MSCIT 5210: Knowledge Discovery and Data Mining

Acknowledgement: Slides modified by Dr. Lei Chen based on the slides provided by Jiawei Han, Micheline Kamber, and Jian Pei

Chapter 9. Classification: Advanced Methods

- Classification by Backpropagation
- Support Vector Machines
- Additional Topics Regarding Classification
- Summary

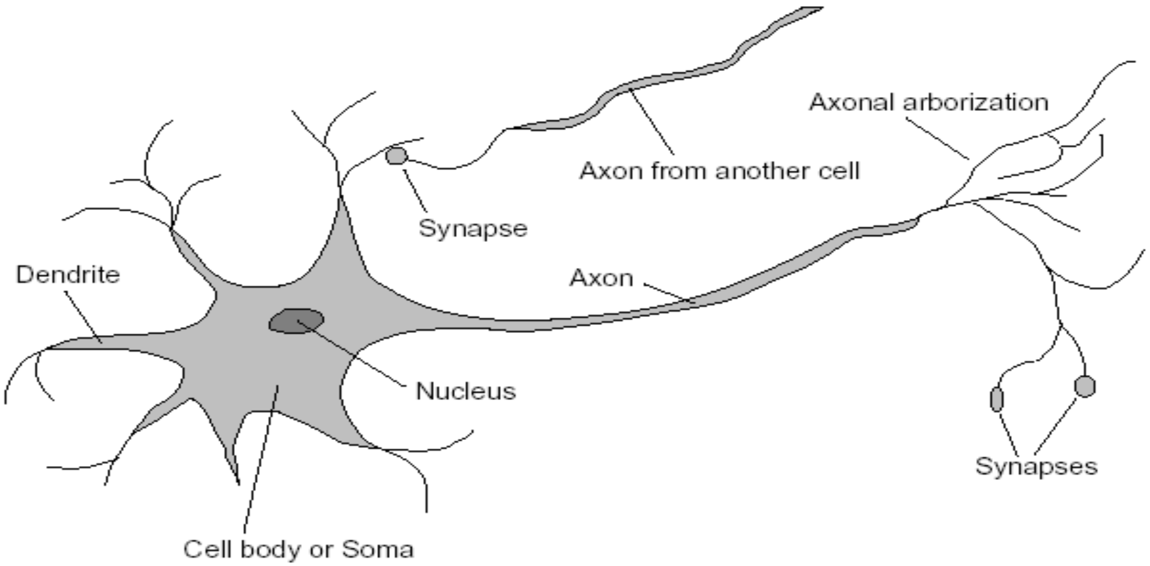
Biological Neural Systems

- Neuron switching time : > 10^{-3} secs
- Number of neurons in the human brain: ~10^{10}
- Connections (synapses) per neuron : ~10^4–10^5
- Face recognition : 0.1 secs
- High degree of distributed and parallel computation
  - Highly fault tolerent
  - Highly efficient
  - Learning is key

Excerpt from Russell and Norvig

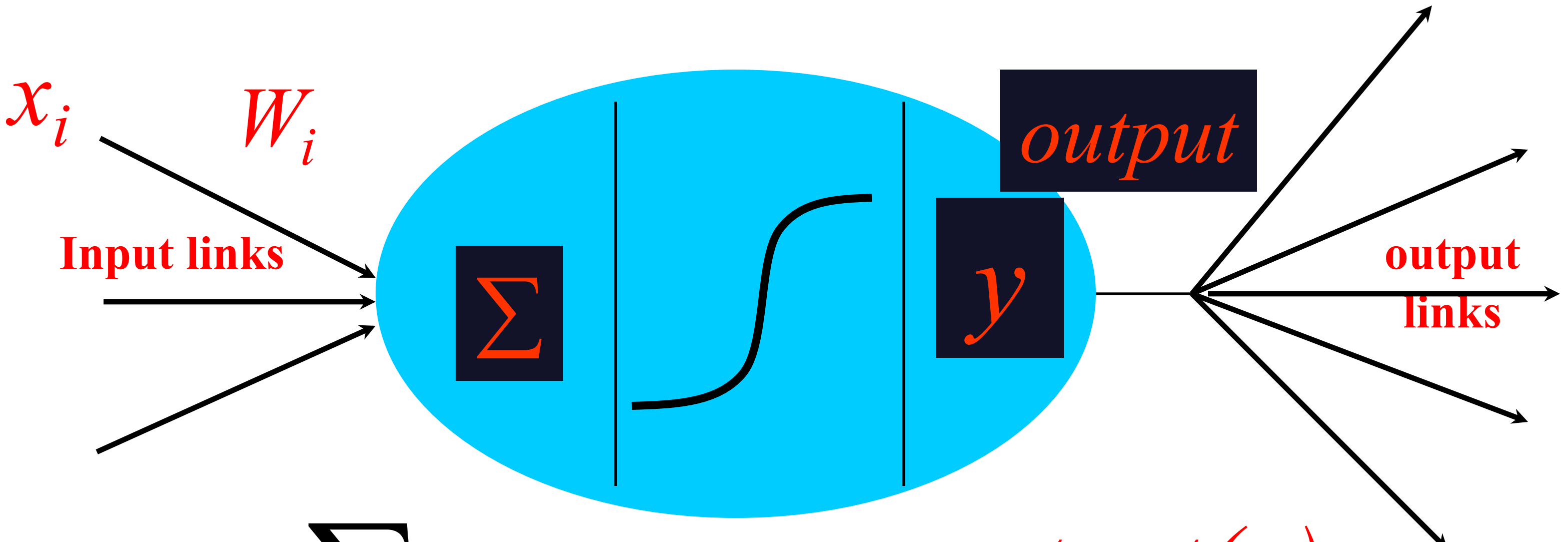
Brains

10^{11} neurons of > 20 types, 10^{14} synapses, 1ms–10ms cycle time
Signals are noisy "spike trains" of electrical potential



http://faculty.washington.edu/chudler/cells.html

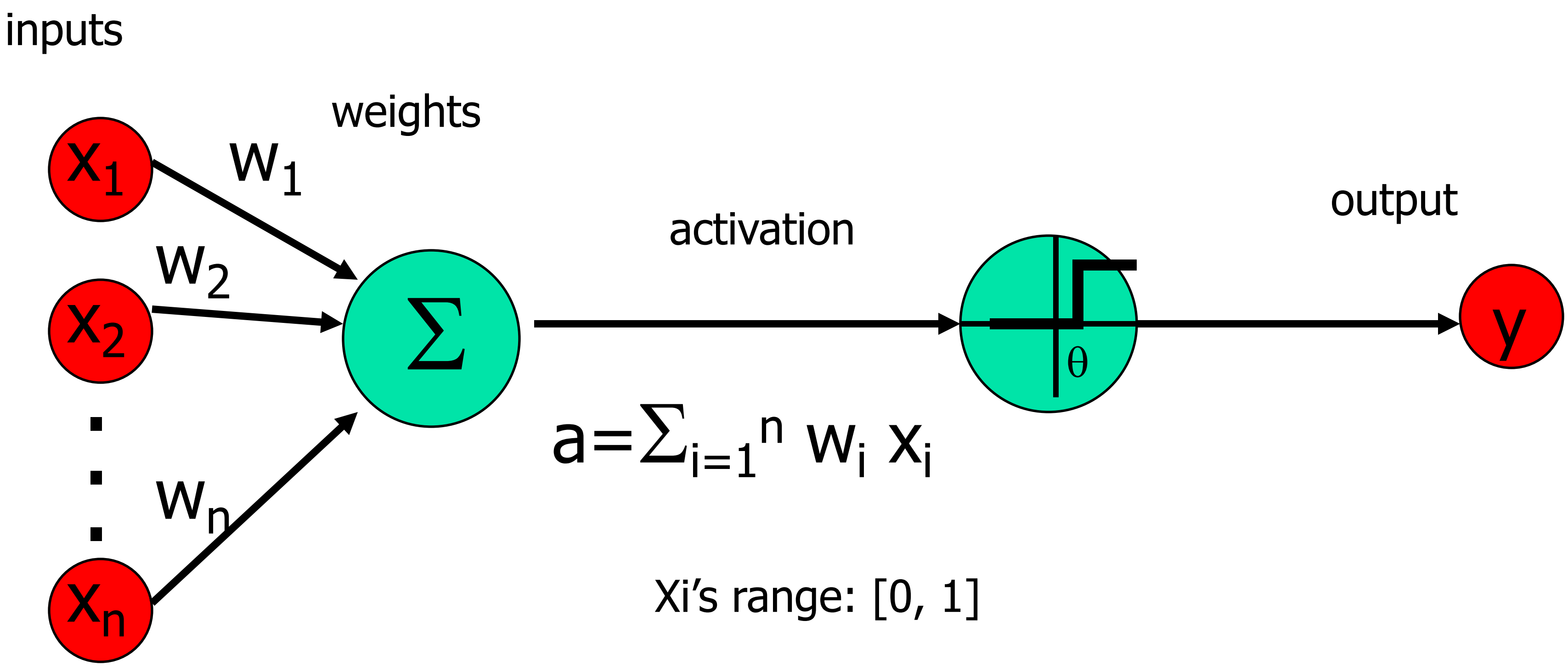
Modeling A Neuron on Computer



$$a = \sum W_i * x_i \qquad y = output(a)$$

- Computation:
  - input signals → input function(linear) → activation function(nonlinear) → output signal

Part 1. Perceptrons: Simple NN

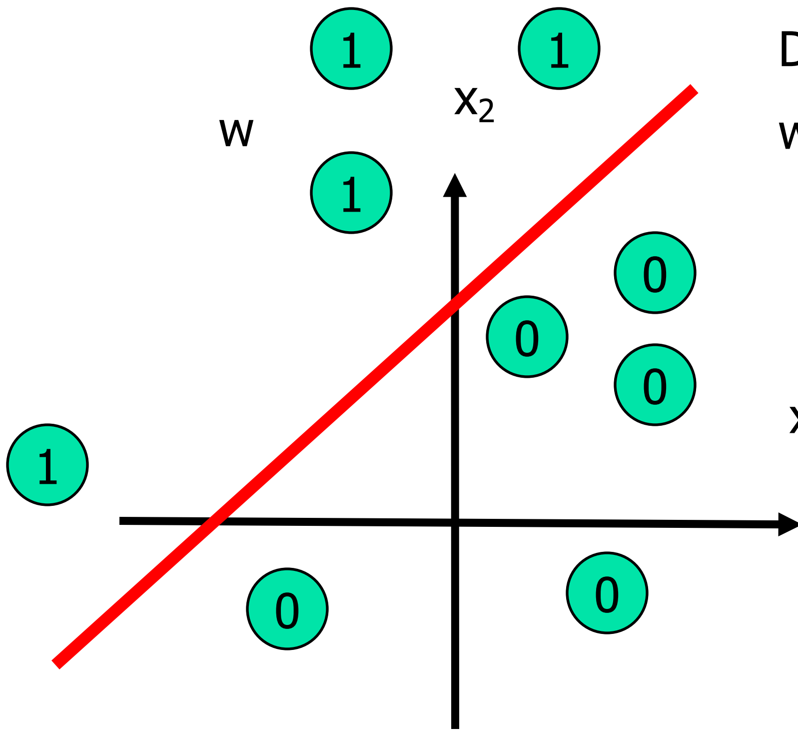


$$a = \sum_{i=1}^{n} w_i \, x_i$$

Xi's range: [0, 1]

$$y = \begin{cases} 1 \text{ if } a \geq \theta \\ 0 \text{ if } a < \theta \end{cases}$$

To be learned: W_i and θ



Decision line

$w_1 x_1 + w_2 x_2 = \theta$

Converting θ To W_0

$$\sum_{i=1}^{N} W_i * X_i \geq \theta \qquad (1)$$

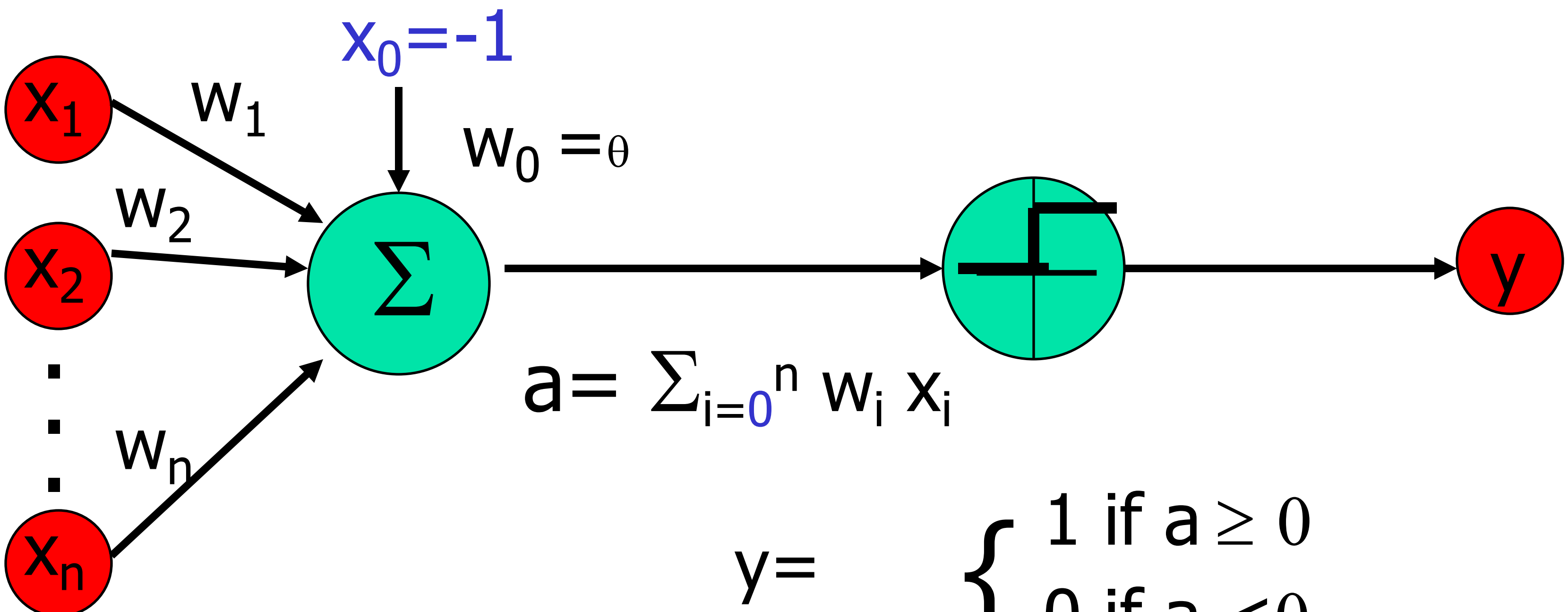
$$\Leftrightarrow \sum_{i=1}^{N} W_i * X_i - \theta \geq 0 \qquad (2)$$

$$\Leftrightarrow \sum_{i=1}^{N} W_i * X_i + (\theta) * (-1) \geq 0 \qquad (3)$$

$$\Leftrightarrow \sum_{i=1}^{N} W_i * X_i + W_0 * X_0 \geq 0 \qquad (4)$$

$$\Leftrightarrow \sum_{i=0}^{N} W_i * X_i \geq 0 \qquad (5)$$

Threshold as Weight: W_0

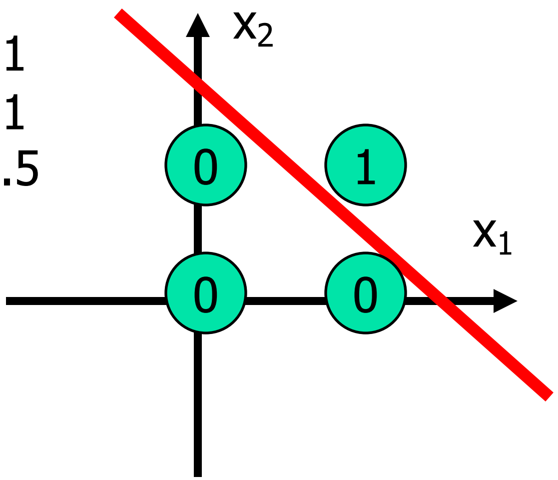


$$a = \sum_{i=0}^{n} w_i x_i$$

$$y = \begin{cases} 1 \text{ if } a \geq 0 \\ 0 \text{ if } a < 0 \end{cases}$$

Linear Separability

$w_1=1$
$w_2=1$
$\theta=1.5$



Logical AND

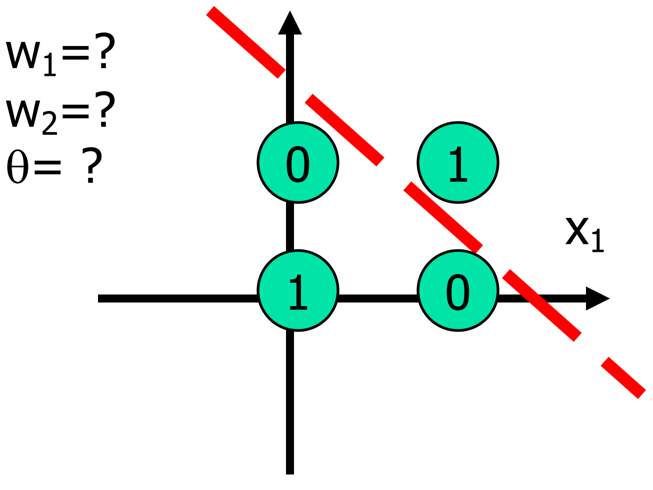
$$a = \sum_{i=0}^{n} w_i x_i$$

$$y = \begin{cases} 1 \text{ if } a \geq 0 \\ 0 \text{ if } a < 0 \end{cases}$$

x_1	x_2	a	y
0	0	-1.5	0
0	1	-0.5	0
1	0	-0.5	0
1	1	0.5	1

t
0
0
0
1

XOR cannot be separated!



Logical XOR

x_1	x_2	t	
0	0	0	
0	1	1	
1	0	1	
1	1	0	

Thus, one level neural network can only learn linear functions (straight lines)

Training the Perceptron

- Training set S of examples {**x**,**t**}
  - **x** is an input vector and
  - **T** the desired target vector (Teacher)
  - Example: Logical And
- Iterative process
  - Present a training example x , compute network output y , compare output y with target t, adjust weights and thresholds
- Learning rule
  - Specifies how to change the weights W of the network as a function of the inputs x, output Y and target t.

x_1	x_2	t
0	0	0
0	1	0
1	0	0
1	1	1

Perceptron Learning Rule

$$w_i := w_i + \Delta w_i = w_i + \alpha\,(t-y)\,x_i \quad (i=1..n)$$

- The parameter α is called the *learning rate*.
  - In Han's book it is lower case L
  - It determines the magnitude of weight updates Δw_i .
- If the output is correct ($t=y$) the weights are not changed ($\Delta w_i = 0$).
- If the output is incorrect ($t \neq y$) the weights w_i are changed such that the output of the Perceptron for the new weights w'_i is *closer/further* to the input x_i.

Perceptron Training Algorithm

Repeat

for each training vector pair ($\mathbf{x}$,t)

evaluate the output y when $\mathbf{x}$ is the input

if $y \neq t$ then

form a new weight vector $\mathbf{w}'$ according to $\mathbf{w}' = \mathbf{w} + \alpha\,(t-y)\,\mathbf{x}$

else

α: set by the user; typically = 0.01

do nothing

end if

end for

Until fixed number of iterations; or error less than a predefined value

Example: Learning the AND Function : Step 1.

x_1	x_2	t
0	0	0
0	1	0
1	0	0
1	1	1

W_0	W_1	W_2
0.5	0.5	0.5

a=(-1)*0.5+0*0.5+0*0.5=-0.5,
Thus, y=0. Correct. No need to change W

$\alpha := 0.1$

Example: Learning the AND Function : Step 2.

x_1	x_2	t
0	0	0
0	1	0
1	0	0
1	1	1

W_0	W_1	W_2
0.5	0.5	0.5

a=(-1)*0.5+0*0.5 + 1*0.5=0,
Thus, y=1. t=0, Wrong.
ΔW_0= 0.1*(0-1)*(-1)=0.1,
ΔW_1= 0.1*(0-1)*(0)=0
ΔW_2= 0.1*(0-1)*(1)=-0.1

α: = 0.1

W_0=0.5+0.1=0.6
W_1=0.5+0=0.5
W_2=0.5-0.1=0.4

Example: Learning the AND Function : Step 3.

x_1	x_2	t
0	0	0
0	1	0
1	0	0
1	1	1

W_0	W_1	W_2
0.6	0.5	0.4

a=(-1)*0.6+1*0.5 + 0*0.4=-0.1,
Thus, y=0. t=0, Correct!

α: = 0.1

Example: Learning the AND Function : Step 2.

x_1	x_2	t
0	0	0
0	1	0
1	0	0
1	1	1

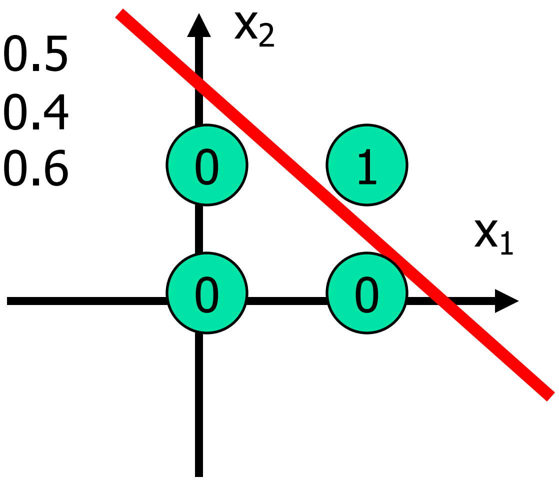
W_0	W_1	W_2
0.6	0.5	0.4

a=(-1)*0.6+1*0.5 + 1*0.4=0.3,
Thus, y=1. t=1, Correct

$\alpha := 0.1$

Final Solution:

$w_1=0.5$
$w_2=0.4$
$w_0=0.6$



Logical AND

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

$$a = 0.5x_1 + 0.4 * x_2 - 0.6$$

$$y = \begin{cases} 1 \text{ if } a \geq 0 \\ 0 \text{ if } a < 0 \end{cases}$$

Perceptron Convergence Theorem

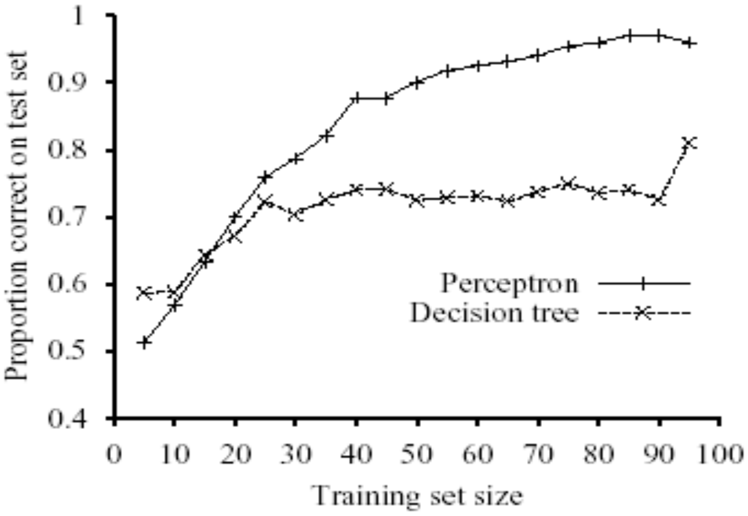
- The algorithm converges to the correct classification
  - if the training data is linearly separable
  - and learning rate is sufficiently small

  (Rosenblatt 1962).
- The final weights in the solution **w** is not unique: there are many possible lines to separate the two classes.

Experiments

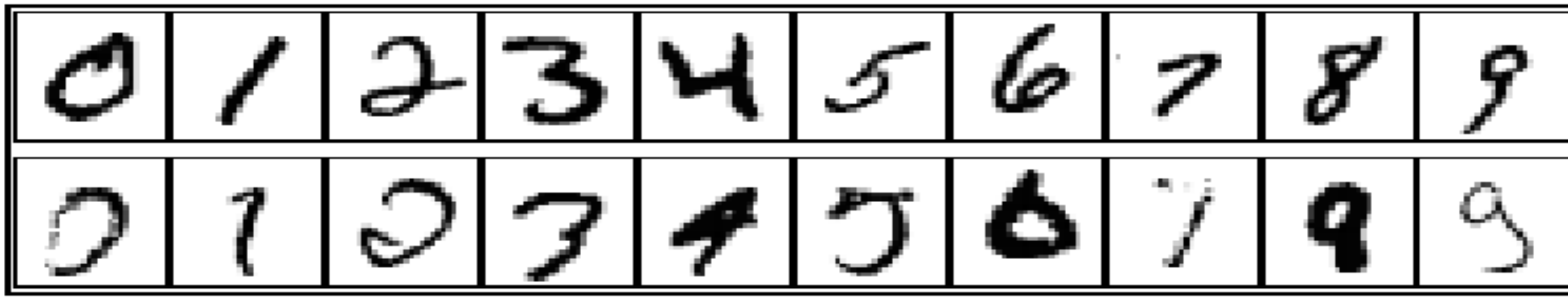
Perceptron learning contd.

Perceptron learning rule converges to a consistent function
for any linearly separable data set



Handwritten Recognition Example

Handwritten digit recognition

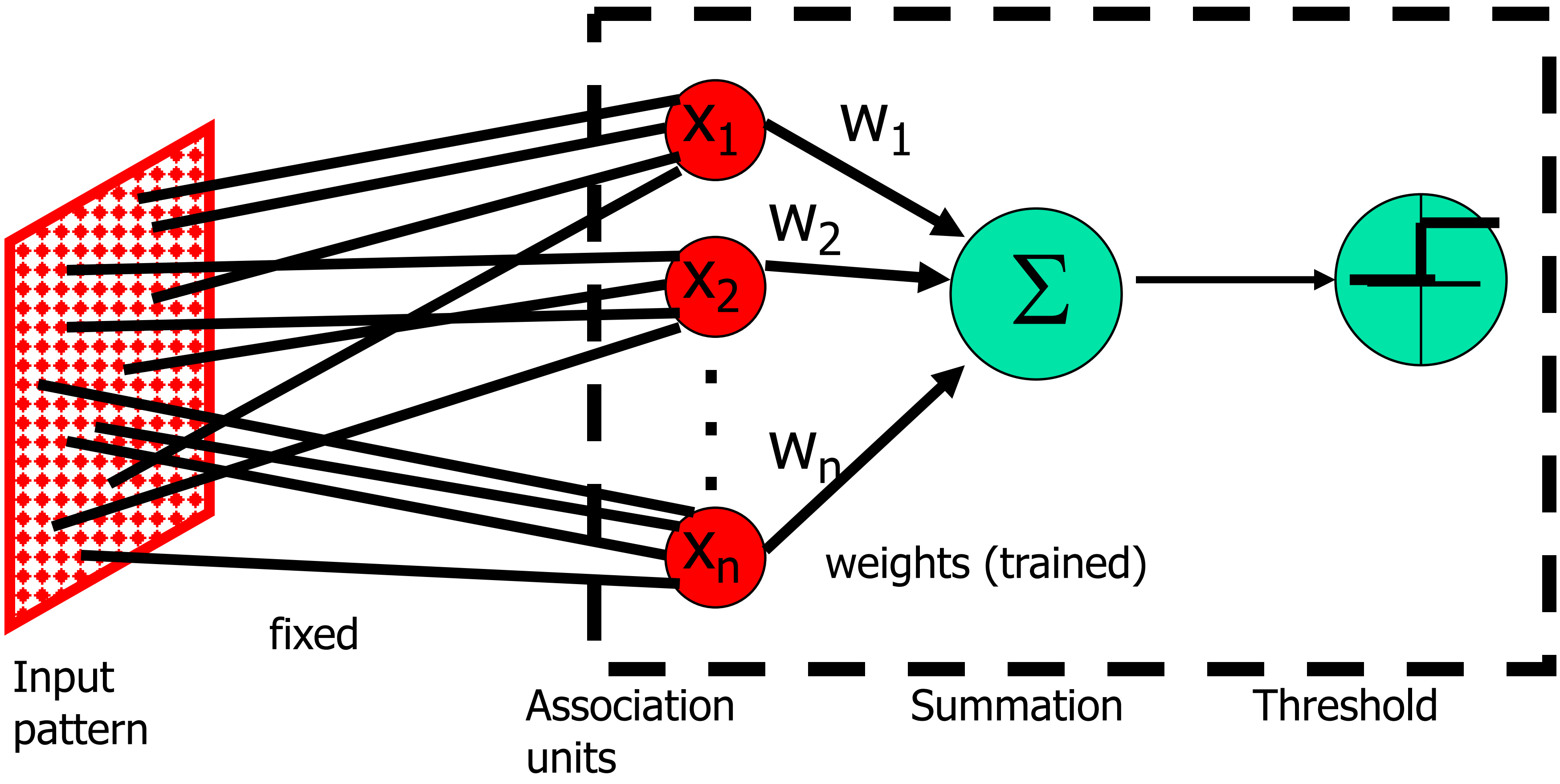


3-nearest-neighbor = 2.4% error

400–300–10 unit MLP = 1.6% error

LeNet: 768–192–30–10 unit MLP = 0.9%

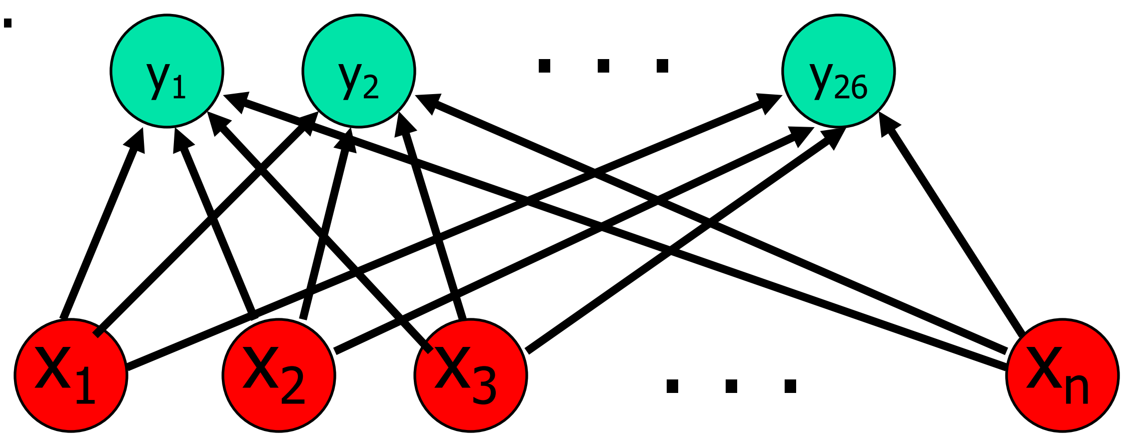
Each letter → one output unit y



Multiple Output Perceptrons

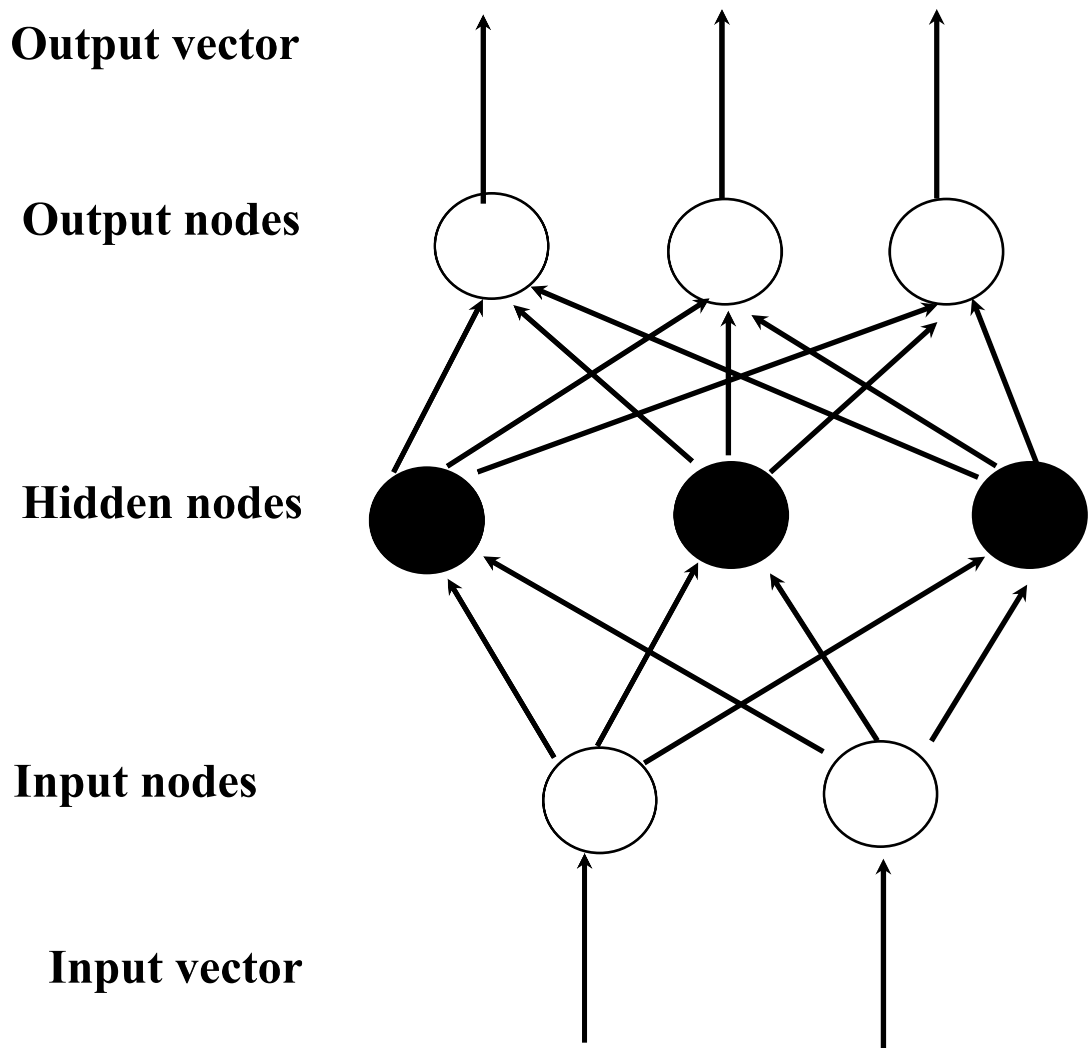
- Handwritten alphabetic character recognition
  - 26 classes : A,B,C…,Z
  - First output unit distinguishes between "A"s and "non-A"s, second output unit between "B"s and "non-B"s etc.

w_{ji} connects x_i with y_j

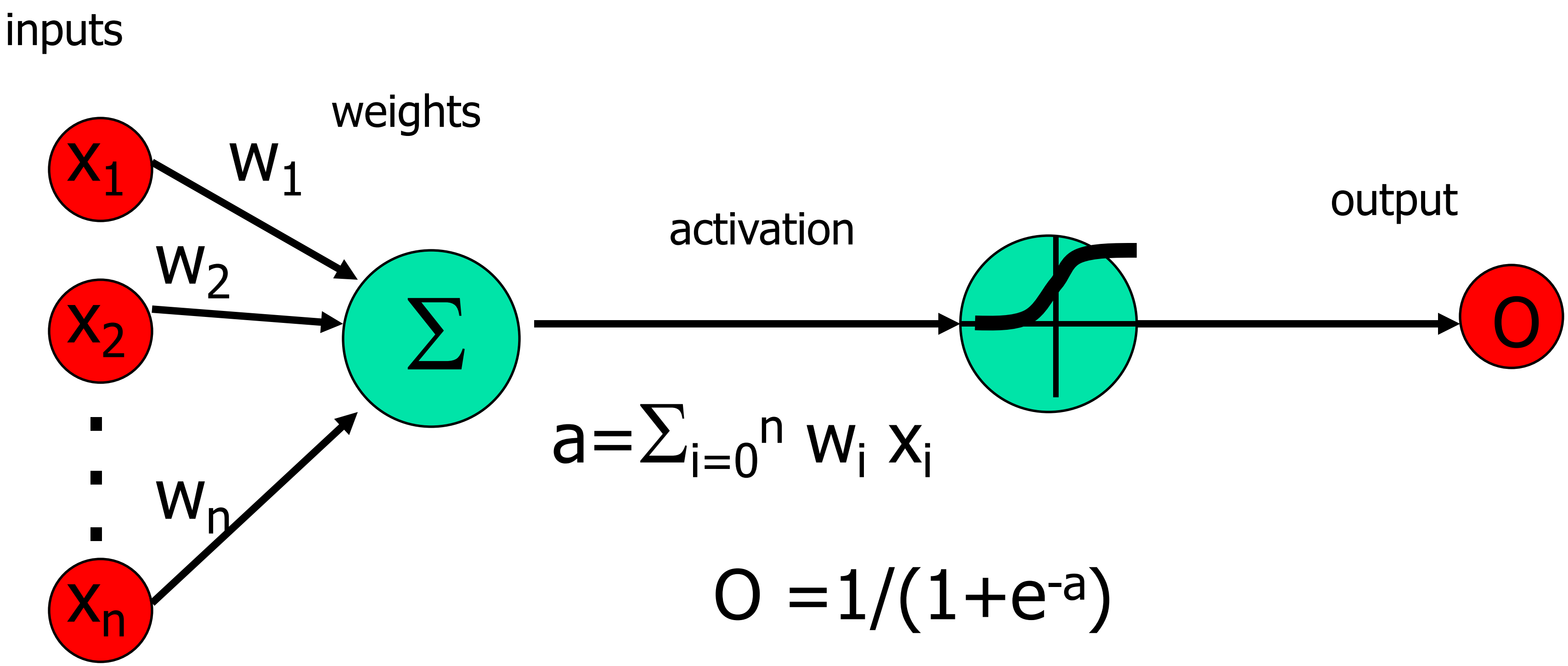


$$w'_{ji} = w_{ji} + \alpha\ (t_j - y_j)\ x_i$$

Part 2. Multi Layer Networks



Sigmoid-Function for Continuous Output



$$a=\sum_{i=0}^{n} w_i x_i$$

$$O = 1/(1+e^{-a})$$

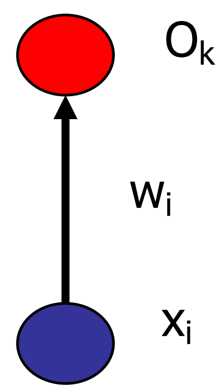
Output between 0 and 1 (when a = negative infinity, O = 0; when a= positive infinity, O=1.

Gradient Descent Learning Rule

- For each training example X,
  - Let O be the output (bewteen 0 and 1)
  - Let T be the correct target value
- Continuous output O
  - $a = w_1 x_1 + \ldots + w_n x_n + \theta$
  - $O = 1/(1+e^{-a})$
- Train the w_i's such that they minimize the squared error
  - $E[w_1,\ldots,w_n] = ½ \sum_{k \in D} (T_k - O_k)^2$

  where D is the set of training examples

Explanation: Gradient Descent Learning Rule



$$\Delta w_i = \alpha \;\; \underline{O_k (1-O_k)} \;\; (T_k - O_k) \;\; x_i^k$$

learning rate

derivative of
activation function

error δ_k of
post-synaptic neuron

activation of
pre-synaptic neuron

Backpropagation Algorithm (Han, Figure 9.5)

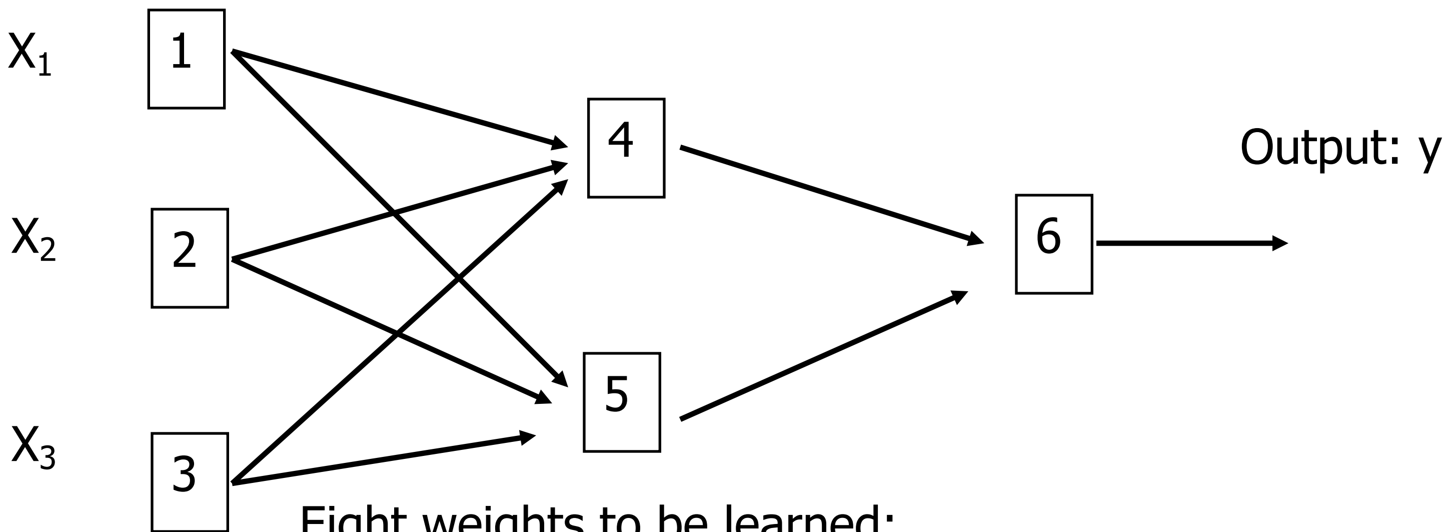
- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - For each training example <$(x_1,\ldots x_n),t$> Do
    - Input the instance $(x_1,\ldots,x_n)$ to the network and compute the network outputs O_k
    - For each output unit k
      - $Err_k=O_k(1-O_k)(t_k-O_k)$
    - For each hidden unit h
      - $Err_h=O_h(1-O_h)\sum_k w_{h,k} Err_k$
    - For each network weight $w_{i,j}$ Do
    - $w_{i,j}=w_{i,j}+\Delta w_{i,j}$ where

      $\Delta w_{i,j}= \alpha\, Err_{j*}\, O_{i,}$
    - $\theta_j=\theta_j+\Delta\theta_j$ where

      $\Delta\theta_j = \alpha\, Err_{j,}$

α: is learning rate, set by the user;

Example 6.9 (HK book, page 333)



Eight weights to be learned:

W_{ij}: W_{14}, W_{15}, … W_{46}, W_{56}, …, and θ

Training example:

x1	x2	x3	t
1	0	1	1

Learning rate: =0.9

Initial Weights: randomly assigned (HK: Tables 7.3, 7.4)

x1	x2	x3	w14	w15	w24	w25	w34	w35	w46	w56	θ4	θ5	θ6
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1

Net input at unit 4:

$$X_1 * W_{14} + X_2 * W_{24} + X_3 * W_{34} + \theta_4 =$$

$$1*(0.2)+0*0.4+1*(-0.5)+(-0.4) = -0.7$$

Output at unit 4:

$$\frac{1}{1+e^{0.7}} = 0.332$$

Feed Forward: (Table 7.4)

- Continuing for units 5, 6 we get:
  - Output at unit 6 = 0.474

Calculating the error (Tables 7.5)

- Error at Unit 6: (t-y)=(1-0.474)

- Error to be backpropagated from unit 6:

$$y(1-y)(t-y) = (0.474)(1-0.474)(1-0.474) = 0.1311$$

- Weight update :

$$\Delta w_{46} = (l)Err_6 y_4 = 0.9*(0.1311)(0.332)$$

$$w_{46} = w_{46} + (l)Err_6 y_4 = -0.3 + 0.9*(0.1311)(0.332)$$

$$= -0.261$$

Weight update (Table 7.6)

Thus, new weights after training with {(1, 0, 1), t=1}:

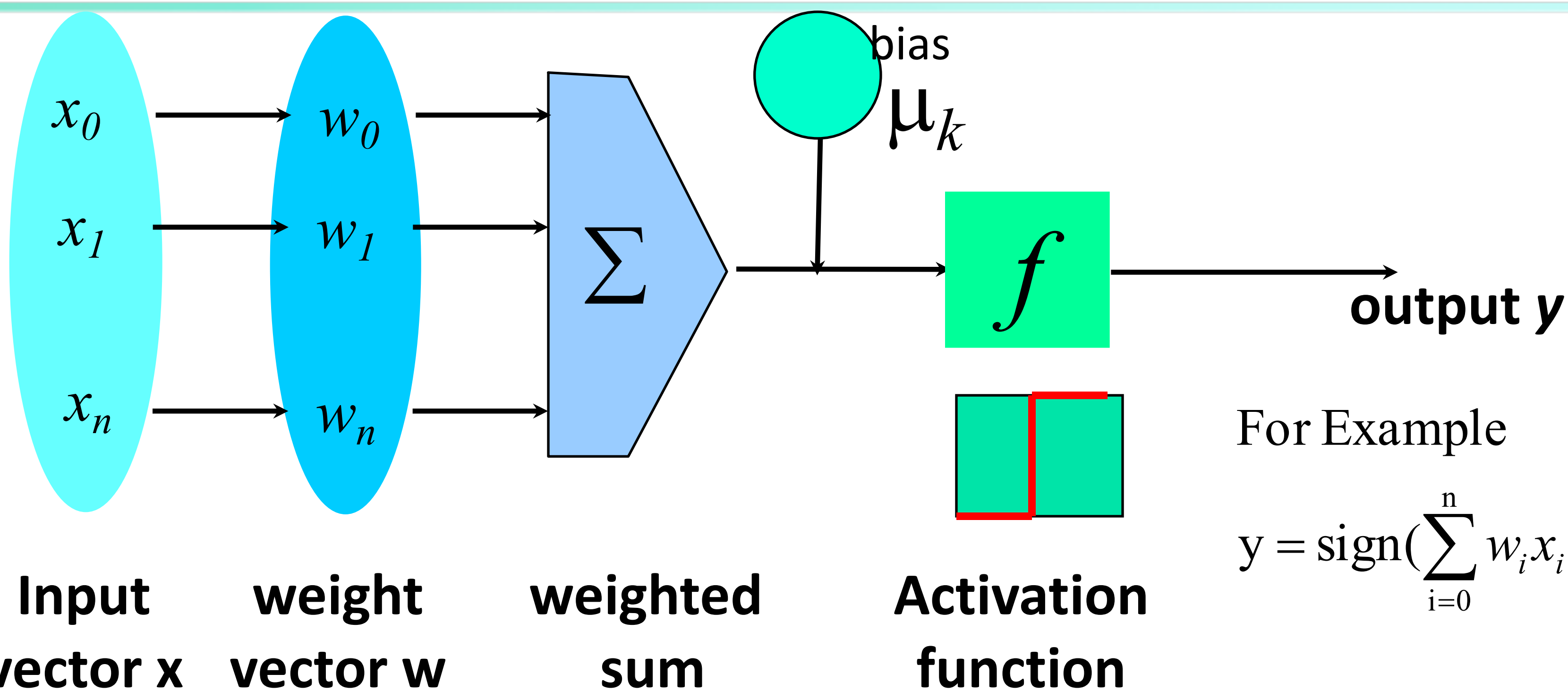
w14	w15	w24	w25	w34	w35	w46	w56	$\theta 4$	$\theta 5$	$\theta 6$
0.192	-0.306	0.4	0.1	-0.506	0.194	-0.261	-0.138	-0.408	0.194	0.218

- If there are more training examples, the same procedure is followed as above.
- Repeat the rest of the procedures.

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

Neuron: A Hidden/Output Layer Unit



For Example

$$y = \text{sign}\left(\sum_{i=0}^{n} w_i x_i - \mu_k\right)$$

- An *n*-dimensional input vector **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.

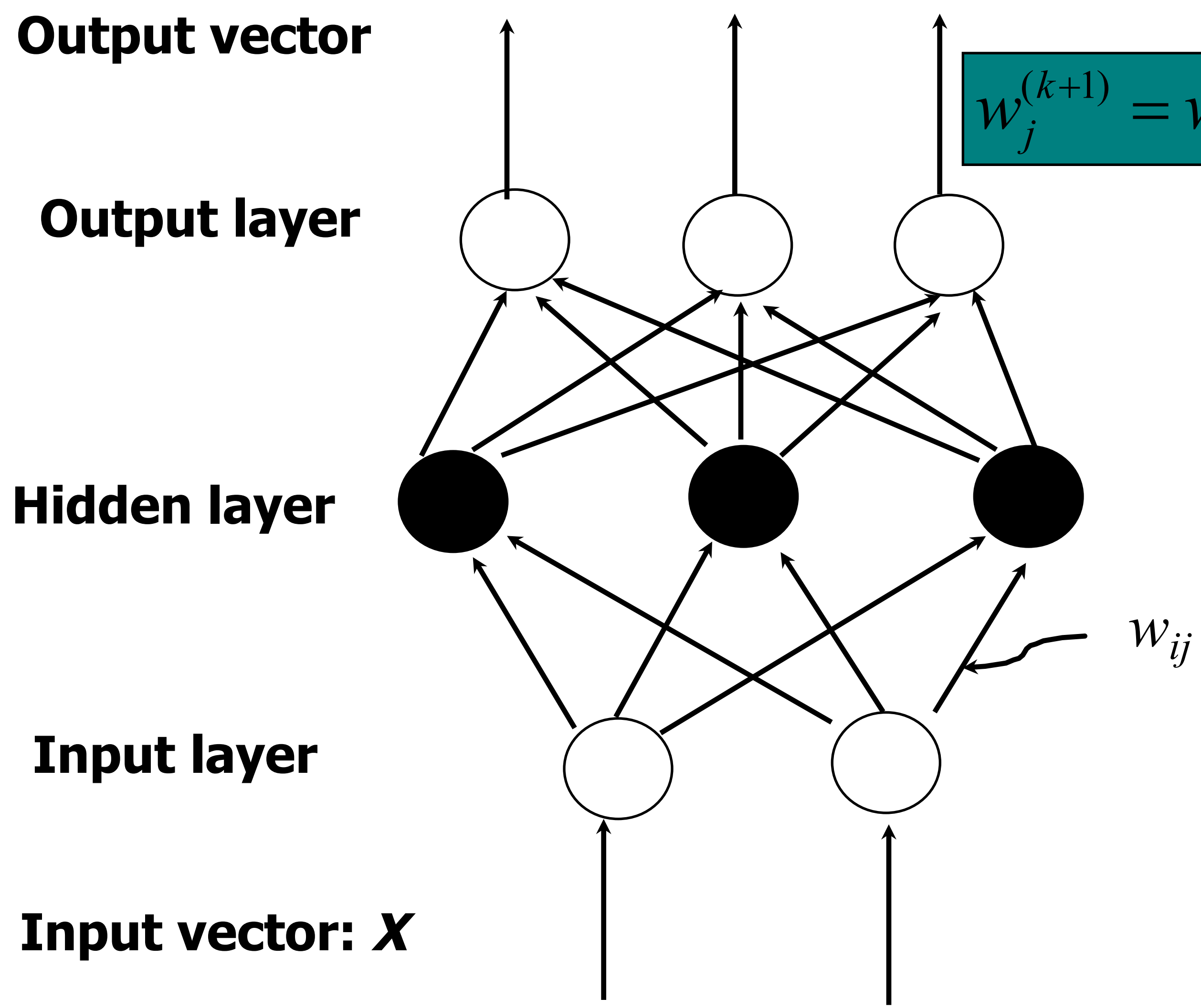
How A Multi-Layer Neural Network Works

- 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*

A Multi-Layer Feed-Forward Neural Network



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

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

Neural Network as a Classifier

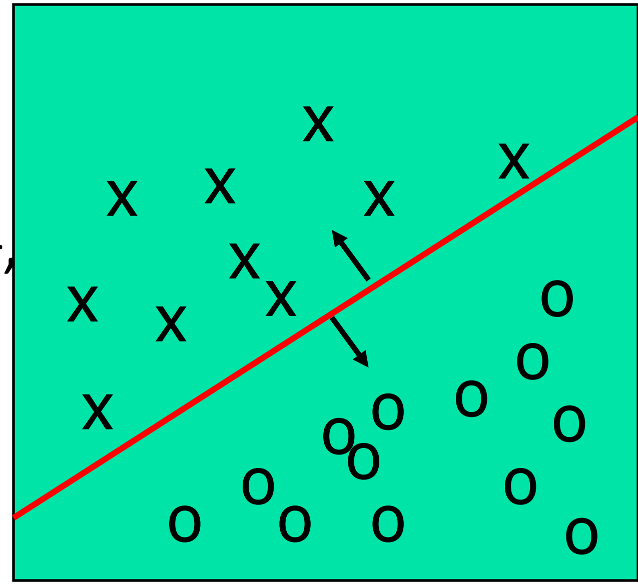
- Weakness
  - Long training time
  - Require a number of parameters typically best determined empirically, e.g., the network topology or "structure."
  - Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of "hidden units" in the network
- Strength
  - 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

Chapter 9. Classification: Advanced Methods

- Classification by Backpropagation
- Support Vector Machines
- Additional Topics Regarding Classification
- Summary

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

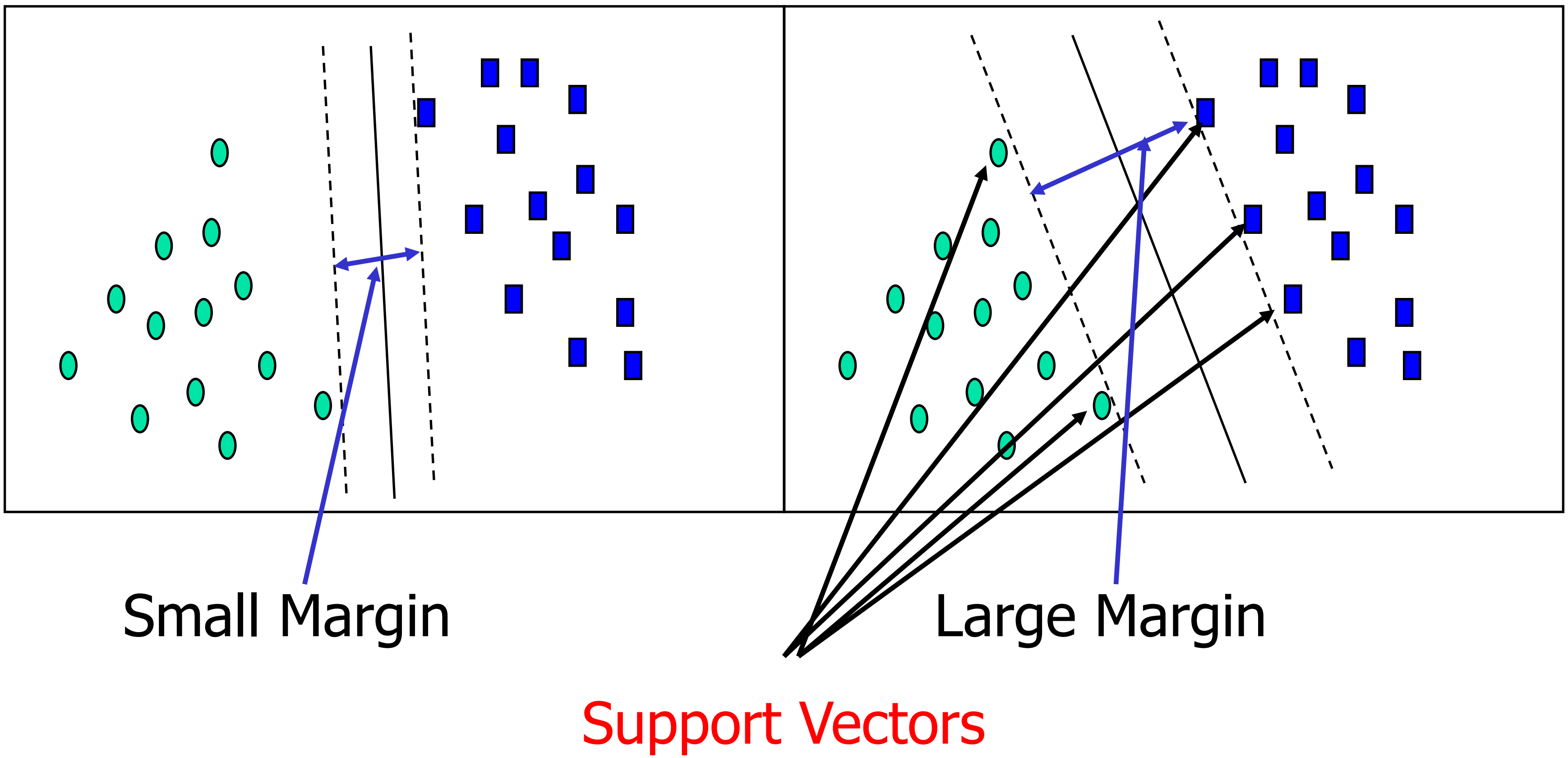
SVM—Support Vector Machines

- 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)

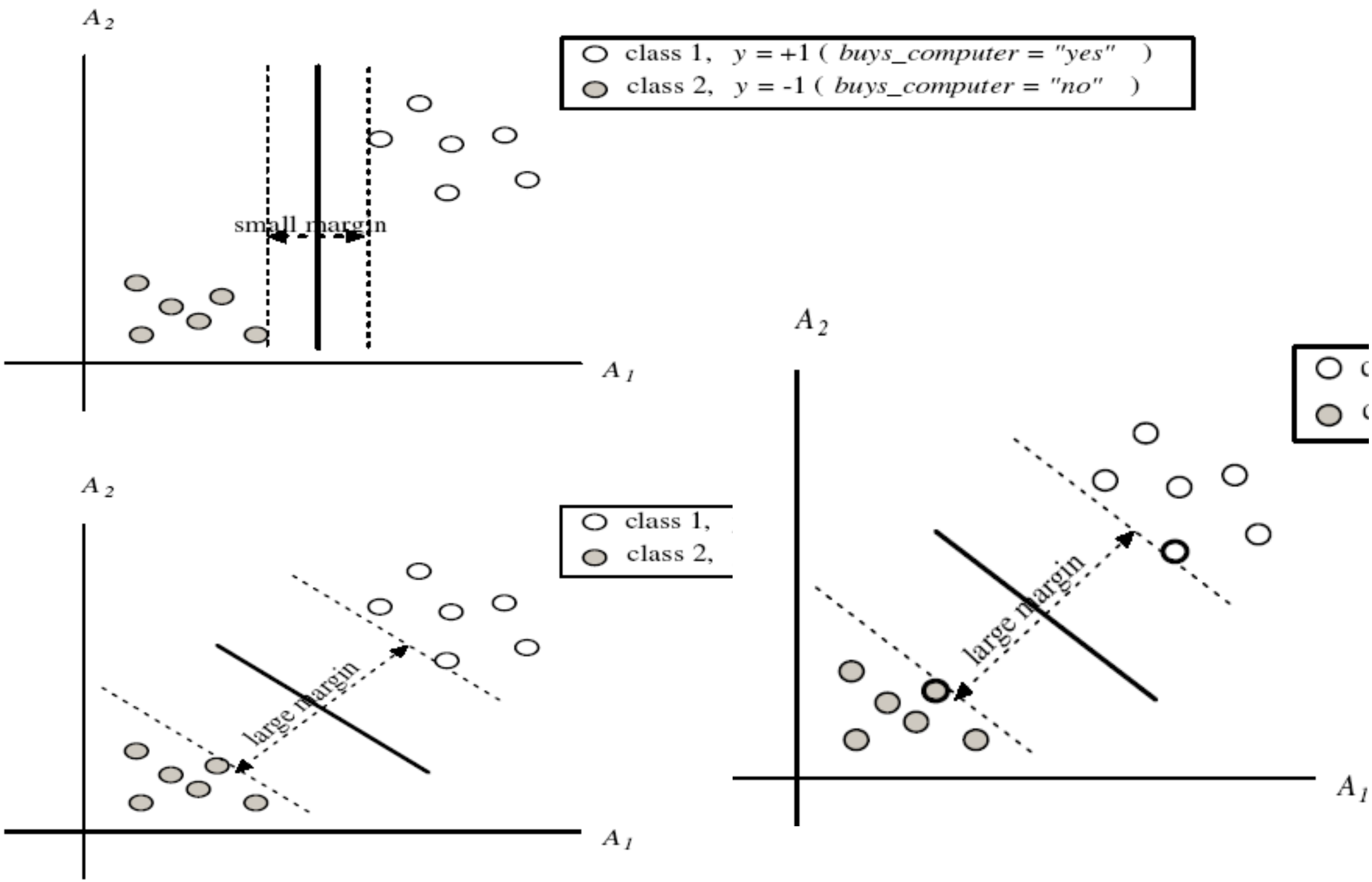
SVM—History and Applications

- 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

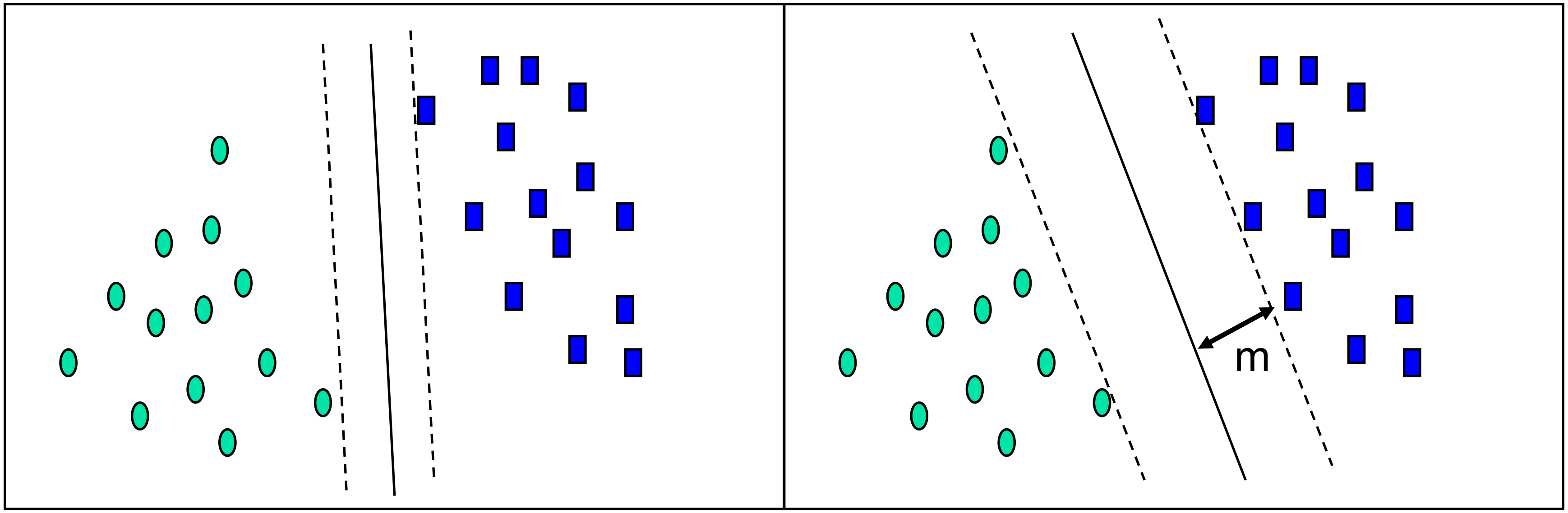
SVM—General Philosophy



SVM—Margins and Support Vectors



SVM—When Data Is Linearly Separable



Let data D be ($\mathbf{X}_1$, y_1), …, ($\mathbf{X}_{|D|}$, $y_{|D|}$), 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 **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:

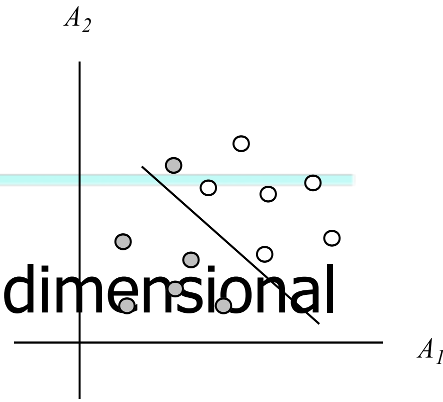
  H_1: $w_0 + w_1\, x_1 + w_2\, x_2 \geq 1$ for $y_i = +1$, and

  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,

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

- Search for a linear separating hyperplane in the new space

SVM: Different Kernel functions

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

- Consider the following data points. Please use SVM to train a classifier, and then classify these data points. Points with a_i=1 means this point is **support vector**. For example, point 1 (1,2) is the support vector, but point 5 (5,9) is not the support vector.
- Training data:

ID	ai	x1	x2	y
1	1	1	2	1
2	1	2	1	−1
3	1	0	1	1
4	0	1	−2	−1
5	0	5	9	1
6	0	6	2	−1
7	0	3	9	1
8	0	7	1	−1

- Testing data:

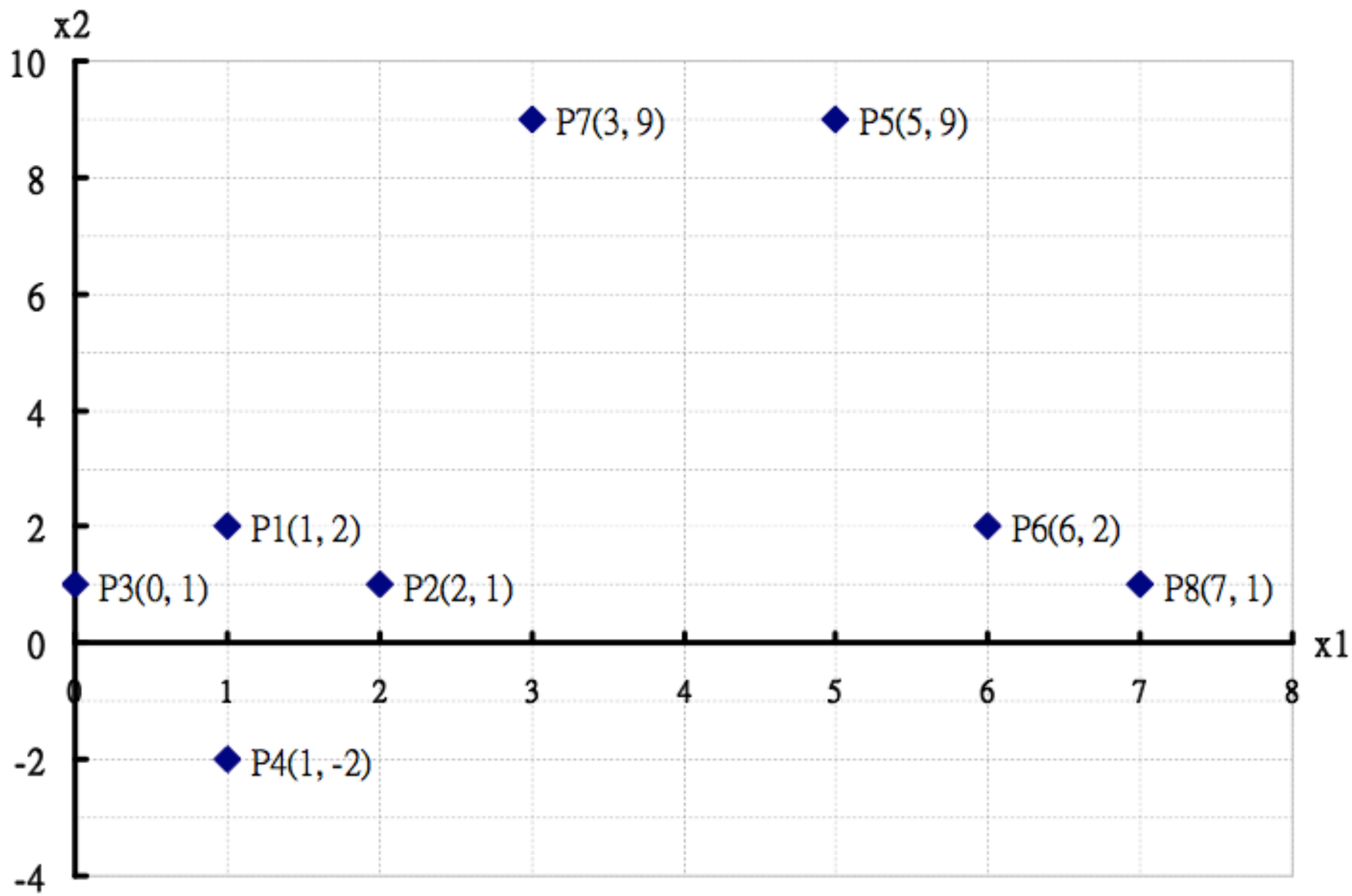
ID	x1	x2	y
9	2	5	
10	7	2	

SVM

- Question:
  - (a) Find the decision boundary, show detail calculation process.
  - (b) Use the decision boundary you found to classify the Testing data. Show all calculation process in detail, including the intermediate result and the formula you used.

SVM

- Answer:
- a) As the picture shows, P1, P2, P3 are support vectors.



SVM

- Suppose w is (w_1,w_2). Since both P1(1,2) and P3(0,1) have y = 1, while P2(2,1) has y =-1:
  - $w_1*1+w_2*2+b = 1$
  - $w_1*0+w_2*1+b = 1$
  - $w_1*2+w_2*1+b =-1$

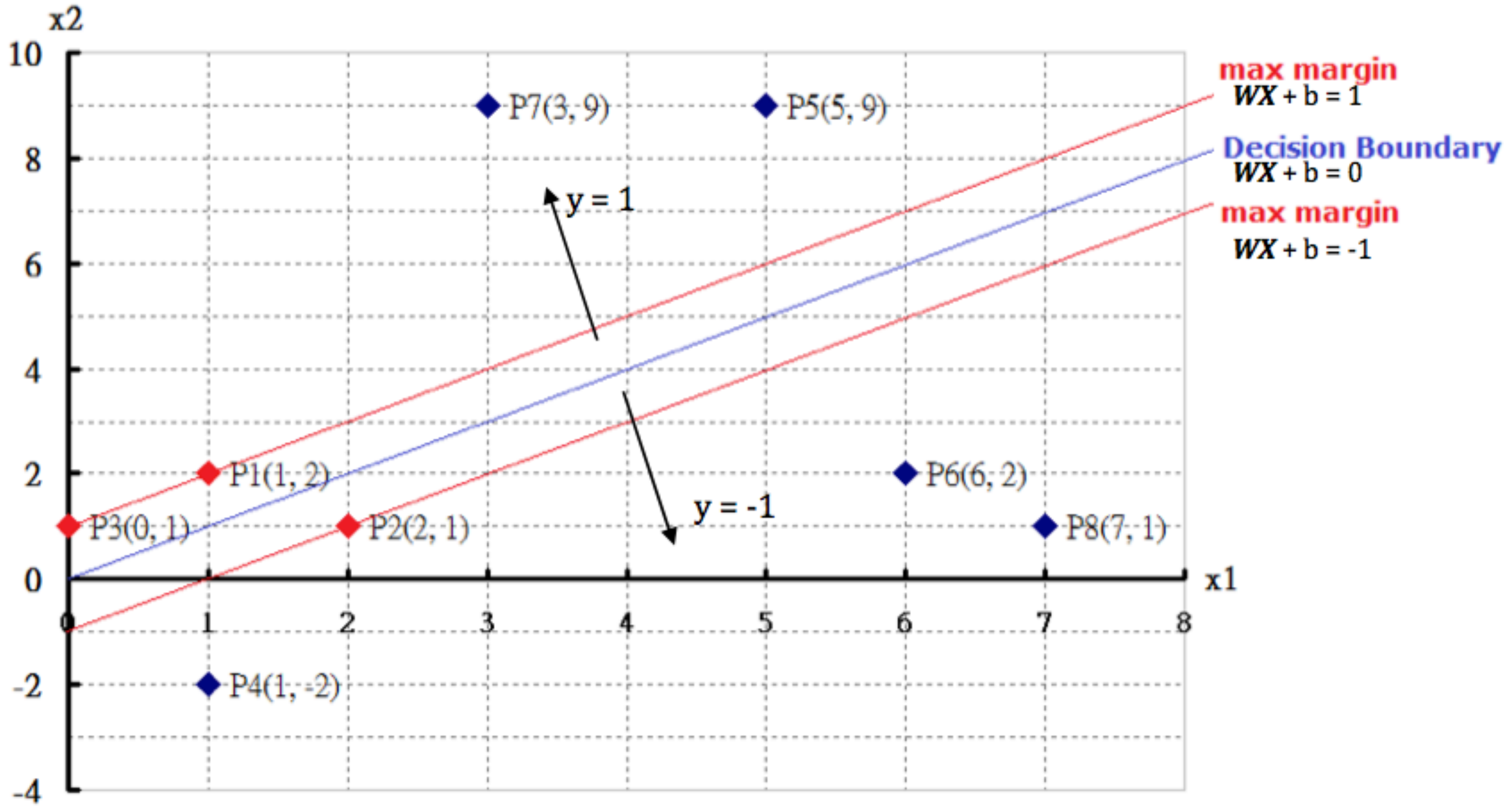
  ⇨ $w_1= -1$, $w_2 = 1$, $b = 0$

  then, the decision boundary is:
  - $w_1 * x_1+w_2 * x_2 + b =0$

  ⇨ **-x1+x2 = 0**

- Showed in the picture next page.

SVM


x2
10
8
6
4
2
0
-2
-4
0
1
2
3
4
5
6
7
8
x1
P7(3, 9)
P5(5, 9)
P1(1, 2)
P3(0, 1)
P2(2, 1)
P6(6, 2)
P8(7, 1)
P4(1, -2)
y = 1
y = -1
max margin
WX + b = 1
Decision Boundary
WX + b = 0
max margin
WX + b = -1

SVM

- b) Use the decision boundary to classify the testing data:
  - For the point P9 (2,5)

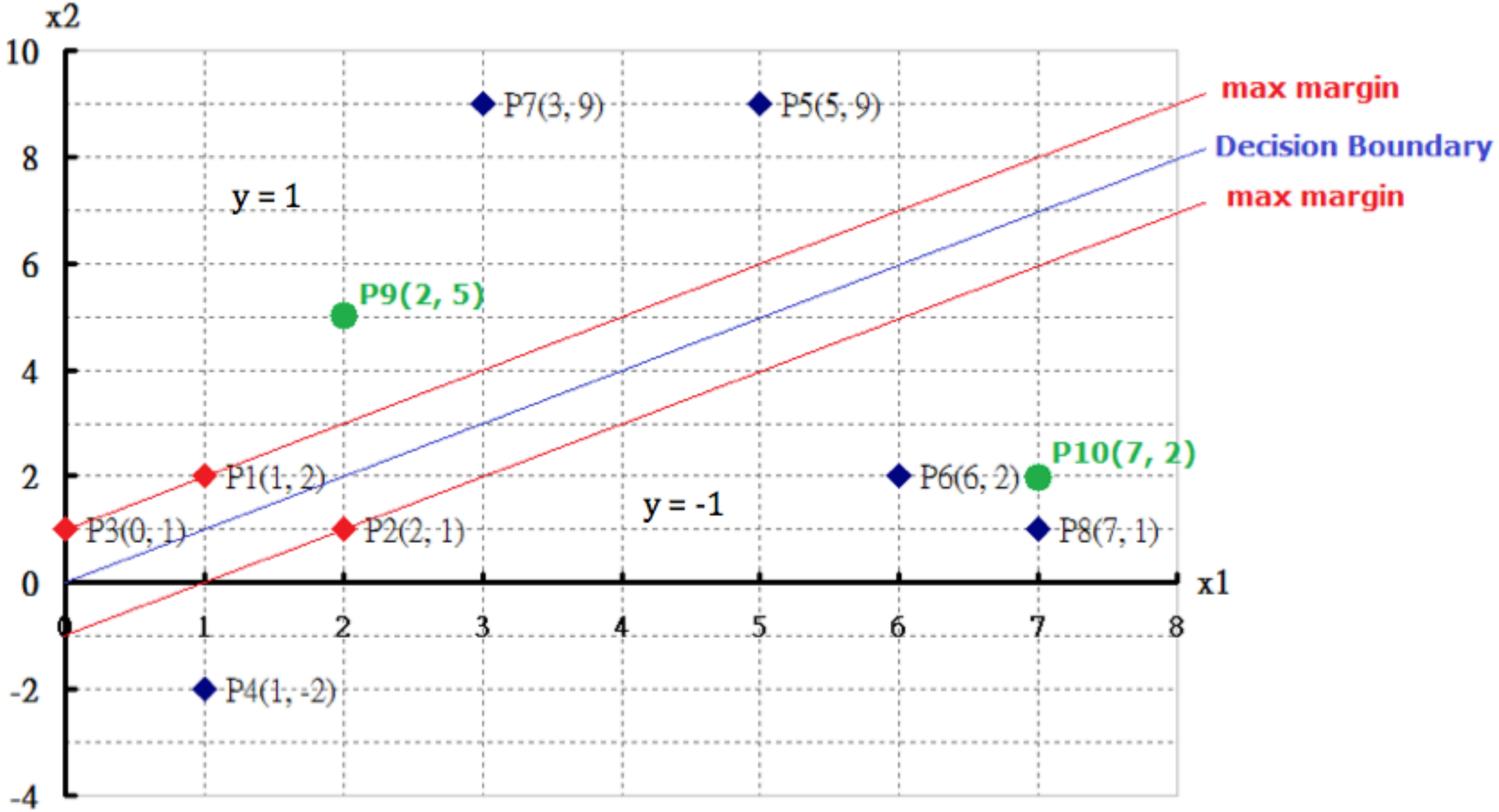
    $-x_1+x_2 = -2+5 = 3 >= 1$

    So we choose y = 1
  - For the point P10 (7,2)

    $-x_1+x_2 = -7+2 = -5 <= -1$

    So we choose y = -1
- Showed in the picture next page.

SVM


x2
10
8
6
4
2
0
-2
-4
0
1
2
3
4
5
6
7
8
x1
P7(3, 9)
P5(5, 9)
max margin
Decision Boundary
max margin
y = 1
P9(2, 5)
P10(7, 2)
P1(1, 2)
P6(6, 2)
y = -1
P3(0, 1)
P2(2, 1)
P8(7, 1)
P4(1, -2)

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)

SVM Related Links

- SVM Website: http://www.kernel-machines.org/
- Representative implementations
  - **LIBSVM**: an efficient implementation of SVM, multi-class classifications, nu-SVM, one-class SVM, including also various interfaces with java, python, etc.
  - **SVM-light**: simpler but performance is not better than LIBSVM, support only binary classification and only in C
  - **SVM-torch**: another recent implementation also written in C

Chapter 9. Classification: Advanced Methods

- Classification by Backpropagation
- Support Vector Machines
- Additional Topics Regarding Classification
- Summary

Multiclass Classification

- Classification involving more than two classes (i.e., > 2 Classes)
- Method 1. **One-vs.-all** (OVA): Learn a classifier one at a time
  - Given m classes, train m classifiers: one for each class
  - Classifier j: treat tuples in class j as *positive* & all others as *negative*
  - To classify a tuple **X**, the set of classifiers vote as an ensemble
- Method 2. **All-vs.-all** (AVA): Learn a classifier for each pair of classes
  - Given m classes, construct m(m-1)/2 binary classifiers
  - A classifier is trained using tuples of the two classes
  - To classify a tuple **X**, each classifier votes. X is assigned to the class with maximal vote
- Comparison
  - All-vs.-all tends to be superior to one-vs.-all
  - Problem: Binary classifier is sensitive to errors, and errors affect vote count

Error-Correcting Codes for Multiclass Classification

- Originally designed to correct errors during data transmission for communication tasks by exploring data redundancy
- Example
  - A 7-bit codeword associated with classes 1-4

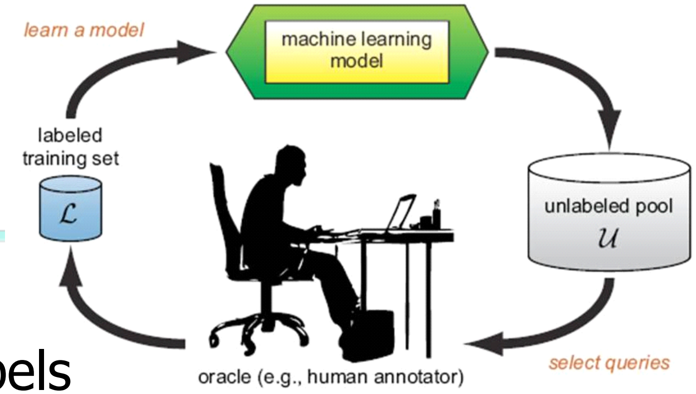
Class	Error-Corr. Codeword						
C_1	1	1	1	1	1	1	1
C_2	0	0	0	0	1	1	1
C_3	0	0	1	1	0	0	1
C_4	0	1	0	1	0	1	0

  - Given a unknown tuple **X**, the 7-trained classifiers output: 0001010
  - Hamming distance: # of different bits between two codewords
  - H(**X**, C_1) = 5, by checking # of bits between [1111111] & [0001010]
  - H(**X**, C_2) = 3, H(**X**, C_3) = 3, H(**X**, C_4) = 1, thus C_4 as the label for **X**
- Error-correcting codes can correct up to (h-1)/h 1-bit error, where h is the minimum Hamming distance between any two codewords
- If we use 1-bit per class, it is equiv. to one-vs.-all approach, the code are insufficient to self-correct
- When selecting error-correcting codes, there should be good row-wise and col.-wise separation between the codewords

Semi-Supervised Classification

- Semi-supervised: Uses labeled and unlabeled data to build a classifier
- Self-training:
  - Build a classifier using the labeled data
  - Use it to label the unlabeled data, and those with the most confident label prediction are added to the set of labeled data
  - Repeat the above process
  - Adv: easy to understand; disadv: may reinforce errors
- Co-training: Use two or more classifiers to teach each other
  - Each learner uses a mutually independent set of features of each tuple to train a good classifier, say f_1
  - Then f_1 and f_2 are used to predict the class label for unlabeled data X
  - Teach each other: The tuple having the most confident prediction from f_1 is added to the set of labeled data for f_2, & vice versa
- Other methods, e.g., joint probability distribution of features and labels

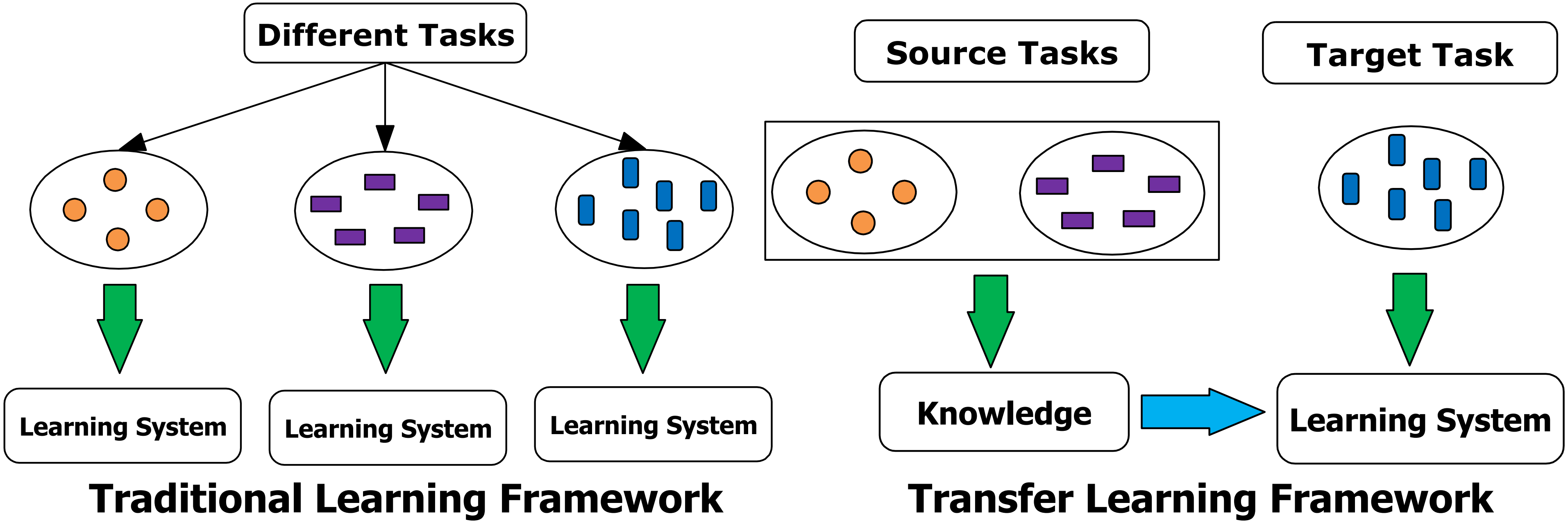
Active Learning



- Class labels are expensive to obtain
- Active learner: query human (oracle) for labels
- Pool-based approach: Uses a pool of unlabeled data
  - L: a small subset of D is labeled, U: a pool of unlabeled data in D
  - Use a query function to carefully select one or more tuples from U and request labels from an oracle (a human annotator)
  - The newly labeled samples are added to L, and learn a model
  - Goal: Achieve high accuracy using as few labeled data as possible
- Evaluated using *learning curves*: Accuracy as a function of the number of instances queried (# of tuples to be queried should be small)
- Research issue: How to choose the data tuples to be queried?
  - Uncertainty sampling: choose the least certain ones
  - Reduce *version space*, the subset of hypotheses consistent w. the training data
  - Reduce expected entropy over U: Find the greatest reduction in the total number of incorrect predictions

Transfer Learning: Conceptual Framework

- Transfer learning: Extract knowledge from one or more source tasks and apply the knowledge to a target task
- Traditional learning: Build a new classifier for each new task
- Transfer learning: Build new classifier by applying existing knowledge learned from source tasks



Transfer Learning: Methods and Applications

- Applications: Especially useful when data is outdated or distribution changes, e.g., Web document classification, e-mail spam filtering
- *Instance-based transfer learning*: Reweight some of the data from source tasks and use it to learn the target task
- TrAdaBoost (Transfer AdaBoost)
  - Assume source and target data each described by the same set of attributes (features) & class labels, but rather diff. distributions
  - Require only labeling a small amount of target data
  - Use source data in training: When a source tuple is misclassified, reduce the weight of such tupels so that they will have less effect on the subsequent classifier
- Research issues
  - Negative transfer: When it performs worse than no transfer at all
  - Heterogeneous transfer learning: Transfer knowledge from different feature space or multiple source domains
  - Large-scale transfer learning

Chapter 9. Classification: Advanced Methods

- Classification by Backpropagation
- Support Vector Machines
- Additional Topics Regarding Classification
- Summary

Summary

- Effective and advanced classification methods
  - Bayesian belief network (probabilistic networks)
  - Backpropagation (Neural networks)
  - Support Vector Machine (SVM)
  - Pattern-based classification
  - Other classification methods: lazy learners (KNN, case-based reasoning), genetic algorithms, rough set and fuzzy set approaches
- Additional Topics on Classification
  - Multiclass classification
  - Semi-supervised classification
  - Active learning
  - Transfer learning

SVM—Introductory Literature

- "Statistical Learning Theory" by Vapnik: extremely hard to understand, containing many errors too.
- C. J. C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Knowledge Discovery and Data Mining*, 2(2), 1998.
  - Better than the Vapnik's book, but still written too hard for introduction, and the examples are so not-intuitive
- The book "An Introduction to Support Vector Machines" by N. Cristianini and J. Shawe-Taylor
  - Also written hard for introduction, but the explanation about the mercer's theorem is better than above literatures
- The neural network book by Haykins
  - Contains one nice chapter of SVM introduction

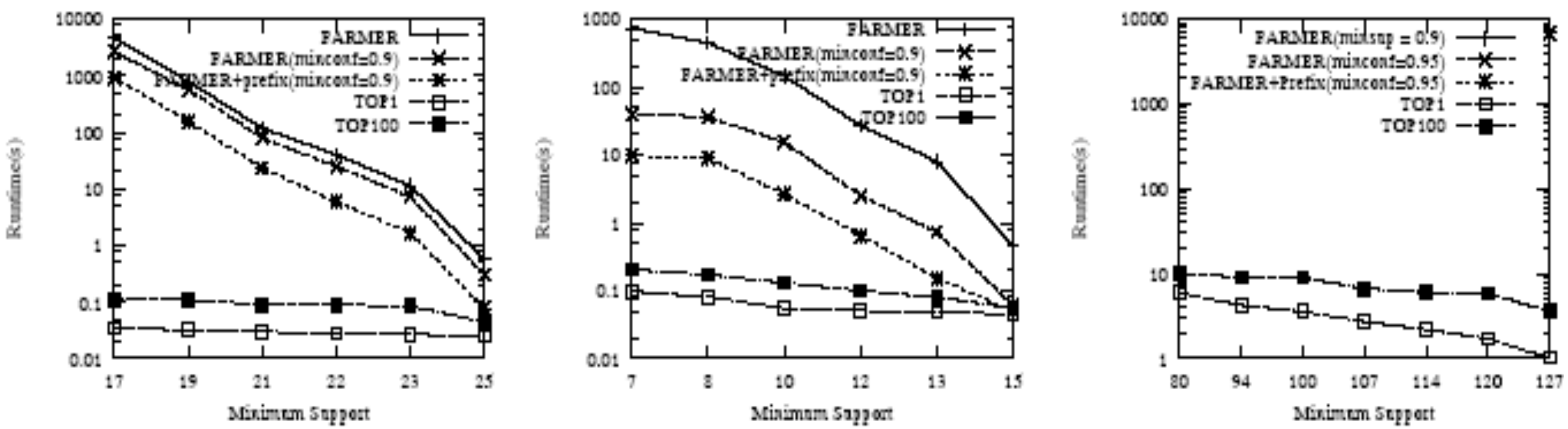
Notes about SVM— Introductory Literature

- "Statistical Learning Theory" by **Vapnik**: difficult to understand, containing many errors.
- C. J. C. **Burges**. A Tutorial on Support Vector Machines for Pattern Recognition. *Knowledge Discovery and Data Mining*, 2(2), 1998.
  - Easier than Vapnik's book, but still not introductory level; the examples are not so intuitive
- The book An Introduction to Support Vector Machines by **Cristianini and Shawe-Taylor**
  - Not introductory level, but the explanation about Mercer's Theorem is better than above literatures
- Neural Networks and Learning Machines by **Haykin**
  - Contains a nice chapter on SVM introduction

Associative Classification Can Achieve High Accuracy and Efficiency (Cong et al. SIGMOD05)

Dataset	RCBT	CBA	IRG Classifier	C4.5 family			SVM
				single tree	bagging	boosting	
AML/ALL (ALL)	91.18%	91.18%	64.71%	91.18%	91.18%	91.18%	97.06%
Lung Cancer(LC)	97.99%	81.88%	89.93%	81.88%	96.64%	81.88%	96.64%
Ovarian Cancer(OC)	97.67%	93.02%	-	97.67%	97.67%	97.67%	97.67%
Prostate Cancer(PC)	97.06%	82.35%	88.24%	26.47%	26.47%	26.47%	79.41%
Average Accuracy	95.98%	87.11%	80.96%	74.3%	77.99%	74.3%	92.70%

Table 2: Classification Results



(a) ALL-AML leukemia

(b) Lung Cancer

(c) Ovarian Cancer

A Closer Look at CMAR

- **CMAR** (Classification based on Multiple Association Rules: Li, Han, Pei, ICDM'01)
- Efficiency: Uses an enhanced FP-tree that maintains the distribution of class labels among tuples satisfying each frequent itemset
- Rule pruning whenever a rule is inserted into the tree
  - Given two rules, R_1 and R_2, if the antecedent of R_1 is more general than that of R_2 and $conf(R_1) \geq conf(R_2)$, then prune R_2
  - Prunes rules for which the rule antecedent and class are not positively correlated, based on a χ^2 test of statistical significance
- Classification based on generated/pruned rules
  - If only *one rule* satisfies tuple X, assign the class label of the rule
  - If a *rule set* S satisfies X, CMAR
    - divides S into groups according to class labels
    - uses a weighted χ^2 measure to find the strongest group of rules, based on the statistical correlation of rules within a group
    - assigns X the class label of the strongest group