
Data Mining & Knowledge Discovery

Lesson 8 Classification (II)

Lan Man

Department of Computer Science and Technology

East China Normal University

©2017 All rights reserved.

Lesson 8 Classification (II)

- Basic Concepts
- Decision Tree Induction
- Bayesian Classification
- Backpropagation
- Support Vector Machines (SVM)
- Lazy Learners (kNN)
- Other Classification Methods
- Additional Topics
- Prediction
- Model Evaluation and Selection
- Techniques to Improve Classification Accuracy:
Ensemble Methods
- Summary

Bayesian Classification: Why?

- **A statistical classifier**: performs *probabilistic prediction*, *i.e.*, predicts class membership probabilities
- **Foundation**: Based on **Bayes' Theorem**.
- **Performance**: A simple Bayesian classifier, *naïve Bayesian classifier*, has comparable performance with decision tree and selected neural network classifiers
- **Incremental**: Each training example can incrementally increase/decrease the probability that a hypothesis is correct — prior knowledge can be combined with observed data

Bayesian Theorem: Basics

- Let \mathbf{X} be a data sample ("*evidence*"): class label is unknown
- Let \mathbf{H} be a *hypothesis*: \mathbf{X} belongs to class C
- Classification is to determine $\mathbf{P}(\mathbf{H} | \mathbf{X})$, (i.e., *posteriori probability*): the probability that the hypothesis \mathbf{H} holds given the observed data sample \mathbf{X}
- $P(\mathbf{H})$ (*prior probability*), the initial probability
 - E.g., \mathbf{X} will buy computer, regardless of age, income, ...
- $P(\mathbf{X})$: prior probability that sample data is observed
- $P(\mathbf{X} | \mathbf{H})$ (likelihood, *posteriori probability \mathbf{X} conditioned on \mathbf{H}*), the probability of observing the sample \mathbf{X} , given that the hypothesis holds,
 - E.g., Given that \mathbf{X} buy computer, the prob. that \mathbf{X} is 31..40, medium income

Prediction Based on Bayesian Theorem

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

$$P(H \mid \mathbf{X}) = \frac{P(\mathbf{X} \mid H)P(H)}{P(\mathbf{X})}$$

- Informally, this can be written as

$$\textit{posteriori} = \frac{\textit{likelihood} * \textit{prior}}{\textit{evidence}}$$

- Predicts \mathbf{X} belongs to C_i iff the probability $P(C_i|\mathbf{X})$ is the highest among all the $P(C_k|\mathbf{X})$ for all the k ($i=1\dots k$) classes
- Practical difficulty: require initial knowledge of many probabilities, significant computational cost

Classification Is to Derive the Maximum Posteriori

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

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

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

needs to be maximized $P(C_i | \mathbf{X}) \equiv P(\mathbf{X} | C_i)P(C_i)$

Naïve Bayes Classifier

- A simplified assumption: attributes are conditionally independent (i.e., no dependence relation between attributes):

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

- This greatly reduces the computation cost: Only counts the class distribution
- If A_k is categorical, $P(x_k | C_i)$ is the # of tuples in C_i having value x_k for A_k divided by $|C_i, D|$ (# of tuples of C_i in D)
- If A_k is continuous-valued, $P(x_k | C_i)$ is usually computed based on Gaussian distribution with a mean μ and standard deviation σ

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

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

Naïve Bayesian Classifier: Training Dataset

Class:

C1:buys_computer = 'yes'

C2:buys_computer = 'no'

Data sample

$X = (\text{age} \leq 30,$

Income = medium,

Student = yes

Credit_rating = Fair)

Which is maximal, $P(C1|X)$
or $P(C2|X)$?

age	income	student	credit_rating	buy
≤ 30	high	no	fair	no
≤ 30	high	no	excellent	no
31...40	high	no	fair	yes
> 40	medium	no	fair	yes
> 40	low	yes	fair	yes
> 40	low	yes	excellent	no
31...40	low	yes	excellent	yes
≤ 30	medium	no	fair	no
≤ 30	low	yes	fair	yes
> 40	medium	yes	fair	yes
≤ 30	medium	yes	excellent	yes
31...40	medium	no	excellent	yes
31...40	high	yes	fair	yes
> 40	medium	no	excellent	no

Naïve Bayesian Classifier: An Example

- $P(C_i)$: $P(\text{buys_computer} = \text{"yes"}) = 9/14 = 0.643$
 $P(\text{buys_computer} = \text{"no"}) = 5/14 = 0.357$

- Compute $P(X|C_i)$ for each class C_i (i.e., C_1, C_2)

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

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

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

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

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

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

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

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

train/learn/build/construct a model/classifier !

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

Naïve Bayesian Classifier: An Example

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

$P(X | C_i)$:

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

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

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

$$P(X | \text{buys_computer} = \text{"yes"}) * P(\text{buys_computer} = \text{"yes"}) = 0.028 \quad \checkmark$$

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

Therefore, X belongs to class C_1 ("buys_computer = yes")

Avoiding the Zero-Probability Problem

- Naïve Bayesian prediction requires each conditional prob. be **non-zero**. Otherwise, the predicted prob. will be zero

$$P(X | C_i) = \prod_{k=1}^n P(x_k | C_i)$$

- Ex. Suppose a dataset with 1000 tuples, income=low (0), income= medium (990), and income = high (10),
- Use **Laplacian correction** (or **Laplacian estimator**)
 - *Adding 1 to each case*
Prob(income = low) = 1/1003
Prob(income = medium) = 991/1003
Prob(income = high) = 11/1003
 - The “corrected” prob. estimates are close to their “uncorrected” counterparts

Avoiding the underflow Problem

- Often, computing a product of many small probabilities leads to underflow
- Therefore, it's convenient to implement product of probabilities as a sum of log probabilities

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

$$\begin{aligned} \underset{c}{\operatorname{argmax}}(p(C_i | X)) &= \underset{c}{\operatorname{argmax}}(\log(p(X | C_i)) + \log(p(C_i))) \\ &= \underset{c}{\operatorname{argmax}}\left(\sum \log(p(x | C_i)) + \log(p(C_i))\right) \end{aligned}$$

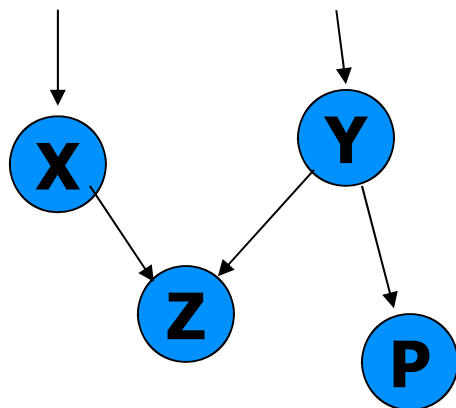
- We can rank log probabilities to find the class that gives maximum probability

Naïve Bayesian Classifier: Comments

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

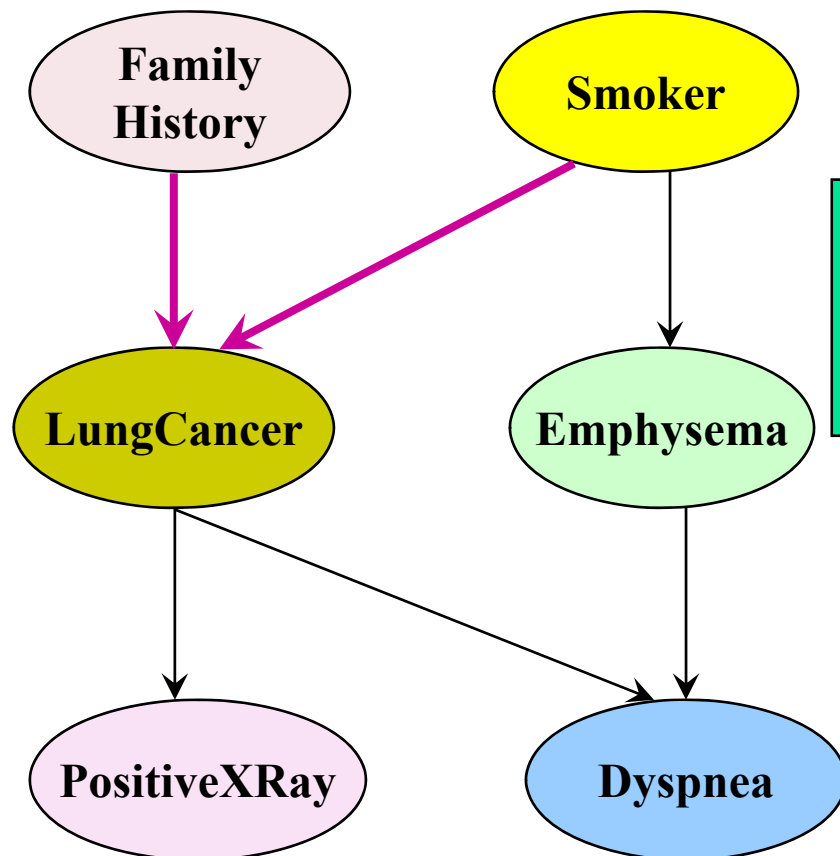
Bayesian Belief Networks

- Two components: (1) A *directed acyclic graph* (called a structure) and (2) a set of *conditional probability tables* (CPTs)
- A (*directed acyclic*) graphical model of *causal influence relationships*
 - Represents dependency among the variables
 - Gives a specification of joint probability distribution



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

Bayesian Belief Network: An Example



The **conditional probability table (CPT)** for variable *LungCancer*:

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

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

Derivation of the probability of a particular combination of values of \mathbf{X} , from CPT:

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

Bayesian Belief Networks

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 given 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

- Given both the network structure and all variables observable: *compute only the CPT entries*
- Network structure known, some hidden variables: *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
- Network structure unknown, all variables observable: search through the model space to *reconstruct network topology*
- 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.

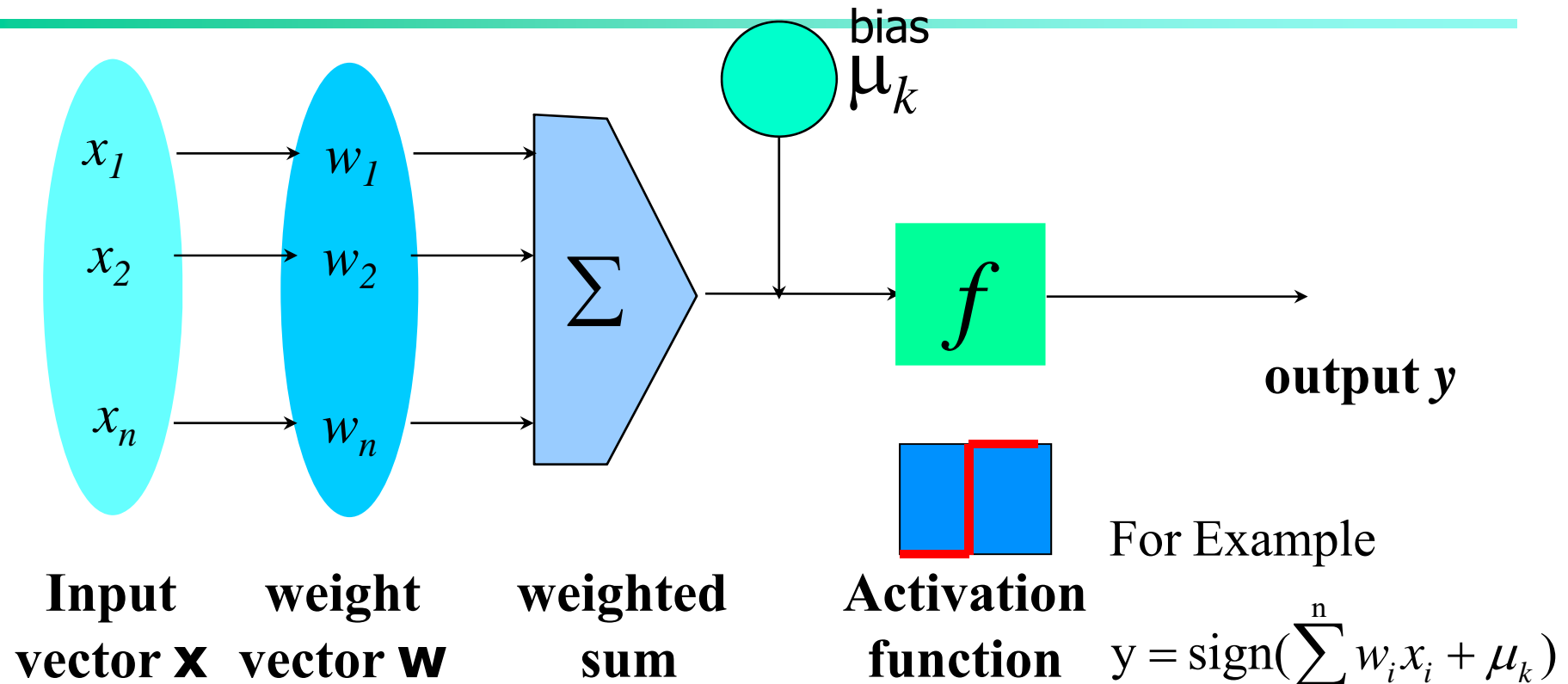
Lesson 8 Classification (II)

- Basic Concepts
- Decision Tree Induction
- Bayesian Classification
- Backpropagation
- Support Vector Machines (SVM)
- Lazy Learners (kNN)
- Other Classification Methods
- Additional Topics
- Prediction
- Model Evaluation and Selection
- Techniques to Improve Classification Accuracy:
Ensemble Methods
- Summary

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 (= perceptron): A Hidden/Output Layer Unit



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

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** in that 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

A Multi-Layer Feed-Forward Neural Network

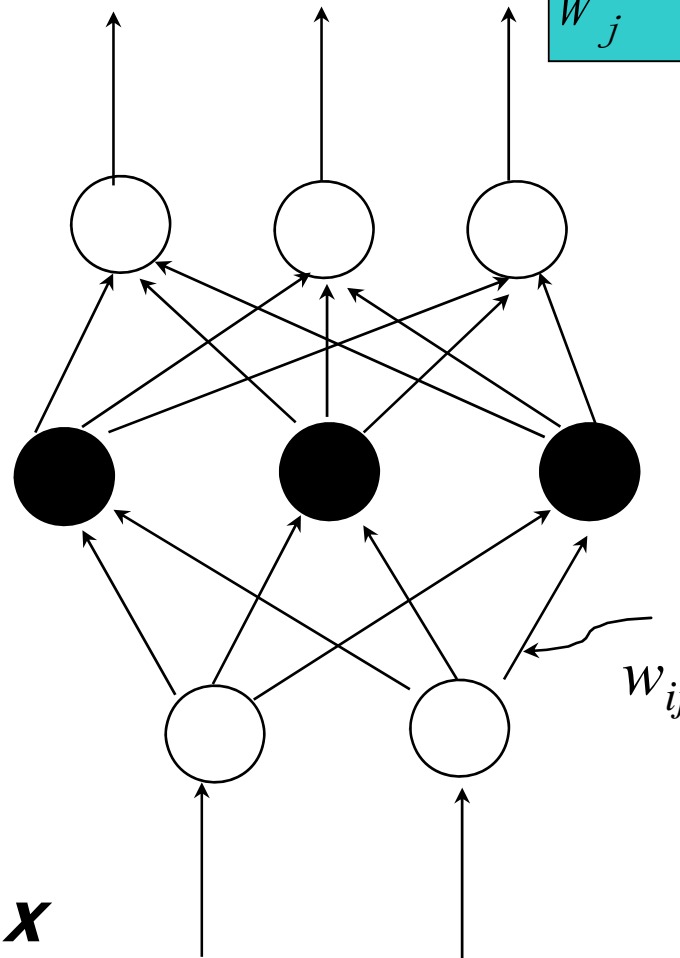
Output vector

Output layer

Hidden layer

Input layer

Input vector: X



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

$$Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk}$$

$$\theta_j = \theta_j + (l)Err_j$$

$$w_{ij} = w_{ij} + (l)Err_j O_i$$

$$Err_j = O_j(1 - O_j)(T_j - O_j)$$

$$O_j = \frac{1}{1 + e^{-I_j}}$$

$$I_j = \sum_i w_{ij} O_i + \theta_j$$

Defining a Network Topology

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

Backpropagation

- Iteratively process a set of training tuples & compare the network's prediction with the actual known target value
- For each training tuple, the weights are modified to **minimize the mean squared error** between the network's prediction and the actual target value
- Modifications are made in the “**backwards**” direction: from the output layer, through each hidden layer down to the first hidden layer, hence “**backpropagation**”
- Steps
 - Initialize weights (to small random #s) and biases in the network
 - 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 a wide array of real-world data
- Algorithms are inherently parallel
- Techniques have recently been developed for the extraction of rules from trained neural networks

Lesson 8 Classification (II)

- Basic Concepts
- Decision Tree Induction
- Bayesian Classification
- Backpropagation
- Support Vector Machines (SVM)
- Lazy Learners (kNN)
- Other Classification Methods
- Additional Topics
- Prediction
- Model Evaluation and Selection
- Techniques to Improve Classification Accuracy:
Ensemble Methods
- Summary

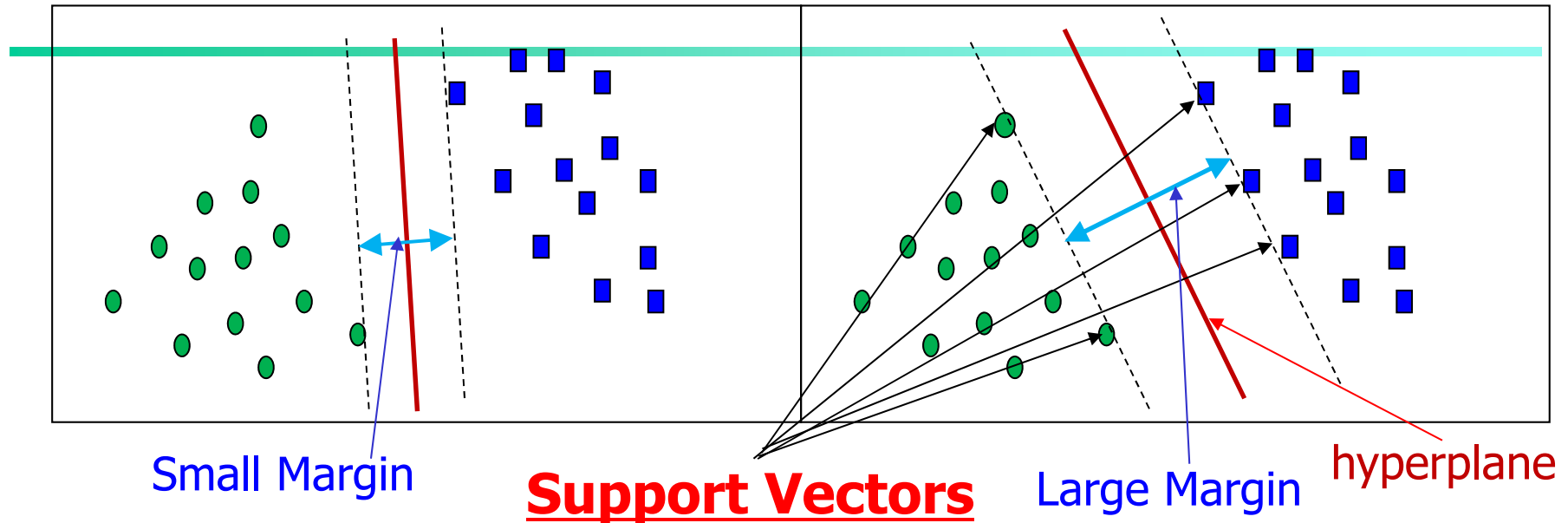
SVM—Support Vector Machines

- 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, a wide range of applications

SVM — General Philosophy



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 — When Data is 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:

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

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

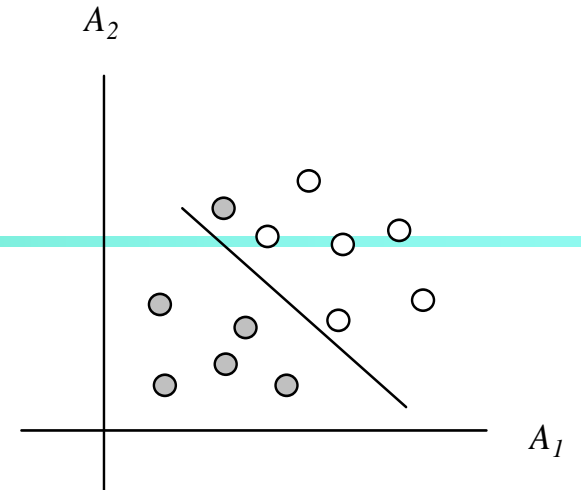
- Any training tuples that fall on hyperplanes H_1 or H_2 (i.e., the sides defining the margin) are **support vectors**
- This becomes a **constrained (convex) quadratic optimization** problem: Quadratic objective function and linear constraints → *Quadratic Programming (QP)* → Lagrangian multipliers

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—When Data Is Linearly Inseparable

- Transform the original input data into a higher dimensional space



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

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

- Search for a linear separating hyperplane in the new space

Kernel functions for Nonlinear Classification

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

- Typical Kernel Functions

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

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

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

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

Scaling SVM by Hierarchical Micro-Clustering

- SVM is not scalable to the number of data objects in terms of training time and memory usage
- “Classifying Large Datasets Using SVMs with Hierarchical Clusters Problem” by Hwanjo Yu, Jiong Yang, Jiawei Han, KDD’03

SVM vs. Neural Network

■ SVM

- Relatively new concept
- 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

- Relatively old
- 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 (not that trivial)

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 C language
 - **SVM-torch**: another recent implementation also written in C.

SVM—Introduction 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

Lesson 8 Classification (II)

- Basic Concepts
- Decision Tree Induction
- Bayesian Classification
- Backpropagation
- Support Vector Machines (SVM)
- Lazy Learners (kNN)
- Other Classification Methods
- Additional Topics
- Prediction
- Model Evaluation and Selection
- Techniques to Improve Classification Accuracy:
Ensemble Methods
- Summary