



CS 412 Intro. to Data Mining

Chapter 9. Classification: Advanced Methods

Jiawei Han, Computer Science, Univ. Illinois at Urbana-Champaign, 2017





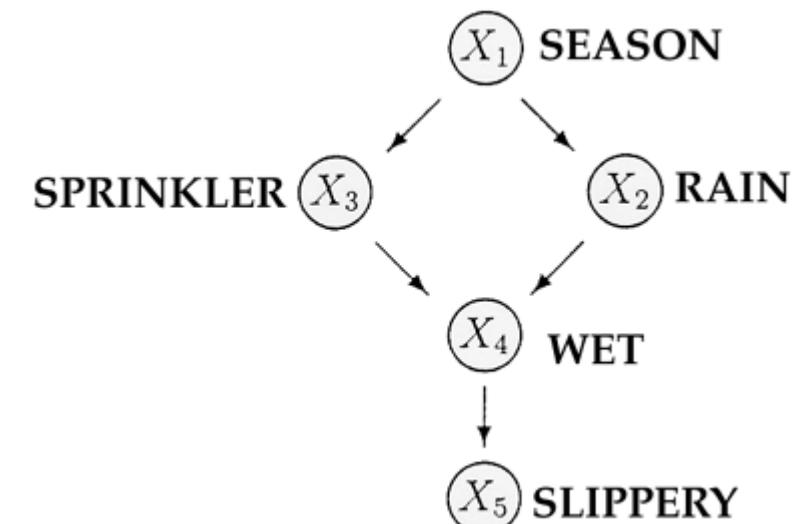
Chapter 9. Classification: Advanced Methods

- Bayesian Belief Networks
- Support Vector Machines
- Neural Networks and Deep Learning
- Pattern-Based Classification
- Lazy Learners and K-Nearest Neighbors
- Other Classification Methods
- Summary



From Naïve Bayes to Bayesian Networks

- Naïve Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable
 - This assumption is often too simple to model the real world well
- Bayesian network (or Bayes network, belief network, Bayesian model or probabilistic directed acyclic graphical model) is a probabilistic **graphical model**
 - Represented by a set of *random variables* and *their conditional dependencies* via a *directed acyclic graph* (DAG)
 - Ex. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases



Bayesian Belief Networks

- Bayesian belief network (or Bayesian network, probabilistic network):

- Allows *class conditional independencies* between *subsets* of variables

- Two components:

- A *directed acyclic graph* (called a structure)
 - A set of *conditional probability tables* (CPTs)

- *Directed Acyclic Graph* (DAG):

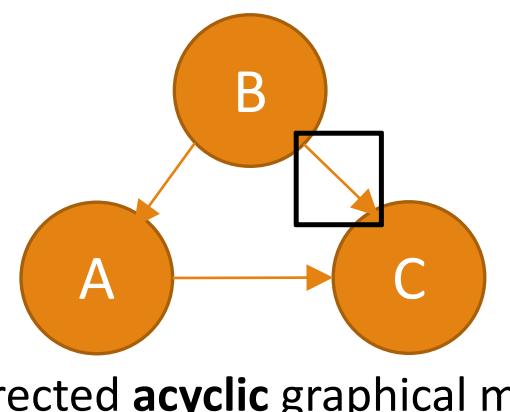
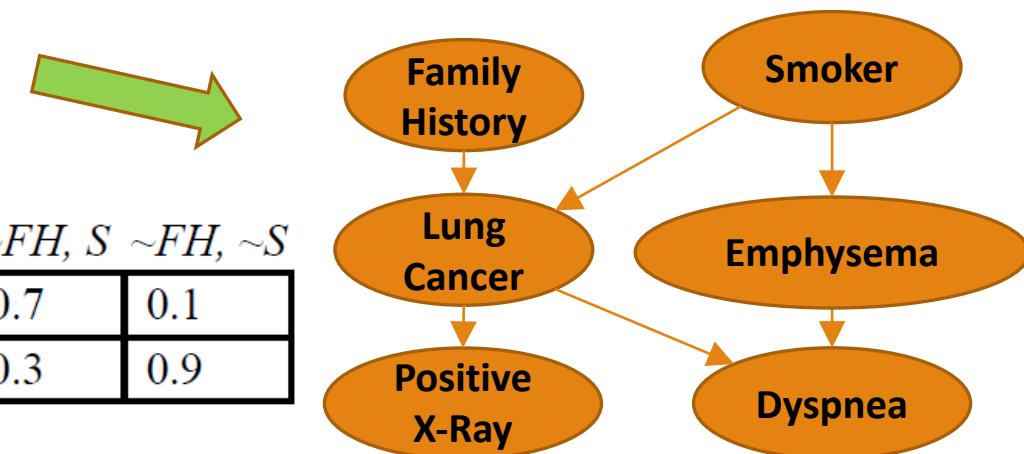
		<i>FH, S</i>	<i>FH, ~S</i>	<i>~FH, S</i>	<i>~FH, ~S</i>	
		<i>LC</i>	0.8	0.5	0.7	0.1
<i>~LC</i>		0.2	0.5	0.3	0.9	

- Represents dependency among the variables (*causal influence* relationship)

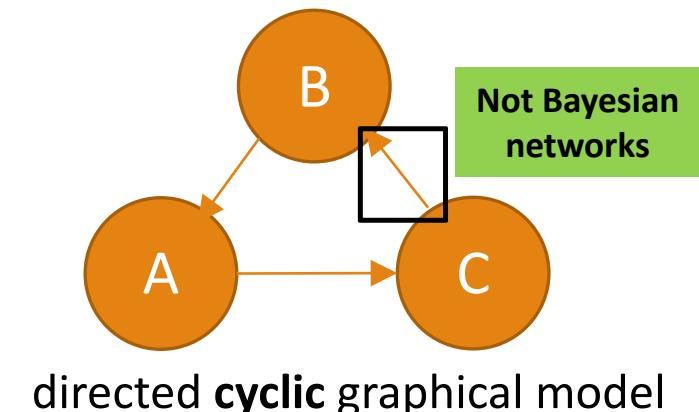
- Gives a specification of joint probability distribution

$$p(A, B, C) = p(B) \cdot p(A|B) \cdot p(C|A, B)$$

Nodes: random variables Links: dependency



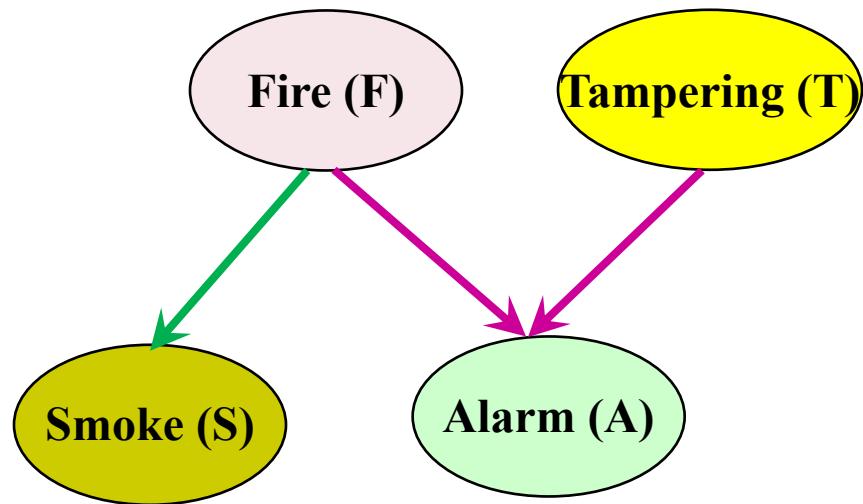
directed acyclic graphical model



Not Bayesian networks

directed cyclic graphical model

A Bayesian Network and Its CPTs



Conditional Probability Tables (CPT)

Fire	Smoke	$\Theta_{s f}$
True	True	.90
False	True	.01

Fire	Tampering	Alarm	$\Theta_{a f,t}$
True	True	True	.5
True	False	True	.99
False	True	True	.85
False	False	True	.0001

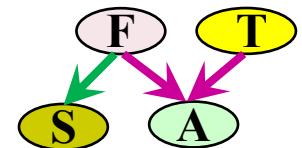
CPT shows the conditional probability for each possible combination of its parents:

$$p(X) = \prod_k p(x_k | \text{Parents}(x_k))$$

$$p(F, S, A, T) = p(F) \cdot p(T) \cdot p(S|F) \cdot p(A|F, T)$$

How Are Bayesian Networks Constructed?

- **Subjective construction:** Identification of (direct) causal structure
 - People are quite good at identifying direct causes from a given set of variables & whether the set contains all relevant direct causes
 - Markovian assumption: Each variable becomes independent of its non-effects once its direct causes are known
 - E.g., $S \leftarrow F \rightarrow A \leftarrow T$, path $S \rightarrow A$ is blocked once we know $F \rightarrow A$
 - HMM (Hidden Markov Model): often used to model dynamic systems whose states are not observable, yet their outputs are
- **Synthesis from other specifications**
 - E.g., from a formal system design: block diagrams & info flow
- **Learning from data** (e.g., from medical records or student admission record)
 - Learn parameters give its structure or learn both structure and parms
 - Maximum likelihood principle: favors Bayesian networks that maximize the probability of observing the given data set

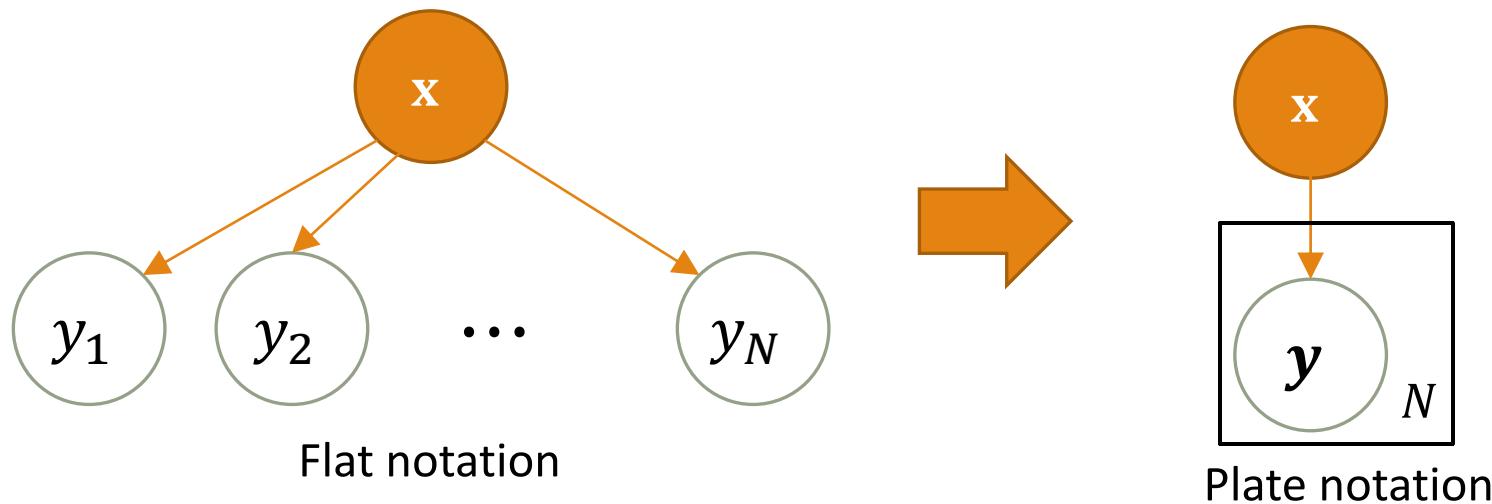
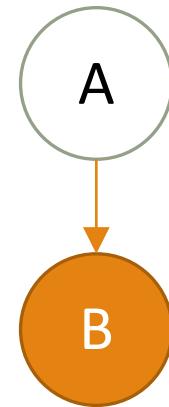


Training Bayesian Networks: Several Scenarios

- 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, without 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

Probabilistic Graphic Model: Plate Notations

- Represent variables that repeat in a graphical model
- Variables
 - A solid (or shaded) circle means the corresponding variable is *observed*; otherwise it is *hidden*
- Dependency among variables:
 - A Directed Acyclic Graphical (DAG) model
- Using plate notation instead of flat notation



An Example of Plate Notation

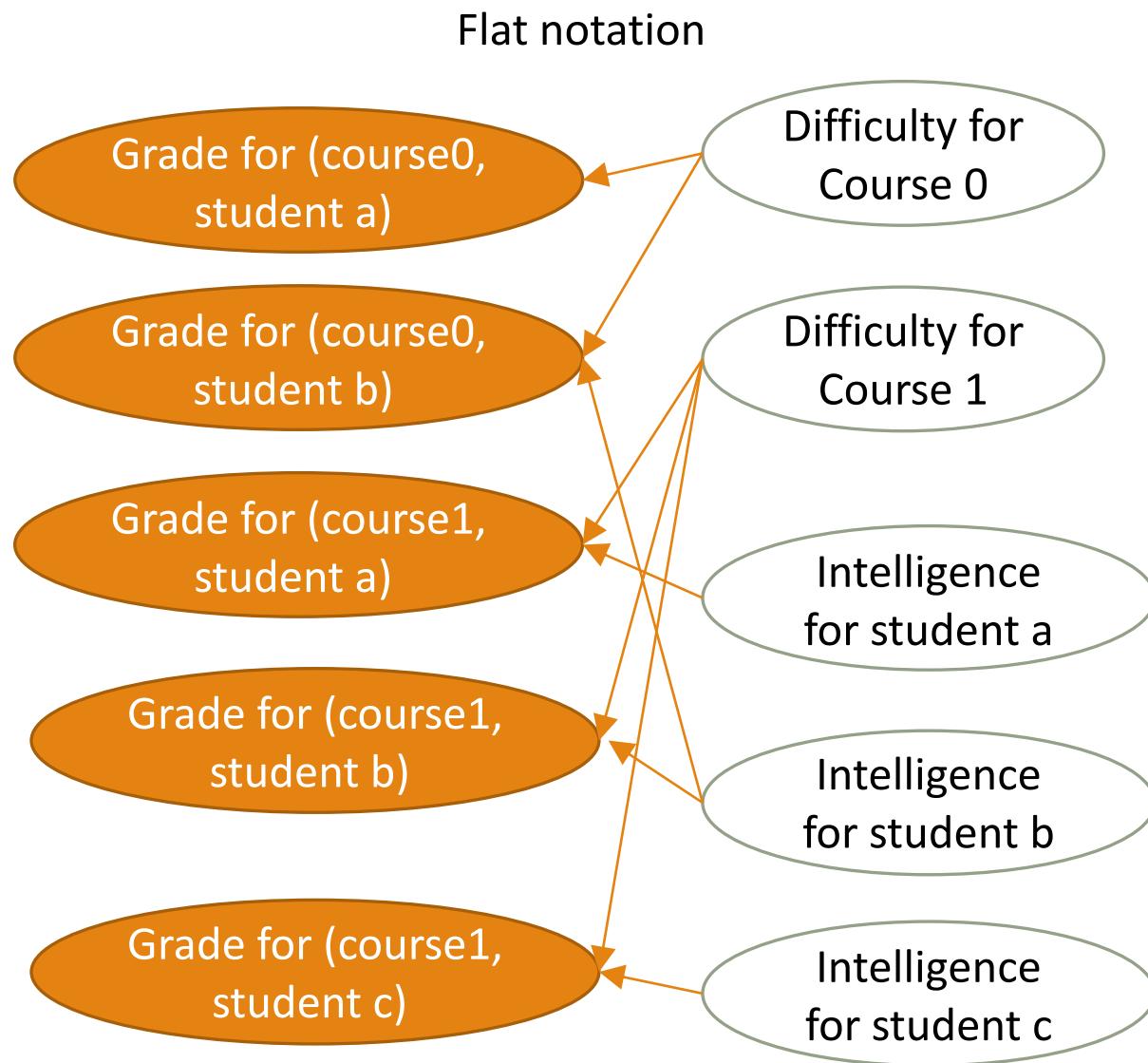
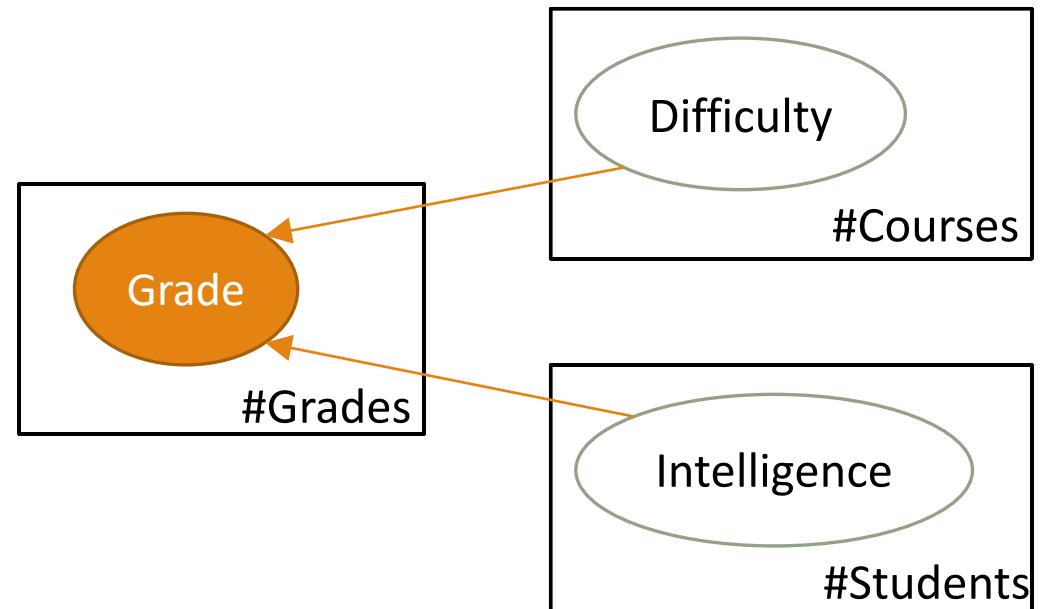


Plate notation



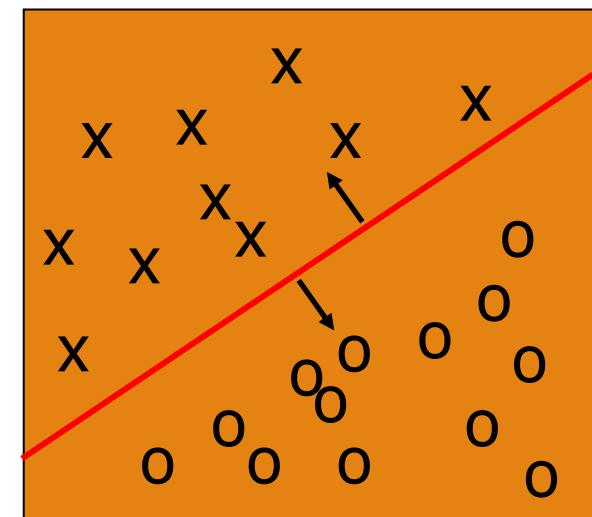
Chapter 9. Classification: Advanced Methods

- Bayesian Belief Networks
- Support Vector Machines
- Neural Networks and Deep Learning
- Pattern-Based Classification
- Lazy Learners and K-Nearest Neighbors
- Other Classification Methods
- Summary



Classification: A Mathematical Mapping

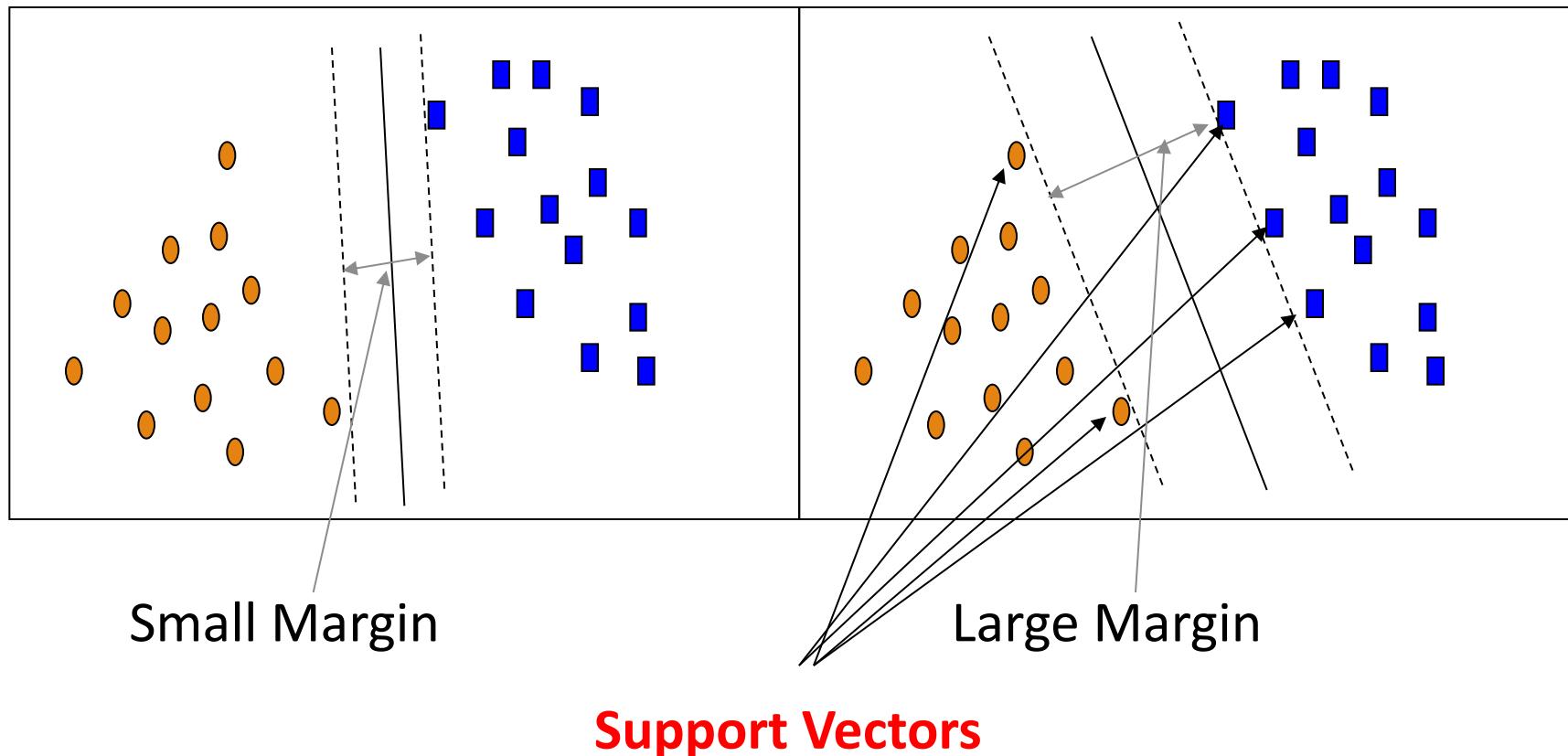
- **Classification:** Predicts categorical class labels
 - E.g., Personal homepage classification
 - $x_i = (x_1, x_2, x_3, \dots)$, $y_i = +1$ or -1
 - x_1 : # of word “homepage”
 - x_2 : # of word “welcome”
 - Mathematically, $x \in X = \mathbb{R}^n$, $y \in Y = \{+1, -1\}$,
 - We want to derive a 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



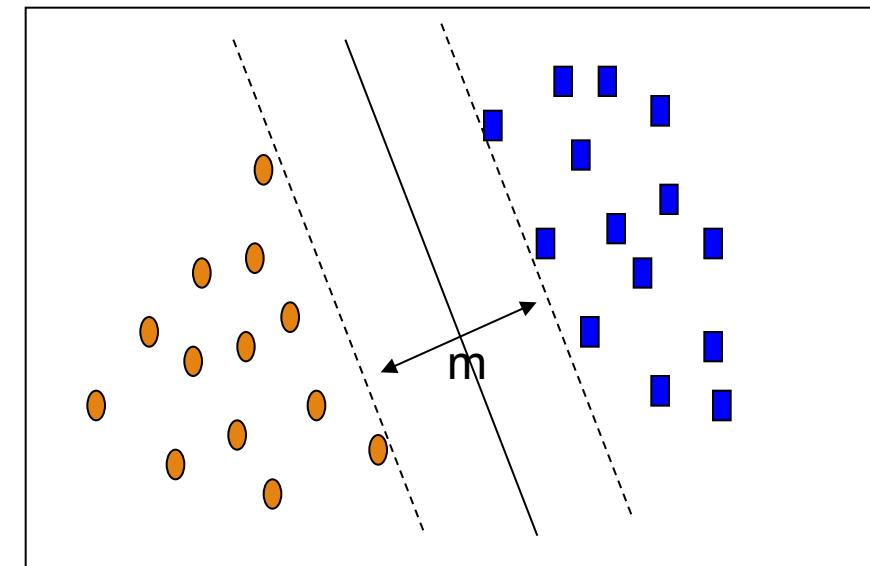
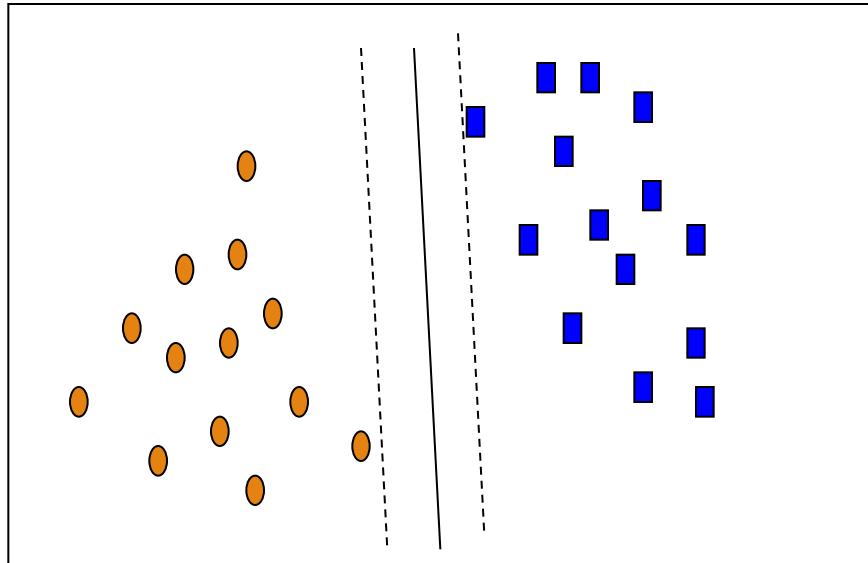
SVM—Support Vector Machines

- A relatively new classification method for both linear and nonlinear data
 - Vapnik and colleagues (1992)—groundwork from Vapnik & Chervonenkis' statistical learning theory in 1960s
- 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—General Philosophy



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 → *Quadratic Programming (QP)* → Lagrangian multipliers

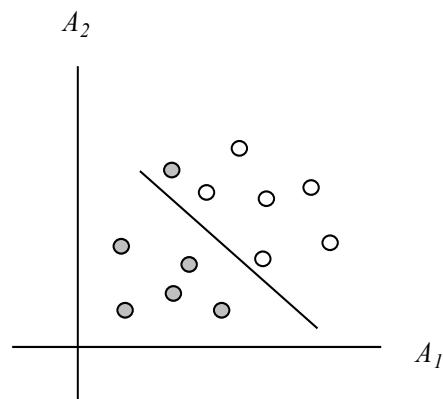
SVM—Linearly Inseparable

- 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 $d(Z) = \mathbf{WZ} + b$, 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 (Z) space corresponds to a nonlinear second order polynomial in the original 3-D input space,

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

- Search for a linear separating hyperplane in the new space



Kernel Functions for Nonlinear Classification

- Instead of computing the dot product on the transformed data, it is mathematically equivalent to applying a kernel function $K(\mathbf{X}_i, \mathbf{X}_j)$ to the original data, i.e.,
 - $K(\mathbf{X}_i, \mathbf{X}_j) = \Phi(\mathbf{X}_i) \Phi(\mathbf{X}_j)$
- Typical Kernel Functions

Polynomial kernel of degree h : $K(\mathbf{X}_i, \mathbf{X}_j) = (\mathbf{X}_i \cdot \mathbf{X}_j + 1)^h$

Gaussian radial basis function kernel : $K(\mathbf{X}_i, \mathbf{X}_j) = e^{-\|\mathbf{X}_i - \mathbf{X}_j\|^2 / 2\sigma^2}$

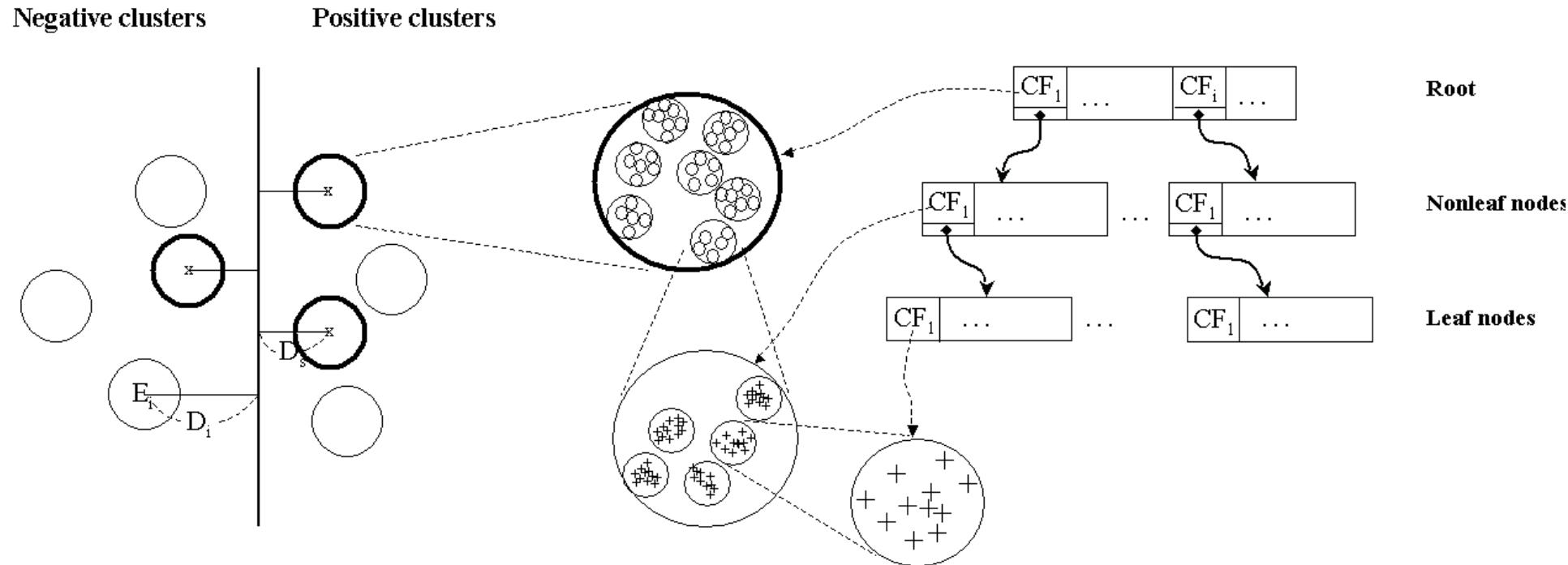
Sigmoid kernel : $K(\mathbf{X}_i, \mathbf{X}_j) = \tanh(\kappa \mathbf{X}_i \cdot \mathbf{X}_j - \delta)$

- SVMs can efficiently perform a non-linear classification using kernel functions, implicitly mapping their inputs into high-dimensional feature spaces

Is SVM Scalable on Massive Data?

- SVM is 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)
 - Thus, an SVM with a small number of support vectors can have good generalization, even when the dimensionality of the data is high
- SVM is not scalable to the # of data objects in terms of training time and memory usage
 - Scaling SVM by a hierarchical micro-clustering approach
 - H. Yu, J. Yang, and J. Han, “Classifying Large Data Sets Using SVM with Hierarchical Clusters”, KDD'03

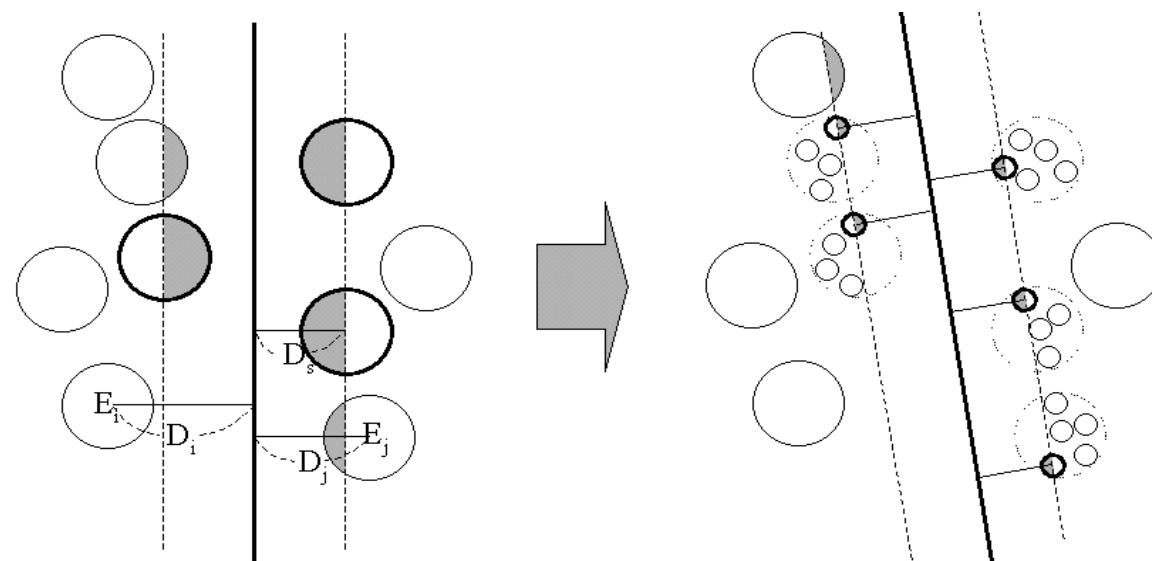
Scaling SVM by Hierarchical Micro-Clustering



- ❑ Construct two CF-trees (i.e., statistical summary of the data) from positive and negative data sets independently (with one scan of the data set)
- ❑ Micro-clustering: Hierarchical indexing structure
 - ❑ Provide finer samples closer to the boundary and coarser samples farther from the boundary

Selective Declustering: Ensure High Accuracy

- De-cluster only the cluster E_i such that
 - $D_i - R_i < D_s$, where D_i is the distance from the boundary to the center point of E_i and R_i is the radius of E_i
 - Decluster only the cluster whose subclusters have possibilities to be the support cluster of the boundary
 - “Support cluster”: The cluster whose centroid is a support vector



Accuracy and Scalability on Synthetic Dataset

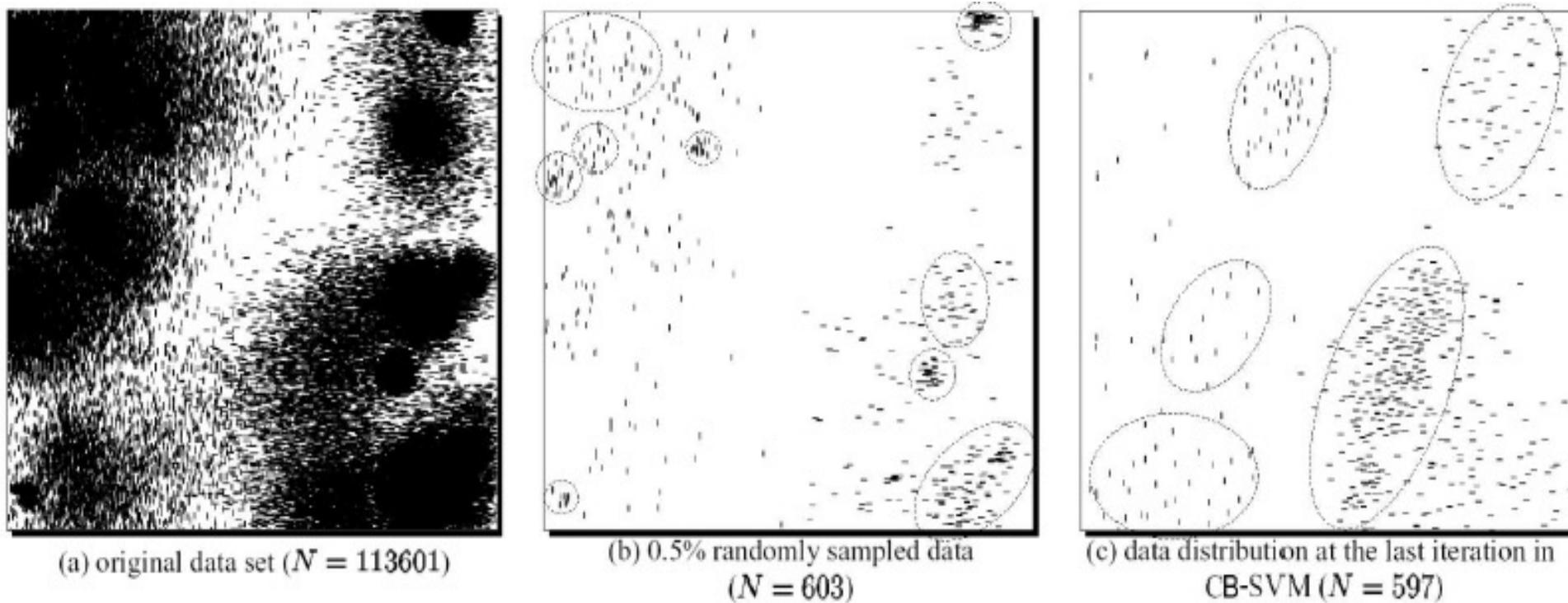


Figure 6: Synthetic data set in a two-dimensional space. '|': positive data; '—': negative data

- ❑ Experiments on large synthetic data sets shows better accuracy than random sampling approaches and far more scalable than the original SVM algorithm

SVM: Applications

- 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
 - SVM can also be used for classifying multiple (> 2) classes and for regression analysis (with additional parameters)
- Applications:
 - handwritten digit recognition, object recognition, speaker identification, benchmarking time-series prediction tests

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

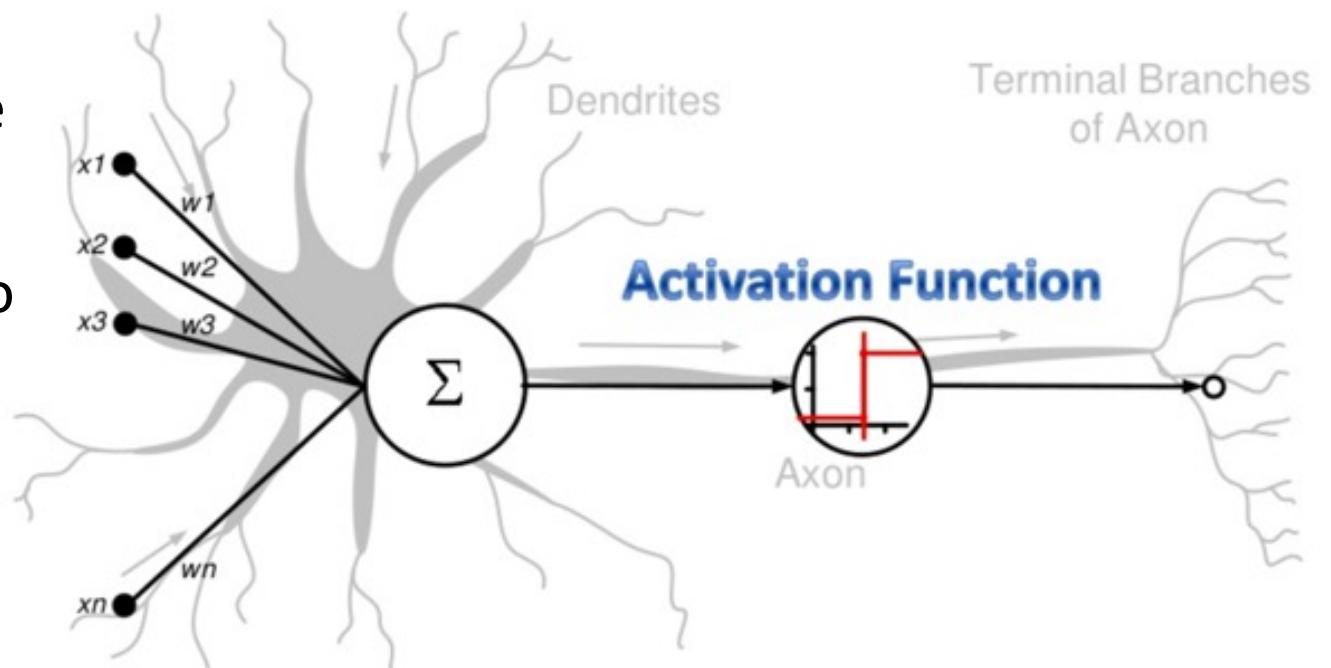
Chapter 9. Classification: Advanced Methods

- Bayesian Belief Networks
- Support Vector Machines
- Neural Networks and Deep Learning
- Pattern-Based Classification
- Lazy Learners and K-Nearest Neighbors
- Other Classification Methods
- Summary



Neural Network for Classification

- 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



Artificial Neural Networks as an analogy of Biological Neural Networks



คก

cp.eng.chula.ac.th



f api61.ilov

f cdn.fbsbx.c...

f api43.ilovep...

f api39.ilovep...

f cdn.fbsbx.c...

x www.cp.eng...

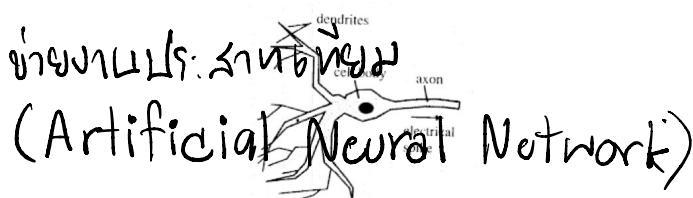
183 จาก 231

เวอร์ชัน 1.0.2: 15 มี.ค.2548 : 9:05 PM boonserm.k@chula.ac.th

6 การเรียนรู้ของเครื่อง 169

6.7 ข่ายงานประสาทเทียม

ช่วยงานประสาทเทียม (*Artificial Neural Network*) เป็นการจำลองการทำงานของสมองมนุษย์ เชลล์ประสาท (neuron) ในสมองของคนเราประกอบด้วยนิวเคลียส (nucleus) ตัวเซลล์ (cell body) ใบประสาทนำเข้า (dendrite) แกนประสาทน้ำออก (axon) และตั้งใน [รูปที่ 6-34](#)

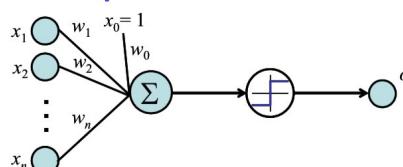


รูปที่ 6-34 เซลล์ประสาท

เด่นไดร์ท์ที่ทำหน้าที่รับสัญญาณไฟฟ้าและมีเชิงสัมภาระจากเซลล์ประสาทใกล้เคียง เชลล์ประสาทด้วยหนึ่ง จะเรียบรวมต่อกันเป็นตัวอ่อนๆ ประมาณ 10,000 ตัว เสื่อสัญญาณไฟฟ้าให้มีที่รับเข้ามาเกินค่าหนึ่ง เชลล์จะถูกกระตุ้นและส่งสัญญาณไปทางแกนประสาทน้ำออกไปยังเซลล์อื่นๆ ต่อไป ประมาณกันว่าสมองของคนเรามีเซลล์ประสาทอยู่ทั้งสิ้นประมาณ 10^{11} ตัว

6.7.1 เพอร์เซปตรอน

เพอร์เซปตรอน (*perceptron*) เป็นข่ายงานประสาทเทียมแบบง่ายมีหน่วยเดียวที่จำลองลักษณะของเซลล์ประสาทดัง [รูปที่ 6-35](#)



รูปที่ 6-35 เพอร์เซปตรอน

170 ปัญญาประดิษฐ์

เวอร์ชัน 1.0.2: 15 มีค.2548 : 9:05 PM boonserm.k@chula.ac.th

เพอร์เซปตรอนรับอินพุตเป็นวงแหวนจึงแล้วคำนวณหาผลรวมเชิงเส้น (linear combination) แบบถ่วงน้ำหนักของอินพุต (x_1, x_2, \dots, x_n) โดยที่ค่า w_1, w_2, \dots, w_n ในรูปเป็นค่าน้ำหนักของอินพุตและให้เอาต์พุต (o) เป็น 1 ถ้าผลรวมที่ได้มามากกว่าค่าศิกไบ (θ) และเป็น -1 ถ้าไม่มาก ส่วน w_0 ในรูปเป็นเกล็ดของค่าศึกแบ่งจังหวัดออกชิบะไป และ x_0 เป็นอินพุตเพื่อยกเว้นให้มีค่าเป็น 1 เสมอ

 **ในรูปแสดงฟังก์ชันกราดตุ้น (activation function)** ชนิดที่เรียกว่า **ฟังก์ชันสองขั้ว (bipolar function)** ซึ่งแสดงผลของเอาต์พุตเป็น 1 กับ -1 ฟังก์ชันกราดตุ้นอื่นๆ ที่นิยมใช้ก็อย่างเช่น **ฟังก์ชันไบนาเรี่ย (binary function)** ซึ่งแสดงผลของเอาต์พุตเป็น 1 กับ 0 และเขียน

 แทนด้วยรูป

ความสามารถแสดงเอาต์พุต (o) ในรูปของฟังก์ชันของอินพุต (x_1, x_2, \dots, x_n) ได้ดังนี้

$$o(x_1, x_2, \dots, x_n) = \begin{cases} 1 & \text{if } w_1x_1 + w_2x_2 + \dots + w_nx_n > \theta \\ -1 & \text{if } w_1x_1 + w_2x_2 + \dots + w_nx_n < \theta \end{cases} \quad (6.7)$$

เอาต์พุตเป็นฟังก์ชันของอินพุตในรูปของผลรวมเชิงเส้นแบบถ่วงน้ำหนัก น้ำหนักจะเป็นตัวกำหนดว่าในจำนวนอินพุตนั้น อินพุต (x_i) ตัวใดมีความสำคัญต่อการกำหนดค่าเอาต์พุต ตัวที่มีความสำคัญมากจะมีค่าสัมภารณ์ของน้ำหนักมาก ส่วนตัวที่มีความสำคัญน้อยจะมีค่าใกล้ศูนย์ ในการนี้เพื่อรวมเท่ากับตัวคูณแบ่งค่าเอาต์พุตไม่ได้ยาน (จะเป็น 1 หรือ -1 而已)

จากฟังก์ชันในสูตรที่ (6.7) เราจัดรูปใหม่โดยยก θ ไปรวมกับผลรวมเชิงเส้นแล้วแทน $-\theta$ ด้วย w_0 เราจะได้ฟังก์ชันของเอาต์พุตดังด้านล่างนี้

$$o(x_1, x_2, \dots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n > 0 \\ -1 & \text{if } w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n < 0 \end{cases} \quad (6.8)$$

กำหนดให้ $g(\vec{x}) = \sum_{i=0}^n w_i x_i = \vec{w} \cdot \vec{x}$ โดยที่ \vec{x} แทนวงแหวนเอาต์พุต เราสามารถเขียนฟังก์ชันของเอาต์พุตได้ใหม่ดังนี้

$$o(x_1, x_2, \dots, x_n) = \begin{cases} 1 & \text{if } g(\vec{x}) > 0 \\ -1 & \text{if } g(\vec{x}) < 0 \end{cases} \quad (6.9)$$

สมมติว่าเรามีอินพุตสองตัวคือ x_1 และ x_2 ซึ่งแสดงค่าส่วนสูงและน้ำหนักของเด็กนักเรียน ประคบกันและหลังจากที่แพทช์ตรวจสอบว่าง่ายของเด็กโดยจะเรียดแล้วได้จำแนกนักเรียน

ออกเป็นสองกลุ่มคือเด็กอ้วนและเด็กไม่อ้วน เราให้เอาต์พุตเป็นค่าที่แสดงเด็กอ้วนแทนด้วย +1 กับไม่อ้วนแทนด้วย -1 ดังตารางที่ 6-16

ตารางที่ 6-16 ข้อมูลเด็กอ้วนและเด็กไม่อ้วน

เด็กคนที่	ส่วนสูง (ซม.)	น้ำหนัก (กг.)	อ้วน/ไม่อ้วน
1	100.0	20.0	-1
2	100.0	26.0	1
3	100.0	30.4	1
4	100.0	32.4	1
5	101.6	27.0	1
6	101.6	32.0	1
7	102.0	21.0	-1
8	103.6	29.6	1
9	104.4	30.4	1
10	104.9	22.0	-1
11	105.2	20.0	-1
12	105.6	34.4	1
13	107.2	32.4	1
14	109.9	34.9	1
15	111.0	25.4	-1
16	114.2	23.5	-1
17	115.5	36.3	1
18	117.8	26.9	-1

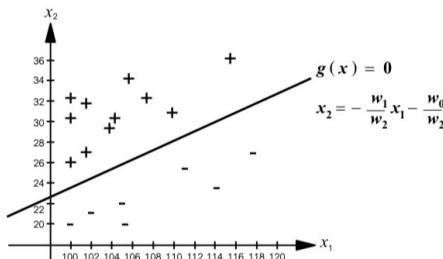
ในการแก้ที่มีอินพุต 2 ตัว (ไม่รวม x_0) เราจะได้ $g(\vec{x}) = w_0 + w_1x_1 + w_2x_2$ ซึ่งถ้าเราให้ $g(\vec{x}) = 0$ จะได้ว่า $w_0 + w_1x_1 + w_2x_2 = 0$ ซึ่งแทนสมการเส้นตรงในระบบสองมิติ x_1 ,

x_2 สมการนี้มีจุดตัดแกนอยู่ที่ $-\frac{w_0}{w_2}$ และมีความชันเท่ากับ $-\frac{w_1}{w_2}$ เมื่อนำสมการนี้ไปวัดใน

ระบบสองมิติร่วมกับตัวอย่างสอนในตารางที่ 6-16 โดยกำหนดค่า w_0, w_1, w_2 ที่เหมาะสมจะได้ดังรูปที่ 6-36

172 ปัญญาประดิษฐ์

เวลาอ่าน 1.0.2: 15 มีค.2548 : 9:05 PM boonserm.k@chula.ac.th



รูปที่ 6-36 สมการเส้นตรงสว่างโดยเพอร์เซปตอรอน

เครื่องหมาย + และ – ในรูปแทนด้วยย่างบวก (เด็กอ้วน) และด้วยย่างลง (เด็กไม่อ้วน) ตามลำดับ ดังจะเห็นได้ในรูปว่าเส้นตรงนี้เมื่อกำหนดจุดตัดแกนและความชันที่เหมาะสมซึ่งกำหนดโดย w_0 , w_1 , w_2 เส้นตรงนี้จะแบ่งตัวอย่างออกเป็นสองกลุ่มซึ่งอยู่คนละด้านของเส้นตรง และเมื่อมีข้อมูลส่วนใหญ่แล้วหัวนักของเด็กคนอื่นที่เราต้องการทำนายว่าจะเป็นเด็กอ้วนหรือไม่ ก็ใช้เส้นตรงนี้โดยคุณว่าข้อมูลใหม่นี้อยู่ด้านใดของเส้นตรง ถ้าด้านบนก็ทำนายว่า เป็นเด็กอ้วน (+) ถ้าด้านล่างก็ทำนายว่าเด็กไม่อ้วน (-)

**ระนาบตัดสินใจ
คลาสิกมิติ**

ตัวอย่างด้านบนแสดงกรณีของอินพุตในสองมิติ จะเห็นได้ว่าเพอร์เซปตอรอนจะเป็นเส้นตรง ในกรณีที่อินพุตมากกว่าสองมิติเพอร์เซปตอรอนจะเป็น **ระนาบตัดสินใจคลาสิกมิติ (hyperplane decision surface)** ปัญหาการเรียนรู้เพอร์เซปตอรอนนี้คือการทำให้เราสามารถจำแนกข้อมูลสองเพื่อให้เพอร์เซปตอรอนแสดงเอาร์พุตได้ตรงกับค่าที่สอน **กฎการเรียนรู้เพอร์เซปตอรอน (perceptron learning rule)** ใช้สำหรับสอนเพอร์เซปตอรอนโดยจะหาค่าเวกเตอร์น้ำหนักดังแสดงในตารางที่ 6-17

อัลกอริทึมรับมือด้านจากส่วนค่าเวกเตอร์น้ำหนัก ซึ่งโดยมากค่าที่สูงมากจะไม่ได้รับระนาบคลาสิกมิติที่แบ่งตัวอย่างได้ถูกต้องทุกตัวดังนั้นจึงต้องมีการแก้ไขน้ำหนักโดยเทียบเพอร์เซปตอรอนกับตัวอย่างที่สอน หมายถึงว่าเมื่อเราป้อนตัวอย่างสองเข้าไปในเพอร์เซปตอรอน เราจะคำนวณค่าเอาร์พุตได้ นำค่าเอาร์พุตที่คำนวณได้โดยเพอร์เซปตอรอนเทียบกับเอาร์พุตเป้าหมาย ถ้าตรงกันแสดงว่าจำแนกด้วยตัวอย่างได้ถูกต้อง ไม่ต้องปรับน้ำหนักสำหรับตัวอย่างนั้น แต่ถ้าไม่ตรงกันก็จะทำการปรับน้ำหนักตามสมการในอัลกอริทึม ส่วนอัตราการเรียนรู้เป็นตัวเลขบวกจำนวนน้อยๆ เช่น 0.01, 0.005 เป็นต้น อัตราการเรียนรู้นี้จะส่งผลต่อการลู่เข้าของเพอร์เซปตอรอน ถ้าอัตราการเรียนรู้มีค่ามากเพอร์เซปตอรอนก็จะเรียนรู้ได้เร็ว แต่ก็อาจเรียนรู้ไม่สำเร็จเนื่องจากการปรับค่ามีความหมายเกินไป อัตราการเรียนรู้ที่มีค่า น้อยก็จะทำให้การปรับน้ำหนักทำได้อย่างลento เนื่องจากเสียเวลาในการเรียนรู้นาน

ตารางที่ 6-17 อัลกอริทึมกฎการเรียนรู้เพอร์เซปตรอน

Algorithm: Perceptron-Learning-Rule

1. Initialize weights w_i of the perceptron.
2. UNTIL the termination condition is met DO
 - 2.1 FOR EACH training example DO
 - Input the example and compute the output.
 - Change the weights if the output from the perceptron is not equal to the target output using the following rule.

$$w_i \leftarrow w_i + \Delta w_i$$

input ของมันเอง
 learning rate
 $\Delta w_i \leftarrow \alpha(t-o)x_i$
 เป็นตัวบวกตัวลบ: บวกมาก
 หักน้อย
 target = หนึ่งในสองอย่าง

where t , o and α are the target output, the output from the perceptron and the learning rate, respectively.

การปรับน้ำหนักตามกฎการเรียนรู้เพอร์เซปตรอนโดยใช้อัตราการเรียนรู้ที่มีค่าน้อยเพียงพอ จะได้รับผลลัพธิ์ที่จะถูกเข้าสู่ระบบหนึ่งที่สามารถแบ่งข้อมูลออกเป็นสองส่วน (ในกรณีที่ข้อมูลสามารถแบ่งได้) เพื่อบริษัทผลิตที่เกิดจากการปรับค่าน้ำหนัก เราจะลองพิจารณาพฤติกรรมของกฎการเรียนรู้นี้ดูว่าทำไงการปรับน้ำหนักเช่นนี้ถูกเข้าสู่ระบบหนึ่งที่แบ่งข้อมูลได้อย่างถูกต้อง

- พิจารณากรณีแรกที่เพอร์เซปตรอนแยกตัวอย่างสองส่วนด้วยเส้นตัวหนึ่งที่รับเข้ามาได้ถูกต้องกรณีนี้จะพบว่า $(t-o)$ จะมีค่าเป็น 0 ดังนั้น Δw_i ไม่เปลี่ยนแปลง เพราะ $\Delta w_i = \alpha(t-o)x_i$
- พิจารณาในกรณีที่เพอร์เซปตรอนให้ເອົາດົດເປັນ -1 ແຕ່ເອົາດົດເປັນທີ່ກີບໄວ້ 1 ໃນกรณีนี้หมายความว่าค่าที่เราต้องการคือ 1 ແຕ່ค่าน้ำหนักໄມ່ແທນະສົມ ດັ່ງນັ້ນເພື່ອທີ່ຈະກຳໄຫ້ເພື່ອຮັບເອົາດົດເປັນ 1 ນ້ຳນັກຕ້ອງກູກປັບໃຫ້ສາມາດເພີ່ມຄໍາຂອງ $t \cdot x$ ໃນกรณีນີ້หมายความວ່າຜລຣວມເຊີງເສັ້ນນັ້ນຍເກີນໄປແລະນັ້ນຍກວ່າ 0 ຈຶ່ງໄດ້ເອົາດົດເປັນ -1 ດັ່ງນັ້ນສິ່ງທີ່ເຮັດວຽກມີຄ່າຜລຣວມເຊີງເສັ້ນເພີ່ມຄໍາເຊີ້ມເວື່ອຍໆ ຈົນນັ້ນກວ່າ 0 ເພື່ອຮັບຜລຣວມທີ່ເອົາດົດເປັນ 1 ສິ່ງທີ່ກີບໄວ້ພວກເຮົາກັນ $(1-(-1))$ ມີຄ່າເປັນ 2 ແລະລອງພິຈານາຄ່າຂອງອິນພຸດ x_i ແຍກກົດຕັ້ງນີ້

174 ปัญญาประดิษฐ์

เวลาอ่าน 1.0.2: 15 มีค.2548 : 9:05 PM boonserm.k@chula.ac.th

- ถ้า $x_i > 0$ จะได้ว่า Δw_i มากกว่า 0 เพราะว่า $\Delta w_i \leftarrow \alpha(t-o)x_i$ และ α มากกว่า 0, $(t-o) = 2$ และ $x_i > 0$ จากสมการการปรับน้ำหนัก $w_i \leftarrow w_i + \Delta w_i$ เมื่อ Δw_i มากกว่า 0 จะทำให้ w_i มีค่าเพิ่มขึ้นและ $\sum w_i x_i$ ก็จะมีค่าเพิ่มขึ้น เมื่อผลรวมมีค่ามากขึ้นแสดงว่าการปรับไปในทิศทางที่ถูกต้องคือเมื่อปรับไปจนกระทั่งได้ผลรวมมากกว่า 0 จะทำให้เพอร์เซปตรอนอาดัพดูถูกต้องยิ่งขึ้น
- ถ้า $x_i < 0$ เราจะได้ว่า $\alpha(t-o)x_i$ จะมีค่าน้อยกว่า 0 แสดงว่า w_i ตัวที่คุณกับ x_i ที่น้อยกว่า 0 จะลดลงทำให้ $\sum w_i x_i$ เพิ่มขึ้นเหมือนเดิม เพราะ x_i เป็นค่าลบและ w_i ฝ่ายลดลง ในที่สุดก็จะทำให้เพอร์เซปตรอนให้อาดัพดูถูกต้องยิ่งขึ้น
- ในการนี้ที่เพอร์เซปตรอนให้อาดัพเป็น 1 แต่อาดัพเป้าหมายหรือค่าที่แท้จริงเท่ากับ -1 จะได้ว่า w_i ของ x_i ที่เป็นค่านegativeลดลง ส่วน w_i ของ x_i ที่เป็นค่าลบจะเพิ่มขึ้นและทำให้การปรับเป็นไปในทิศทางที่ถูกต้องเช่นเดียวกับในการนี้แรก



6.7.2 ตัวอย่างการเรียนฟังก์ชัน AND และ XOR ด้วยกฎเรียนรู้เพอร์เซปตรอน

พิจารณาตัวอย่างการเรียนรู้ของเพอร์เซปตรอนโดยจะให้เรียนรู้ฟังก์ชัน 2 ฟังก์ชัน ฟังก์ชัน แรกคือฟังก์ชัน AND แสดงในตารางที่ 6-18 ในกรณีเราใช้ฟังก์ชันในนี้เป็นฟังก์ชันกราฟดูนี้

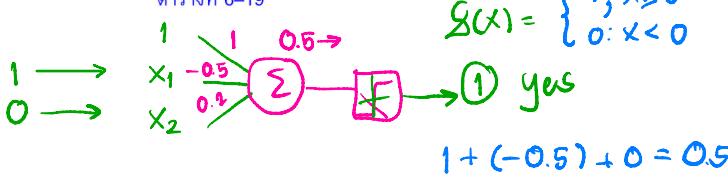
ตารางที่ 6-18 ฟังก์ชัน AND(x_1, x_2)

กฎ นิกบท

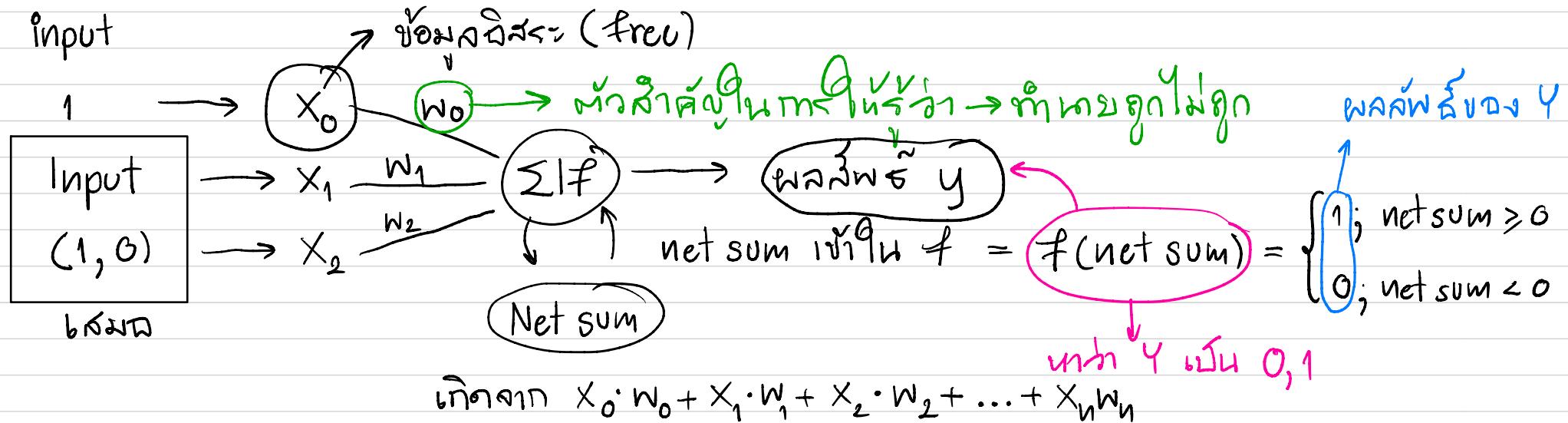
x_1	x_2	อาดัพ เป้าหมาย
0	0	0
0	1	0
1	0	0
1	1	1

Data

ฟังก์ชัน AND ตามตารางด้านบนนี้จะให้ค่าที่เป็นจริงถ้าเมื่อ x_1 และ x_2 เป็นจริงทั้งคู่ (ถ้าที่แสดงมีอาดัพเป้าหมาย) ผลการใช้กฎการเรียนรู้เพอร์เซปตรอนกับฟังก์ชัน AND และในตารางที่ 6-19



HW 5 ข่ายงานประสาทเทียม (ANN: Artificial Neural Network)



$$W = \text{ห\u00e1ม\u00e1น\u00e1} \rightarrow W_i(a_{\text{ใหม}}) = W_i(a_{\text{เดิม}}) + (\alpha (+-0) x_i)$$

Output = ผลลัพธ\u00f3ท\u00f3ป\u00f4ด
\u00d7 (not sum)

$t = \text{target} = \text{ผลลัพธ\u00f3ท\u00f3ตาม\u00e1น\u00e1}$

$$W_i(q_{NN}) = W_i(b_{NN}) + [(\psi(t-\tau)) \times_i]$$

$$w_1(q_{un}) = w_1(\sigma) + [\alpha(t-\sigma)x_1]$$

ตารางที่ 6-19 ผลการเรียนรู้ฟังก์ชัน AND โดยกฎการเรียนรู้เพอร์เซปตรอน

Perceptron Learning Example - Function AND												
		Bias Input $x_0=+1$						Alpha = $0.5 = \alpha$				
Input	Input				Net Sum	Target	Actual	Alpha*	Weight Values			
x_0	x_1	x_2	$1.0*w_0$	x_1*w_1	x_2*w_2	Input	Output	Error	w_0	w_1	w_2	
			↓	↓	↓		t	0	$\alpha(t-0)$	0.5	0.5	0.5
1	0	0	0.5	0	0	0.5	0	1	$0.5(0-1)$ = -0.5	$0.5 + (-0.5 \times 1)$ = 0	$0.5 + (-0.5 \times 0)$ = 0.5	$0.5 + (-0.5 \times 0)$ = 0.5
1	0	1	0	0	0.5	0.5	0	1	$0.5(0-1)$ = -0.5	$0.5 + (-0.5 \times 1)$ = -0.5	$0.5 + (-0.5 \times 0)$ = 0.5	$0.5 + (-0.5 \times 1)$ = 0
1	1	0	-0.5	0.5	0	0	0	1	$0.5(0-1)$ = -0.5	$0.5 + (-0.5 \times 1)$ = -1	$0.5 + (-0.5 \times 0)$ = 0	$0.5 + (-0.5 \times 0)$ = 0
1	1	1	-1	0	0	-1	1	0	$0.5(1-0)$ = 0.5	$0.5 + (-0.5 \times 1)$ = -0.5	$0.5 + (-0.5 \times 1)$ = 0.5	$0.5 + (-0.5 \times 1)$ = 0.5

$$f(\text{net sum}) = \begin{cases} 1 & ; \text{net sum} \geq 0 \\ 0 & ; \text{net sum} < 0 \end{cases}$$

ตารางที่ 6-19 ผลการเรียนรู้ฟังก์ชัน AND โดยกฎการเรียนรู้เพอร์เซปตรอน

Perceptron Learning Example - Function AND												
								Alpha = 0.5				
		Bias Input $x_0=+1$						Alpha = 0.5				
	Input	Input				Net Sum	Target	Actual	Alpha*	Weight Values		
x_0	x_1	x_2	$1.0 * w_0$	$x_1 * w_1$	$x_2 * w_2$	Input	Output	Output	Error	w_0	w_1	w_2
1	0	0	0.5	0	0	0.5	0	1	-0.5	0	0.5	0.5
1	0	1	0	0	0.5	0.5	0	1	-0.5	-0.5	0.5	0
1	1	0	-0.5	0.5	0	0	0	1	-0.5	-1	0	0
1	1	1	-1	0	0	-1	1	0	0.5	-0.5	0.5	0.5
1	0	0	-0.5	0	0	-0.5	0	0	0	-0.5	0.5	0.5
1	0	1	-0.5	0	0.5	0	0	1	-0.5	-1	0.5	0
1	1	0	-1	0.5	0	-0.5	0	0	0	-1	0.5	0
1	1	1	-1	0.5	0	-0.5	1	0	0.5	-0.5	1	0.5
1	0	0	-0.5	0	0	-0.5	0	0	0	-0.5	1	0.5
1	0	1	-0.5	0	0.5	0	0	1	-0.5	-1	1	0
1	1	0	-1	1	0	0	0	1	-0.5	-1.5	0.5	0
1	1	1	-1.5	0.5	0	-1	1	0	0.5	-1	1	0.5
1	0	0	-1	0	0	-1	0	0	0	-1	1	0.5
1	0	1	-1	0	0.5	-0.5	0	0	0	-1	1	0.5
1	1	0	-1	1	0	0	0	1	-0.5	-1.5	0.5	0.5
1	1	1	-1.5	0.5	0.5	-0.5	1	0	0.5	-1	1	1

		Bias Input $x_0=+1$					Alpha =	0.5				
	Input	Input				Net Sum	Target	Actual	Alpha*	Weight Values		
x_0	x_1	x_2	$1.0 \cdot w_0$	$x_1 \cdot w_1$	$x_2 \cdot w_2$	Input	Output	Output	Error	w_0	w_1	w_2
1	0	0	-1	0	0	-1	0	0	0	-1	1	1
1	0	1	-1	0	1	0	0	1	-0.5	-1.5	1	0.5
1	1	0	-1.5	1	0	-0.5	0	0	0	-1.5	1	0.5
1	1	1	-1.5	1	0.5	0	1	1	0	-1.5	1	0.5
1	0	0	-1.5	0	0	-1.5	0	0	0	-1.5	1	0.5
1	0	1	-1.5	0	0.5	-1	0	0	0	-1.5	1	0.5
1	1	0	-1.5	1	0	-0.5	0	0	0	-1.5	1	0.5
1	1	1	-1.5	1	0.5	0	1	1	0	-1.5	1	0.5

6 epoch

$$Acc = \frac{TP+TN}{All} = \frac{4+0}{4} \times 100$$

$$= 1 \times 100$$

$$= 100 \% \ast$$

b43021266-5 និង ៩៧០០

Actual class \ Predicted class	yes	no	Total
yes	TP = 4	FN = 2	P = 6
no	FP = 2	TN = 2	N = 4
Total	P' = 6	N' = 4	10

$$\text{Accuracy} = \frac{(TP+TN)}{\text{All}} = \frac{(4+2)}{10} = \frac{6}{10}$$

Actual = Yes
Predicted = Yes

Actual = no
Predicted = no

		Bias Input $x_0=+1$						Alpha = 0.5		
	Input	Input				Net Sum	Target	Actual	Alpha*	
x_0	x_1	x_2	$1.0*w_0$	x_1*w_1	x_2*w_2	Input	Output	Output	Error	w
0	0	-1	0	0	-1	-1	0	0	0	-
0	1	-1	0	1	0	0	0	1	-0.5	-1.
1	0	-1.5	1	0	-0.5	0	0	0	0	-1.
1	1	-1.5	1	0.5	0	1	1	1	0	-1.
0	0	-1.5	0	0	-1.5	0	0	0	0	-1.
0	1	-1.5	0	0.5	-1	0	0	0	0	-1.
1	1	-1.5	1	0	-0.5	0	0	0	0	-1.
1	1	-1.5	1	0.5	0	1	1	1	0	-1.

6 epoch

$$\text{Acc} = \frac{TP+TN}{\text{All}} = \frac{4+0}{4+1} \times 100 = \frac{4}{5} \times 100 = 100 \% \times$$

249/24
TP = 4
TN = 0
Predict = Yes
Actual = Yes
Predict = No
Actual = No
 $\frac{4}{5} = 0$

Ex 11/11/11

	Target	Actual
t	Output	Output
t	0	0
0	0	1
0	0	0

$$\text{Acc} =$$

$$=$$

$$=$$

$$=$$

		Bias Input $x_0 = +1$						Alpha = 0.5		
	Input	Input				Net Sum	Target	Actual	Alpha*	
x_0	x_1	x_2	$1.0 * w_0$	$x_1 * w_1$	$x_2 * w_2$	Input	Output	Output	Error	w
0	0	0	-1	0	0	-1	0	0	0	-
0	1	-1	0	1	0	0	0	1	-0.5	-1.
1	0	-1.5	1	0	-0.5	0	0	0	0	-1.
1	1	-1.5	1	0.5	0	1	1	0	0	-1.
0	0	-1.5	0	0	-1.5	0	0	0	0	-1.
0	1	-1.5	0	0.5	-1	0	0	0	0	-1.
1	1	0	-1.5	1	0	-0.5	0	0	0	-1.
1	1	1	-1.5	1	0.5	0	1	1	0	-1.

6 epoch

• Confusion matrix

$$\text{Accurate : } AC = \frac{TP + TN}{all} = \frac{1+3}{4} \times 100\%.$$

รวมผลผิดพลาดทั้งหมด = 100%.

		Actual output		
		1	0	
		1	0	
Target output				
1	0	1	0	1
0	1	0	1	3
		1	3	4

แสดงค่า weight Value ซึ่งทั้งที่ปรับจนถูกต้องเท่านั้น

$$w_0 = -1.5$$

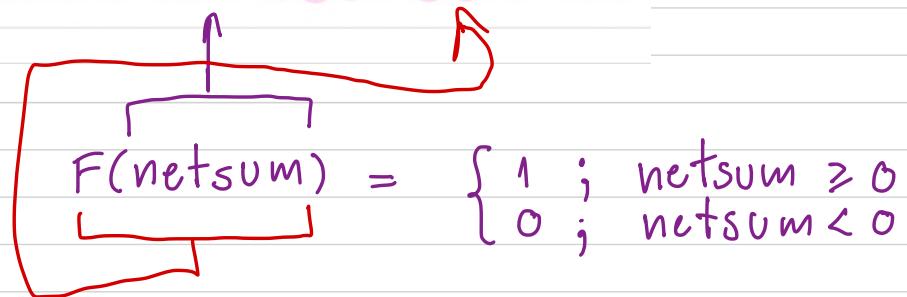
$$w_1 = 1$$

$$w_2 = 0.5$$

ตอบ

		Bias Input $x_0=+1$						Alpha = 0.5		
Input	Input				Net Sum	Target	Actual	Alpha*	w	
x_0	x_1	x_2	$1.0 \cdot w_0$	$x_1 \cdot w_1$	$x_2 \cdot w_2$	Input	Output	Output	Error	w
						t	0	$w(t-0)$	Δ	
0	0	-1	0	0	-1	0	0	0	-	
0	1	-1	0	1	0	0	0	-0.5	-1.	
1	0	-1.5	1	0	-0.5	0	0	0	-1.	
1	1	-1.5	1	0.5	0	1	1	0	-1.	
0	0	-1.5	0	0	-1.5	0	0	0	-1.	
0	1	-1.5	0	0.5	-1	0	0	0	-1.	
1	1	0	-1.5	1	0	-0.5	0	0	-1.	
1	1	1	-1.5	1	0.5	0	1	0	-1.	

b epoch



$$f(\Sigma) \begin{cases} 1; \Sigma \geq 0 \\ 0; \Sigma < 0 \end{cases}$$

เวลาอ่าน 1.0.2: 15 มีค.2548 : 9:05 PM boonserm.k@chula.ac.th

6 การเรียนรู้ของเครื่องประดิษฐ์

175

แบบ
การ
เรียนรู้
ของเครื่องประดิษฐ์

ห้อง
เรียน

} Epoch
จำนวน

ตารางที่ 6-19 ผลการเรียนรู้ฟังก์ชัน AND โดยกฎการเรียนรู้เพอร์เซปตอร์

Perceptron Learning Example - Function AND											
		Bias Input $x_0 = +1$		Net Sum	Target	Actual Output	Error	Alpha*	Weight Values		
Input	Input	$1.0 * w_0$	$x_1 * w_1$	$x_2 * w_2$	Input	Output			w_0	w_1	w_2
x_1	x_2										
0	0	0.10	0.00	0.10	0	0	-0.50	-0.40	0.10	0.10	0.10
0	1	-0.40	0.00	-0.40	0	0	0.00	-0.40	0.10	0.10	0.10
1	0	-0.40	0.10	-0.30	0	0	0.60	-0.40	0.10	0.10	0.10
1	1	-0.40	0.10	-0.20	1	0	0.50	0.10	0.60	0.60	0.60
0	0	0.10	0.00	0.10	0	1	-0.50	-0.40	0.60	0.60	0.60
0	1	-0.40	0.00	-0.60	0.20	0	1	-0.50	-0.90	0.60	0.10
1	0	-0.90	0.60	-0.30	0	0	0.00	-0.90	0.60	0.10	0.10
1	1	-0.90	0.60	-0.20	1	0	0.50	-0.40	1.10	0.60	0.60
0	0	-0.40	0.00	-0.40	0	0	0.00	-0.40	1.10	0.60	0.60
0	1	-0.40	0.00	0.60	0.20	0	1	-0.50	-0.90	1.10	0.10
1	0	-0.90	1.10	-0.20	0	1	-0.50	-1.40	0.60	0.10	0.10
1	1	-1.40	0.60	-0.70	1	0	0.50	-0.90	1.10	0.60	0.60
0	0	-0.90	0.00	-0.90	0	0	0.00	-0.90	1.10	0.60	0.60
0	1	-0.90	0.00	1.10	0.20	0	1	-0.50	-1.40	1.10	0.60
1	0	-1.40	1.10	-0.30	0	0	0.00	-1.40	1.10	0.60	0.60
1	1	-1.40	1.10	0.60	0.30	1	1	0.00	-1.40	1.10	0.60
0	0	-1.40	0.00	-1.40	0	0	0.00	-1.40	1.10	0.60	0.60
0	1	-1.40	0.00	0.60	-0.80	0	0	0.00	-1.40	1.10	0.60
1	0	-1.40	1.10	0.00	-0.30	0	0	0.00	-1.40	1.10	0.60
1	1	-1.40	1.10	0.60	0.30	1	0	0.00	-1.40	1.10	0.60

ขั้นตอนแรกเริ่มจากการสุ่มค่า w_0 จนถึง w_2 ในที่นี้กำหนดให้เป็น 0.1 ทั้งสามตัว จากนั้นก็เริ่มป้อนด้วยอย่างเข้าไป (ทีละแก้ว) ด้วยอย่างแรกได้ผลรวมเริงเส้น (Net Sum) เป็น 0.10 ซึ่งมากกว่า 0 ดังนั้นเบอร์เซปตอร์จะให้อัตราต่อพุ่ดวิธี (Actual Output) ออกมาเป็น 1 ซึ่งผิดเพระอาจเป็นสาเหตุของการล้มเหลว (Target Output) จะต้องได้เป็น 0 ทำให้อัตราการเรียนรู้คูณค่าผิดพลาด ($\text{Alpha} \times \text{Error}$) ได้ -0.50 หลังจากนั้นนำไปปรับหนักตาม $w_i \leftarrow w_i + \Delta w_i$ และ $\Delta w_i \leftarrow \alpha(t-o)x_i$ ดังนั้นจะได้เป็น $w_0 \leftarrow w_0 + \alpha(t-o)x_0 = w_0 + 0.50(-1) \times 1 = 0.10 + (-0.5) = -0.4$ ต่อไปก็ปรับค่า w_1 ในท่านองเดียวกัน $w_1 \leftarrow w_1 + \alpha(t-o)x_1 = w_1 + 0.50(-1) \times 0 = 0.10 + 0.50(-1) \times 0 = 0.10$ จะเท่ากับ 0.10 คือไม่เปลี่ยนแปลง เช่นเดียวกับ w_2 ที่ไม่เมื่อยล้ายแปลง จะเห็นได้ว่าแม้มีค่าผิดพลาดแต่ไม่มีการปรับค่า w_1 และ w_2 เนื่องจากอินพุตที่ใส่เข้าไปเป็น 0 ทำ

กระบวนการต่อไปนี้เป็น
แบบเรียนรู้
ของเครื่องประดิษฐ์

ຕາງໆທີ່ 6-19 ຜລກາເຮັດວຽກກັບ AND ໂດຍກູກເຮັດວຽກນູ້ເພື່ອເປົ້າຕ່ອນ

Perceptron Learning Example - Function AND												
		Bias Input $x_0=+1$					Alpha = 0.5					
Input	Input				Net Sum	Target	Actual	Alpha*	Weight Values			
x_1	x_2	$1.0 * w_0$	$x_1 * w_1$	$x_2 * w_2$	Input	Output	Output	Error	w_0	w_1	w_2	
0	0	0.10	0.00	0.00	0.10	0	1	-0.50	-0.40	0.10	0.10	
0	1	-0.40	0.00	0.10	-0.30	0	0	0.00	-0.40	0.10	0.10	
1	0	-0.40	0.10	0.00	-0.30	0	0	0.00	-0.40	0.10	0.10	
1	1	-0.40	0.10	0.10	-0.20	1	0	0.50	0.10	0.60	0.60	
0	0	0.10	0.00	0.00	0.10	0	1	-0.50	-0.40	0.60	0.60	
0	1	-0.40	0.00	0.60	0.20	0	1	-0.50	-0.90	0.60	0.10	
1	0	-0.90	0.60	0.00	-0.30	0	0	0.00	-0.90	0.60	0.10	
1	1	-0.90	0.60	0.10	-0.20	1	0	0.50	-0.40	1.10	0.60	
0	0	-0.40	0.00	0.00	-0.40	0	0	0.00	-0.40	1.10	0.60	
0	1	-0.40	0.00	0.60	0.20	0	1	-0.50	-0.90	1.10	0.10	
1	0	-0.90	1.10	0.00	0.20	0	1	-0.50	-1.40	0.60	0.10	
1	1	-1.40	0.60	0.60	-0.70	1	0	0.50	-0.90	1.10	0.60	
0	0	-0.90	0.00	0.00	-0.90	0	0	0.00	-0.90	1.10	0.60	
0	1	-0.90	0.00	0.60	-0.30	0	0	0.00	-0.90	1.10	0.60	
1	0	-0.90	1.10	0.00	0.20	0	1	-0.50	-1.40	0.60	0.60	
1	1	-1.40	0.60	0.60	-0.20	1	0	0.50	-0.90	1.10	1.10	
0	0	-0.90	0.00	0.00	-0.90	0	0	0.00	-0.90	1.10	1.10	
0	1	-0.90	0.00	1.10	0.20	0	1	-0.50	-1.40	1.10	0.60	
1	0	-1.40	1.10	0.00	-0.30	0	0	0.00	-1.40	1.10	0.60	
1	1	-1.40	1.10	0.60	0.30	1	1	0.00	-1.40	1.10	0.60	
0	0	-1.40	0.00	0.00	-1.40	0	0	0.00	-1.40	1.10	0.60	
0	1	-1.40	0.00	0.60	-0.80	0	0	0.00	-1.40	1.10	0.60	
1	0	-1.40	1.10	0.00	-0.30	0	0	0.00	-1.40	1.10	0.60	
1	1	-1.40	1.10	0.60	0.30	1	1	0.00	-1.40	1.10	0.60	

ຂັ້ນດອນແຮງເຈັ້ມຈາກການສຸ່ມຄ່າ w_0 ຈະຄື່ງ w_1 ໃນທີ່ກໍາທັນດໄດ້ເປັນ 0.1 ທັ້ງສາມດ້ວຍ ຈາກນັ້ນ ກີ່ເຮັດວຽກດ້ວຍຢ່າງເຂົ້າໄປ (ກິລະແກວ) ດ້ວຍຢ່າງເຮັດວຽກໄດ້ຜລວມເຮີງເສັ້ນ (Net Sum) ເປັນ 0.10 ຜື້ນາກົກກ່າວ່າ 0 ດັ່ງນັ້ນເປົ້າຕ່ອນຈະໄຫ້ເຄົາຕຸດຈົງ (Actual Output) ອອກນາເປັນ 1 ຜື້ນົດເພຣະເຄົາຕຸດເປົ້າຫມາຍ (Target Output) ຈະຕ້ອງໄດ້ເປັນ 0 ທຳມະຊາດການເຮັດວຽກນູ້ຄູຄ່າ ພຶດພລາດ (Alpha x Error) ໄດ້ -0.50 ລ່ວງຈາກນີ້ນໍາໄປປັບປຸງໜັກດາມ $w_i \leftarrow w_i + \Delta w_i$ ແລະ $\Delta w_i \leftarrow \alpha(t-o)x_i$ ດັ່ງນັ້ນຈະໄດ້ເປັນ $w_0 \leftarrow w_0 + \alpha(t-o)x_0 = w_0 + 0.50(-1) \times 1 = 0.10 + (-0.5) = -0.4$ ຕ້ອງໄປກັບປົບຄ່າ w_1 ໃນທ່ານອອງເຕີຍກັບ $w_1 \leftarrow w_1 + \alpha(t-o)x_1 = w_1 + 0.50(-1) \times 0$ ດັ່ງນັ້ນ w_1 ຈະເຖິກກັບ 0.10 ຕີ້ໄມ່ເປັນແປງແປງ ເຊິ່ງເຕີຍກັບ w_2 ທີ່ໄມ່ເປັນແປງແປງ ຈະເທັນໄດ້ວ່າແນວໝັ້ນຄໍາພຶດພລາດແຕ່ໄໝມີການປັບຄ່າ w_1 ແລະ w_2 ເນື່ອງຈາກອິນພຸດທີ່ໄສເຂົ້າໄປເປັນ 0 ທໍາ

ຕາມາດທີ 6-19 ຜລກາເຮັດວຽກກັບ AND ໂດຍກູກາເຮັດວຽກນູ້ເພື່ອເປົ້າຕອນ

6 = Epoch

Perceptron Learning Example - Function AND										
		Bias Input $x_0=+1$			Alpha = 0.5	Actual	Alpha*	Weight Values	0.5 0.5 0.5	
Input	Input			Net Sum	Target	Output	Error	w0	w1	w2
x1	x2	$1.0 \cdot w_0$	$x_1 \cdot w_1$	$x_2 \cdot w_2$	Input	Output	Output	0.1	-0.4	0.1
0	0	0.10	0.00	0.00	0.10	0	1	-0.50	-0.40	0.10
0	1	-0.40	0.00	0.10	-0.30	0	0	0.00	-0.40	0.10
1	0	-0.40	0.10	0.00	-0.30	0	0	0.00	-0.40	0.10
1	1	-0.40	0.10	0.10	-0.20	1	0	0.50	0.40	0.60
0	0	0.10	0.00	0.00	0.10	0	1	-0.50	-0.40	0.60
0	1	-0.40	0.00	0.60	0.20	0	1	-0.50	-0.90	0.60
1	0	-0.90	0.60	0.00	-0.30	0	0	0.00	-0.90	0.60
1	1	-0.90	0.60	0.10	-0.20	1	0	0.50	-0.40	1.10
0	0	-0.40	0.00	0.00	-0.40	0	0	0.00	-0.40	1.10
0	1	-0.40	0.00	0.60	0.20	0	1	-0.50	-0.90	1.10
1	0	-0.90	1.10	0.00	0.20	0	1	-0.50	-1.40	0.60
1	1	-1.40	0.60	0.60	-0.20	1	0	0.50	-0.90	1.10
0	0	-0.90	0.00	0.00	-0.90	0	0	0.00	-0.90	1.10
0	1	-0.90	0.00	1.10	0.20	0	1	-0.50	-1.40	1.10
1	0	-1.40	1.10	0.00	-0.30	0	0	0.00	-1.40	1.10
1	1	-1.40	1.10	0.60	0.30	1	0	0.50	-1.40	1.10
0	0	-1.40	0.00	0.00	-1.40	0	0	0.00	-1.40	1.10
0	1	-1.40	0.00	0.60	-0.80	0	0	0.00	-1.40	1.10
1	0	-1.40	1.10	0.00	-0.30	0	0	0.00	-1.40	1.10
1	1	-1.40	1.10	0.60	0.30	1	1	0.00	-1.40	1.10

ຂັ້ນດອນແຮງເຈັ້ມຈາກການສຸ່ນຄໍາ w_0 ຈະເຄີ່ງ w_2 ໃນທີ່ນີ້ກ່າວກັດໄດ້ເປັນ 0.1 ທັງສາມດ້ວຍ ຈາກນັ້ນ ກີ່ເຮັດວຽກດ້ວຍຢ່າງເຂົາໄປ (ກີ່ລະແກວ) ດ້ວຍຢ່າງເຮັດວຽກໄດ້ພລວມເຮີງເສັ້ນ (Net Sum) ເປັນ 0.10 ຊື່ມາກວ່າ 0 ດັ່ງນັ້ນເປົ້າຕົວອະນະໄຫ້ເອົາຕຸດຈຸວິງ (Actual Output) ອອກນາເປັນ 1 ຊື່ຜິດເພຣະເຄົ້າຕຸດເປົ້າຫມາຍ (Target Output) ຈະຕ້ອງໄດ້ເປັນ 0 ທໍາທີ່ອັດຕາການເຮັດວຽກນູ້ຄຸນຄໍາ ຜິດພລາດ ($\text{Alpha} \times \text{Error}$) ໄດ້ -0.50 ລ່ວງຈາກນີ້ນໍາໄປປັບນ້ຳໜັກດາມ $w_i \leftarrow w_i + \Delta w_i$ ແລະ $\Delta w_i \leftarrow \alpha(t-o)x_i$ ດັ່ງນັ້ນຈະໄດ້ເປັນ $w_0 \leftarrow w_0 + \alpha(t-o)x_0 = w_0 + 0.50(-1) \times 1 = 0.10 + (-0.5) = -0.4$ ຕ້ອງໄປກັບຄໍາ w_1 ໃນກ່າວອອງເຕີຍກັນ $w_1 \leftarrow w_1 + \alpha(t-o)x_1 = w_1 + 0.50(-1) \times 0 = 0.10 + 0.50(-1) \times 0 = 0.10$ ຈະເຖິງກັນ w_1 ຈະເຖິງກັນ 0.10 ຕື່ອມໄປເປັນແປງ ເຊິ່ງເຕີຍກັນ w_2 ທີ່ໄໝເປັນແປງ ຈະເທັນໄດ້ ວ່າແມ່ນຄໍາຜິດພລາດແຕ່ໄໝມີການປັບຄໍາ w_1 ແລະ w_2 ເນື່ອງຈາກອິນພຸດທີ່ໄສເຂົາໄປເປັນ 0 ທໍາ

data 01
ກວດກຳກົດ
Decision

176 ปัญญาประดิษฐ์

เวลาชั้น 1.0.2: 15 มี.ค.2548 : 9:05 PM boonserm.k@chula.ac.th

ให้ผลคูณเป็น 0 ซึ่งไม่ได้ปรับ และเป็นขอเสียงของฟังก์ชันกระดุมแบบบินารีซึ่งถ้าผลลูกมาเป็น 0 จะไม่มีการปรับค่าให้ (ถ้าเราเปลี่ยน 0 เป็น -1 การปรับค่าจะตีข้อ w_i จะถูกปรับหันให้ตื้นๆต่อรอบแรก)

ด้วยอย่างที่สองนี้ด้วยที่สำคัญที่สุดคือการอ่านค่าเดียวทุกepoch แล้วเมื่อทำการวน 1 รอบการสอน (epoch) และจะต้องทำการสอนซ้ำด้วยข้อมูลชุดเดิม นี่คือวิธีการสอนของภ่ายงานประสาทเทียมซึ่งต่างจากวิธีอื่นๆ ที่ต้องใช้ข้อมูลชุดเดิมสอนซ้ำไปจนกว่าจะตั้งค่าพิเศษลดลงจนถึงจุดที่เราต้องการ ในที่นี้คือ 0 เนื่องจากเราต้องการให้มีการแบ่งข้อมูลอย่างเด็ดขาด และสมการเส้นตรงที่ได้จะมีค่า $w_0 = -1.40$, $w_1 = 1.10$ และ $w_2 = 0.60$

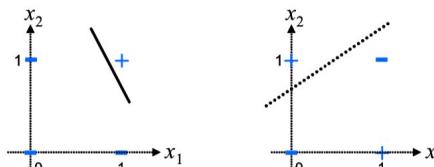
ฟังก์ชันที่สองที่จะทดลองเรียนรู้ด้วยภ่ายการเรียนรู้เพอร์เซปตรอนคือฟังก์ชัน XOR แสดงในตารางที่ 6-20

ตารางที่ 6-20 ฟังก์ชัน XOR(x_1, x_2)

x_1	x_2	เอาต์พุตเป้าหมาย
0	0	0
0	1	1
1	0	1
1	1	0

ฟังก์ชัน XOR ตามตารางข้างบนนี้จะได้ค่าที่เป็นจริงก็ต่อเมื่อ x_1 หรือ x_2 ตัวใดตัวหนึ่งเพียงตัวเดียวเป็นจริง (ถ้าที่สอนก็เอาต์พุตเป้าหมาย) ผลการใช้ภ่ายการเรียนรู้เพอร์เซปตรอนกับฟังก์ชัน XOR แสดงในตารางที่ 6-21

ในการนี้ของฟังก์ชัน XOR นี้พบว่าค่าพิเศษลดลง แต่ค่าหักจะแก่วงไปทางโดยไม่ลุบข้ามไว้จะสอนต่อจากนี้ไปอีกที่รวมการสอนก็ตาม จึงสรุปว่าฟังก์ชัน XOR เรียนไม่สำเร็จด้วยเพอร์เซปตรอน เมื่อเราใช้ฟังก์ชัน AND และ XOR ไปคาดการณ์ในสองมิติจะได้กราฟดังรูปที่ 6-37



(ก) ฟังก์ชันแยกเชิงเส้นได้ (AND)

(ข) ฟังก์ชันแยกเชิงเส้นไม่ได้ (XOR)

รูปที่ 6-37 ฟังก์ชันแยกเชิงเส้นได้และไม่ได้

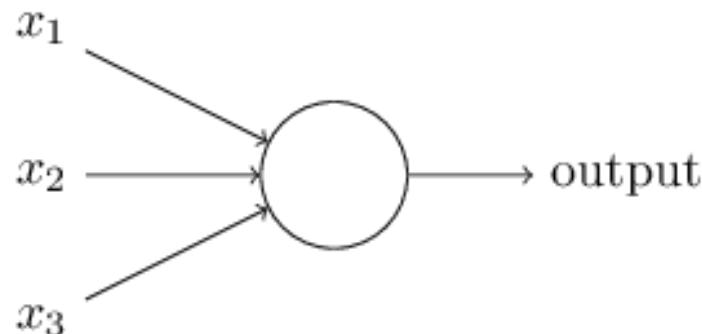
ตารางที่ 6-21 ผลการเรียนรู้ฟังก์ชัน XOR โดยกฎการเรียนรู้เพอร์เซปตรอน

Perceptron Learning Example: XOR												
Input	Input	Bias Input X0=1			Net Sum	Target	Actual	Alpha*	Weight Values			
		x1	x2	1.0*w0	x1*w1	x2*w2	Input	Output	Error	w0	w1	w2
0	0	0.10	0.00	0.00	0.10	0	1	-0.50	-0.40	0.10	0.10	0.10
0	1	-0.40	0.00	0.10	-0.30	1	0	0.50	0.10	0.10	0.60	
1	0	0.10	0.10	0.00	0.20	1	1	0.00	0.10	0.10	0.60	
1	1	0.10	0.10	0.60	0.80	0	1	-0.50	-0.40	-0.40	0.10	
0	0	-0.40	0.00	0.00	-0.40	0	0	0.00	-0.40	-0.40	0.10	
0	1	-0.40	0.00	0.10	-0.30	1	0	0.50	0.10	-0.40	0.60	
1	0	0.10	-0.40	0.00	-0.30	1	0	0.50	0.60	0.10	0.60	
1	1	0.60	0.10	0.60	1.30	0	1	-0.50	0.10	-0.40	0.10	
0	0	0.10	0.00	0.00	0.10	0	1	-0.50	-0.40	-0.40	0.10	
0	1	-0.40	0.00	0.10	-0.30	1	0	0.50	0.10	-0.40	0.60	
1	0	0.10	-0.40	0.00	-0.30	1	0	0.50	0.60	0.10	0.60	
1	1	0.60	0.10	0.60	1.30	0	1	-0.50	0.10	-0.40	0.10	
0	0	0.10	0.00	0.00	0.10	0	1	-0.50	-0.40	-0.40	0.10	
0	1	-0.40	0.00	0.10	-0.30	1	0	0.50	0.10	-0.40	0.60	
1	0	0.10	-0.40	0.00	-0.30	1	0	0.50	0.60	0.10	0.60	
1	1	0.60	0.10	0.60	1.30	0	1	-0.50	0.10	-0.40	0.10	
0	0	0.10	0.00	0.00	0.10	0	1	-0.50	-0.40	-0.40	0.10	
0	1	-0.40	0.00	0.10	-0.30	1	0	0.50	0.10	-0.40	0.60	
1	0	0.10	-0.40	0.00	-0.30	1	0	0.50	0.60	0.10	0.60	
1	1	0.60	0.10	0.60	1.30	0	1	-0.50	0.10	-0.40	0.10	
0	0	0.10	0.00	0.00	0.10	0	1	-0.50	-0.40	-0.40	0.10	
0	1	-0.40	0.00	0.10	-0.30	1	0	0.50	0.10	-0.40	0.60	
1	0	0.10	-0.40	0.00	-0.30	1	0	0.50	0.60	0.10	0.60	
1	1	0.60	0.10	0.60	1.30	0	1	-0.50	0.10	-0.40	0.10	
0	0	0.10	0.00	0.00	0.10	0	1	-0.50	-0.40	-0.40	0.10	
0	1	-0.40	0.00	0.10	-0.30	1	0	0.50	0.10	-0.40	0.60	
1	0	0.10	-0.40	0.00	-0.30	1	0	0.50	0.60	0.10	0.60	
1	1	0.60	0.10	0.60	1.30	0	1	-0.50	0.10	-0.40	0.10	
0	0	0.10	0.00	0.00	0.10	0	1	-0.50	-0.40	-0.40	0.10	
0	1	-0.40	0.00	0.10	-0.30	1	0	0.50	0.10	-0.40	0.60	
1	0	0.10	-0.40	0.00	-0.30	1	0	0.50	0.60	0.10	0.60	
1	1	0.60	0.10	0.60	1.30	0	1	-0.50	0.10	-0.40	0.10	
0	0	0.10	0.00	0.00	0.10	0	1	-0.50	-0.40	-0.40	0.10	
0	1	-0.40	0.00	0.10	-0.30	1	0	0.50	0.10	-0.40	0.60	
1	0	0.10	-0.40	0.00	-0.30	1	0	0.50	0.60	0.10	0.60	
1	1	0.60	0.10	0.60	1.30	0	1	-0.50	0.10	-0.40	0.10	

จากข้อจะเห็นได้ว่าฟังก์ชัน AND เป็นฟังก์ชันที่แยก (ระหว่างตัวอย่างบวกกับตัวอย่างลบ) ได้ด้วยเส้นตรง ส่วนฟังก์ชัน XOR เรายังสามารถหาเส้นตรงที่ไม่แบ่งตัวอย่างบวกและลบออกจากกัน (ไม่สามารถถูกเลี้ยงเส้นตรงให้ตัวอย่างบวกและลบให้อยู่คนละด้านของเส้น) ตัวอย่างการเรียนรู้ฟังก์ชัน XOR ข้างต้นให้แสดงให้เห็นว่า เพอร์เซปตรอนเรียนรู้บวกฟังก์ชันไม่ได้ ฟังก์ชันเหล่านี้เรียกว่า **ฟังก์ชันแยกเชิงเส้นไม่ได้ (linearly non-separable function)** ส่วนฟังก์ชันที่แยกได้เรียกว่า **ฟังก์ชันแยกเชิงเส้นได้ (linearly separable function)** ซึ่งเป็นข้อจำกัดของเพอร์เซปตรอน เมื่อเราลองกลับไปดูตารางที่ 6-21 จะพบว่า นอกจากการเรียนรู้ฟังก์ชัน XOR ไม่สำเร็จแล้ว การเรียนรู้ก็จะไม่สำเร็จสักเส้นตรงได้เส้นตรงหนึ่งก็ตัวเดียว ดังจะเห็นได้จากการที่เวกเตอร์น้ำหนักจะแก่วงไปมา การไม่สำเร็จก็อีกเกิดบัญหาในการเรียนรู้ เพราะเราจะไม่รู้ว่าเมื่อไรจะหยุดอัลกอริทึม พิจารณาตัวอย่างใน

ฟังก์ชันแยก
เส้นไม่ได้

Perceptron: Predecessor of a Neural Network



$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

- The perceptron algorithm: invented in 1957 by Frank Rosenblatt
- Input: An n -dimensional input vector \mathbf{x} (with n variables)
- Output: 1 or 0 depending on if the weighted sum passes a threshold
- Perceptron: A device that makes decisions by weighing up evidence
- Often written in the vector form, using bias (b) instead of threshold, as

$$\text{output} = \begin{cases} 0 & \text{if } \mathbf{w} \cdot \mathbf{x} + b \leq 0 \\ 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \end{cases}$$

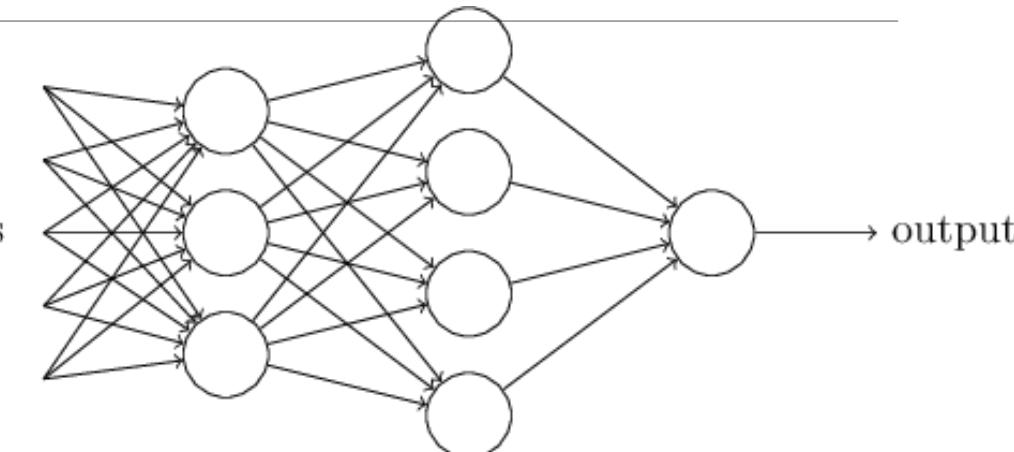
Bias: a measure of how easy it is to get the perceptron to output a 1

Sigmoid Neurons

- A many-layer network of perceptrons can engage in sophisticated decision making
- Instead of assigning weights of the edges by a person, we can devise *learning algorithms* that can automatically tune the weights and biases of a network of artificial neurons
- Use sigmoid neuron instead of perceptron: Output is not 0/1 but a sigmoid function: $\sigma(w \bullet x + b)$, i.e.,
- The smoothness of σ means that small changes in the Δw_j weights and in the Δb bias will produce a small change Δ_{output} in the output from the neuron

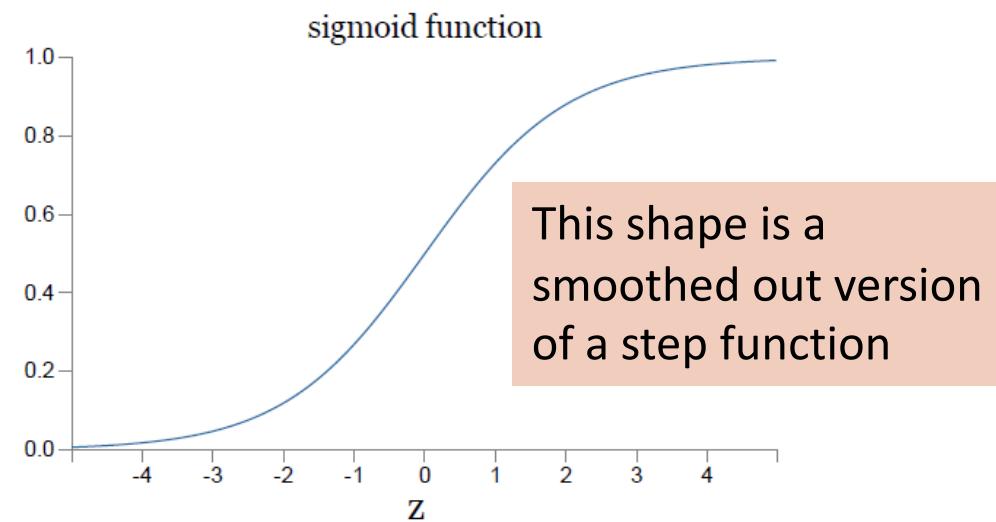
$$\Delta_{\text{output}} \approx \sum_j \frac{\partial \text{output}}{\partial w_j} \Delta w_j + \frac{\partial \text{output}}{\partial b} \Delta b$$

i.e., Δ_{output} is a *linear function* of the changes Δw_j and Δb



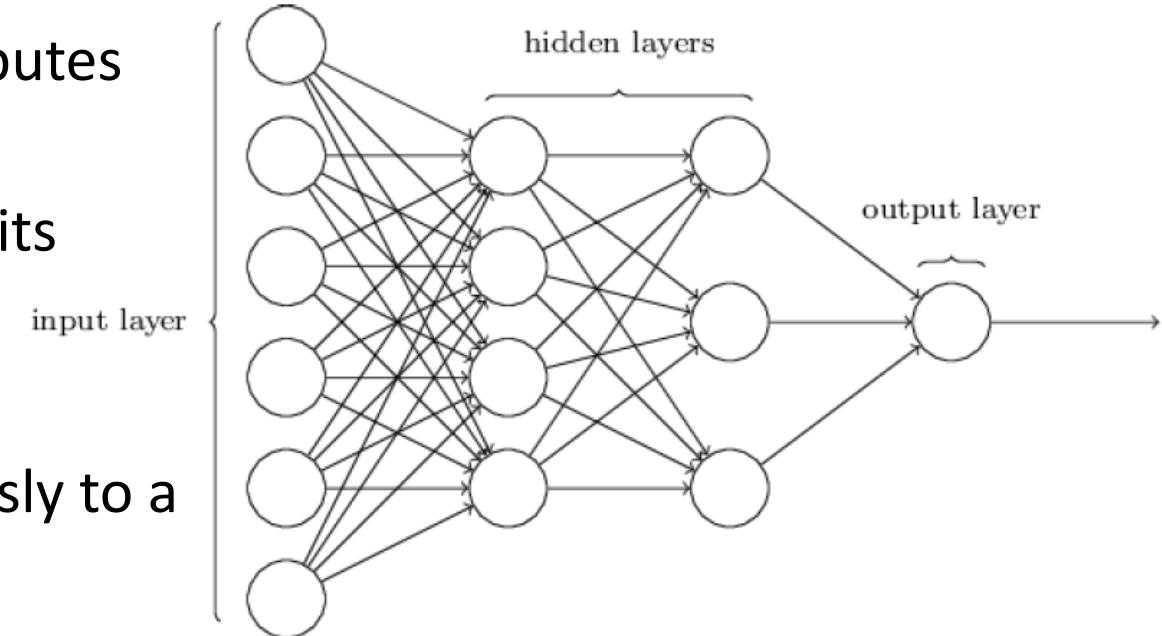
Sigmoid function: $\sigma(z) \equiv \frac{1}{1 + e^{-z}}$

$$\frac{1}{1 + \exp(-\sum_j w_j x_j - b)}$$



Architecture of a (Feed-Forward) Neural Network (NN)

- **Input layer**
 - The **inputs** to NN correspond to the attributes measured for each training tuple
 - Inputs are fed simultaneously into the units making up the **input layer**
- **Hidden layer(s)**
 - Inputs are weighted and fed simultaneously to a hidden layer
 - The number of hidden layers is arbitrary
- **Output layer**
 - The weighted outputs of the last hidden layer are input to units making up the **output layer**, which emits the network's prediction



Neural Network Architecture: Feed-Forward vs. Recurrent

- **Feed-Forward Neural Network:** Typical neural network architecture
 - The output from one layer is used as input to the next layer (no loops)
 - Information is always fed forward, never fed back
 - From a statistical point of view, networks perform **nonlinear regression**
 - Given enough hidden units and enough training samples, they can closely approximate any function
- **Recurrent neural network:** Feedback loops are possible (cascade of neurons firing)
 - Some neurons fire for some limited duration of time, before becoming quiescent
 - That firing can stimulate other neurons, which may fire a little while later, also for a limited duration, which causes still more neurons to fire, and so on
 - Loops do not cause problems since a neuron's output only affects its input at some later time, not instantaneously

Learning with Gradient Descent

- A quadratic cost (objective) function C (or mean square error, MSE)

$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2$$

where w : the collection of all weights in the network, b : all the biases, n : the total # of training inputs, a : the vector of outputs from the network when x is input

- Goal of training a network: Find weights and biases which minimize the cost $C(w, b)$

- That is, choose Δv_1 and Δv_2 to make ΔC negative; i.e., the ball is rolling down into the valley:

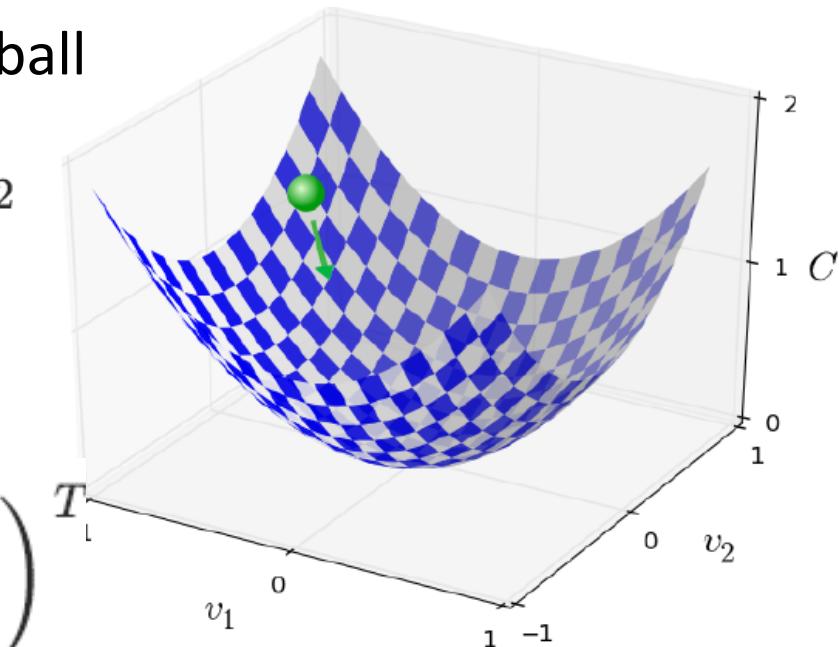
$$\Delta C \approx \frac{\partial C}{\partial v_1} \Delta v_1 + \frac{\partial C}{\partial v_2} \Delta v_2$$

- The change ΔC in C by a small change in v , Δv :

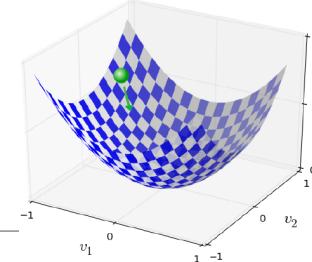
$$\Delta C \approx \nabla C \cdot \Delta v$$

where ∇C is the gradient vector:

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \dots, \frac{\partial C}{\partial v_m} \right)^T$$



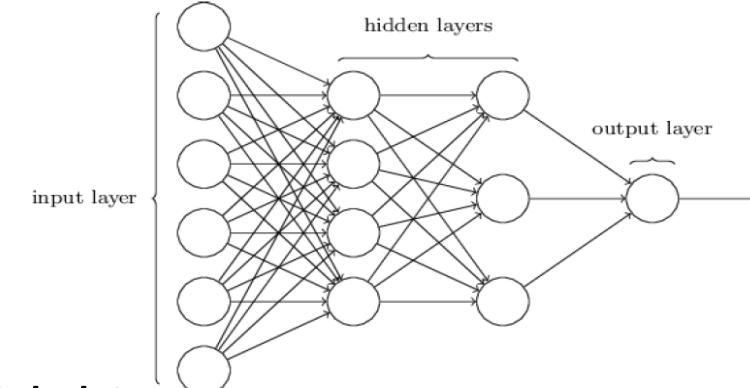
Stochastic Gradient Descent



- Gradient descent can be viewed as a way of taking small steps in the direction which does the most to immediately decrease C
- To compute gradient ∇C , we need to compute the gradients ∇C_x separately for each training input, x , and then average them: slow when the # of training inputs is large
- *Stochastic gradient descent (SGD)*: Speed up learning
 - Computing for a small sample of randomly chosen training inputs and *averaging over them*, we can quickly get a good estimate of the true gradient
 - Method: Randomly pick out a small number (**mini-batch**) m of randomly chosen training inputs. Provided the sample size is large enough, we expect that the average value will be roughly equal to the average over all, that is, $\nabla C \approx \frac{1}{m} \sum_{j=1}^m \nabla C_{X_j}$
- Stochastic gradient descent in neural networks:
 - Pick out a randomly chosen minibatch of training inputs and train with them; then pick out another minibatch, until inputs exhausted—complete an *epoch* of training
 - Then we start over with a new training epoch

Backpropagation for Fast Gradient Computation

- **Backpropagation:** Reset weights on the “front” neural units and this is sometimes done in combination with training where the correct result is known
- 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 back 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.)



More on Backpropagation

- With backpropagation, we distribute the “blame” backward through the network
 - Each hidden node sending input to the current node is somewhat “responsible” for some portion of the error in each neuron to which it has forward connection
- Local minima and backpropagation
 - Backpropagation can be stuck at local minima
 - But in practice it generally performs well
- Is backpropagation too slow?
 - Historically, backpropagation has been considered slow
 - Recent advances in computer power through parallelism and GPUs (graphics processing units) have reduced time substantially for training neural networks

From Neural Networks to Deep Learning

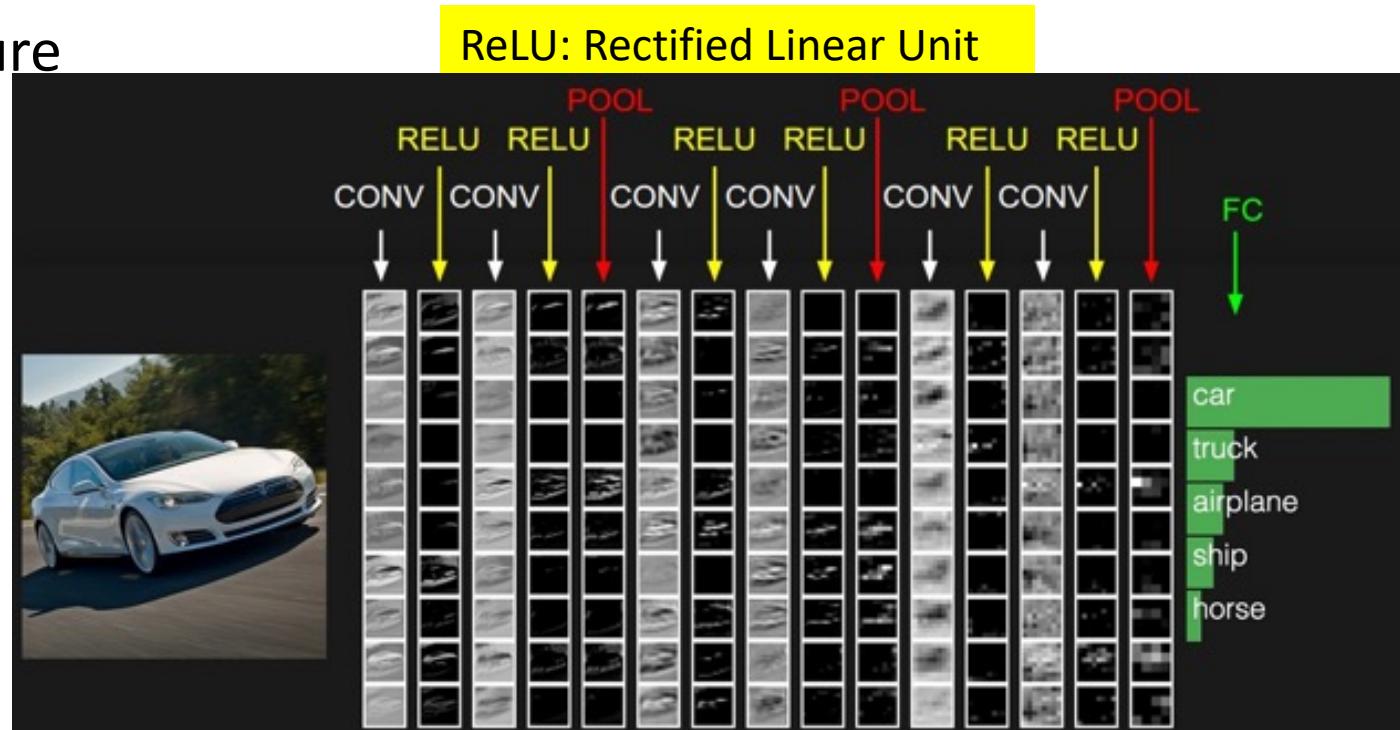
- ❑ Train networks with many layers (vs. shallow nets with just a couple of layers)
 - ❑ More neurons than previous networks
 - ❑ More complex ways to connect layers
 - ❑ Tremendous computing power to train networks
 - ❑ Automatic feature extraction
- ❑ Multiple layers work to build an improved feature space
 - ❑ Analogy: Signals passing through regions of the visual cortex
 - ❑ Example: For face recognition: edge → nose → face, layer-by-layer
- ❑ Popular Deep Learning Frameworks for Classification
 - ❑ Deep Feedforward Neural Networks
 - ❑ Convolutional Neural Networks
 - ❑ Recurrent Neural Networks

Deep (Feed Forward) Neural Networks

- How multiple layers work to build an improved feature space?
 - First layer learns 1st order features (e.g., edges, ...)
 - 2nd layer learns higher order features (combinations of first layer features, combinations of edges, etc.)
 - In Deep Belief Networks (DBNs), layers often learn in an unsupervised mode and discover general features of the input space—serving multiple tasks related to the unsupervised instances (image recognition, etc.)
 - Then final layer features are fed into supervised layer(s)
 - And entire network is often subsequently tuned using supervised training of the entire net, using the initial weightings learned in the unsupervised phase
 - Could also do fully supervised versions (back-propagation)

Convolutional Neural Networks: General Architecture

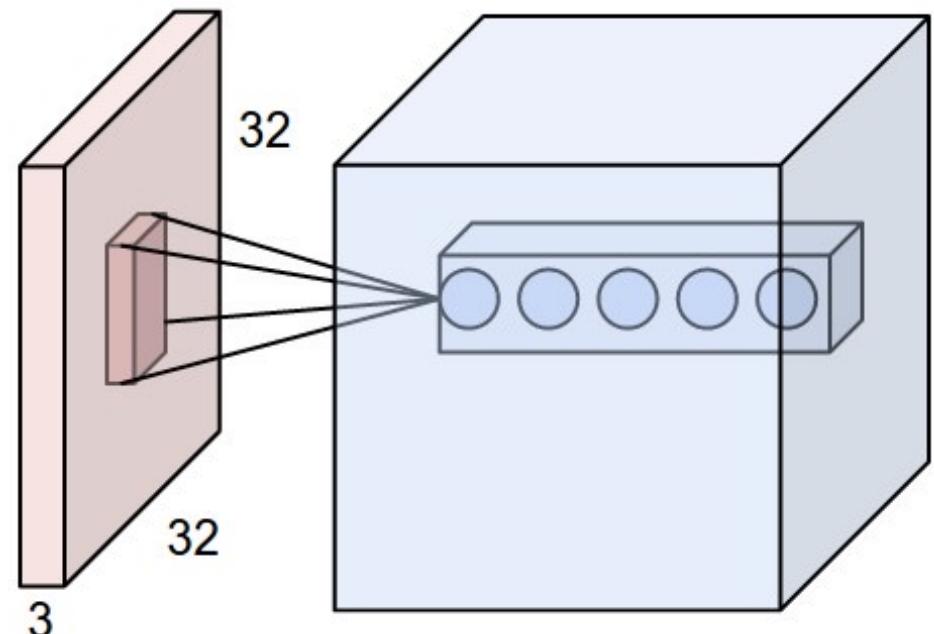
- Learn high-order features in the data via convolutions
 - Well suited to object recognition with images (e.g., computer vision)
 - Build position- and (somewhat) rotation-invariant features from raw image data
- CNN leverages learnable visual filters and globally shared local features
 - Specifics: high dimensional, 2D topology of pixels, invariance to translations, etc.
- High-level general CNN architecture
 - Input layer
 - Feature-extraction layers
(Convolution—ReLU—Pool)
 - Classification layers
- CNN properties
 - Local connectivity
 - Parameter sharing
 - Subsampling



Convolutional Neural Networks: Local Connectivity

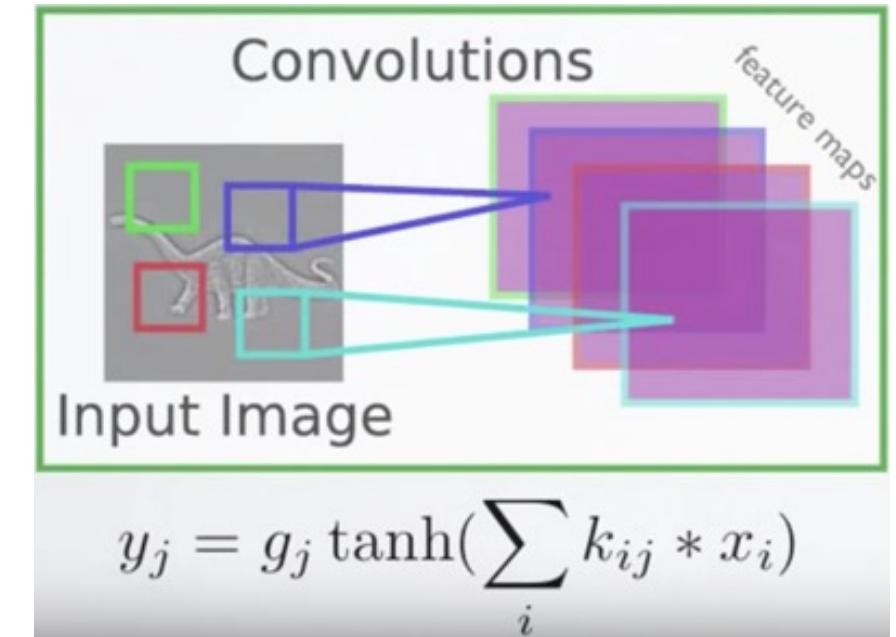
- Local Connectivity
 - Receptive fields: Each hidden unit is connected only to a sub-region of the image
 - Manageable number of parameters
 - Efficient computation of pre-activation
- Spatial arrangements
 - Depth: Number of filters
 - Stride: how to slide the filter
 - Zero-padding: deal with the border

$$(x * k)_{ij} = \sum_{pq} x_{i+p,j+q} k_{r-p,r-q}$$



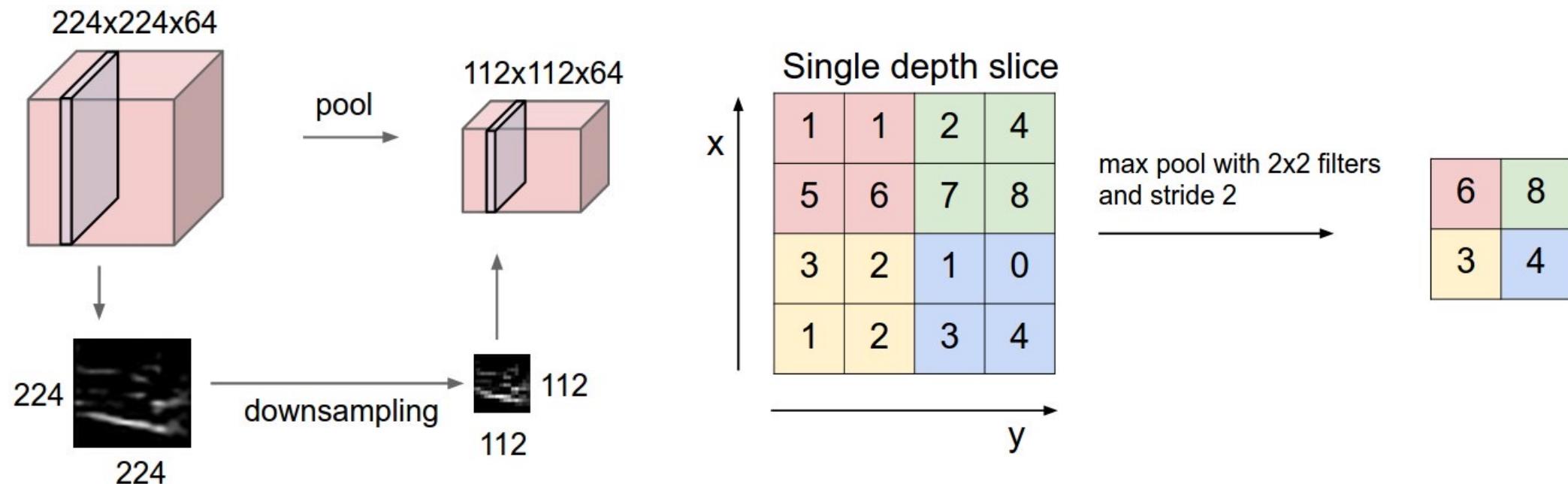
Convolutional Neural Networks: Parameter Sharing

- ❑ Parameter sharing
 - ❑ Discrete convolution: share matrix of parameters across certain units
 - ❑ Reduces even more the number of parameters
 - ❑ Extract the same feature at every position



Convolutional Neural Networks: Subsampling

- ❑ Subsampling
 - ❑ Pooling: pool hidden units in the same neighborhood
 - ❑ Introduces invariance to local translations
 - ❑ Reduces the number of hidden units in hidden layer



Recurrent Neural Networks: General Concepts

- Modeling the time dimension: by creating cycles in the network (thus “recurrent”)
 - Adding feedback loops connected to past decisions
 - Long-term dependencies: Use hidden states to preserve sequential information
- RNNs are trained to generate sequences: Output at each timestamp is based on both the current input and the inputs at all previous timestamps

$$\mathbf{h}_t = \phi(W\mathbf{x}_t + U\mathbf{h}_{t-1}),$$

- Compute a gradient with the algorithm BPTT (backpropagation through time)
- Major obstacles of RNN: Vanishing and Exploding Gradients
 - When the gradient becomes too large or too small, it is difficult to model long-range dependencies (10 timestamps or more)
 - Solution: Use a variant of RNN: LSTM (1997, by Hochreiter and Schmidhuber)

LSTM: One Variant of Recurrent Neural Network

- Critical components of LSTM

- Memory cells

- 3 Gates (input, forget, output)

- Use gated cells to

- Write, store, forget information

- When both gates are closed

- The contents of the memory cell will remain unmodified

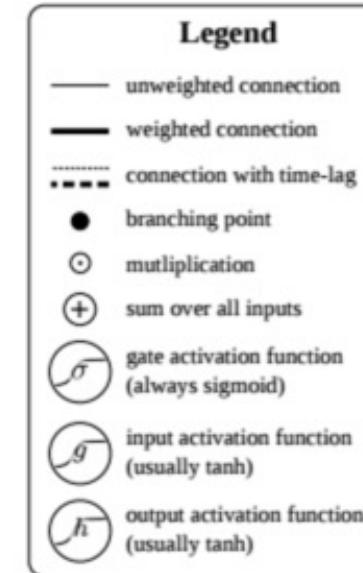
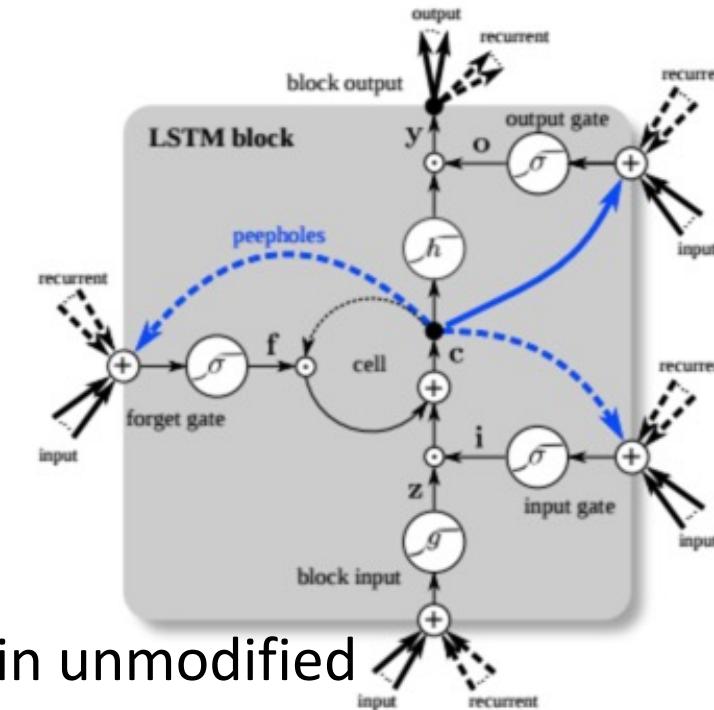
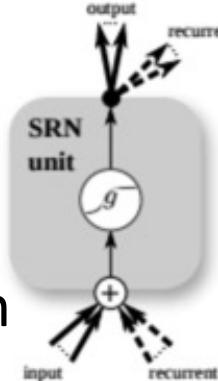
- The gating structure allows information to be retained across many timestamps

- Also allows gradient to flow across many timestamps

- By back-propagating errors and adjusting weights, to learn what to store, and when to allow reads, writes and erasures

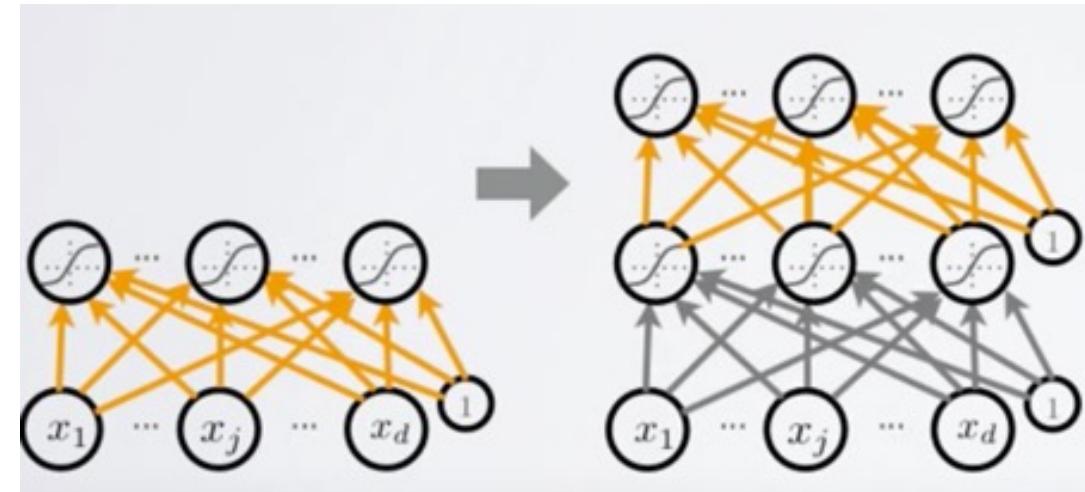
- Applications: Handling sequence and time series data

- E.g., NLP, video analysis, image captioning, robotics control



Difficulties of Training and Improvements

- ❑ Vanishing gradient problem: Saturated units block gradient propagation
 - ❑ Need better optimization (than SGD)
- ❑ Overfitting: high variance/low bias situation
 - ❑ Better regularization (than L1, L2 norm)
- ❑ Unsupervised pre-training
 - ❑ Statistical dropout
 - ❑ Other popular approaches
 - ❑ Batch normalization, residual networks, highway networks, attention, etc.



Pre-training of stacked autoencoders

Chapter 9. Classification: Advanced Methods

- Bayesian Belief Networks
- Support Vector Machines
- Neural Networks and Deep Learning
- Pattern-Based Classification
- Lazy Learners and K-Nearest Neighbors
- Other Classification Methods
- Summary



Using IF-THEN Rules for Classification

- Represent the knowledge in the form of **IF-THEN** rules

R_1 : 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_1

- n_{correct} = # of tuples correctly classified by R_1

$\text{coverage}(R_1) = n_{\text{covers}} / |D|$ /* D: training data set */

$\text{accuracy}(R_1) = n_{\text{correct}} / n_{\text{covers}}$

- If more than one rule are triggered, need **conflict resolution**

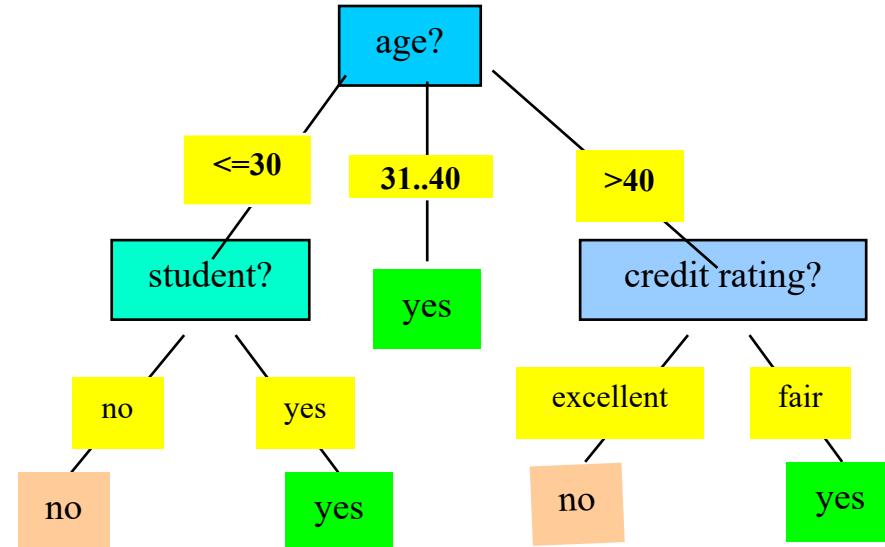
- **Size ordering**: assign the highest priority to the triggering rules that has the “toughest” requirement (i.e., with the *most attribute tests*)

- **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

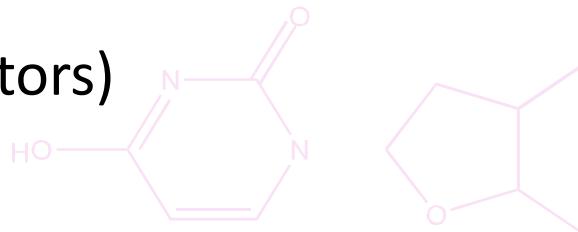
Rule Induction: Sequential Covering Method

- Sequential covering algorithm: Extracts rules directly from training data
- Typical sequential covering algorithms: FOIL, AQ, CN2, RIPPER
- Rules are learned *sequentially*, each for a given class C_i will cover many tuples of C_i but none (or few) of the tuples of other classes
- Steps:
 - Rules are learned one at a time
 - Each time a rule is learned, the tuples covered by the rules are removed
 - Repeat the process on the remaining tuples until *termination condition*, e.g., when no more training examples or when the quality of a rule returned is below a user-specified threshold
- Comp. w. decision-tree induction: learning a set of rules *simultaneously*

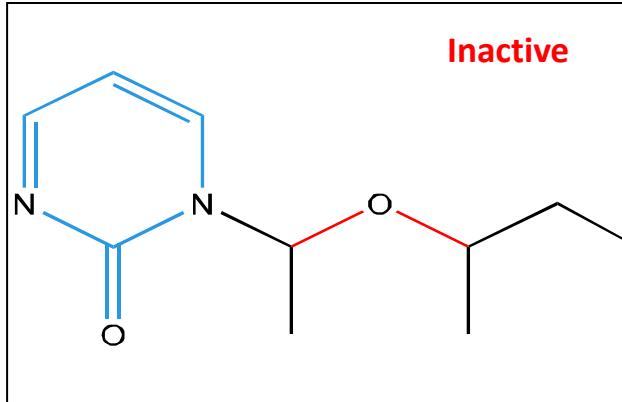
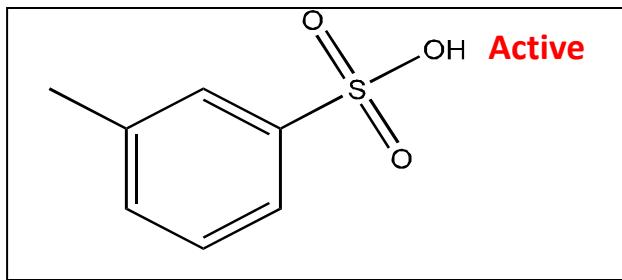
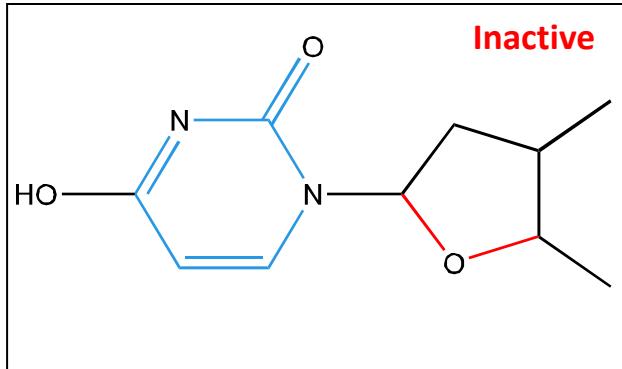
Pattern-Based Classification, Why?



- **Pattern-based classification:** An integration of both themes
- **Why pattern-based classification?**
 - **Feature construction**
 - Higher order; compact; discriminative
 - E.g., single word → phrase (Apple pie, Apple i-pad)
 - **Complex data modeling**
 - Graphs (no predefined feature vectors)
 - Sequences
 - Semi-structured/unstructured Data

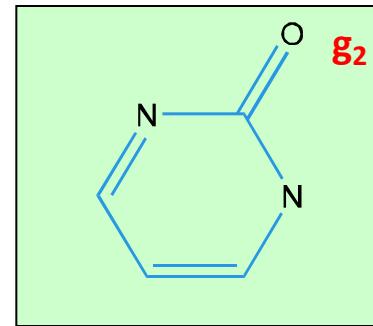
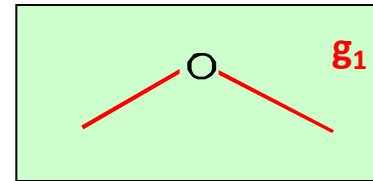


Pattern-Based Classification on Graphs



Mining
 $\text{min_sup}=2$

Frequent subgraphs



Transform

Use frequent patterns as features for classification

g_1	g_2	Class
1	1	0
0	0	1
1	1	0

Associative or Pattern-Based Classification

- **Data:** Transactions, microarray data, ... → **Patterns or association rules**
- **Classification Methods** (Some interesting work):
 - CBA [Liu, Hsu & Ma, KDD'98]: Use high-conf., high-support *class association rules* to build classifiers To be discussed here
 - Emerging patterns [Dong & Li, KDD'99]: Patterns whose support changes significantly between the two classes
 - CMAR [Li, Han & Pei, ICDM'01]: Multiple rules in prediction To be discussed here
 - CPAR [Yin & Han, SDM'03]: Beam search on multiple prediction rules
 - RCBT [Cong et al., SIGMOD'05]: Build classifier based on mining top-k covering rule groups with row enumeration (for high-dimensional data)
 - Lazy classifier [Veloso, Meira & Zaki, ICDM'06]: For a test t , project training data D on t , mine rules from D_t , predict on the best rule
 - Discriminative pattern-based classification [Cheng et al., ICDE'07] To be discussed here

CBA: Classification Based on Associations

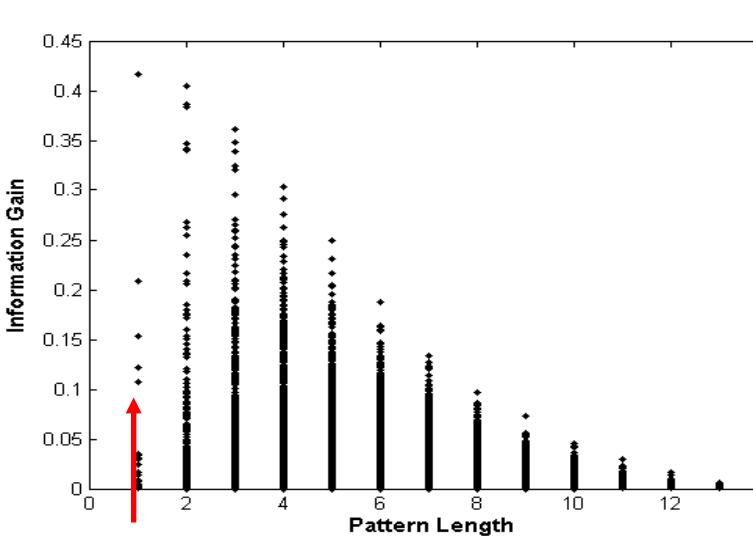
- CBA [Liu, Hsu and Ma, KDD'98]
- Method
 - Mine high-confidence, high-support class association rules
 - LHS: conjunctions of attribute-value pairs); RHS: class labels
 $p_1 \wedge p_2 \dots \wedge p_l \rightarrow "A_{\text{class-label}} = C"$ (confidence, support)
 - Rank rules in descending order of confidence and support
 - Classification: Apply the first rule that matches a test case; o.w. apply the default rule
 - Effectiveness: Often found more accurate than some traditional classification methods, such as C4.5
 - Why? — Exploring high confident associations among multiple attributes may overcome some constraints introduced by some classifiers that consider only one attribute at a time

Discriminative Pattern-Based Classification

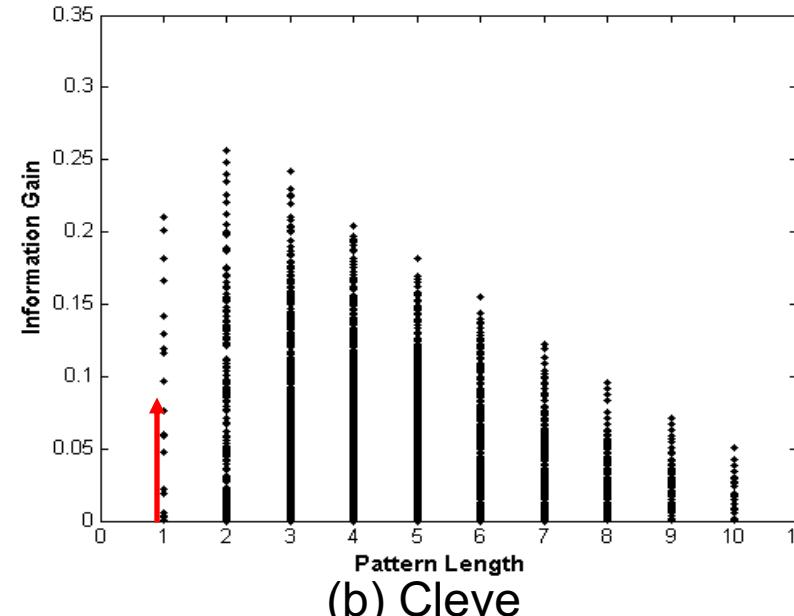
- Discriminative patterns as features for classification [Cheng et al., ICDE'07]
- **Principle:** Mining discriminative frequent patterns as high-quality features and then apply any classifier
- **Framework (PatClass)**
 - Feature construction by *frequent itemset mining*
 - Feature selection (e.g., using **Maximal Marginal Relevance (MMR)**)
 - Select discriminative features (i.e., that are relevant but minimally similar to the previously selected ones)
 - Remove redundant or closely correlated features
 - Model learning
 - Apply a general classifier, such as SVM or C4.5, to build a classification model

On the Power of Discriminative Patterns

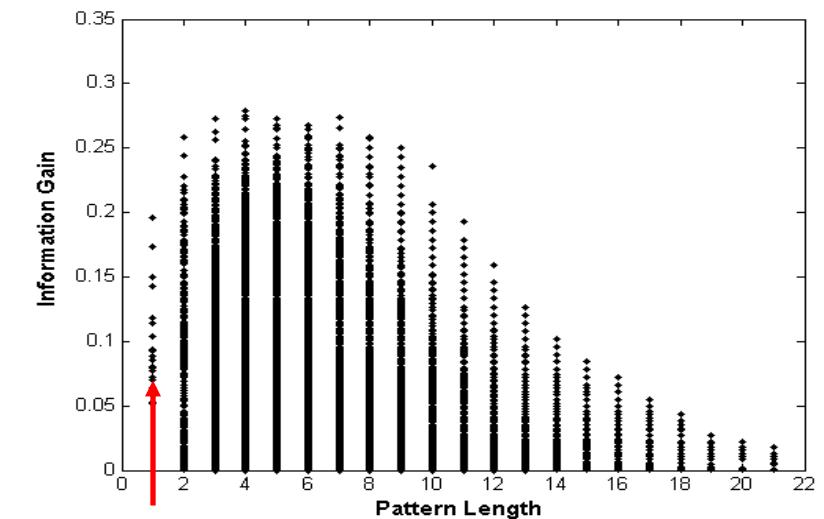
- K-itemsets are often more informative than single features (1-itemsets) in classification
- Computation on real datasets shows: The discriminative power of k-itemsets (for $k > 1$ but often ≤ 10) is higher than that of single features



(a) Austral



(b) Cleve

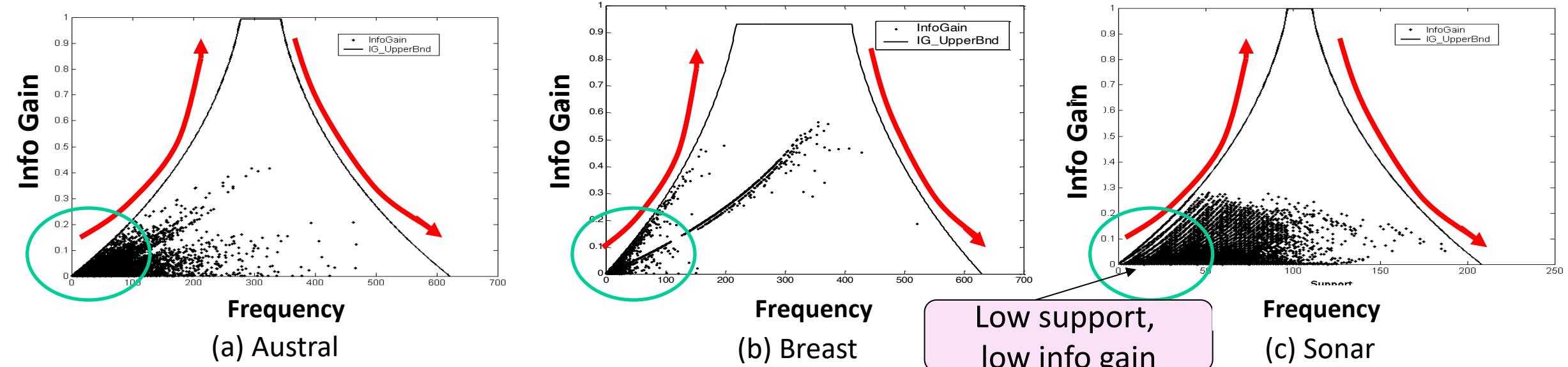


(c) Sonar

Information Gain vs. Pattern Length

Information Gain vs. Pattern Frequency

- Computation on real datasets shows: Pattern frequency (but not too frequent) is strongly tied with the discriminative power (information gain)
- Information gain upper bound monotonically increases with pattern frequency



Information Gain Formula: $IG(C | X) = H(C) - H(C | X)$ Conditional entropy of study focus

Entropy of given data

$$H(C) = -\sum_{i=1}^m p_i \log_2(p_i)$$

$$H(C | X) = \sum_j P(X = x_j)H(Y | X = x_j)$$

Discriminative Pattern-Based Classification: Experimental Results

Table 1. Accuracy by SVM on Frequent Combined Features vs. Single Features

Data	Single Feature			Freq. Pattern	
	<i>Item_All</i>	<i>Item_FS</i>	<i>Item_RBF</i>	<i>Pat_All</i>	<i>Pat_FS</i>
anneal	99.78	99.78	99.11	99.33	99.67
austral	85.01	85.50	85.01	81.79	91.14
auto	83.25	84.21	78.80	74.97	90.79
breast	97.46	97.46	96.98	96.83	97.78
cleve	84.81	84.81	85.80	78.55	95.04
diabetes	74.41	74.41	74.55	77.73	78.31
glass	75.19	75.19	74.78	79.91	81.32
heart	84.81	84.81	84.07	82.22	88.15
hepatic	84.50	89.04	85.83	81.29	96.83
horse	83.70	84.79	82.36	82.35	92.39
iono	93.15	94.30	92.61	89.17	95.44
iris	94.00	96.00	94.00	95.33	96.00
labor	89.99	91.67	91.67	94.99	95.00
lymph	81.00	81.62	84.29	83.67	96.67
pima	74.56	74.56	76.15	76.43	77.16
sonar	82.71	86.55	82.71	84.60	90.86
vehicle	70.43	72.93	72.14	73.33	76.34
wine	98.33	99.44	98.33	98.30	100
zoo	97.09	97.09	95.09	94.18	99.00

Table 2. Accuracy by C4.5 on Frequent Combined Features vs. Single Features

Dataset	Single Features		Frequent Patterns	
	<i>Item_All</i>	<i>Item_FS</i>	<i>Pat_All</i>	<i>Pat_FS</i>
anneal	98.33	98.33	97.22	98.44
austral	84.53	84.53	84.21	88.24
auto	71.70	77.63	71.14	78.77
breast	95.56	95.56	95.40	96.35
cleve	80.87	80.87	80.84	91.42
diabetes	77.02	77.02	76.00	76.58
glass	75.24	75.24	76.62	79.89
heart	81.85	81.85	80.00	86.30
hepatic	78.79	85.21	80.71	93.04
horse	83.71	83.71	84.50	87.77
iono	92.30	92.30	92.89	94.87
iris	94.00	94.00	93.33	93.33
labor	86.67	86.67	95.00	91.67
lymph	76.95	77.62	74.90	83.67
pima	75.86	75.86	76.28	76.72
sonar	80.83	81.19	83.67	83.67
vehicle	70.70	71.49	74.24	73.06
wine	95.52	93.82	96.63	99.44
zoo	91.18	91.18	95.09	97.09

Discriminative Pattern-Based Classification: Scalability Tests

Table 3. Accuracy & Time on Chess Data

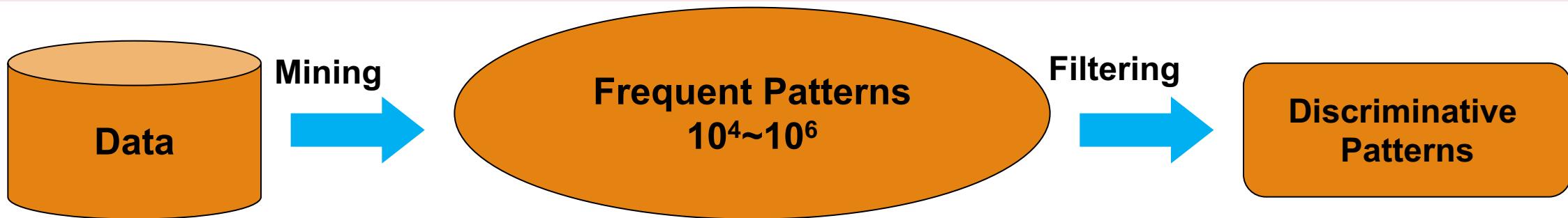
<i>min_sup</i>	#Patterns	Time (s)	SVM (%)	C4.5 (%)
1	N/A	N/A	N/A	N/A
2000	68,967	44.703	92.52	97.59
2200	28,358	19.938	91.68	97.84
2500	6,837	2.906	91.68	97.62
2800	1,031	0.469	91.84	97.37
3000	136	0.063	91.90	97.06

Table 4. Accuracy & Time on Waveform Data

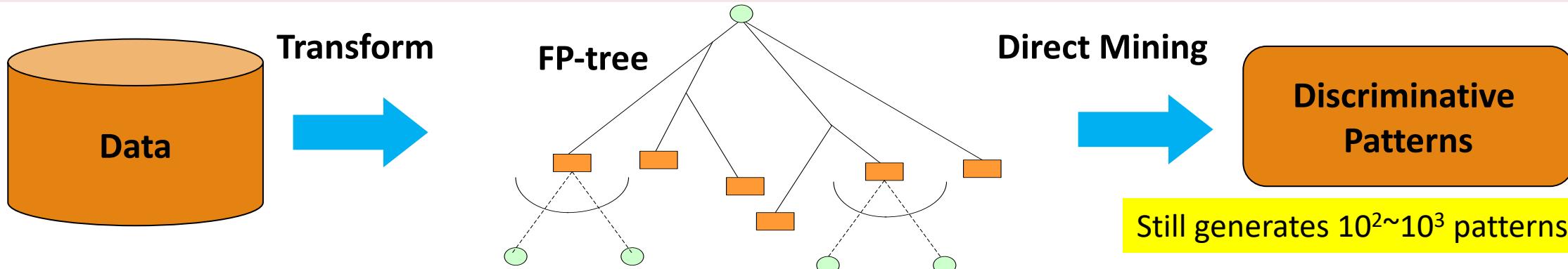
<i>min_sup</i>	#Patterns	Time (s)	SVM (%)	C4.5 (%)
1	9,468,109	N/A	N/A	N/A
80	26,576	176.485	92.40	88.35
100	15,316	90.406	92.19	87.29
150	5,408	23.610	91.53	88.80
200	2,481	8.234	91.22	87.32

Mining Concise Set of Discriminative Patterns

Frequent pattern mining, then getting discriminative patterns: Expensive, large model

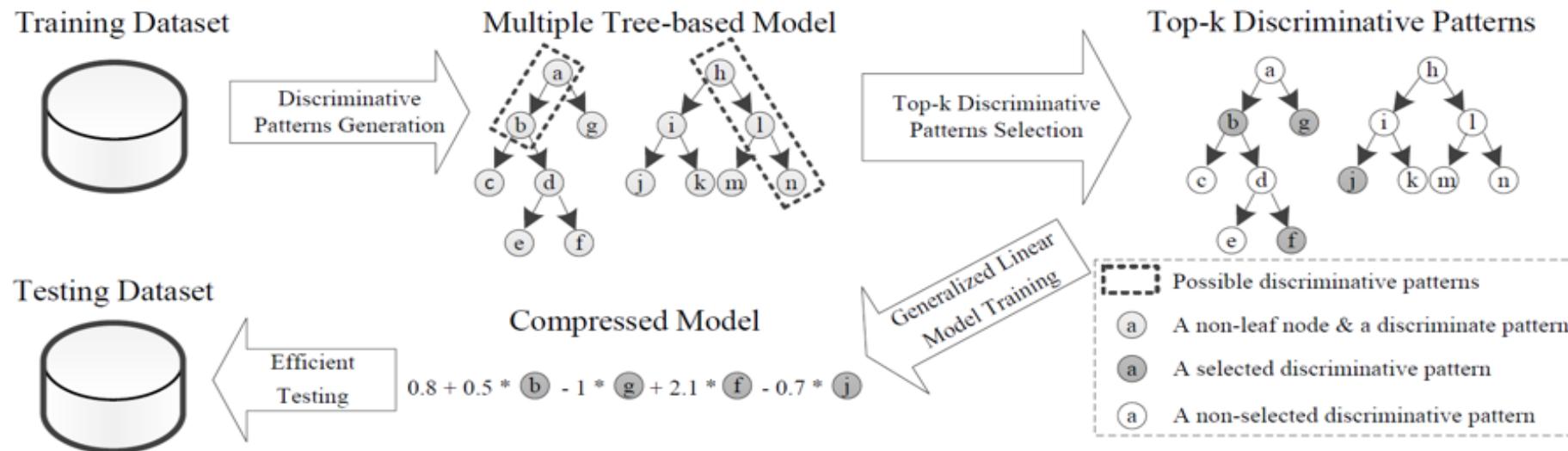


DDPMine [Cheng et al., ICDE'08]: Direct mining of discriminative patterns: Efficient



DPClass [Shang et al, SDM'16]: A better solution—Efficient, effective, and generating a very limited number of (such as only 20 or so) patterns

DPClass: Discriminative Pattern-based Classification



Input: A feature table for training data

- Adopt every prefix path in an (extremely) random forest as a candidate pattern
 - The split points of continuous variables are automatically chosen by random forest → No discretization!
- Run top-k (e.g., top-20) pattern selection based on training data
- Train a generalized linear model (e.g., logistic regression) based on “bag-of-patterns” representations of training data

Explanatory Discriminative Patterns: Generation

- Example: For each patient, we have several uniformly sampled features as follows
 - Age (A): Positive Integers no more than 60
 - Gender (G): Male or Female.
 - Lab Test 1 (LT1): Categorical values from (A, B, O, AB)
 - Lab Test 2 (LT2): Continuous values in [0..1]
- The positive label of the hypo-disease will be given when at least one of the following rules holds
 - $(age > 18) \text{ and } (gender = \text{Male}) \text{ and } (LT1 = AB) \text{ and } (LT2 \geq 0.6)$
 - $(age > 18) \text{ and } (gender = \text{Female}) \text{ and } (LT1 = O) \text{ and } (LT2 \geq 0.5)$
 - $(age \leq 18) \text{ and } (LT2 \geq 0.9)$
- Training: 10^5 random patients + 0.1% noise
 - Flip the binary labels with 0.1% probability
- Testing: 5×10^4 random patients in test

Explanatory Discriminative Patterns: Evaluation

- ❑ Accuracy:
 - ❑ DPClass **99.99% (perfect)**
 - ❑ DDPMine 95.64% (reasonable)
- ❑ Top-3 Discriminative Patterns:
 - ❑ DPClass (**perfect**):
 - ❑ (age > 18) and (gender = Female) and (LT1 = O) and (LT2 \geq 0.496)
 - ❑ (age \leq 18) and (LT2 \geq 0.900)
 - ❑ (age > 18) and (gender = Male) and (LT1 = AB) and (LT2 \geq 0.601)
 - ❑ DDPMine (poor):
 - ❑ (LT2 > 0.8)
 - ❑ (gender = Male) and (LT1 = AB) and (LT2 \geq 0.6) and (LT2 < 0.8)
 - ❑ (gender = Female) and (LT1 = O) and (LT2 \geq 0.6) and (LT2 < 0.8)

A Comparison on Classification Accuracy

- ❑ DPClass: Discriminative & frequent at the same time, then select top-k
- ❑ Only top-20 patterns are used in DPClass
- ❑ Two methods on pattern selection
 - ❑ Forward vs. LASSO
- ❑ In comparison with DDPMine and Random Forest, DPClass maintains high accuracy

	Dataset	DPClass (Forward)	DPClass (LASSO)	DDPMine	Random Forest
low-dimensional data	adult	85.66%	84.33%	83.42%	85.45%
	hypo	99.58%	99.28%	92.69%	97.22%
	sick	98.35%	98.87%	93.82%	94.03%
	crx	89.35%	87.96%	87.96%	89.35%
	sonar	85.29%	83.82%	73.53%	83.82%
	chess	92.25%	92.05%	90.04%	94.22%
high-dimensional data	namao	97.17%	96.94%	96.83%	97.86%
	musk	95.92%	95.71%	93.29%	96.60%
	madelon	74.50%	76.00%	59.84%	56.50%

- ❑ An extension of DPClass has been applied to health study
 - ❑ Cheng et al, "Mining Discriminative Patterns to Predict Health Status for Cardiopulmonary Patients", ACM-BCB'16

Chapter 9. Classification: Advanced Methods

- Bayesian Belief Networks
- Support Vector Machines
- Neural Networks and Deep Learning
- Pattern-Based Classification
- Lazy Learners and K-Nearest Neighbors
- Other Classification Methods
- Summary



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

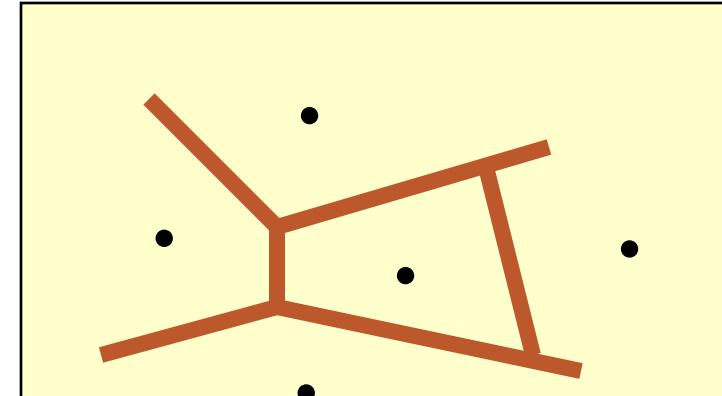
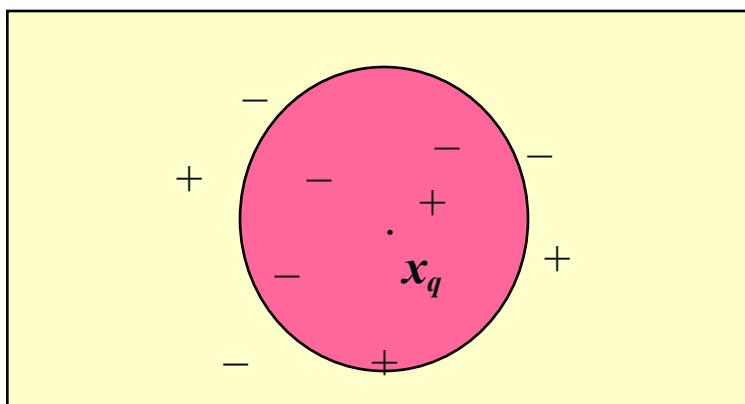
Lazy Learner: Instance-Based Methods

- 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

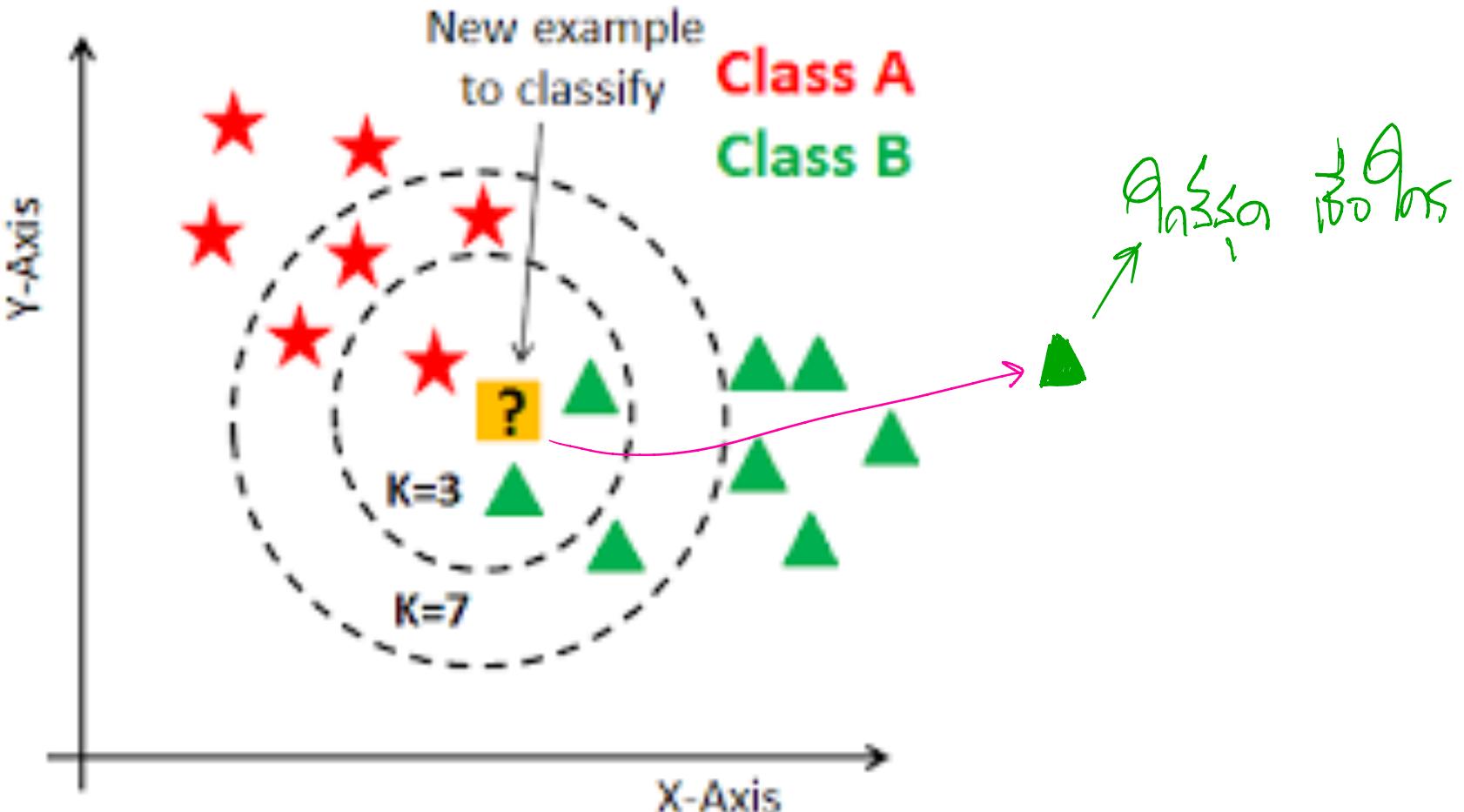
The k -Nearest Neighbor Algorithm

គេណាជាអ្នកសម្រេចក្នុងក្នុង K នាម

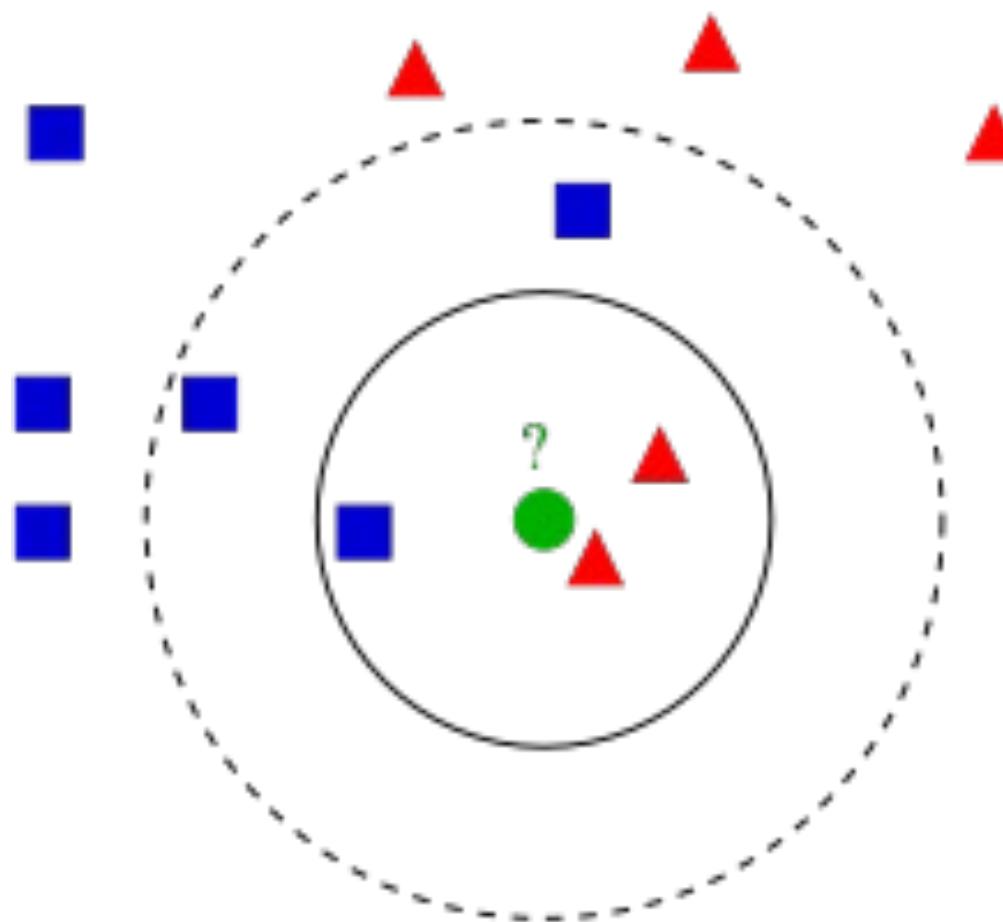
- All instances correspond to points in the n-D space
- The nearest neighbor are defined in terms of Euclidean distance, $\text{dist}(x_1, x_2)$
- Target function could be discrete- or real- valued
- For discrete-valued, k -NN returns the most common value among the k training examples nearest to x_q
- Voronoi diagram: the decision surface induced by 1-NN for a typical set of training examples



KNN



KNN



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

Chapter 9. Classification: Advanced Methods

- Bayesian Belief Networks
- Support Vector Machines
- Neural Networks and Deep Learning
- Pattern-Based Classification
- Lazy Learners and K-Nearest Neighbors
- Other Classification Methods
- Summary



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 pre-specified threshold*
- Slow but easily parallelizable

Chapter 9. Classification: Advanced Methods

- Bayesian Belief Networks
- Support Vector Machines
- Neural Networks and Deep Learning
- Pattern-Based Classification
- Lazy Learners and K-Nearest Neighbors
- Other Classification Methods
- Summary



Summary

- Bayesian belief network (probabilistic networks)
- Support Vector Machine (SVM)
- Neural networks and Deep Learning
- Pattern-Based classification
- Other classification methods
 - lazy learners (KNN, case-based reasoning)
 - genetic algorithms
 - rough set and fuzzy set

References (1)

- C. M. Bishop, Neural Networks for Pattern Recognition. Oxford University Press, 1995
- L. Breiman, J. Friedman, R. Olshen, and C. Stone. Classification and Regression Trees. Wadsworth International Group, 1984
- C. J. C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2): 121-168, 1998
- N. Cristianini and J. Shawe-Taylor, Introduction to Support Vector Machines and Other Kernel-Based Learning Methods, Cambridge University Press, 2000
- H. Yu, J. Yang, and J. Han. Classifying large data sets using SVM with hierarchical clusters. KDD'03
- A. J. Dobson. An Introduction to Generalized Linear Models. Chapman & Hall, 1990
- R. O. Duda, P. E. Hart, and D. G. Stork. Pattern Classification, 2ed. John Wiley, 2001
- T. Hastie, R. Tibshirani, and J. Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer-Verlag, 2001
- S. Haykin, Neural Networks and Learning Machines, Prentice Hall, 2008
- D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. Machine Learning, 1995

References (2): Rule and Pattern-Based Classification

- H. Cheng, X. Yan, J. Han & C.-W. Hsu, Discriminative Frequent Pattern Analysis for Effective Classification, ICDE'07
- H. Cheng, X. Yan, J. Han & P. S. Yu, Direct Discriminative Pattern Mining for Effective Classification, ICDE'08
- W. Cohen. Fast effective rule induction. ICML'95
- G. Cong, K. Tan, A. Tung & X. Xu. Mining Top-k Covering Rule Groups for Gene Expression Data, SIGMOD'05
- M. Deshpande, M. Kuramochi, N. Wale & G. Karypis. Frequent Substructure-based Approaches for Classifying Chemical Compounds, TKDE'05
- G. Dong & J. Li. Efficient Mining of Emerging Patterns: Discovering Trends and Differences, KDD'99
- W. Fan, K. Zhang, H. Cheng, J. Gao, X. Yan, J. Han, P. S. Yu & O. Verscheure. Direct Mining of Discriminative and Essential Graphical and Itemset Features via Model-based Search Tree, KDD'08
- W. Li, J. Han & J. Pei. CMAR: Accurate and Efficient Classification based on Multiple Class-association Rules, ICDM'01
- B. Liu, W. Hsu & Y. Ma. Integrating Classification and Association Rule Mining, KDD'98
- J. R. Quinlan and R. M. Cameron-Jones. FOIL: A midterm report. ECML'93
- Jingbo Shang, Wenzhu Tong, Jian Peng, and Jiawei Han, "[DPClass: An Effective but Concise Discriminative Patterns-Based Classification Framework](#)", SDM'16
- J. Wang and G. Karypis. HARMONY: Efficiently Mining the Best Rules for Classification, SDM'05
- X. Yin & J. Han. CPAR: Classification Based on Predictive Association Rules, SDM'03

