ORACLE
Certified Professional
Sun Certified
JAVA Programmer
Microsoft Certified
Professional

by

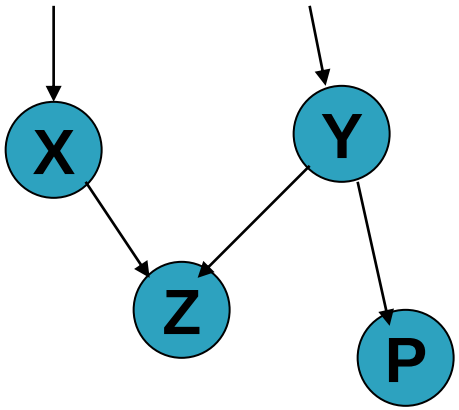
Sohail IMRAN سهیل عمران

Ref. : 1-Data Mining: Concepts and Techniques

Classification: Advanced Methods

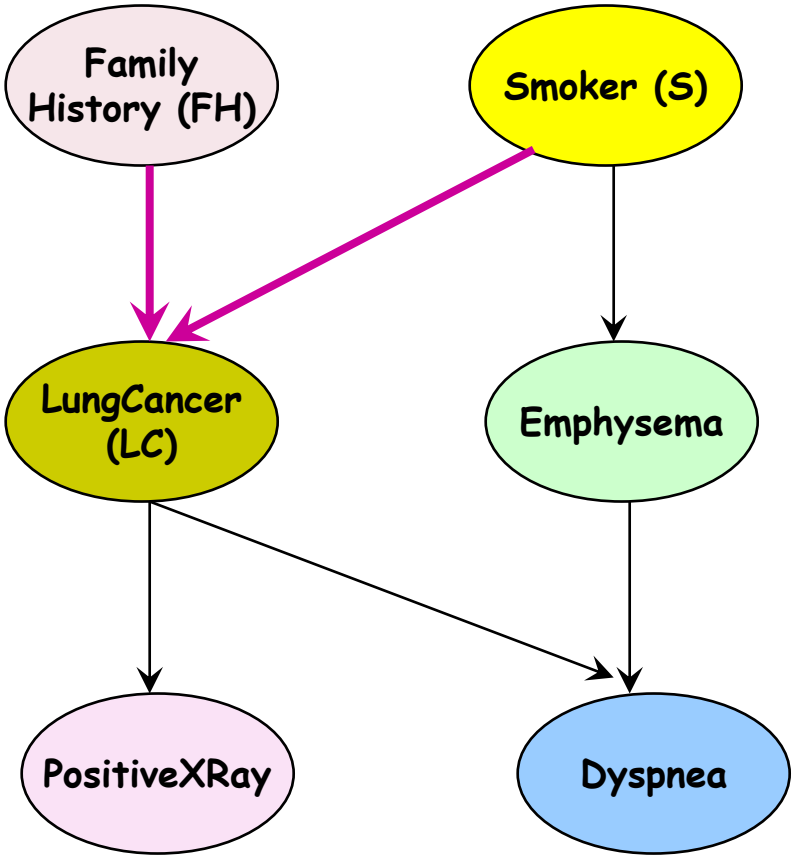
Bayesian Belief Networks

- **Bayesian belief networks** (also known as **Bayesian networks**, **probabilistic networks**): allow *class conditional independencies* between *subsets* of variables
- *A (directed acyclic) graphical model of causal relationships*
  - Represents dependency among the variables
  - Gives a specification of joint probability distribution



- ❑ Nodes: random variables
- ❑ Links: dependency
- ❑ X and Y are the parents of Z, and Y is the parent of P
- ❑ No dependency between Z and P
- ❑ Has no loops/cycles

Bayesian Belief Network: An Example



Bayesian Belief Network

CPT: Conditional Probability Table for variable LungCancer:

	(FH, S)	(FH, ~S)	(~FH, S)	(~FH, ~S)
LC	0.8	0.5	0.7	0.1
~LC	0.2	0.5	0.3	0.9

shows the conditional probability for each possible combination of its parents

Derivation of the probability of a particular combination of values of **X**, from CPT:

$$P(x_1,\ldots,x_n) = \prod_{i=1}^{n} P(x_i \mid Parents(Y_i))$$

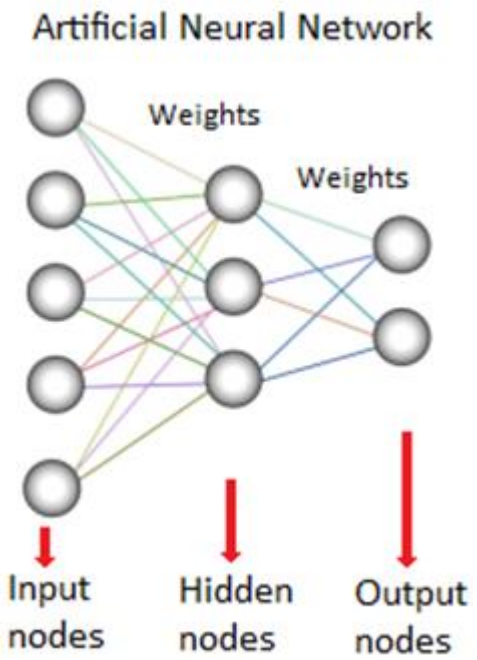
Training Bayesian Networks: Several Scenarios

- Scenario 1: Given both the network structure and all variables observable: *compute only the CPT entries*
- Scenario 2: Network structure known, some variables hidden: *gradient descent* (greedy hill-climbing) method, i.e., search for a solution along the steepest descent of a criterion function
  - Weights are initialized to random probability values
  - At each iteration, it moves towards what appears to be the best solution at the moment, w.o. backtracking
  - Weights are updated at each iteration & converge to local optimum
- Scenario 3: Network structure unknown, all variables observable: search through the model space to *reconstruct network topology*
- Scenario 4: Unknown structure, all hidden variables: No good algorithms known for this purpose
- D. Heckerman. A Tutorial on Learning with Bayesian Networks. In *Learning in Graphical Models*, M. Jordan, ed.. MIT Press, 1999.

Artificial Neutral Network

An artificial neutral network (**ANN**) is a system that is based on the biological neural network, such as the brain. The brain has approximately 100 billion neurons, which communicate through electro-chemical signals. The neurons are connected through junctions called synapses. Each neuron receives thousands of connections with other neurons, constantly receiving incoming signals to reach the cell body. If the resulting sum of the signals surpasses a certain threshold, a response is sent through the axon.

An ANN is comprised of a network of artificial neurons (also known as "**nodes**"). These nodes are **connected** to each other, and the strength of their connections to one another is assigned a value based on their **strength**:
inhibition (maximum being -1.0) or **excitation** (maximum being +1.0).
If the value of the connection is **high**, then it indicates that there is a **strong** connection.
Within each node's design, a transfer function is built in.
There are three types of neurons in an ANN:
Input nodes, **Hidden nodes**, and **Output nodes**.



ANN - Process

The input nodes take in information, in the form which can be numerically expressed.

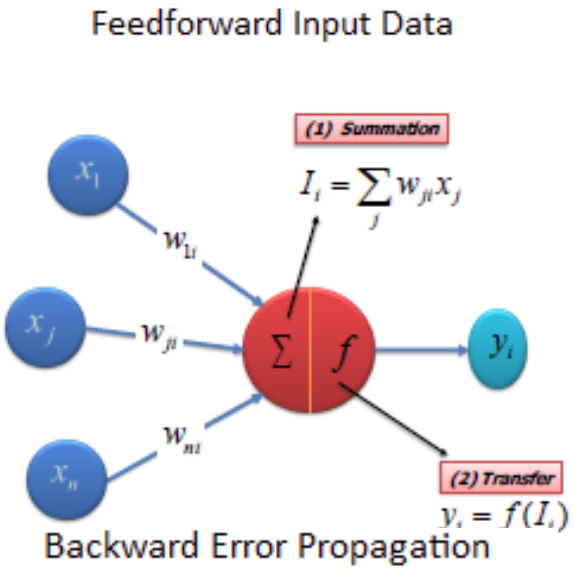
The information is presented as **activation values**, e.g. x_1, x_j, where each node is given a number, the higher the number, the greater the activation. This information is then passed throughout the network.

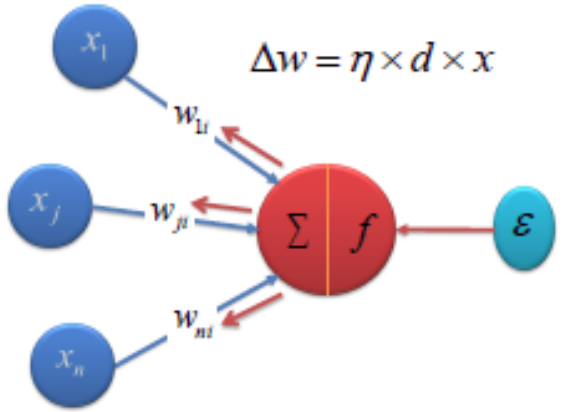
Based on the connection strengths (**weights**) , e.g. w_{1i}, w_{ji}, inhibition or excitation, and transfer functions, the activation value is passed from node to node.

Each of the nodes sums the activation values it receives; it then modifies the value based on its **transfer function**. The activation flows through the network, through hidden layers, until it reaches the output nodes.

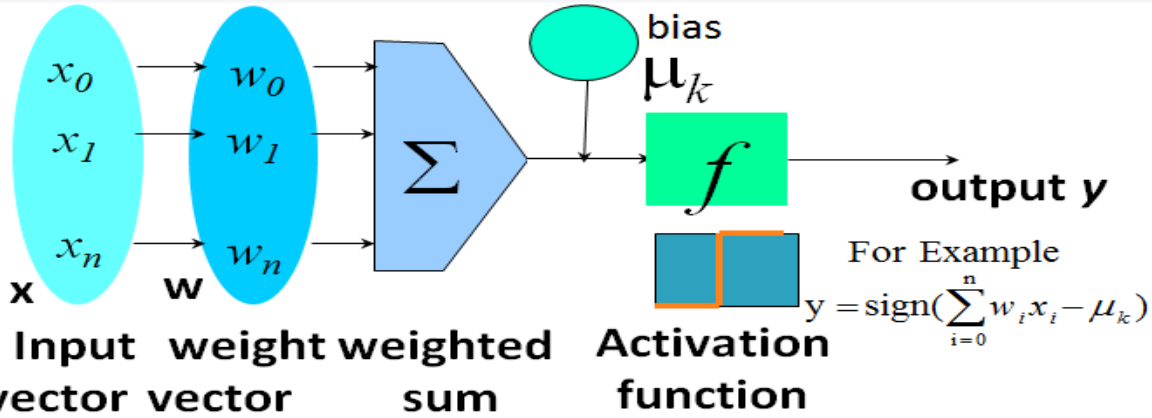
The output nodes then reflect the input in a meaningful way to the outside world.

The difference between predicted value and actual value (error) will be propagated backward by apportioning them to each node's weights according to the amount of this error the node is responsible for (e.g., gradient descent algorithm).





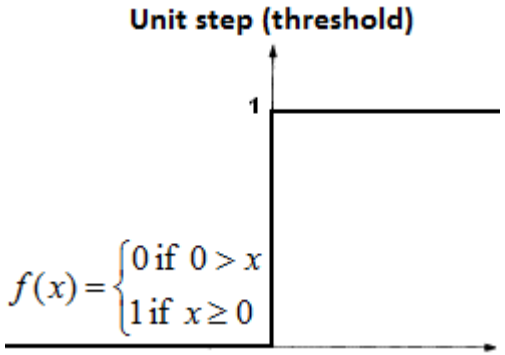
Transfer (Activation) Functions



The transfer function translates the input signals to output signals. Four types of transfer functions are commonly used, Unit step (threshold), sigmoid, piecewise linear, and Gaussian.

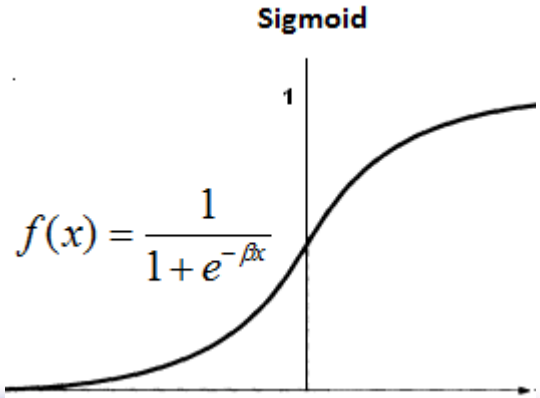
Unit step (threshold)

The output is set at one of two levels, depending on whether the total input is greater than or less than some threshold value.



Sigmoid

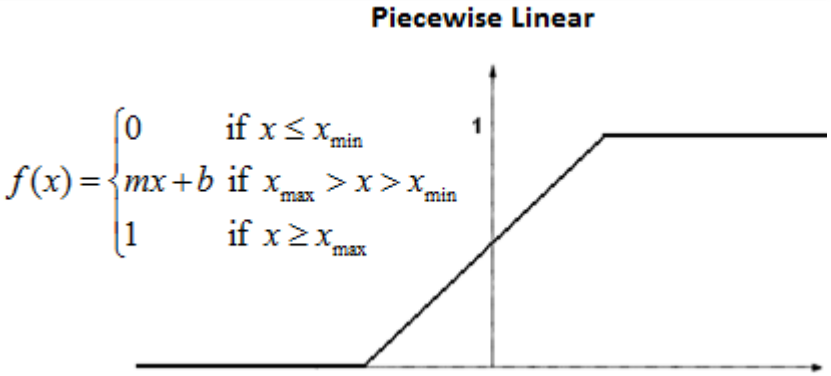
The sigmoid function consists of 2 functions, logistic and tangential. The values of logistic function range from 0 and 1 and -1 to +1 for tangential function.



Transfer (Activation) Functions

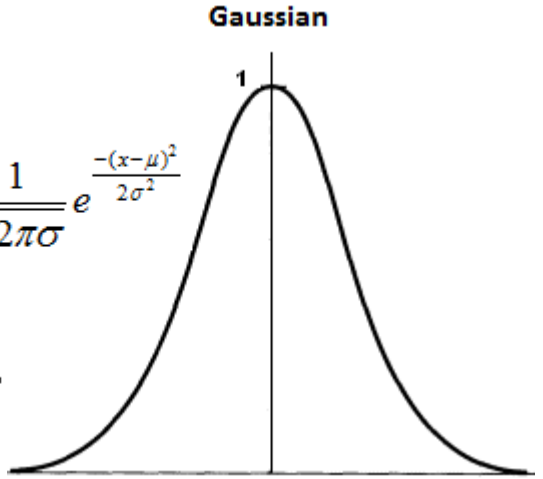
Piecewise Linear

The output is proportional to the total weighted output.



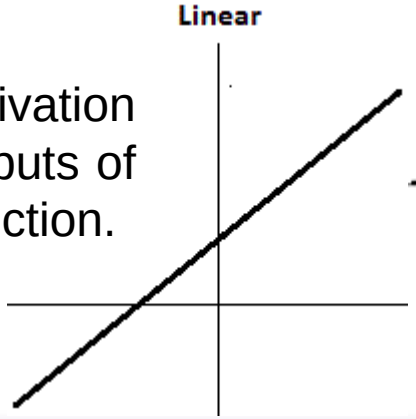
Gaussian

Gaussian functions are bell-shaped curves that are continuous. The node output (high/low) is interpreted in terms of class membership (1/0), depending on how close the net input is to a chosen value of average.



Linear

Like a linear regression, a linear activation function transforms the weighted sum inputs of the neuron to an output using a linear function.



ANN - Types

There are different types of neural networks, but they are generally classified into feed-forward and feed-back networks.

A **feed-forward** network is a non-recurrent network which contains inputs, outputs, and hidden layers; the signals can only travel in one direction. Input data is passed onto a layer of processing elements where it performs calculations. Each processing element makes its computation based upon a weighted sum of its inputs. The new calculated values then become the new input values that feed the next layer. This process continues until it has gone through all the layers and determines the output. A threshold transfer function is sometimes used to quantify the output of a neuron in the output layer. Feed-forward networks include Perceptron (linear and non-linear) and Radial Basis Function networks. Feed-forward networks are often used in data mining.

A **feed-back** network has feed-back paths meaning they can have signals traveling in both directions using loops. All possible connections between neurons are allowed. Since loops are present in this type of network, it becomes a non-linear dynamic system which changes continuously until it reaches a state of equilibrium. Feed-back networks are often used in associative memories and optimization problems where the network looks for the best arrangement of interconnected factors.

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

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

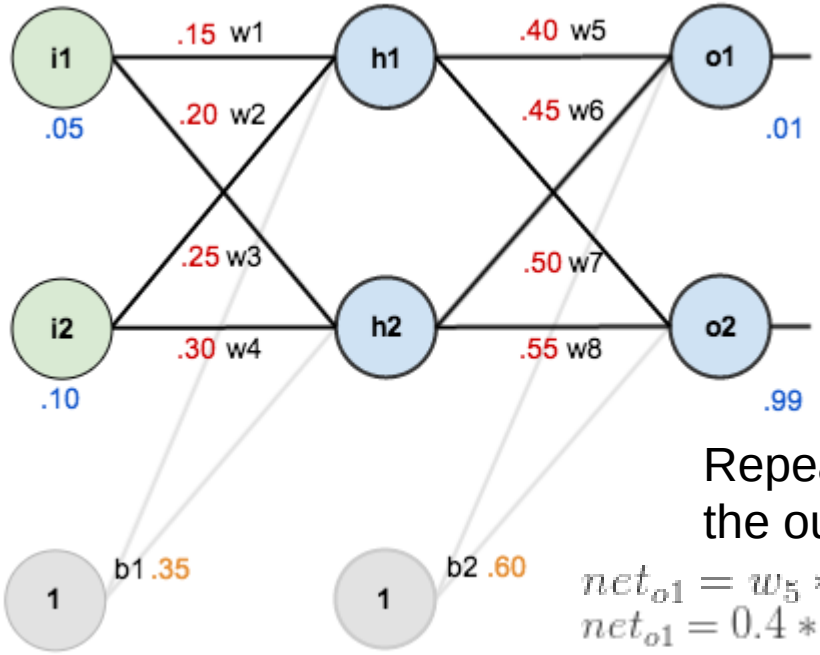
Backpropagation

- 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.)

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

The Forward Pass



The total net input for h1:

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$
$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

Squash it using the logistic function to get the output of h1:

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

$$out_{h2} = 0.596884378$$

Repeat this process for the output layer neurons, using the output from the hidden layer neurons as inputs.

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$
$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$
$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

$$out_{o2} = 0.772928465$$

Calculating the Total Error:

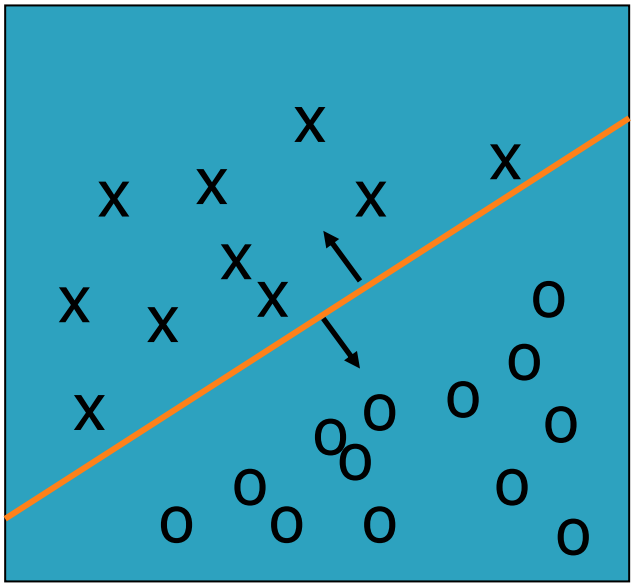
$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

$$E_{o2} = 0.023560026$$

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

Classification: A Mathematical Mapping

- Classification: predicts categorical class labels
  - E.g., Personal homepage classification
    - $x_i = (x_1, x_2, x_3, \ldots)$, $y_i = +1$ or -1
    - x_1 : # of word "homepage"
    - x_2 : # of word "welcome"
- Mathematically, $x \in X = \Re^n$, $y \in Y = \{+1, -1\}$,
  - 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

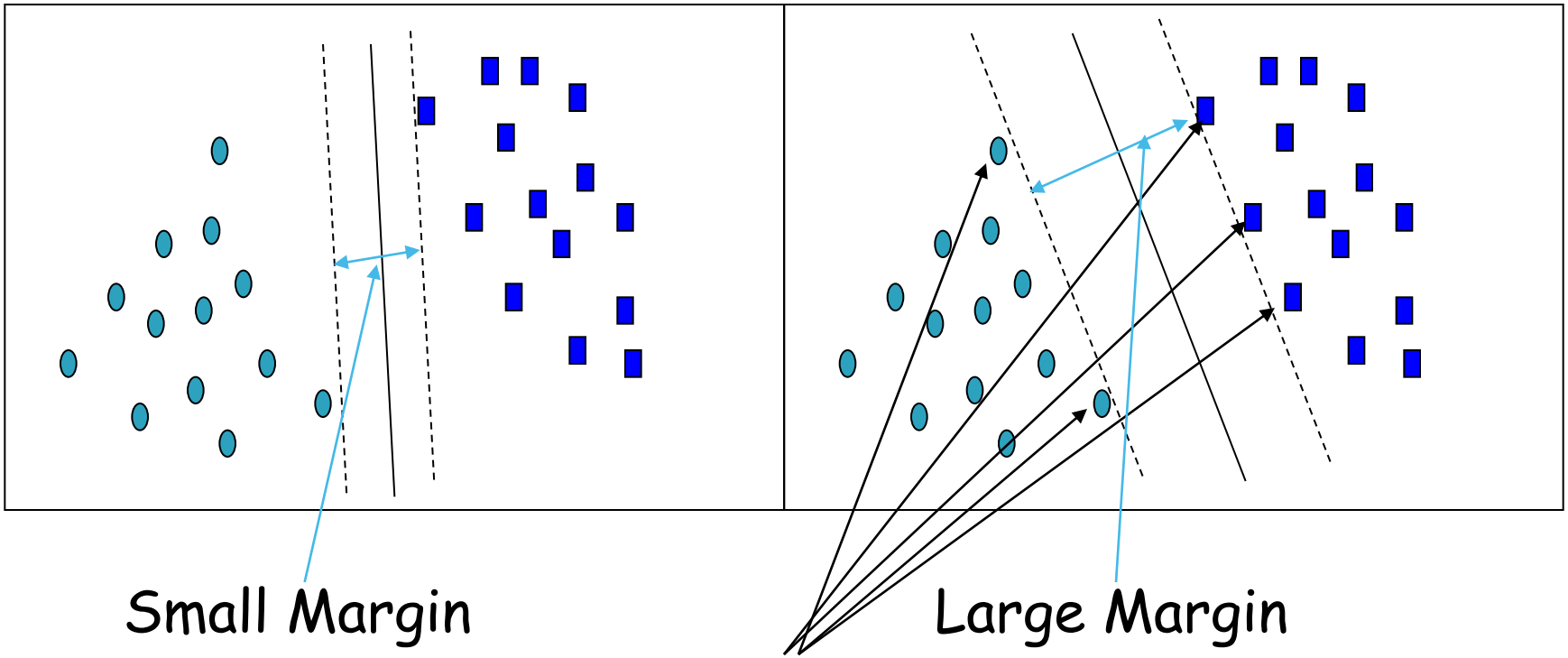
SVM—Support Vector Machines

- 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)

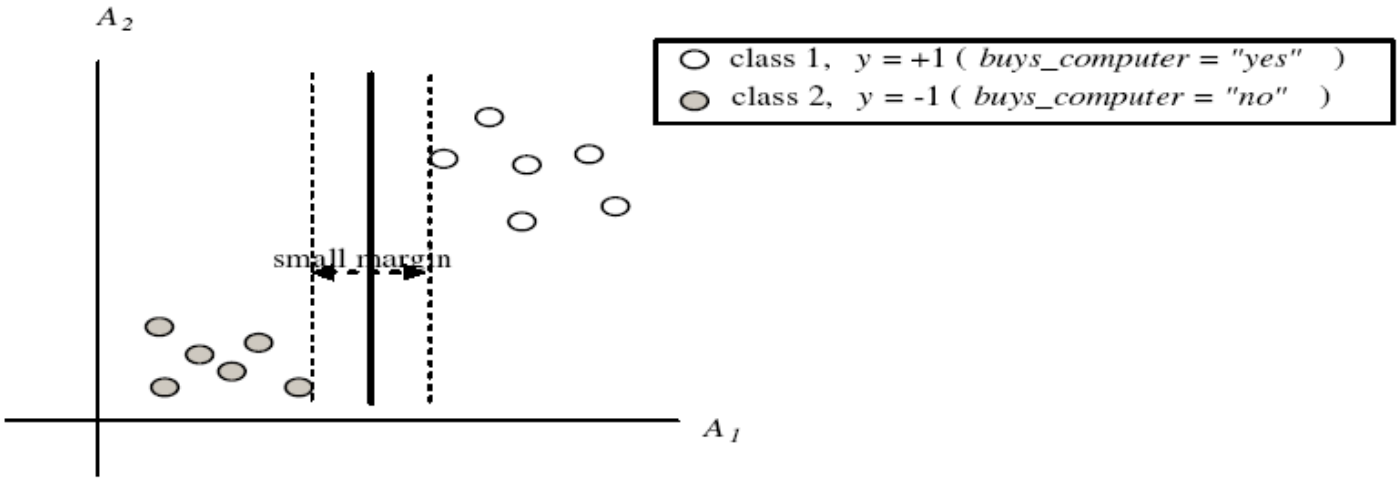
SVM—History and Applications

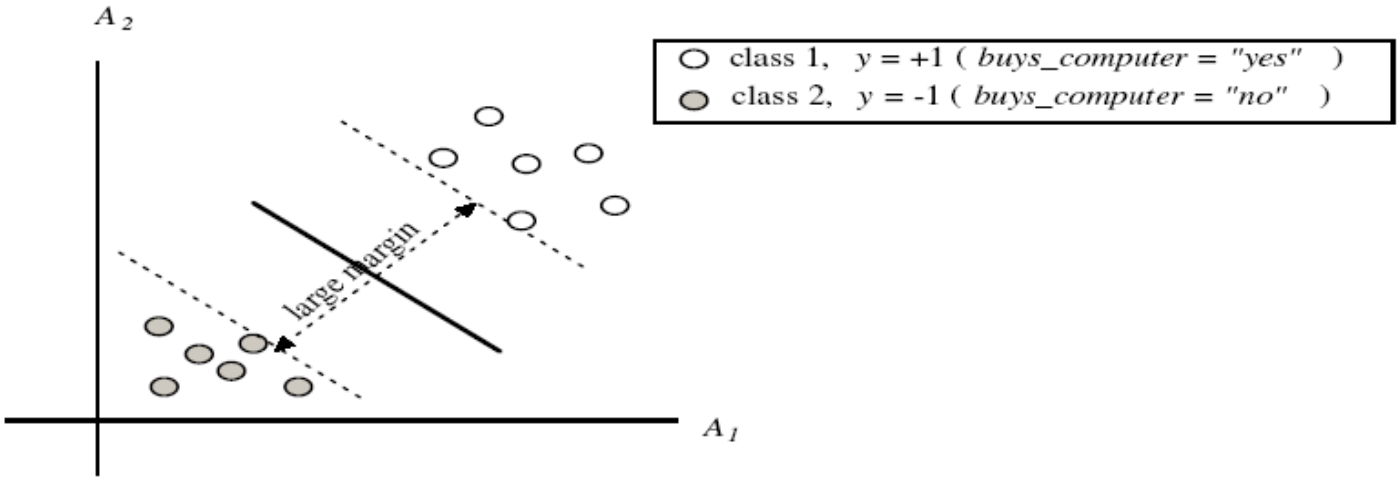
- Vapnik and colleagues (1992)—groundwork from Vapnik & Chervonenkis' statistical learning theory in 1960s
- Features: training can be slow but accuracy is high owing to their ability to model complex nonlinear decision boundaries (margin maximization)
- Used for: classification and numeric prediction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

SVM—General Philosophy

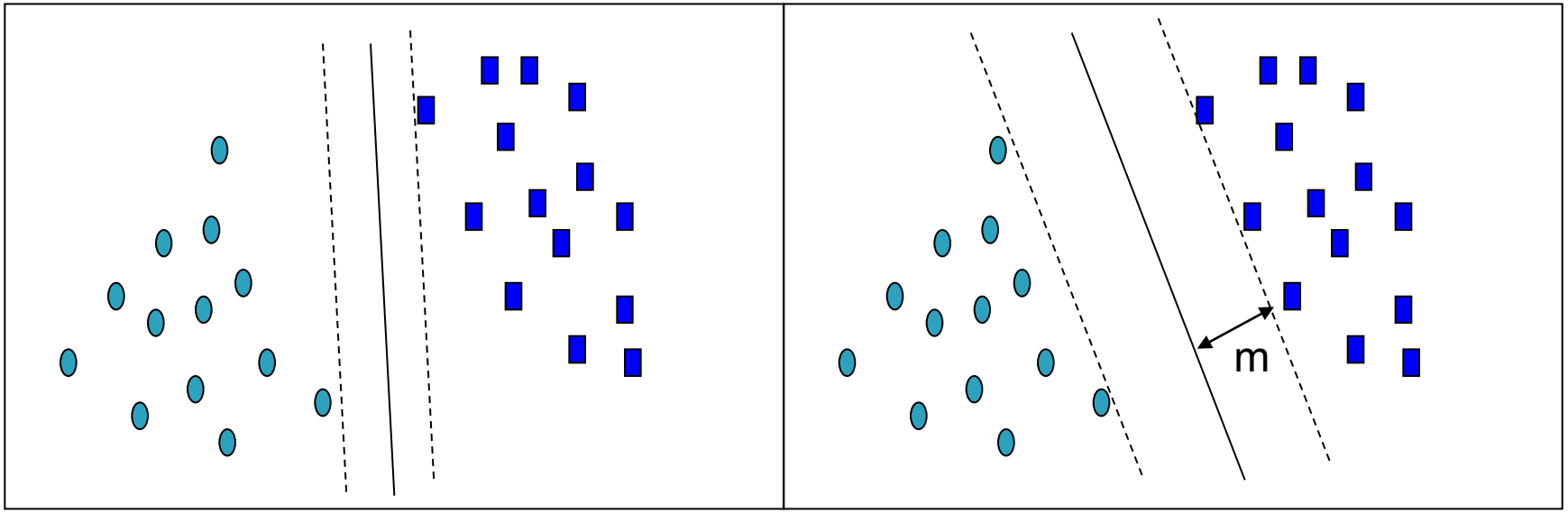


SVM—Margins and Support Vectors





SVM—When Data Is Linearly Separable



Let data D be $(\mathbf{X}_1, y_1), \ldots, (\mathbf{X}_{|D|}, y_{|D|})$, 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 $\mathbf{W}=\{w_1, w_2, \ldots, w_n\}$ 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: $w_0 + w_1 x_1 + w_2 x_2 \geq 1$ for $y_i = +1$, and

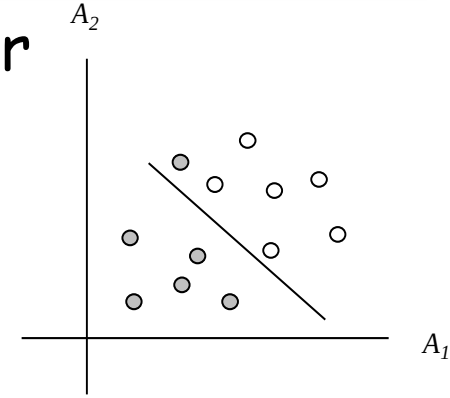
  H_2: $w_0 + w_1 x_1 + w_2 x_2 \leq -1$ for $y_i = -1$
- Any training tuples that fall on hyperplanes H_1 or H_2 (i.e., the sides defining the margin) are **support vectors**
- This becomes a **constrained (convex) quadratic optimization** problem: Quadratic objective function and linear constraints → *Quadratic Programming (QP)* → 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 $\phi_1(X) = x_1, \phi_2(X) = x_2, \phi_3(X) = x_3, \phi_4(X) = (x_1)^2, \phi_5(X) = x_1x_2$, and $\phi_6(X) = x_1x_3$. 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,

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

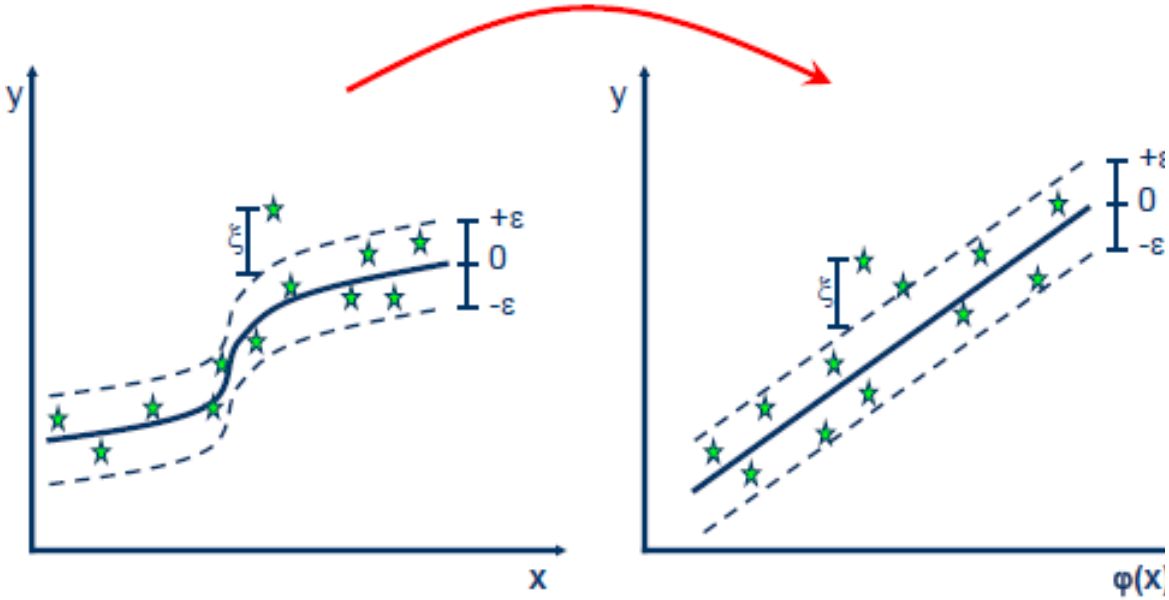
- Search for a linear separating hyperplane in the new space

Non-linear SVM

The kernel functions transform the data into a higher dimensional feature space to make it possible to perform the linear separation.

$$y = \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) \cdot \langle \varphi(x_i), \varphi(x) \rangle + b$$

$$y = \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) \cdot K(x_i, x) + b$$



Kernel functions

Polynomial

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i . \mathbf{x}_j)^d$$

Gaussian Radial Basis function

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$$

SVM: Different Kernel functions

- Instead of computing the dot product on the transformed data, it is math. equivalent to applying a kernel function $K(X_i, X_j)$ to the original data, i.e., $K(X_i, X_j) = \Phi(X_i)\,\Phi(X_j)$
- Typical Kernel Functions

$$\text{Polynomial kernel of degree } h: \quad K(X_i, X_j) = (X_i \cdot X_j + 1)^h$$

$$\text{Gaussian radial basis function kernel}: \quad K(X_i, X_j) = e^{-\|X_i - X_j\|^2/2\sigma^2}$$

$$\text{Sigmoid kernel}: \quad K(X_i, X_j) = \tanh(\kappa X_i \cdot X_j - \delta)$$

- SVM can also be used for classifying multiple (> 2) classes and for regression analysis (with additional parameters)

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)

Associative Classification

- Associative classification: Major steps
  - Mine data to find strong associations between frequent patterns (conjunctions of attribute-value pairs) and class labels
  - Association rules are generated in the form of

    $P_1 \wedge p_2 \ldots \wedge p_l \rightarrow$ "$A_{class} = C$" (conf, sup)
  - Organize the rules to form a rule-based classifier
- Why effective?
  - It explores highly confident associations among multiple attributes and may overcome some constraints introduced by decision-tree induction, which considers only one attribute at a time
  - Associative classification has been found to be often more accurate than some traditional classification methods, such as C4.5

Typical Associative Classification Methods

- **CBA** (Classification Based on Associations: Liu, Hsu & Ma, KDD'98)
  - Mine possible association rules in the form of
    - Cond-set (a set of attribute-value pairs) → class label
  - Build classifier: Organize rules according to decreasing precedence based on confidence and then support
- **CMAR** (Classification based on Multiple Association Rules: Li, Han, Pei, ICDM'01)
  - Classification: Statistical analysis on multiple rules
- **CPAR** (Classification based on Predictive Association Rules: Yin & Han, SDM'03)
  - Generation of predictive rules (FOIL-like analysis) but allow covered rules to retain with reduced weight
  - Prediction using best k rules
  - High efficiency, accuracy similar to CMAR

Lazy vs. Eager Learning

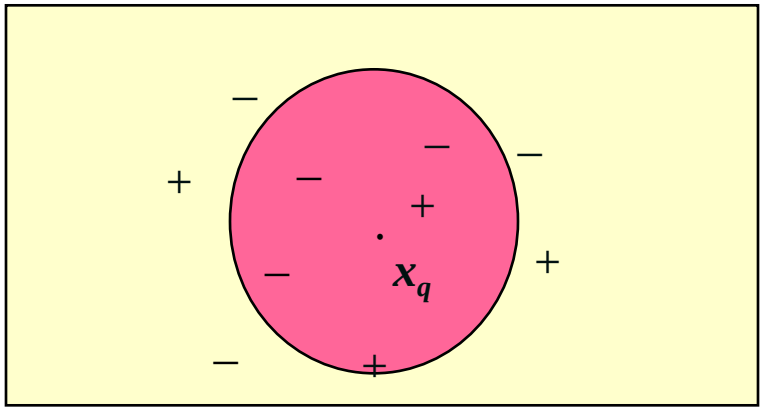
- Lazy vs. eager learning
  - **Lazy learning** (e.g., instance-based learning): Simply stores training data (or only minor processing) and waits until it is given a test tuple
  - **Eager learning** (the above discussed methods): Given a set of training tuples, constructs a classification model before 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

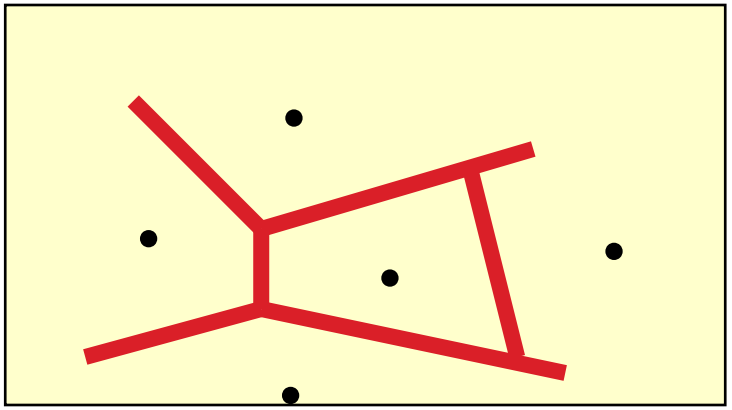
Lazy Learner: Instance-Based Methods

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

Case-Based Reasoning (CBR)

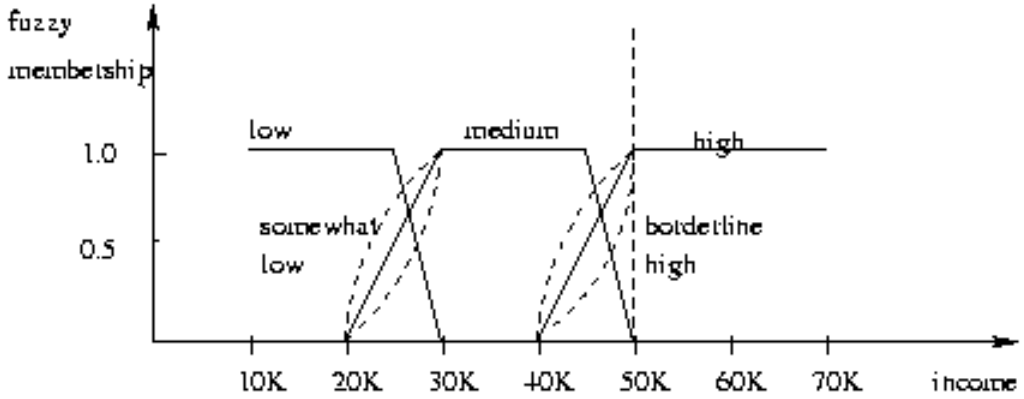
- **CBR**: Uses a database of problem solutions to solve new problems
- Store symbolic description (tuples or cases)—not points in a Euclidean space
- Applications: Customer-service (product-related diagnosis), legal ruling
- Methodology
  - Instances represented by rich symbolic descriptions (e.g., 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

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

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

What Is Prediction?

- (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
- Major method for prediction: regression
  - model the relationship between one or more *independent* or **predictor** variables and a *dependent* or **response** variable
- Regression analysis
  - Linear and multiple regression
  - Non-linear regression
  - Other regression methods: generalized linear model, Poisson regression, log-linear models, regression trees

Linear Regression

- Linear regression: involves a response variable y and a single predictor variable x

  $y = w_0 + w_1 x$

  where w_0 (y-intercept) and w_1 (slope) are regression coefficients
- Method of least squares: estimates the best-fitting straight line

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

- Multiple linear regression: involves more than one predictor variable
  - Training data is of the form $(\mathbf{X}_1, y_1), (\mathbf{X}_2, y_2), \ldots, (\mathbf{X}_{|D|}, y_{|D|})$
  - Ex. For 2-D data, 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

The Perceptrons

Developed by Frank Rosenblatt (1958).

Its learning rule is superior than the Hebb learning rule.

Has been proven by Rosenblatt that the weights can converge on particular applications.

However, the Perceptron does not work for nonlinear applications as proven by Minsky and Papert (1969).

Activation function used is the binary step function with an arbitrary, but fixed threshold.

Weights are adjusted by the Perceptron learning rule:

The Perceptron Algorithm

Step 0. Set up the NN model (which follows the problem to be solved)

Initialize weights and bias.

(For simplicity, set weights and bias to zero or randomize)

Set learning rate, α $(0<\alpha<=1)$ and threshold $(0<\theta<1)$

(For simplicity, α can be set to 1.)

Step 1. While stopping condition is false, do Steps 2-6.

Step 2. For each training pair u:t, do Steps 3-5.

Step 3. Set activations of input units:

$x_i = u_i$.

Step 4. Compute response of output unit:

$$s = b + \sum_i x_i w_i$$

$$y = \begin{cases} 1 & \text{if } s > \theta \\ 0 & \text{if } -\theta \le s \le \theta \\ -1 & \text{if } s < -\theta \end{cases}$$

θ is a threshold value assigned between 0 and 1

The Perceptron Algorithm

Step 5 Update weights and bias if an error occurred for this pattern:

$$If \;\; y \neq t$$

$$w_i(new) = w_i(old) + \alpha \, t x_i .$$

$$b(new) = b(old) + \alpha \, t.$$

else

$$w_i(new) = w_i(old),$$

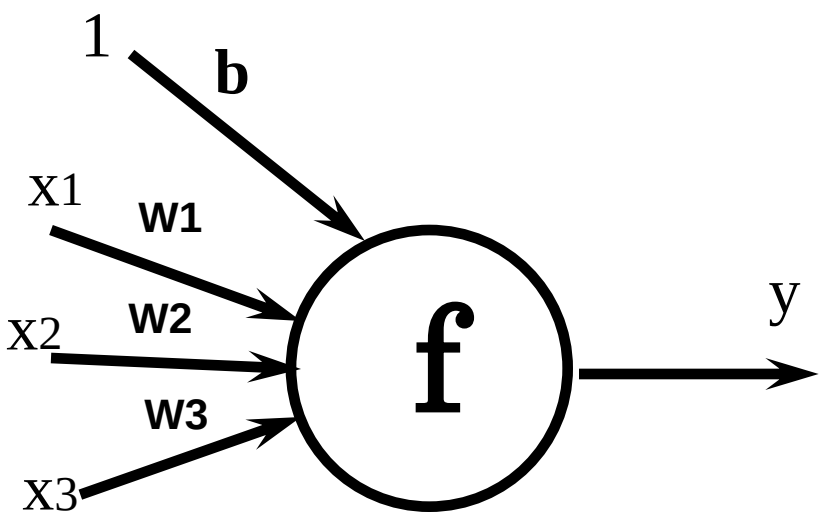
$$b(new) = b(old).$$

Step 6. Test stopping condition :

If no weights changed in Step 2, stop; else, continue.

Example

x1	x2	x3	t
-1	-1	-1	-1
-1	-1	1	1
-1	1	-1	1
-1	1	1	1
1	-1	-1	1
1	-1	1	1
1	1	-1	1
1	1	1	1



- Using the Perceptron network (3-input), solve the OR problem above (using bipolar activations) and show the results of the weights adaptation for each pattern over 1 epoch.
- Choose all initial weights to be -0.5, α=0.25, and θ=0.1
- Fill up the following table.

Iteration # 1

s = b + (w1*x1) + (w2*-x2) + (w3*x3)

Iter. #1 (-1 -1 -1 t=-1) :

s = 1 + (-0.5*-1) + (-0.5*-1) + (-0.5*-1)
s = 1 + 0.5 + 0.5 + 0.5 = **2.5**
y = 1 (2.5>0.1) → (s>θ)
t = -1

y != t, hence

Adapt weights & bias

w1(new) = w1(old) + αtx1
= -0.5 + (0.25*-1*-1)
= -0.25

w2(new) = w2(old) + αtx2
= -0.5 + (0.25*-1*-1)
= -0.25

w3(new) = w3(old) + αtx3
= -0.5 + (0.25*-1*-1)
= -0.25

b(new) = b(old) + αt
= 1 + (0.25*-1)
= 0.75

Iteration # 2 & 3

Iter. #2 (-1 -1 1 t=1) :

s = 0.75 + (-0.25*-1) + (-0.25*-1) + (-0.25*1)
s = 0.75 + 0.25 + 0.25 - 0.25 = **1**
y = 1 (1>0.1)
t = 1

y = t, hence

No Adaptation

w1(new)	=	w1(old)	=	-0.25
w2(new)	=	w2(old)	=	-0.25
w3(new)	=	w3(old)	=	-0.25
b(new)	=	b(old)	=	0.75

Iter. #3 (-1 1 -1 t=1) :

s = 0.75 + (-0.25*-1) + (-0.25*1) + (-0.25*-1)
s = 0.75 + 0.25 + 0.25 - 0.25 = **1**
y = 1 (1>0.1)
t = 1

y = t, hence

No Adaptation

w1(new)	=	w1(old)	=	-0.25
w2(new)	=	w2(old)	=	-0.25
w3(new)	=	w3(old)	=	-0.25
b(new)	=	b(old)	=	0.75

Iteration # 4 & 5

Iter. #4 (-1 1 1 t=1) :

s = 0.75 + (-0.25*-1) + (-0.25*1) + (-0.25*1)
s = 0.75 + 0.25 - 0.25 - 0.25 = **0.5**
y = 1 (0.5>0.1)
t = 1

y = t, hence

No Adaptation

w1(new)	=	w1(old)	=	-0.25
w2(new)	=	w2(old)	=	-0.25
w3(new)	=	w3(old)	=	-0.25
b(new)	=	b(old)	=	0.75

Iter. #5 (1 -1 -1 t=1) :

s = 0.75 + (-0.25*1) + (-0.25*-1) + (-0.25*-1)
s = 0.75 - 0.25 + 0.25 + 0.25 = **1**
y = 1 (1>0.1)
t = 1

y = t, hence

No Adaptation

w1(new)	=	w1(old)	=	-0.25
w2(new)	=	w2(old)	=	-0.25
w3(new)	=	w3(old)	=	-0.25
b(new)	=	b(old)	=	0.75

Iteration # 6 & 7

Iter. #6 (1 -1 1 t=1) :

s = 0.75 + (-0.25*1) + (-0.25*-1) + (-0.25*1)
s = 0.75 - 0.25 + 0.25 - 0.25 = **0.5**
y = 1 (0.5>0.1)
t = 1

y = t, hence

No Adaptation

w1(new)	=	w1(old)	=	-0.25
w2(new)	=	w2(old)	=	-0.25
w3(new)	=	w3(old)	=	-0.25
b(new)	=	b(old)	=	0.75

Iter. #7 (1 1 -1 t=1) :

s = 0.75 + (-0.25*1) + (-0.25*1) + (-0.25*-1)
s = 0.75 - 0.25 - 0.25 + 0.25 = **0.5**
y = 1 (0.5>0.1)
t = 1

y = t, hence

No Adaptation

w1(new)	=	w1(old)	=	-0.25
w2(new)	=	w2(old)	=	-0.25
w3(new)	=	w3(old)	=	-0.25
b(new)	=	b(old)	=	0.75

Iteration #8

Iter. #8 (1 1 1 t=1) :

s = 0.75 + (-0.25*1) + (-0.25*1) + (-0.25*1)
s = 0.75 - 0.25 - 0.25 - 0.25 = **0**
y = 0 (-0.1<0<0.1)
t = 1

y != t, hence

Adapt weights & bias

w1(new) = w1(old) + αtx1
= -0.25 + (0.25*1*1)
= 0

w2(new) = w2(old) + αtx2
= -0.25 + (0.25*1*1)
= 0

w3(new) = w3(old) + αtx3
= -0.25 + (0.25*1*1)
= 0

b(new) = b(old) + αt
= 0.75 + (0.25*1)
= 1

Epoch # 1

x1	x2	x3	s	y	t	w1	w2	w3	b
			-	-	-	-0.5	-0.5	-0.5	1
-1	-1	-1	2.5	1	-1	-0.25	-0.25	-0.25	0.75
-1	-1	1	1	1	1	-0.25	-0.25	-0.25	0.75
-1	1	-1	1	1	1	-0.25	-0.25	-0.25	0.75
-1	1	1	0.5	1	1	-0.25	-0.25	-0.25	0.75
1	-1	-1	1	1	1	-0.25	-0.25	-0.25	0.75
1	-1	1	0.5	1	1	-0.25	-0.25	-0.25	0.75
1	1	-1	0.5	1	1	-0.25	-0.25	-0.25	0.75
1	1	1	0	0	1	0	0	0	1

Epoch # 2

x1	x2	x3	s	y	t	w1	w2	w3	b
			-	-	-	0	0	0	1
-1	-1	-1	1	1	-1	0.25	0.25	0.25	0.75
-1	-1	1	0.5	1	1	0.25	0.25	0.25	0.75
-1	1	-1	0.5	1	1	0.25	0.25	0.25	0.75
-1	1	1	1	1	1	0.25	0.25	0.25	0.75
1	-1	-1	0.5	1	1	0.25	0.25	0.25	0.75
1	-1	1	1	1	1	0.25	0.25	0.25	0.75
1	1	-1	1	1	1	0.25	0.25	0.25	0.75
1	1	1	1.5	1	1	0.25	0.25	0.25	0.75

Epoch # 3

x1	x2	x3	s	y	t	w1	w2	w3	b
			-	-	-	0.25	0.25	0.25	0.75
-1	-1	-1	0	0	-1	0.5	0.5	0.5	0.5
-1	-1	1	0	0	1	0.25	0.25	0.75	0.75
-1	1	-1	0	0	1	0	0.5	0.5	1
-1	1	1	2	1	1	0	0.5	0.5	1
1	-1	-1	0	0	1	0.25	0.25	0.25	1.25
1	-1	1	1.5	1	1	0.25	0.25	0.25	1.25
1	1	-1	1.5	1	1	0.25	0.25	0.25	1.25
1	1	1	2	1	1	0.25	0.25	0.25	1.25

Epoch # 4

x1	x2	x3	s	y	t	w1	w2	w3	b
			-	-	-	0.25	0.25	0.25	1.25
-1	-1	-1	0.5	1	-1	0.5	0.5	0.5	1
-1	-1	1	0.5	1	1	0.5	0.5	0.5	1
-1	1	-1	0.5	1	1	0.5	0.5	0.5	1
-1	1	1	1.5	1	1	0.5	0.5	0.5	1
1	-1	-1	0.5	1	1	0.5	0.5	0.5	1
1	-1	1	1.5	1	1	0.5	0.5	0.5	1
1	1	-1	1.5	1	1	0.5	0.5	0.5	1
1	1	1	2.5	1	1	0.5	0.5	0.5	1

Epoch # 5

x1	x2	x3	s	y	t	w1	w2	w3	b
			-	-	-	0.5	0.5	0.5	1
-1	-1	-1	-0.5	-1	-1	0.5	0.5	0.5	1
-1	-1	1	0.5	1	1	0.5	0.5	0.5	1
-1	1	-1	0.5	1	1	0.5	0.5	0.5	1
-1	1	1	1.5	1	1	0.5	0.5	0.5	1
1	-1	-1	0.5	1	1	0.5	0.5	0.5	1
1	-1	1	1.5	1	1	0.5	0.5	0.5	1
1	1	-1	1.5	1	1	0.5	0.5	0.5	1
1	1	1	2.5	1	1	0.5	0.5	0.5	1