability of observing the sample \mathbf{X} , given that the hypothesis holds
 - E.g., Given that \mathbf{X} will buy computer, the prob. that X is 31..40, medium income

Prediction Based on Bayes' Theorem

- Given training data \mathbf{X} , *posteriori probability of a hypothesis* H , $P(H|\mathbf{X})$, follows the Bayes' theorem

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

- Informally, this can be viewed as
posteriori = likelihood x prior/evidence
- Predicts \mathbf{X} belongs to C_i iff the probability $P(C_i|\mathbf{X})$ is the highest among all the $P(C_k|\mathbf{X})$ for all the k classes
- Practical difficulty: It requires initial knowledge of many probabilities, involving significant computational cost

Classification Is to Derive the Maximum Posteriori

- Let D be a training set of tuples and their associated class labels, and each tuple is represented by an n -D attribute vector $\mathbf{X} = (x_1, x_2, \dots, x_n)$
- Suppose there are m classes C_1, C_2, \dots, C_m .
- Classification is to derive the maximum posteriori, i.e., the maximal $P(C_i | \mathbf{X})$
- This can be derived from Bayes' theorem

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

- Since $P(\mathbf{X})$ is constant for all classes, only

$$P(C_i | \mathbf{X}) = P(\mathbf{X} | C_i)P(C_i)$$

needs to be maximized

Naïve Bayes Classifier

- A simplified assumption: attributes are conditionally independent (i.e., no dependence relation between attributes):
$$P(\mathbf{X} | C_i) = \prod_{k=1}^n P(x_k | C_i) = P(x_1 | C_i) \times P(x_2 | C_i) \times \dots \times P(x_n | C_i)$$
- This greatly reduces the computation cost: Only counts the class distribution
- If A_k is categorical, $P(x_k | C_i)$ is the # of tuples in C_i having value x_k for A_k divided by $|C_{i,D}|$ (# of tuples of C_i in D)
- If A_k is continuous-valued, $P(x_k | C_i)$ is usually computed based on Gaussian distribution with a mean μ and standard deviation σ

and $P(x_k | C_i)$ is

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

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

Naïve Bayes Classifier: Training Dataset

Class:

C1:buys_computer = 'yes'

C2:buys_computer = 'no'

Data to be classified:

X = (age <=30,

Income = medium,

Student = yes

Credit_rating = Fair)

age	income	student	credit_rating	comp
<=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

Naïve Bayes Classifier: An Example

age	income	student	credit_rating	com
<=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

- $P(C_i): P(\text{buys_computer} = \text{"yes"}) = 9/14 = 0.643$

$$P(\text{buys_computer} = \text{"no"}) = 5/14 = 0.357$$

- Compute $P(X|C_i)$ for each class

$$P(\text{age} = \text{"<=30"} \mid \text{buys_computer} = \text{"yes"}) = 2/9 = 0.222$$

$$P(\text{age} = \text{"<= 30"} \mid \text{buys_computer} = \text{"no"}) = 3/5 = 0.6$$

$$P(\text{income} = \text{"medium"} \mid \text{buys_computer} = \text{"yes"}) = 4/9 = 0.444$$

$$P(\text{income} = \text{"medium"} \mid \text{buys_computer} = \text{"no"}) = 2/5 = 0.4$$

$$P(\text{student} = \text{"yes"} \mid \text{buys_computer} = \text{"yes"}) = 6/9 = 0.667$$

$$P(\text{student} = \text{"yes"} \mid \text{buys_computer} = \text{"no"}) = 1/5 = 0.2$$

$$P(\text{credit_rating} = \text{"fair"} \mid \text{buys_computer} = \text{"yes"}) = 6/9 = 0.667$$

$$P(\text{credit_rating} = \text{"fair"} \mid \text{buys_computer} = \text{"no"}) = 2/5 = 0.4$$

- $X = (\text{age} \leq 30, \text{income} = \text{medium}, \text{student} = \text{yes}, \text{credit_rating} = \text{fair})$

$$P(X|C_i) : P(X \mid \text{buys_computer} = \text{"yes"}) = 0.222 \times 0.444 \times 0.667 \times 0.667 = 0.044$$

$$P(X \mid \text{buys_computer} = \text{"no"}) = 0.6 \times 0.4 \times 0.2 \times 0.4 = 0.019$$

$$P(X|C_i) * P(C_i) : P(X \mid \text{buys_computer} = \text{"yes"}) * P(\text{buys_computer} = \text{"yes"}) = 0.028$$

$$P(X \mid \text{buys_computer} = \text{"no"}) * P(\text{buys_computer} = \text{"no"}) = 0.007$$

Therefore, X belongs to class ("buys_computer = yes")

Naïve Bayes Classifier: Comments

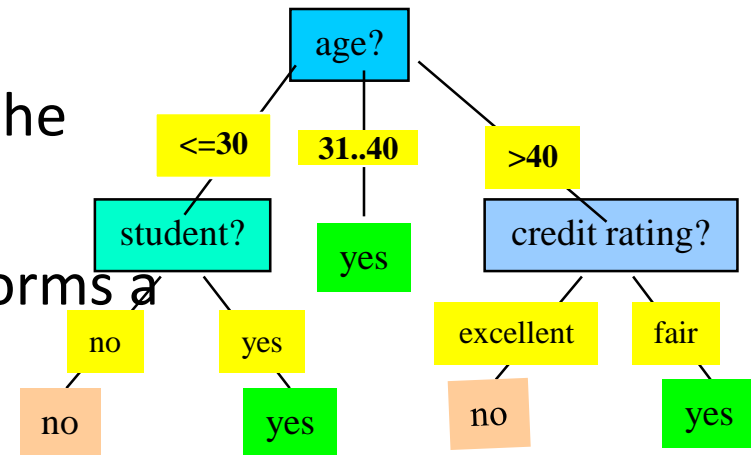
- Advantages
 - Easy to implement
 - Good results obtained in most of the cases
- Disadvantages
 - Assumption: class conditional independence, therefore loss of accuracy
 - Practically, dependencies exist among variables
 - E.g., hospitals: patients: Profile: age, family history, etc.
Symptoms: fever, cough etc., Disease: lung cancer, diabetes, etc.
 - Dependencies among these cannot be modeled by Naïve Bayes Classifier
- How to deal with these dependencies? Bayesian Belief Networks (Chapter 9)

Using IF-THEN Rules for Classification

- Represent the knowledge in the form of **IF-THEN** rules
 - R: IF *age* = youth AND *student* = yes THEN *buys_computer* = yes
 - Rule antecedent/precondition vs. rule consequent
- Assessment of a rule: *coverage* and *accuracy*
 - n_{covers} = # of tuples covered by R
 - n_{correct} = # of tuples correctly classified by R
 - $\text{coverage}(R) = n_{\text{covers}} / |D|$ /* D: training data set */
 - $\text{accuracy}(R) = n_{\text{correct}} / n_{\text{covers}}$
- If more than one rule are triggered, need **conflict resolution**
 - Size ordering: assign the highest priority to the triggering rules that has the “toughest” requirement (i.e., with the *most attribute tests*)
 - Class-based ordering: decreasing order of *prevalence or misclassification cost per class*
 - Rule-based ordering (**decision list**): rules are organized into one long priority list, according to some measure of rule quality or by experts

Rule Extraction from a Decision Tree

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



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

THEN *buys_computer* = *no*

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

THEN *buys_computer* = *yes*

IF *age* = mid-age

THEN *buys_computer* = *yes*

IF *age* = old AND *credit_rating* = *excellent*

THEN *buys_computer* = *no*

IF *age* = old AND *credit_rating* = *fair*

THEN *buys_computer* = *yes*

Classification by Backpropagation

- Backpropagation: A **neural network** learning algorithm
- Started by psychologists and neurobiologists to develop and test computational analogues of neurons
- A neural network: A set of connected input/output units where each connection has a **weight** associated with it
- During the learning phase, the **network learns by adjusting the weights** so as to be able to predict the correct class label of the input tuples
- Also referred to as **connectionist learning** due to the connections between units

Neural Network as a Classifier

- Weakness
 - Long training time
 - Require a number of parameters typically best determined empirically, e.g., the network topology or “structure.”
 - Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of “hidden units” in the network
- Strength
 - High tolerance to noisy data
 - Ability to classify untrained patterns
 - Well-suited for continuous-valued inputs *and outputs*
 - Successful on an array of real-world data, e.g., hand-written letters
 - Algorithms are inherently parallel
 - Techniques have recently been developed for the extraction of rules from trained neural networks

A Multi-Layer Feed-Forward Neural Network

Output vector

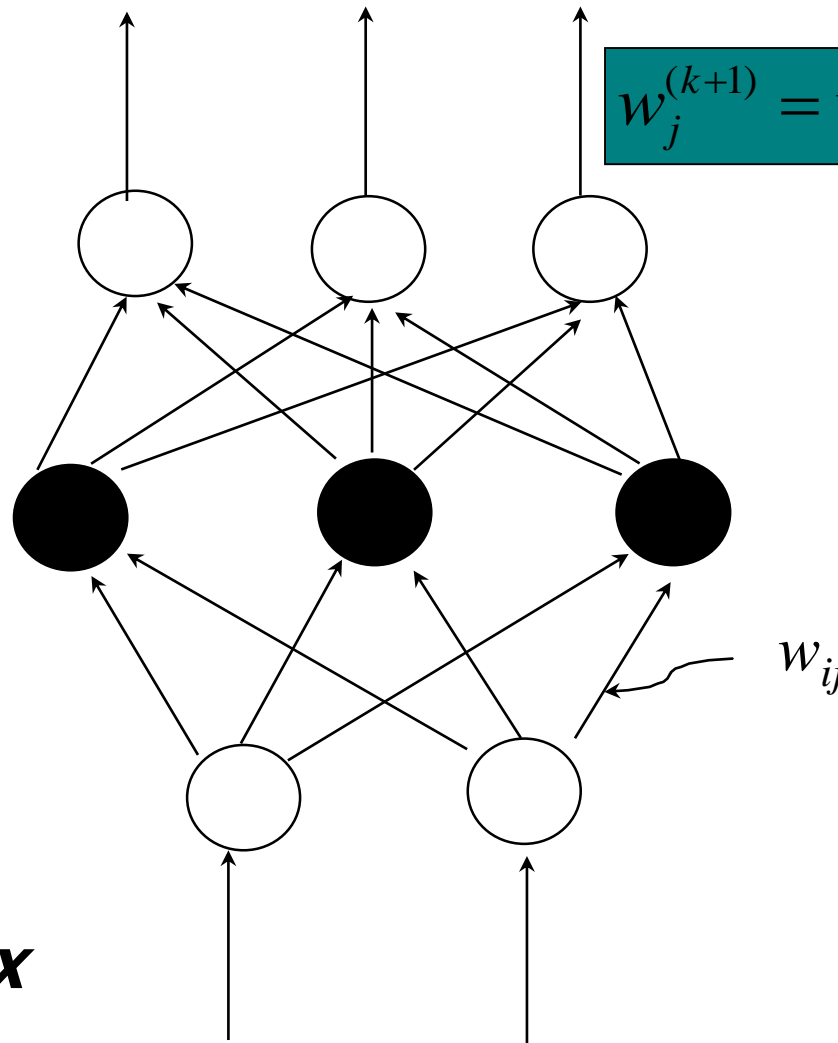
Output layer

$$w_j^{(k+1)} = w_j^{(k)} + \lambda(y_i - \hat{y}_i^{(k)})x_{ij}$$

Hidden layer

Input layer

Input vector: X



How A Multi-Layer Neural Network Works

- The **inputs** to the network correspond to the attributes measured for each training tuple
- Inputs are fed simultaneously into the units making up the **input layer**
- They are then weighted and fed simultaneously to a **hidden layer**
- The number of hidden layers is arbitrary, although usually only one
- The weighted outputs of the last hidden layer are input to units making up the **output layer**, which emits the network's prediction
- The network is **feed-forward**: None of the weights cycles back to an input unit or to an output unit of a previous layer
- From a statistical point of view, networks perform **nonlinear regression**: Given enough hidden units and enough training samples, they can closely approximate any function

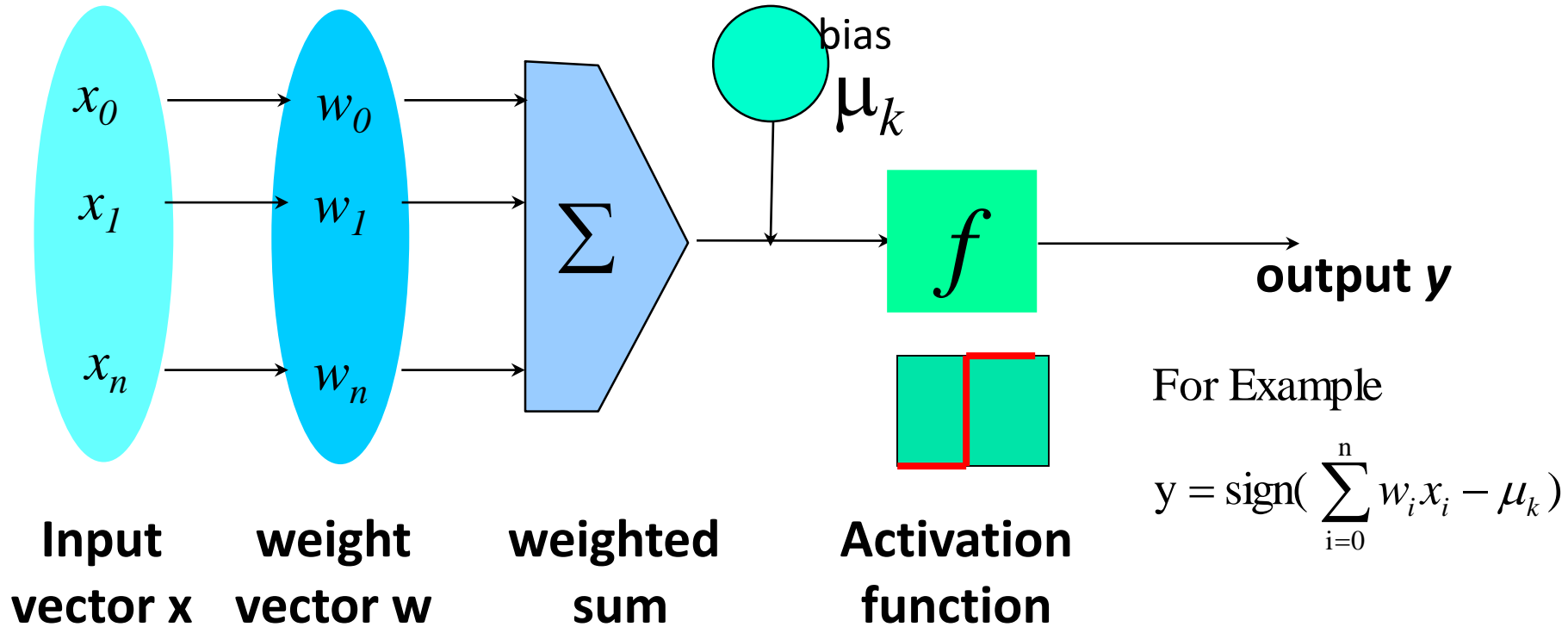
Defining a Network Topology

- Decide the **network topology**: Specify # of units in the *input layer*, # of *hidden layers* (if > 1), # of units in *each hidden layer*, and # of units in the *output layer*
- Normalize the input values for each attribute measured in the training tuples to $[0.0—1.0]$
- One **input** unit per domain value, each initialized to 0
- **Output**, if for classification and more than two classes, one output unit per class is used
- Once a network has been trained and its accuracy is **unacceptable**, repeat the training process with a *different network topology* or a *different set of initial weights*

Backpropagation

- Iteratively process a set of training tuples & compare the network's prediction with the actual known target value
- For each training tuple, the weights are modified to **minimize the mean squared error** between the network's prediction and the actual target value
- Modifications are made in the “**backwards**” direction: from the output layer, through each hidden layer down to the first hidden layer, hence “**backpropagation**”
- Steps
 - Initialize weights to small random numbers, associated with biases
 - Propagate the inputs forward (by applying activation function)
 - Backpropagate the error (by updating weights and biases)
 - Terminating condition (when error is very small, etc.)

Neuron: A Hidden/Output Layer Unit



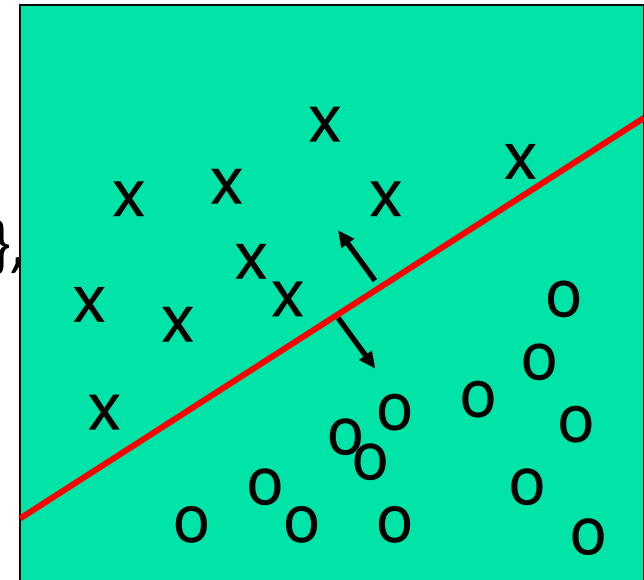
- An n -dimensional input vector \mathbf{x} is mapped into variable y by means of the scalar product and a nonlinear function mapping
- The inputs to unit are outputs from the previous layer. They are multiplied by their corresponding weights to form a weighted sum, which is added to the bias associated with unit. Then a nonlinear activation function is applied to it.

Efficiency and Interpretability

- **Efficiency** of backpropagation: Each epoch (one iteration through the training set) takes $O(|D| * w)$, with $|D|$ tuples and w weights, but # of epochs can be exponential to n , the number of inputs, in worst case
- For easier comprehension: **Rule extraction** by network pruning
 - Simplify the network structure by removing weighted links that have the least effect on the trained network
 - Then perform link, unit, or activation value clustering
 - The set of input and activation values are studied to derive rules describing the relationship between the input and hidden unit layers
- **Sensitivity analysis**: assess the impact that a given input variable has on a network output. The knowledge gained from this analysis can be represented in rules

Classification: A Mathematical Mapping

- **Classification:** predicts categorical class labels
 - E.g., Personal homepage classification
 - $x_i = (x_1, x_2, x_3, \dots)$, $y_i = +1$ or -1
 - x_1 : # of word “homepage”
 - x_2 : # of word “welcome”
- Mathematically, $x \in X = \mathbb{R}^n$, $y \in Y = \{+1, -1\}$,
 - We want to derive a function $f: X \rightarrow Y$
- Linear Classification
 - Binary Classification problem
 - Data above the red line belongs to class ‘x’
 - Data below red line belongs to class ‘o’
 - Examples: SVM, Perceptron, Probabilistic Classifiers



Discriminative Classifiers

- Advantages
 - Prediction accuracy is generally high
 - As compared to Bayesian methods – in general
 - Robust, works when training examples contain errors
 - Fast evaluation of the learned target function
 - Bayesian networks are normally slow
- Criticism
 - Long training time
 - Difficult to understand the learned function (weights)
 - Bayesian networks can be used easily for pattern discovery
 - Not easy to incorporate domain knowledge
 - Easy in the form of priors on the data or distributions

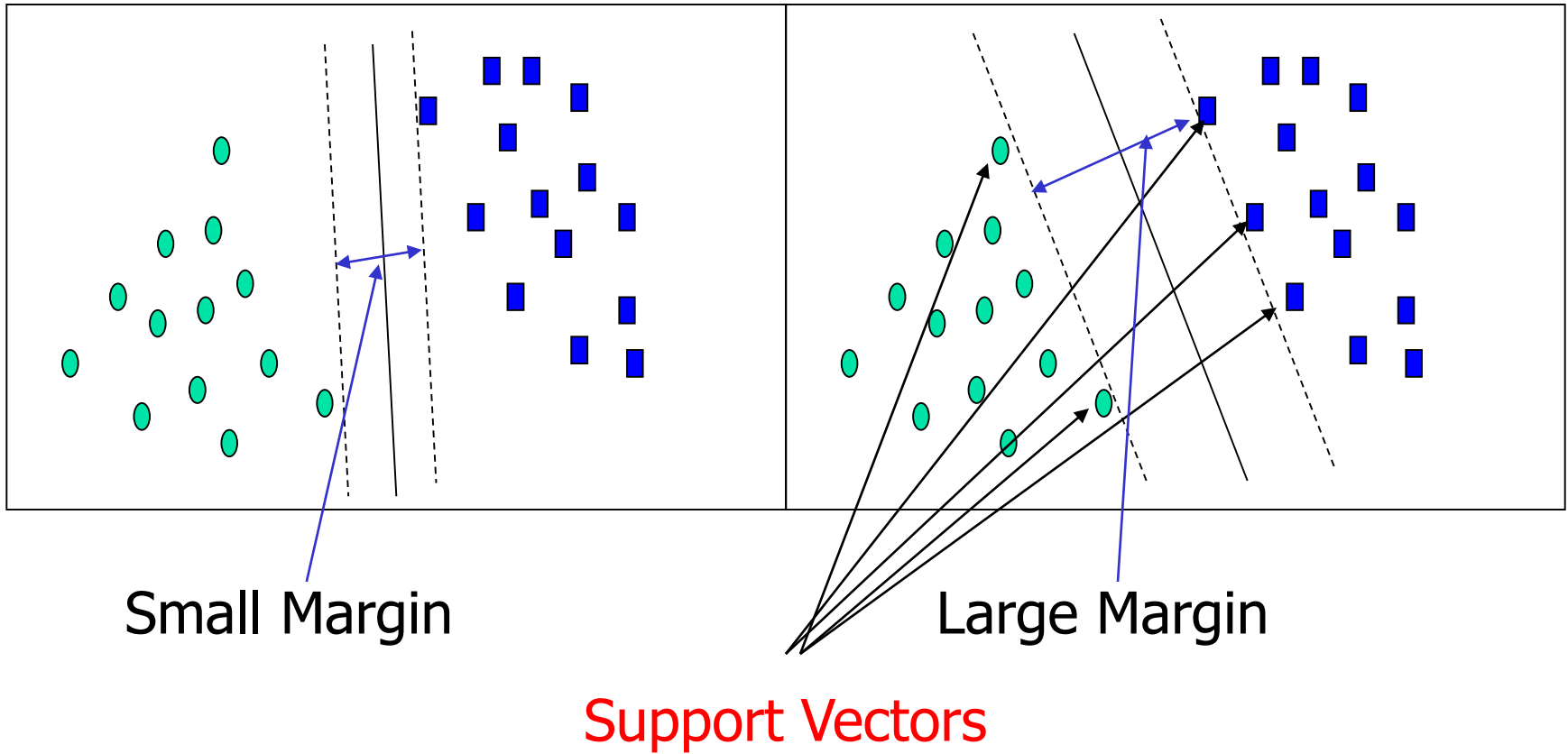
SVM—Support Vector Machines

- A relatively new classification method for both linear and nonlinear data
- It uses a nonlinear mapping to transform the original training data into a higher dimension
- With the new dimension, it searches for the linear optimal separating **hyperplane** (i.e., “decision boundary”)
- With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane
- SVM finds this hyperplane using **support vectors** (“essential” training tuples) and **margins** (defined by the support vectors)

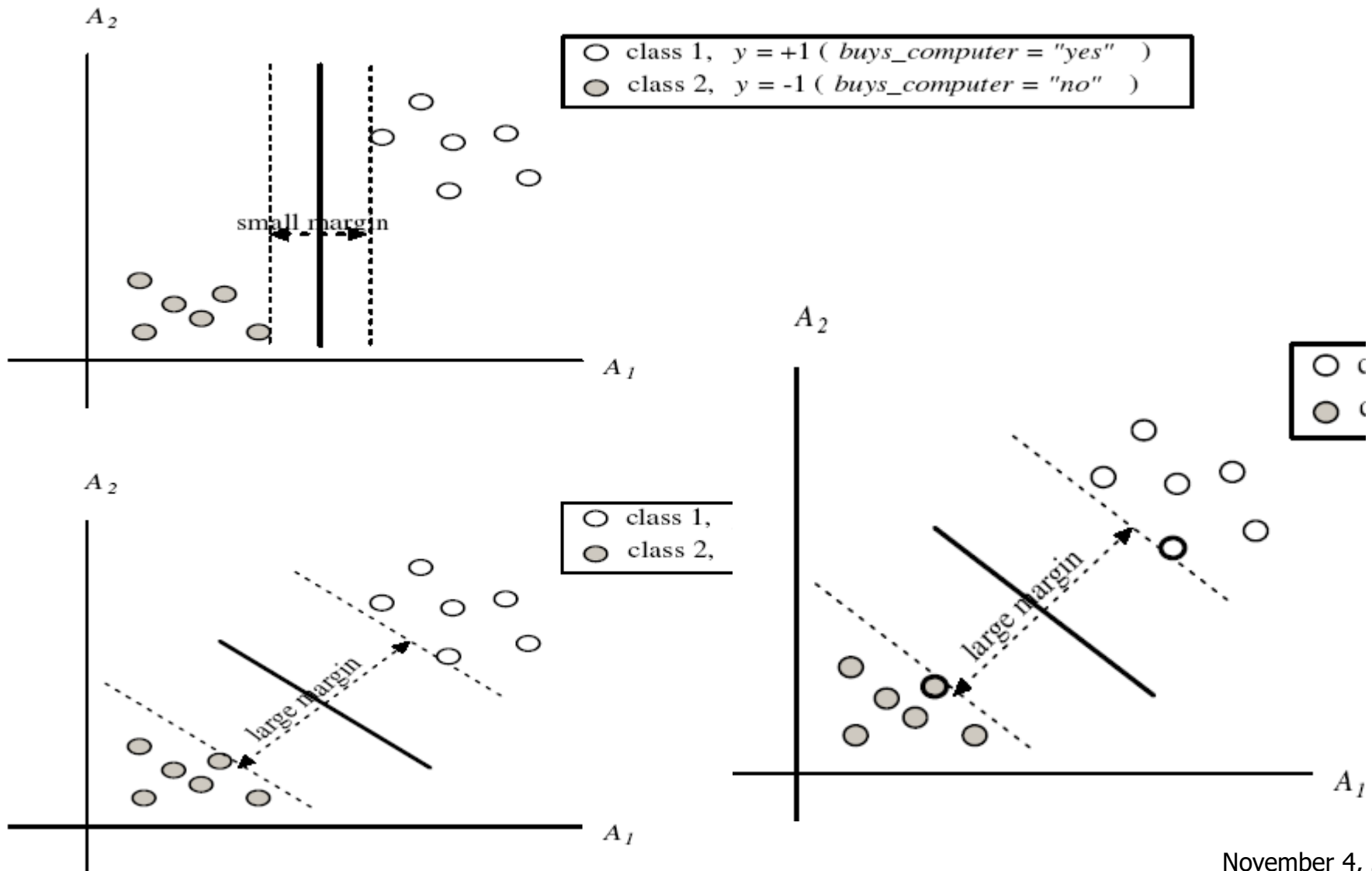
SVM—History and Applications

- Vapnik and colleagues (1992)—groundwork from Vapnik & Chervonenkis' statistical learning theory in 1960s
- Features: training can be slow but accuracy is high owing to their ability to model complex nonlinear decision boundaries (margin maximization)
- Used for: classification and numeric prediction
- Applications:
 - handwritten digit recognition, object recognition, speaker identification, benchmarking time-series prediction tests

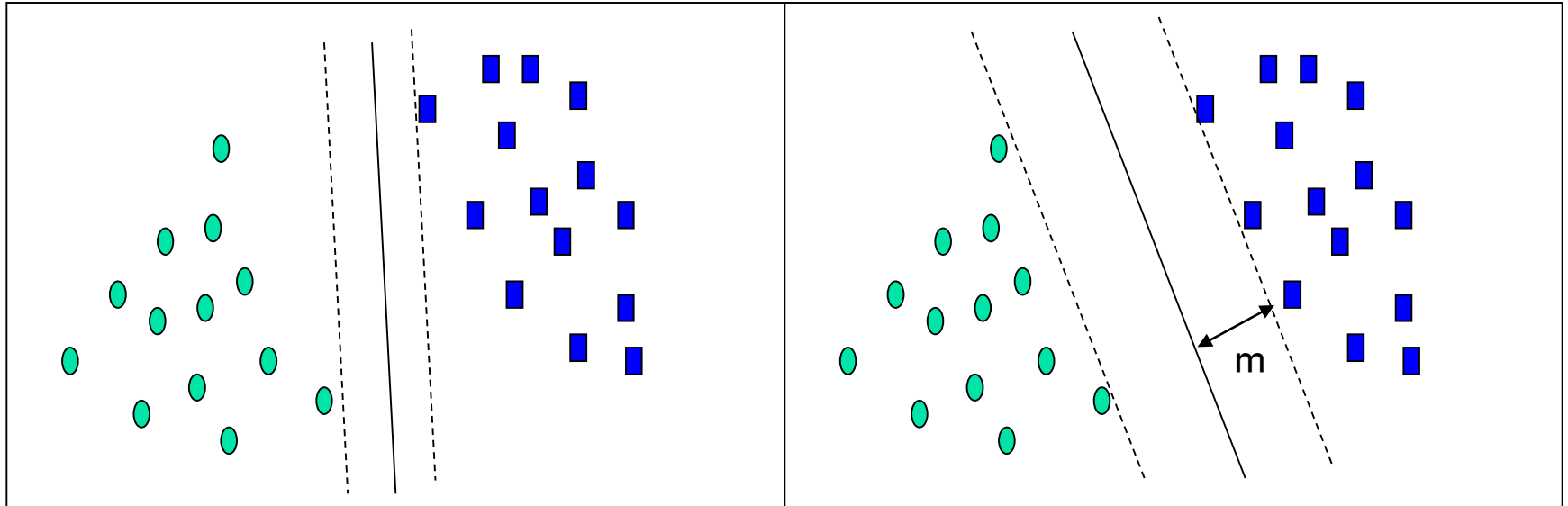
SVM—General Philosophy



SVM—Margins and Support Vectors



SVM—When Data Is Linearly Separable



Let data D be $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_{|D|}, y_{|D|})$, where \mathbf{x}_i is the set of training tuples associated with the class labels y_i

There are infinite lines (hyperplanes) separating the two classes but we want to find the best one (the one that minimizes classification error on unseen data)

*SVM searches for the hyperplane with the largest margin, i.e., **maximum marginal hyperplane** (MMH)*

SVM—Linearly Separable

- A separating hyperplane can be written as

$$\mathbf{W} \bullet \mathbf{X} + b = 0$$

where $\mathbf{W} = \{w_1, w_2, \dots, w_n\}$ is a weight vector and b a scalar (bias)

- For 2-D it can be written as

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

- The hyperplane defining the sides of the margin:

$$H_1: w_0 + w_1 x_1 + w_2 x_2 \geq 1 \quad \text{for } y_i = +1, \text{ and}$$

$$H_2: w_0 + w_1 x_1 + w_2 x_2 \leq -1 \quad \text{for } y_i = -1$$

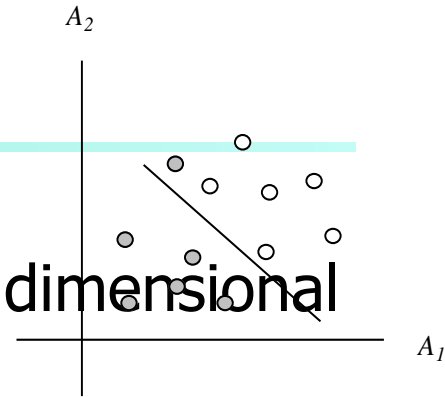
- Any training tuples that fall on hyperplanes H_1 or H_2 (i.e., the sides defining the margin) are **support vectors**
- This becomes a **constrained (convex) quadratic optimization** problem: Quadratic objective function and linear constraints \rightarrow *Quadratic Programming (QP)* \rightarrow Lagrangian multipliers

Why Is SVM Effective on High Dimensional Data?

- The **complexity** of trained classifier is characterized by the # of support vectors rather than the dimensionality of the data
- The **support vectors** are the essential or critical training examples — they lie closest to the decision boundary (MMH)
- If all other training examples are removed and the training is repeated, the same separating hyperplane would be found
- The number of support vectors found can be used to compute an (upper) bound on the expected error rate of the SVM classifier, which is independent of the data dimensionality
- Thus, an SVM with a small number of support vectors can have good generalization, even when the dimensionality of the data is high

SVM—Linearly Inseparable

- Transform the original input data into a higher dimensional space



Example 6.8 Nonlinear transformation of original input data into a higher dimensional space. Consider the following example. A 3D input vector $\mathbf{X} = (x_1, x_2, x_3)$ is mapped into a 6D space Z using the mappings $\phi_1(\mathbf{X}) = x_1, \phi_2(\mathbf{X}) = x_2, \phi_3(\mathbf{X}) = x_3, \phi_4(\mathbf{X}) = (x_1)^2, \phi_5(\mathbf{X}) = x_1x_2$, and $\phi_6(\mathbf{X}) = x_1x_3$. A decision hyperplane in the new space is $d(\mathbf{Z}) = \mathbf{WZ} + b$, where \mathbf{W} and \mathbf{Z} are vectors. This is linear. We solve for \mathbf{W} and b and then substitute back so that we see that the linear decision hyperplane in the new (\mathbf{Z}) space corresponds to a nonlinear second order polynomial in the original 3-D input space,

$$\begin{aligned} d(\mathbf{Z}) &= w_1x_1 + w_2x_2 + w_3x_3 + w_4(x_1)^2 + w_5x_1x_2 + w_6x_1x_3 + b \\ &= w_1z_1 + w_2z_2 + w_3z_3 + w_4z_4 + w_5z_5 + w_6z_6 + b \end{aligned} \quad \blacksquare$$

- Search for a linear separating hyperplane in the new space

SVM vs. Neural Network

■ SVM

- Deterministic algorithm
- Nice generalization properties
- Hard to learn – learned in batch mode using quadratic programming techniques
- Using kernels can learn very complex functions

■ Neural Network

- Nondeterministic algorithm
- Generalizes well but doesn't have strong mathematical foundation
- Can easily be learned in incremental fashion
- To learn complex functions—use multilayer perceptron (nontrivial)

Lazy vs. Eager Learning

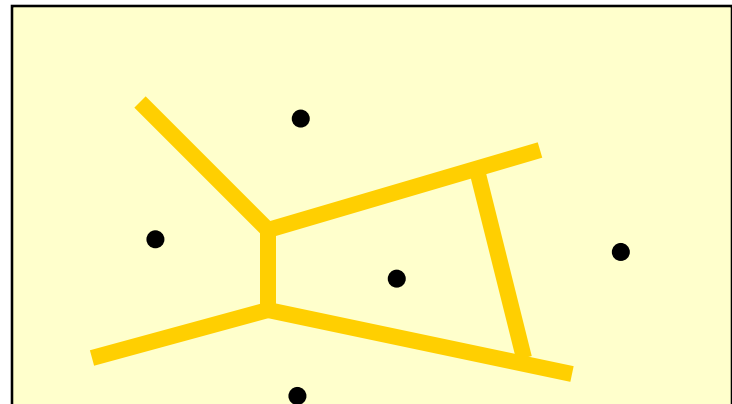
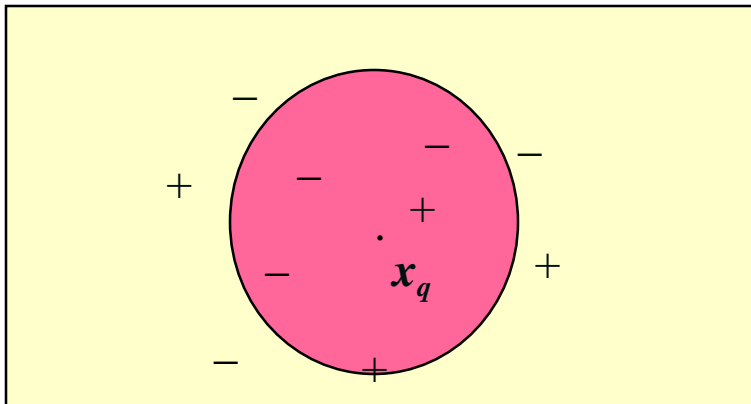
- Lazy vs. eager learning
 - **Lazy learning** (e.g., instance-based learning): Simply stores training data (or only minor processing) and waits until it is given a test tuple
 - **Eager learning** (the above discussed methods): Given a set of training tuples, constructs a classification model before receiving new (e.g., test) data to classify
- Lazy: less time in training but more time in predicting
- Accuracy
 - Lazy method effectively uses a richer hypothesis space since it uses many local linear functions to form an implicit global approximation to the target function
 - Eager: must commit to a single hypothesis that covers the entire instance space

Lazy Learner: Instance-Based Methods

- Instance-based learning:
 - Store training examples and delay the processing (“lazy evaluation”) until a new instance must be classified
- Typical approaches
 - k-nearest neighbor approach
 - Instances represented as points in a Euclidean space.
 - Locally weighted regression
 - Constructs local approximation
 - Case-based reasoning
 - Uses symbolic representations and knowledge-based inference

The k -Nearest Neighbor Algorithm

- All instances correspond to points in the n -D space
- The nearest neighbor are defined in terms of Euclidean distance, $\text{dist}(\mathbf{X}_1, \mathbf{X}_2)$
- Target function could be discrete- or real- valued
- For discrete-valued, k -NN returns the most common value among the k training examples nearest to x_q
- Voronoi diagram: the decision surface induced by 1-NN for a typical set of training examples



Discussion on the ~~k~~-NN Algorithm

- k -NN for real-valued prediction for a given unknown tuple
 - Returns the mean values of the k nearest neighbors
- Distance-weighted nearest neighbor algorithm
 - Weight the contribution of each of the k neighbors according to their distance to the query x_q
 - Give greater weight to closer neighbors
- Robust to noisy data by averaging k -nearest neighbors
- Curse of dimensionality: distance between neighbors could be dominated by irrelevant attributes
 - To overcome it, axes stretch or elimination of the least relevant attributes

$$w \equiv \frac{1}{d(x_q, x_i)^2}$$

Case-Based Reasoning (CBR)

- **CBR:** Uses a database of problem solutions to solve new problems
- Store symbolic description (tuples or cases)—not points in a Euclidean space
- Applications: Customer-service (product-related diagnosis), legal ruling
- Methodology
 - Instances represented by rich symbolic descriptions (e.g., function graphs)
 - Search for similar cases, multiple retrieved cases may be combined
 - Tight coupling between case retrieval, knowledge-based reasoning, and problem solving
- Challenges
 - Find a good similarity metric
 - Indexing based on syntactic similarity measure, and when failure, backtracking, and adapting to additional cases

Other Classification Methods

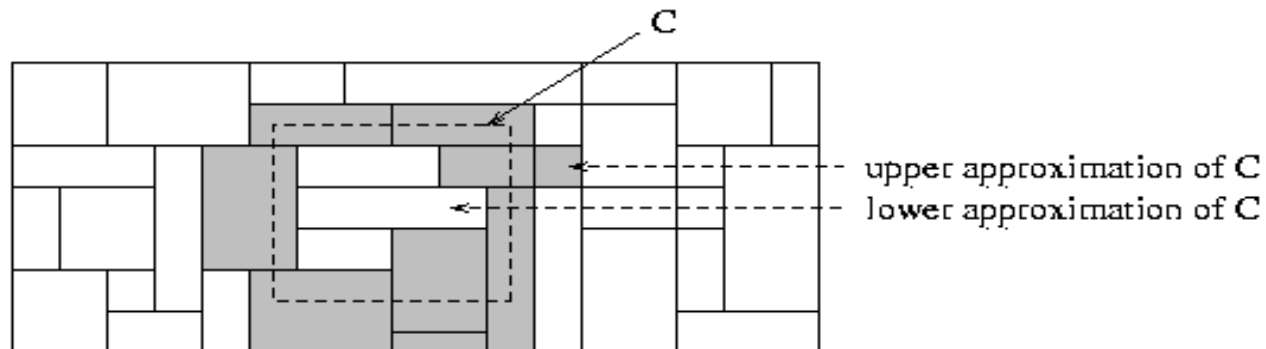
- Genetic Algorithms (GA)
- Rough Set Approach
- Fuzzy Set Approaches

Genetic Algorithms (GA)

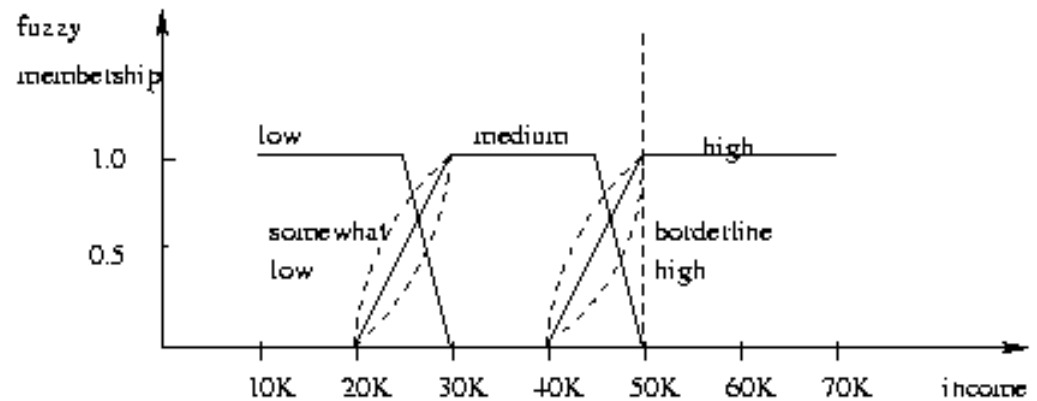
- Genetic Algorithm: based on an analogy to biological evolution
- An initial **population** is created consisting of randomly generated rules
 - Each rule is represented by a string of bits
 - E.g., if A_1 and $\neg A_2$ then C_2 can be encoded as 100
 - If an attribute has $k > 2$ values, k bits can be used
- Based on the notion of survival of the **fittest**, a new population is formed to consist of the fittest rules and their offspring
- The *fitness of a rule* is represented by its classification accuracy on a set of training examples
- Offspring are generated by *crossover* and *mutation*
- The process continues until a population P evolves *when each rule in P satisfies a prespecified threshold*
- Slow but easily parallelizable

Rough Set Approach

- Rough sets are used to **approximately** or “**roughly**” define equivalent classes
- A rough set for a given class C is approximated by two sets: a **lower approximation** (certain to be in C) and an **upper approximation** (cannot be described as not belonging to C)
- Finding the minimal subsets (**reducts**) of attributes for feature reduction is NP-hard but a **discernibility matrix** (which stores the differences between attribute values for each pair of data tuples) is used to reduce the computation intensity



Fuzzy Set Approaches



- Fuzzy logic uses truth values between 0.0 and 1.0 to represent the degree of membership (such as in a *fuzzy membership graph*)
- Attribute values are converted to fuzzy values. Ex.:
 - Income, x , is assigned a **fuzzy membership value** to each of the discrete categories {low, medium, high}, e.g. \$49K belongs to “medium income” with fuzzy value 0.15 but belongs to “high income” with fuzzy value 0.96
 - Fuzzy membership values do not have to sum to 1.
- Each applicable rule contributes a vote for membership in the categories
- Typically, the truth values for each predicted category are summed, and these sums are combined