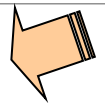
CS 591.03
Introduction to Data Mining
Instructor: Abdullah Mueen

LECTURE 6: CLASSIFICATION: NEURAL NETWORK AND SUPPORT VECTOR MACHINE

Chapter 9. Classification: Advanced Methods

Bayesian Belief Networks 

Classification by Backpropagation

Support Vector Machines

~~Classification by Using Frequent Patterns~~

Lazy Learners (or Learning from Your Neighbors)

~~Other Classification Methods~~

Additional Topics Regarding Classification

Summary

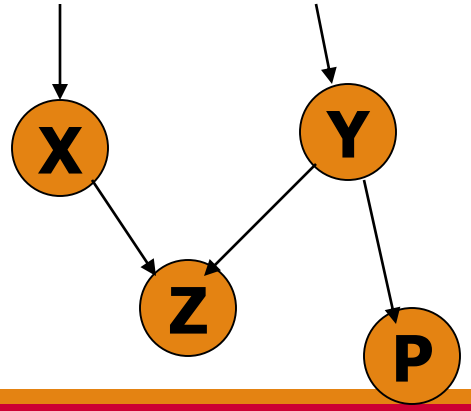
Bayesian Belief Networks

Bayesian belief network (also known as **Bayesian network**, **probabilistic network**): allows *class conditional independencies* between *subsets* of variables

Two components: (1) A *directed acyclic graph* (called a structure) and (2) a set of *conditional probability tables* (CPTs)

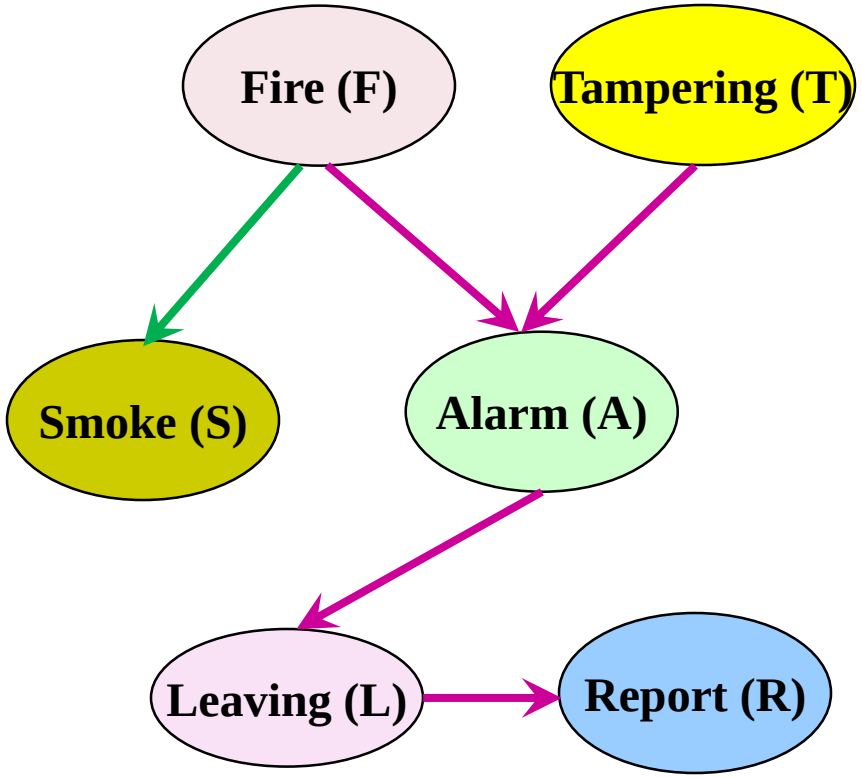
A (*directed acyclic*) graphical model of *causal influence* relationships

- 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/cycles

A Bayesian Network and Some of Its CPTs



CPT: Conditional Probability Tables

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

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

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

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 ‹— F —› A ‹— T, path S—›A is blocked once we know F—›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, w.o. backtracking
- Weights are updated at each iteration & converge to local optimum

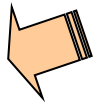
Scenario 3: Network structure unknown, all variables observable: search through the model space to *reconstruct network topology*

Scenario 4: Unknown structure, all hidden variables: No good algorithms known for this purpose

D. Heckerman. A Tutorial on Learning with Bayesian Networks. In *Learning in Graphical Models,* M. Jordan, ed. MIT Press, 1999.

Chapter 9. Classification: Advanced Methods

Bayesian Belief Networks

Classification by Backpropagation 

Support Vector Machines

~~Classification by Using Frequent Patterns~~

Lazy Learners (or Learning from Your Neighbors)

~~Other Classification Methods~~

Additional Topics Regarding Classification

Summary

Classification by Backpropagation

Backpropagation: A **neural network** learning algorithm

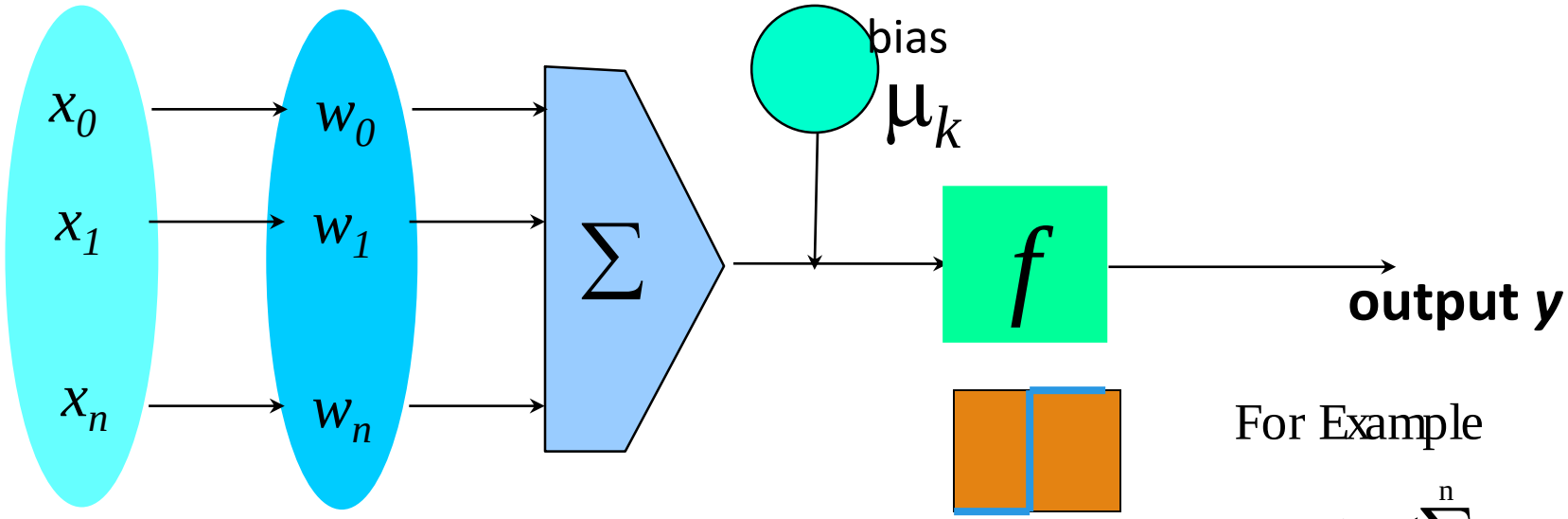
Started by psychologists and neurobiologists to develop and test computational analogues of neurons

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: A Hidden/Output Layer Unit



For Example

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

An *n*-dimensional input vector **x** is mapped into variable y by means of the scalar product and a nonlinear function mapping

The inputs to unit are outputs from the previous layer. They are multiplied by their corresponding weights to form a weighted sum, which is added to the bias associated with unit. Then a nonlinear activation function is applied to it.

How A Multi-Layer Neural Network Works

The **inputs** to the network correspond to the attributes measured for each training tuple

Inputs are fed simultaneously into the units making up the **input layer**

They are then weighted and fed simultaneously to a **hidden layer**

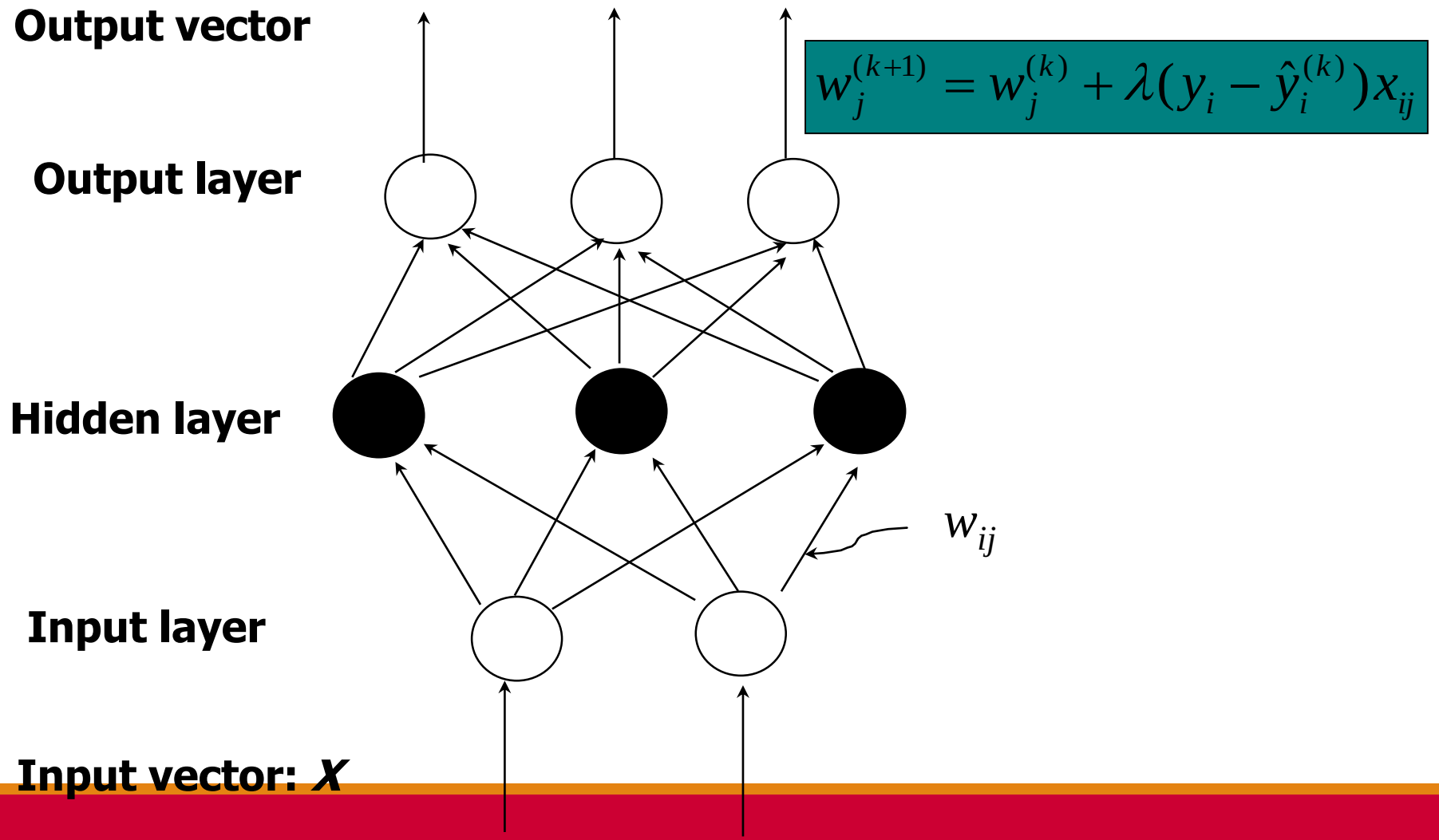
The number of hidden layers is arbitrary, although usually only one

The weighted outputs of the last hidden layer are input to units making up the **output layer**, which emits the network's prediction

The network is **feed-forward**: None of the weights cycles back to an input unit or to an output unit of a previous layer

From a statistical point of view, networks perform **nonlinear regression**: Given enough hidden units and enough training samples, they can closely approximate any function

A Multi-Layer Feed-Forward Neural Network



Defining a Network Topology

Decide the **network topology:** Specify # of units in the *input layer*, # of *hidden layers* (if > 1), # of units in *each hidden layer*, and # of units in the *output layer*

Normalize the input values for each attribute measured in the training tuples to [0.0—1.0]

One **input** unit per domain value, each initialized to 0

Output, if for classification and more than two classes, one output unit per class is used

Once a network has been trained and its accuracy is **unacceptable**, repeat the training process with a *different network topology* or a *different set of initial weights*

Backpropagation

Iteratively process a set of training tuples & compare the network's prediction with the actual known target value

For each training tuple, the weights are modified to **minimize the mean squared error** between the network's prediction and the actual target value

Modifications are made in the "**backwards**" direction: from the output layer, through each hidden layer down to the first hidden layer, hence "**backpropagation**"

Steps

- Initialize weights to small random numbers, associated with biases
- Propagate the inputs forward (by applying activation function)
- Backpropagate the error (by updating weights and biases)
- Terminating condition (when error is very small, etc.)

Efficiency and Interpretability

Efficiency of backpropagation: Each epoch (one iteration through the training set) takes O(|D| * *w*), with |D| tuples and *w* weights, but # of epochs can be exponential to n, the number of inputs, in worst case

For easier comprehension: **Rule extraction** by network pruning

- Simplify the network structure by removing weighted links that have the least effect on the trained network
- Then perform link, unit, or activation value clustering
- The set of input and activation values are studied to derive rules describing the relationship between the input and hidden unit layers

Sensitivity analysis: assess the impact that a given input variable has on a network output. The knowledge gained from this analysis can be represented in rules

Neural Network as a Classifier

Weakness

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

Strength

- High tolerance to noisy data
- Ability to classify untrained patterns
- Well-suited for continuous-valued inputs *and outputs*
- Successful on an array of real-world data, e.g., hand-written letters
- Algorithms are inherently parallel
- Techniques have recently been developed for the extraction of rules from trained neural networks

Chapter 9. Classification: Advanced Methods

Bayesian Belief Networks

Classification by Backpropagation

Support Vector Machines 

~~Classification by Using Frequent Patterns~~

Lazy Learners (or Learning from Your Neighbors)

~~Other Classification Methods~~

Additional Topics Regarding Classification

Summary

Classification: A Mathematical Mapping

Classification: predicts categorical class labels

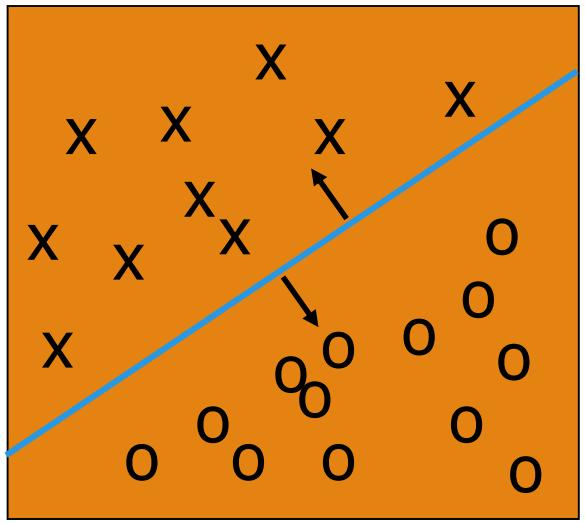
- E.g., Personal homepage classification
  - $x_i = (x_1, x_2, x_3, \ldots)$, $y_i = +1$ or -1
  - x_1 : # of word "homepage"
  - x_2 : # of word "welcome"

Mathematically, $x \in X = \Re^n$, $y \in Y = \{+1, -1\}$,

- We want to derive a function f: X → Y

Linear Classification

- Binary Classification problem
- Data above the red line belongs to class 'x'
- Data below red line belongs to class 'o'
- Examples: SVM, Perceptron, Probabilistic Classifiers



Discriminative Classifiers

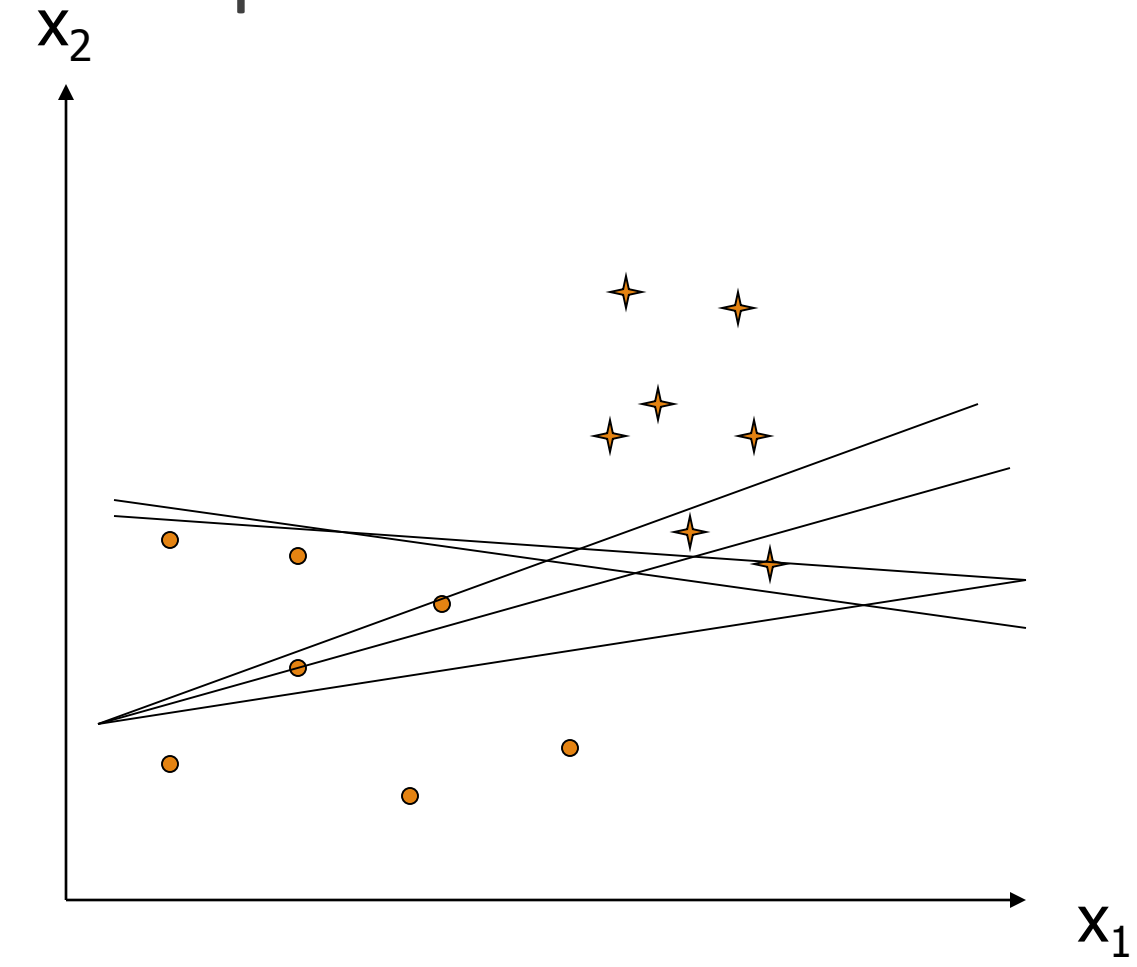
Advantages

- Prediction accuracy is generally high
  - As compared to Bayesian methods – in general
- Robust, works when training examples contain errors
- Fast evaluation of the learned target function
  - Bayesian networks are normally slow

Criticism

- Long training time
- Difficult to understand the learned function (weights)
  - Bayesian networks can be used easily for pattern discovery
- Not easy to incorporate domain knowledge
  - Easy in the form of priors on the data or distributions

Perceptron & Winnow



- Vector: x, w
- Scalar: x, y, w

Input: $\{(\mathrm{x}_1, y_1), \ldots\}$

Output: classification function $f(\mathrm{x})$

$f(\mathrm{x}_i) > 0$ for $y_i = +1$

$f(\mathrm{x}_i) < 0$ for $y_i = -1$

$f(\mathrm{x}) \Rightarrow \mathrm{wx} + b = 0$

or $w_1x_1+w_2x_2+b = 0$

- Perceptron: update W additively
- Winnow: update W multiplicatively

SVM—Support Vector Machines

A relatively new classification method for both linear and nonlinear data

It uses a nonlinear mapping to transform the original training data into a higher dimension

With the new dimension, it searches for the linear optimal separating **hyperplane** (i.e., "decision boundary")

With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane

SVM finds this hyperplane using **support vectors** ("essential" training tuples) and **margins** (defined by the support vectors)

SVM—History and Applications

Vapnik and colleagues (1992)—groundwork from Vapnik & Chervonenkis' statistical learning theory in 1960s

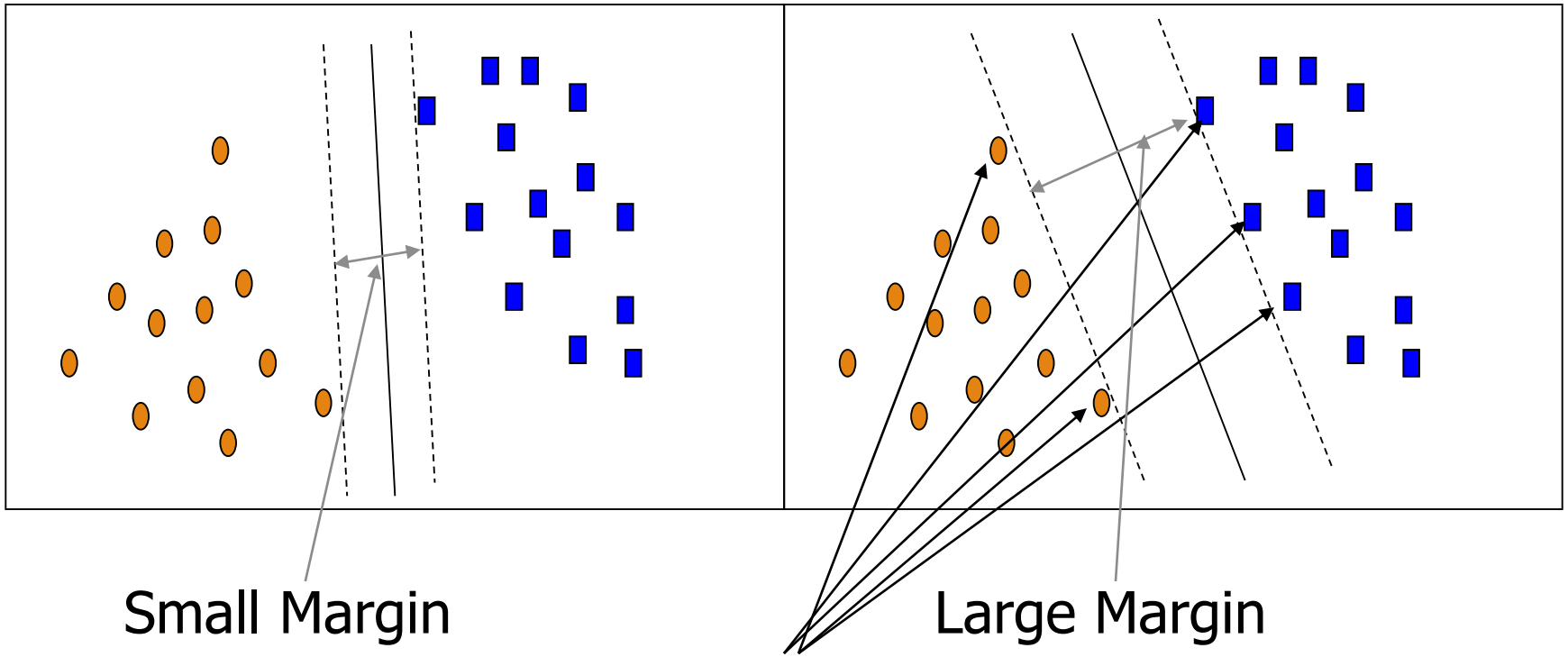
Features: training can be slow but accuracy is high owing to their ability to model complex nonlinear decision boundaries (margin maximization)

Used for: classification and numeric prediction

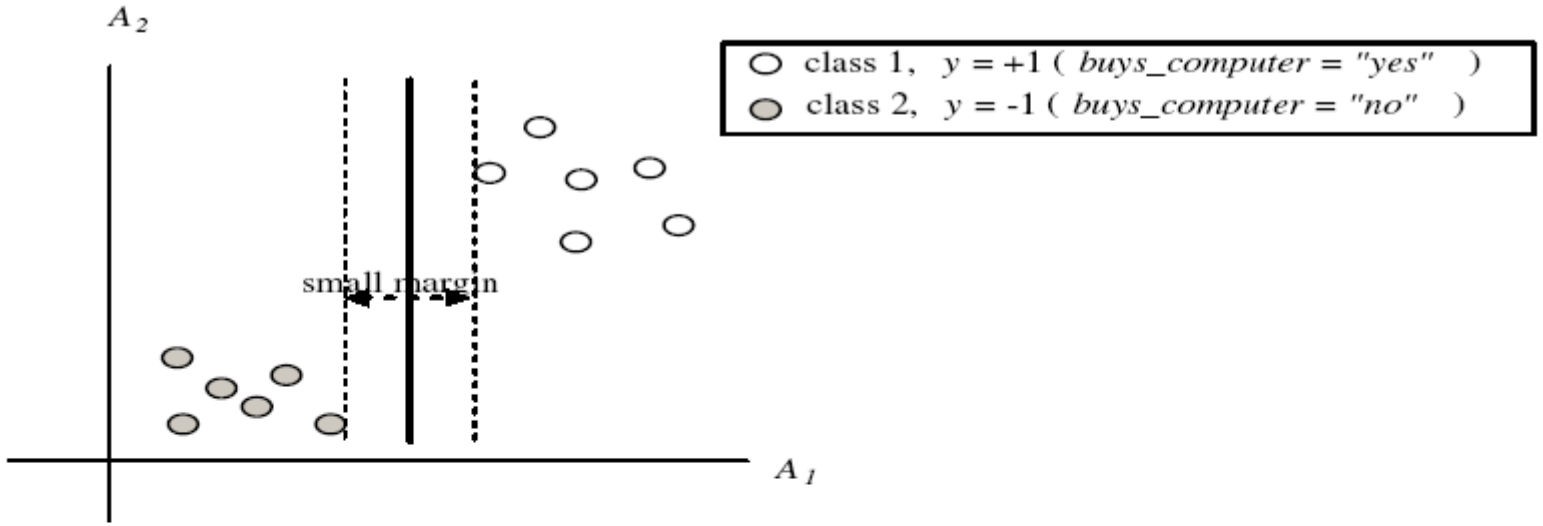
Applications:

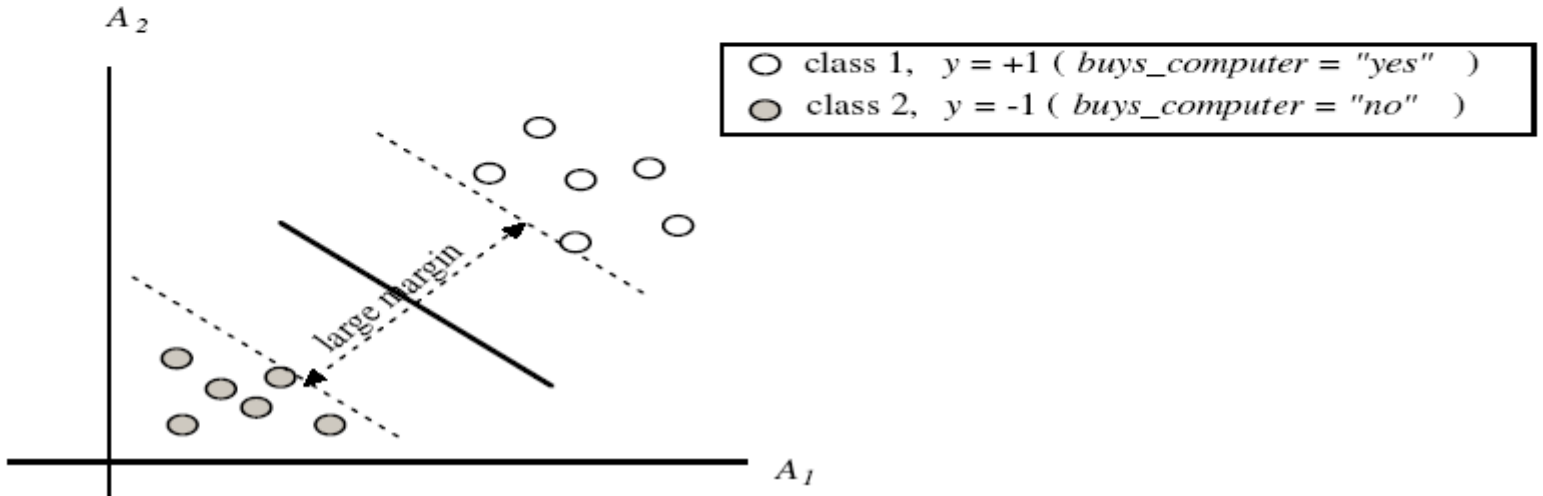
- handwritten digit recognition, object recognition, speaker identification, benchmarking time-series prediction tests

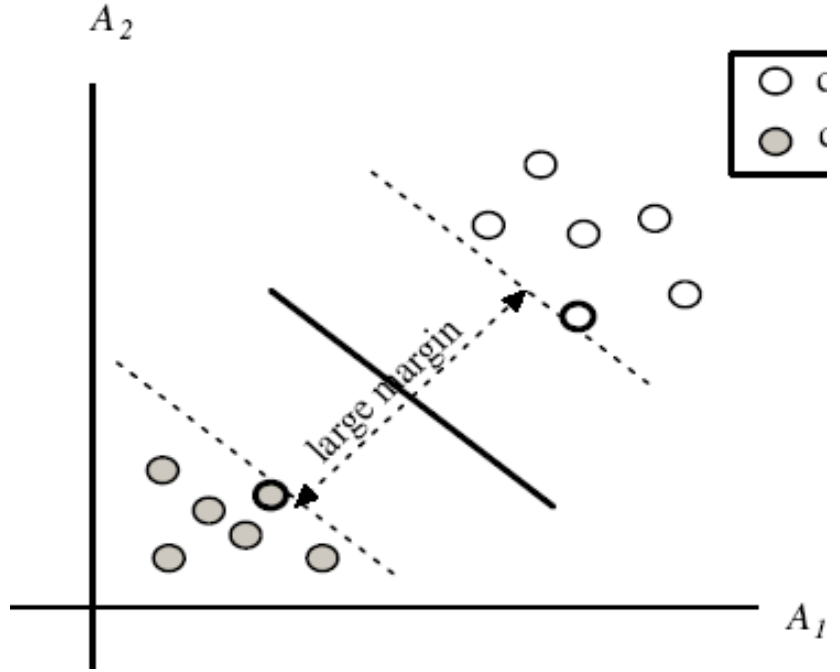
SVM—General Philosophy



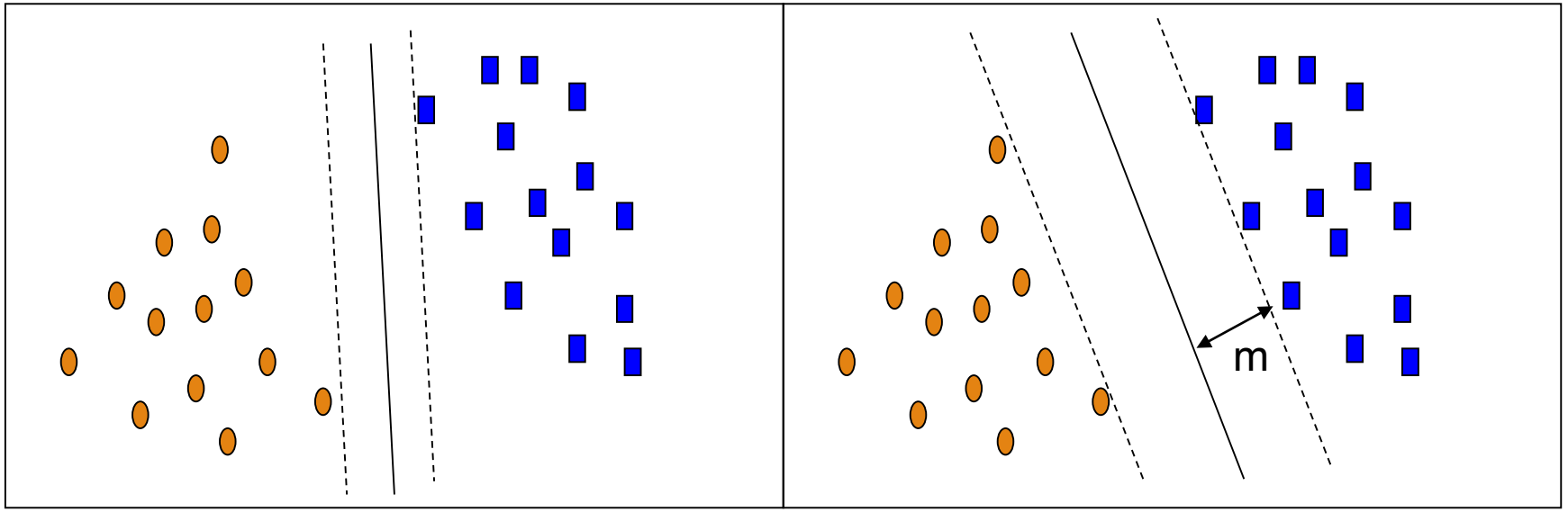
SVM—Margins and Support Vectors







SVM—When Data Is Linearly Separable



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

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

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

SVM—Linearly Separable

- A separating hyperplane can be written as

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

  where $\mathbf{W}=\{w_1, w_2, \ldots, w_n\}$ is a weight vector and b a scalar (bias)
- For 2-D it can be written as

  $w_0 + w_1 x_1 + w_2 x_2 = 0$
- The hyperplane defining the sides of the margin:

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

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

Why Is SVM Effective on High Dimensional Data?

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

SVM—Linearly Inseparable

- Transform the original input data into a higher dimensional space

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

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

■

- Search for a linear separating hyperplane in the new space

Kernel functions for Nonlinear Classification

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

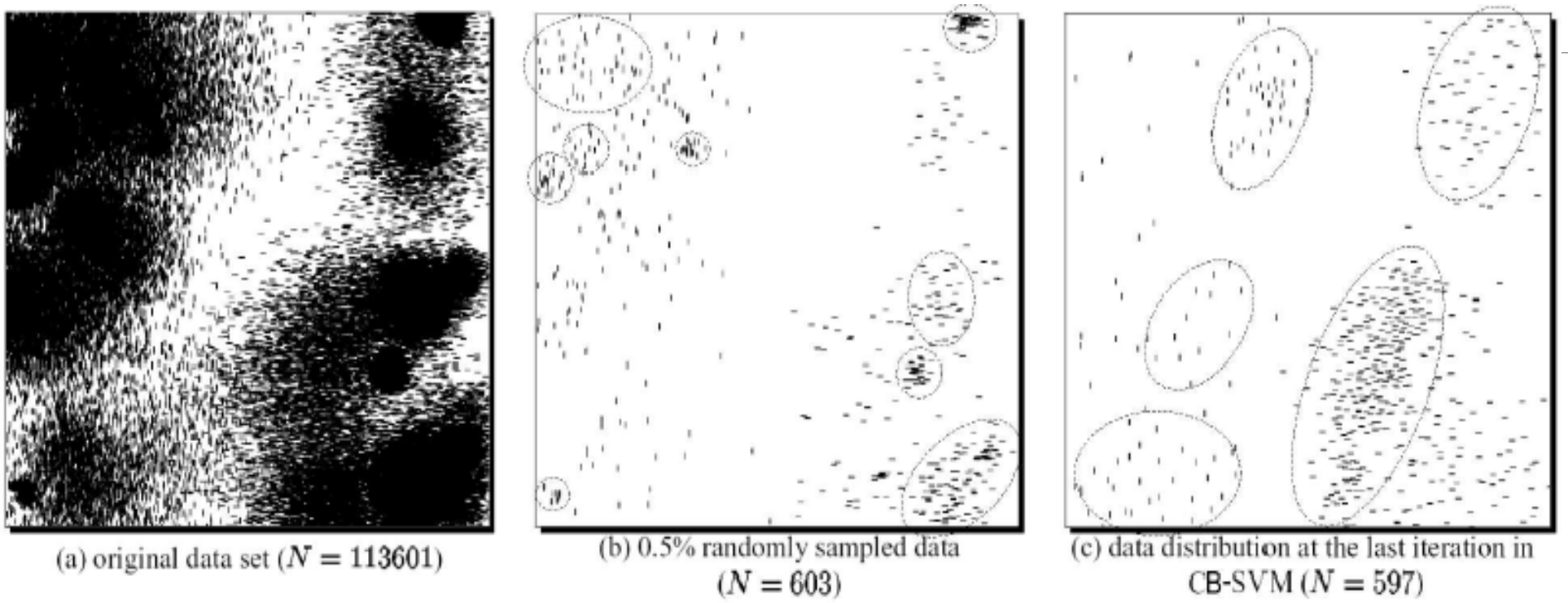
$$\textbf{Polynomial kernel of degree } h: \quad K(X_i, X_j) = (X_i \cdot X_j + 1)^h$$

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

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

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

Accuracy and Scalability on Synthetic Dataset



(a) original data set ($N = 113601$)

(b) 0.5% randomly sampled data ($N = 603$)

(c) data distribution at the last iteration in CB-SVM ($N = 597$)

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 vs. Neural Network

SVM

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

Neural Network

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

SVM Related Links

SVM Website: http://www.kernel-machines.org/

Representative implementations

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

Chapter 9. Classification: Advanced Methods

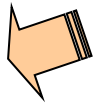
Bayesian Belief Networks

Classification by Backpropagation

Support Vector Machines

~~Classification by Using Frequent Patterns~~

Lazy Learners (or Learning from Your Neighbors)



~~Other Classification Methods~~

Additional Topics Regarding Classification

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

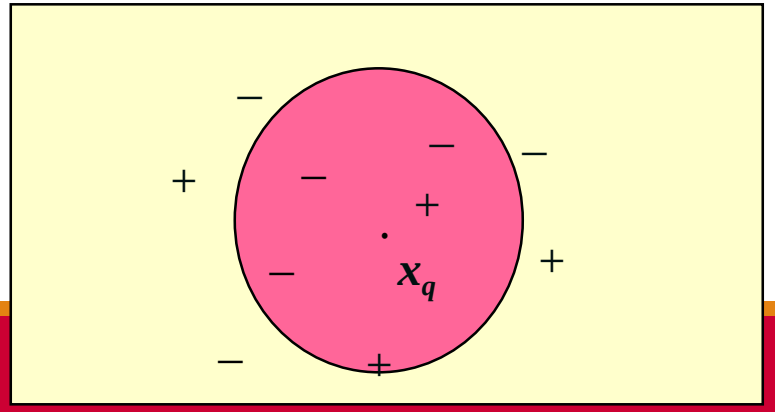
All instances correspond to points in the n-D space

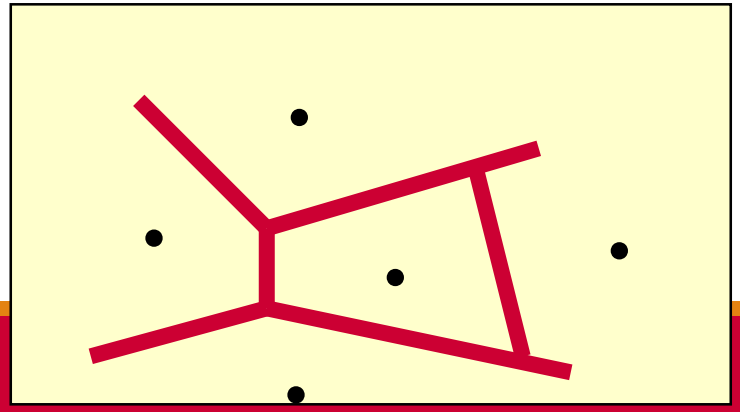
The nearest neighbor are defined in terms of Euclidean distance, dist($\mathbf{X}_1$, $\mathbf{X}_2$)

Target function could be discrete- or real- valued

For discrete-valued, *k*-NN returns the most common value among the *k* training examples nearest to x_q

Vonoroi diagram: the decision surface induced by 1-NN for a typical set of training examples





Discussion on the *k*-NN Algorithm

k-NN for real-valued prediction for a given unknown tuple

- Returns the mean values of the *k* nearest neighbors

Distance-weighted nearest neighbor algorithm

- Weight the contribution of each of the *k* neighbors according to their distance to the query x_q
  - Give greater weight to closer neighbors

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

Robust to noisy data by averaging *k*-nearest neighbors

Curse of dimensionality: distance between neighbors could be dominated by irrelevant attributes

- To overcome it, axes stretch or elimination of the least relevant attributes

Case-Based Reasoning (CBR)

CBR: Uses a database of problem solutions to solve new problems

Store symbolic description (tuples or cases)—not points in a Euclidean space

Applications: Customer-service (product-related diagnosis), legal ruling

Methodology

- Instances represented by rich symbolic descriptions (e.g., function graphs)
- Search for similar cases, multiple retrieved cases may be combined
- Tight coupling between case retrieval, knowledge-based reasoning, and problem solving

Challenges

- Find a good similarity metric
- Indexing based on syntactic similarity measure, and when failure, backtracking, and adapting to additional cases

Chapter 9. Classification: Advanced Methods

Bayesian Belief Networks

Classification by Backpropagation

Support Vector Machines

~~Classification by Using Frequent Patterns~~

Lazy Learners (or Learning from Your Neighbors)

~~Other Classification Methods~~

Additional Topics Regarding Classification

Summary

Multiclass Classification

Classification involving more than two classes (i.e., > 2 Classes)

Method 1. **One-vs.-all** (OVA): Learn a classifier one at a time

- Given m classes, train m classifiers: one for each class
- Classifier j: treat tuples in class j as *positive* & all others as *negative*
- To classify a tuple **X**, the set of classifiers vote as an ensemble

Method 2. **All-vs.-all** (AVA): Learn a classifier for each pair of classes

- Given m classes, construct m(m-1)/2 binary classifiers
- A classifier is trained using tuples of the two classes
- To classify a tuple **X**, each classifier votes. X is assigned to the class with maximal vote

Comparison

- All-vs.-all tends to be superior to one-vs.-all
- Problem: Binary classifier is sensitive to errors, and errors affect vote count

Error-Correcting Codes for Multiclass Classification

Originally designed to correct errors during data transmission for communication tasks by exploring data redundancy

Example

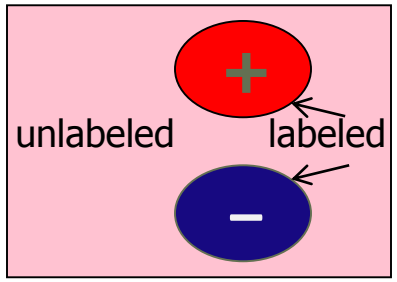
- A 7-bit codeword associated with classes 1-4

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

- Given an unknown tuple **X**, the 7-trained classifiers output: 0001010
- Hamming distance: # of different bits between two codewords
- H(**X**, C_1) = 5, by checking # of bits between [1111111] & [0001010]
- H(**X**, C_2) = 3, H(**X**, C_3) = 3, H(**X**, C_4) = 1, thus C_4 as the label for **X**

- Error-correcting codes can correct up to (h – 1)/2 1-bit error, where h is the minimum Hamming distance between any two codewords
- If we use 1-bit per class, it is equiv. to one-vs.-all approach, the code are insufficient to self-correct
- When selecting error-correcting codes, there should be good row-wise and col.-wise separation between the codewords

Semi-Supervised Classification



Semi-supervised: Uses labeled and unlabeled data to build a classifier

Self-training:

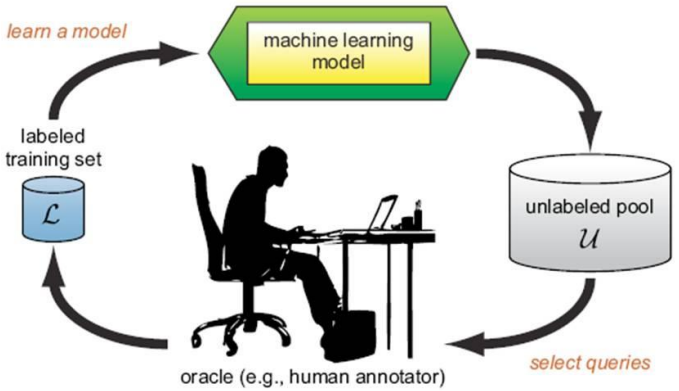
- Build a classifier using the labeled data
- Use it to label the unlabeled data, and those with the most confident label prediction are added to the set of labeled data
- Repeat the above process
- Adv: easy to understand; disadv: may reinforce errors

Co-training: Use two or more classifiers to teach each other

- Each learner uses a mutually independent set of features of each tuple to train a good classifier, say f_1
- Then f_1 and f_2 are used to predict the class label for unlabeled data X
- Teach each other: The tuple having the most confident prediction from f_1 is added to the set of labeled data for f_2, & vice versa

Other methods, e.g., joint probability distribution of features and labels

Active Learning



Class labels are expensive to obtain

Active learner: query human (oracle) for labels

Pool-based approach: Uses a pool of unlabeled data
- L: a small subset of D is labeled, U: a pool of unlabeled data in D
- Use a query function to carefully select one or more tuples from U and request labels from an oracle (a human annotator)
- The newly labeled samples are added to L, and learn a model
- Goal: Achieve high accuracy using as few labeled data as possible

Evaluated using *learning curves*: Accuracy as a function of the number of instances queried (# of tuples to be queried should be small)

Research issue: How to choose the data tuples to be queried?
- Uncertainty sampling: choose the least certain ones
- Reduce *version space*, the subset of hypotheses consistent w. the training data
- Reduce expected entropy over U: Find the greatest reduction in the total number of incorrect predictions

Chapter 9. Classification: Advanced Methods

Bayesian Belief Networks

Classification by Backpropagation

Support Vector Machines

Classification by Using Frequent Patterns

Lazy Learners (or Learning from Your Neighbors)

Other Classification Methods

Additional Topics Regarding Classification

Summary

Summary

Effective and advanced classification methods

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

Additional Topics on Classification

- Multiclass classification
- Semi-supervised classification
- Active learning
- Transfer learning