CS6220: DATA MINING TECHNIQUES

Chapter 8&9: Classification: Part 2

Instructor: Yizhou Sun

yzsun@ccs.neu.edu

February 11, 2013

Announcement

- Homework 1
  - Apriori algorithm: some length-l candidate can be pruned by checking whether all its sub-patterns with length-(l-1) are in frequent
  - FP-growth: Need to scan DB twice
- Proposal due tomorrow
  - Report
  - Sign up meeting time with me on Wednesday afternoon or Thursday morning
- Homework 2 will be out tomorrow
- No class next week
  - The make-up class is canceled
  - With homework 2 due on next Friday
- Midterm exam (Feb. 25, the same location, same time as classes, 6-8pm)
  - You can take a A4 "cheating sheet"
  - Covers to the content taught by today

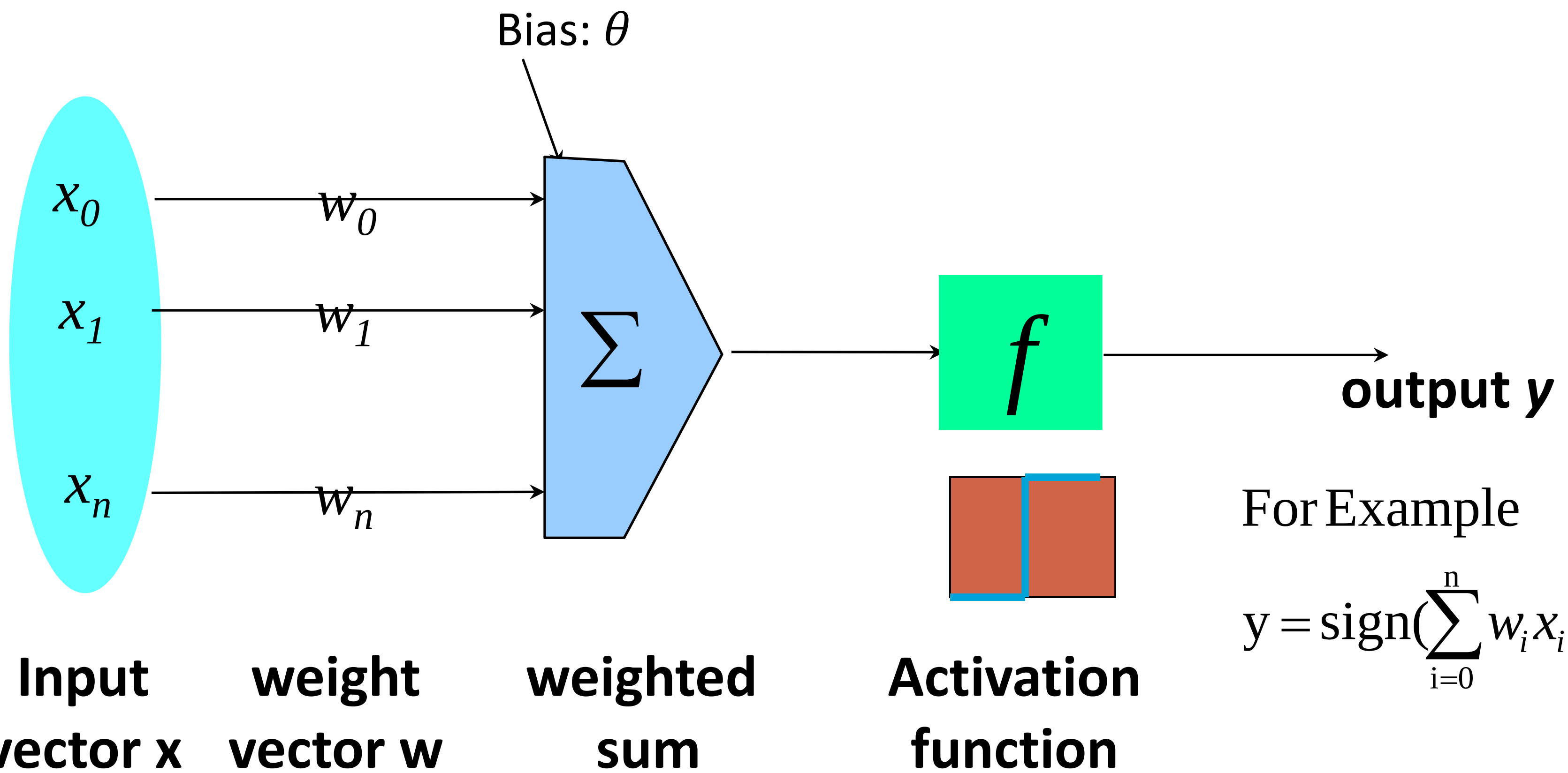
Chapter 8&9. Classification: Part 2

- Neural Networks
- Support Vector Machine
- Summary

Artificial Neural Networks

- Consider humans:
  - Neuron switching time ~.001 second
  - Number of neurons ~10^{10}
  - Connections per neuron ~10^{4-5}
  - Scene recognition time ~.1 second
  - 100 inference steps doesn't seem like enough -> parallel computation
- Artificial neural networks
  - Many neuron-like threshold switching units
  - Many weighted interconnections among units
  - Highly parallel, distributed process
  - Emphasis on tuning weights automatically

Single Unit: Perceptron



For Example

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

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

Perceptron Training Rule

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

where

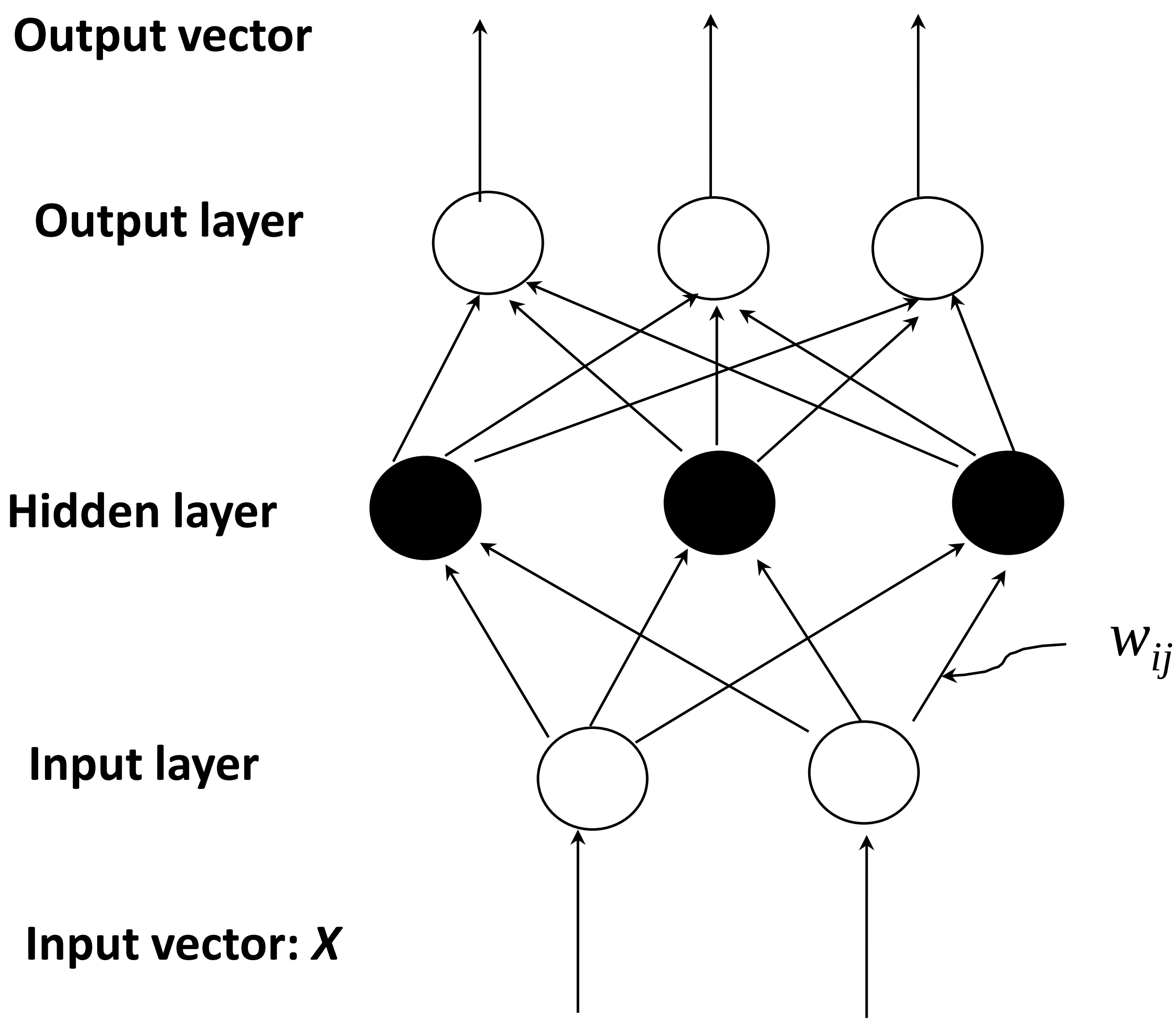
$$\Delta w_i = \eta(t - o)x_i$$

- t: target value (true value)
- o: output value
- η: learning rate (small constant)

- Derived using Gradient Descent method by minimizing the squared error:

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

A Multi-Layer Feed-Forward Neural Network



A **two-layer** network

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

Defining a Network Topology

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

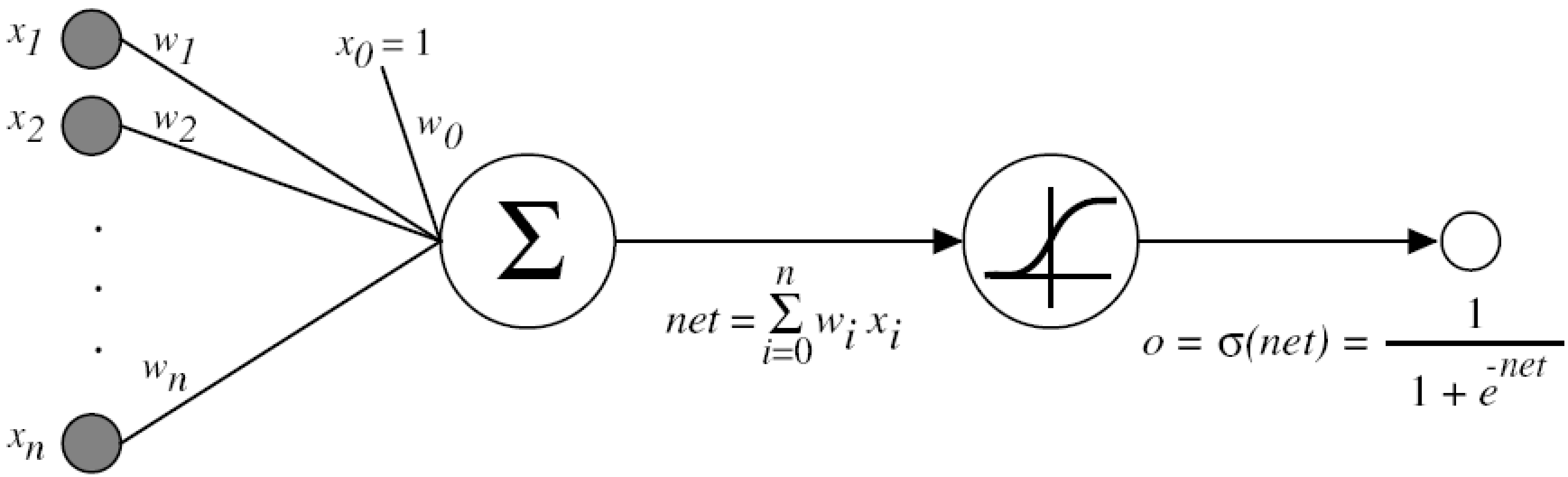
Classification by Backpropagation

- Backpropagation: A **neural network** learning algorithm
- Started by psychologists and neurobiologists to develop and test computational analogues of neurons
- During the learning phase, the **network learns by adjusting the weights** so as to be able to predict the correct class label of the input tuples
- Also referred to as **connectionist learning** due to the connections between units

Backpropagation

- Iteratively process a set of training tuples & compare the network's prediction with the actual known target value
- For each training tuple, the weights are modified to **minimize the mean squared error** between the network's prediction and the actual target value
- Modifications are made in the "**backwards**" direction: from the output layer, through each hidden layer down to the first hidden layer, hence "**backpropagation**"

Sigmoid Unit

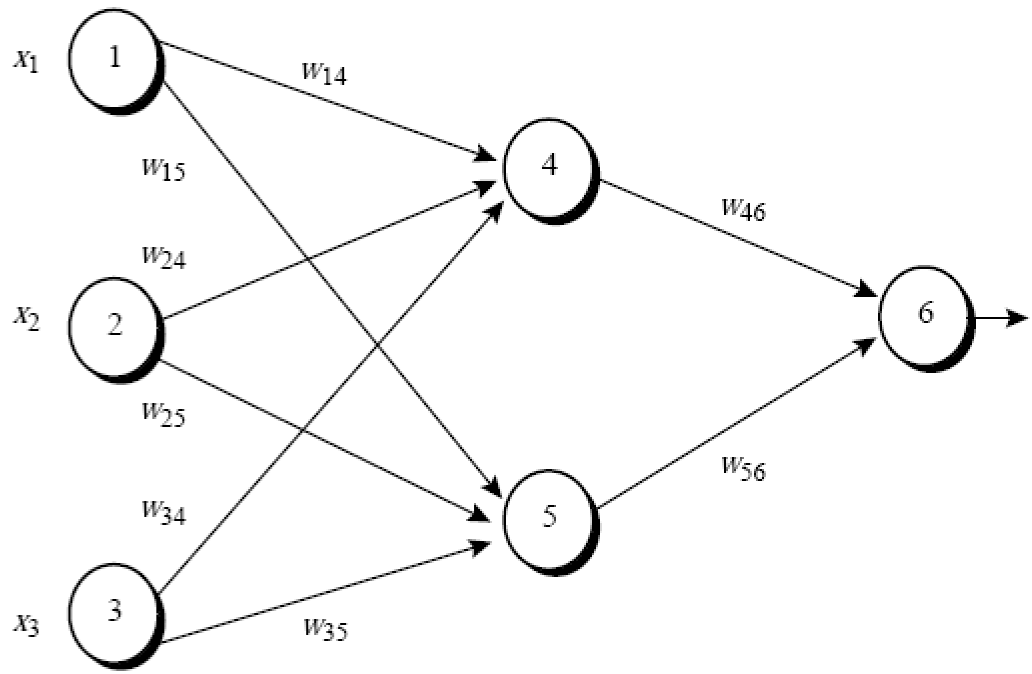


- $\sigma(x) = \frac{1}{1+e^{-x}}$ is a sigmoid function
  - Property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$
  - Will be used in backpropagation

Backpropagation Steps

- Initialize weights to small random numbers, associated with biases
- Repeat until terminating condition meets
  - For each training example
    - **Propagate the inputs forward** (by applying activation function)
      - For a hidden or output layer unit j
        - Calculate net input: $I_j = \sum_i w_{ij} O_i + \theta_j$
        - Calculate output of unit j: $O_j = \frac{1}{1+e^{-I_j}}$
    - **Backpropagate the error** (by updating weights and biases)
      - For unit j in output layer: $Err_j = O_j(1-O_j)(T_j - O_j)$
      - For unit j in a hidden layer: : $Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$
      - Update weights: $w_{ij} = w_{ij} + \eta Err_j O_i$
- Terminating condition (when error is very small, etc.)

Example



A multilayer feed-forward neural network

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1

Initial Input, weight, and bias values

- Input forward:

Table 9.2: The net input and output calculations.

Unit j	*Net input,* I_j	*Output,* O_j
4	$0.2 + 0 - 0.5 - 0.4 = -0.7$	$1/(1 + e^{0.7}) = 0.332$
5	$-0.3 + 0 + 0.2 + 0.2 = 0.1$	$1/(1 + e^{-0.1}) = 0.525$
6	$(-0.3)(0.332) - (0.2)(0.525) + 0.1 = -0.105$	$1/(1 + e^{0.105}) = 0.474$

- Error backpropagation and weight update:

Table 9.3: Calculation of the error at each node.

Unit j	Err_j
6	$(0.474)(1 - 0.474)(1 - 0.474) = 0.1311$
5	$(0.525)(1 - 0.525)(0.1311)(-0.2) = -0.0065$
4	$(0.332)(1 - 0.332)(0.1311)(-0.3) = -0.0087$

Table 9.4: Calculations for weight and bias updating.

Weight or bias	*New value*
w_{46}	$-0.3 + (0.9)(0.1311)(0.332) = -0.261$
w_{56}	$-0.2 + (0.9)(0.1311)(0.525) = -0.138$
w_{14}	$0.2 + (0.9)(-0.0087)(1) = 0.192$
w_{15}	$-0.3 + (0.9)(-0.0065)(1) = -0.306$
w_{24}	$0.4 + (0.9)(-0.0087)(0) = 0.4$

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
  - E.g., If x decreases 5% then y increases 8%

Neural Network as a Classifier

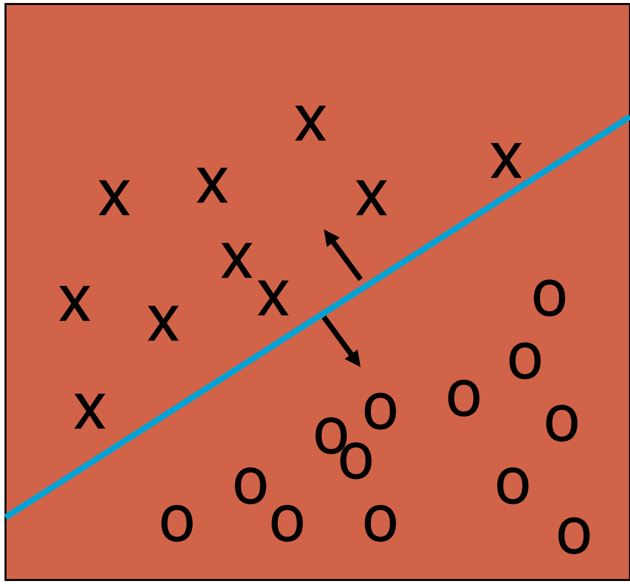
- Weakness
  - Long training time
  - Require a number of parameters typically best determined empirically, e.g., the network topology or "structure."
  - Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of "hidden units" in the network
- Strength
  - High tolerance to noisy data
  - 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 8&9. Classification: Part 2

- Neural Networks
- Support Vector Machine
- Summary

Classification: A Mathematical Mapping

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



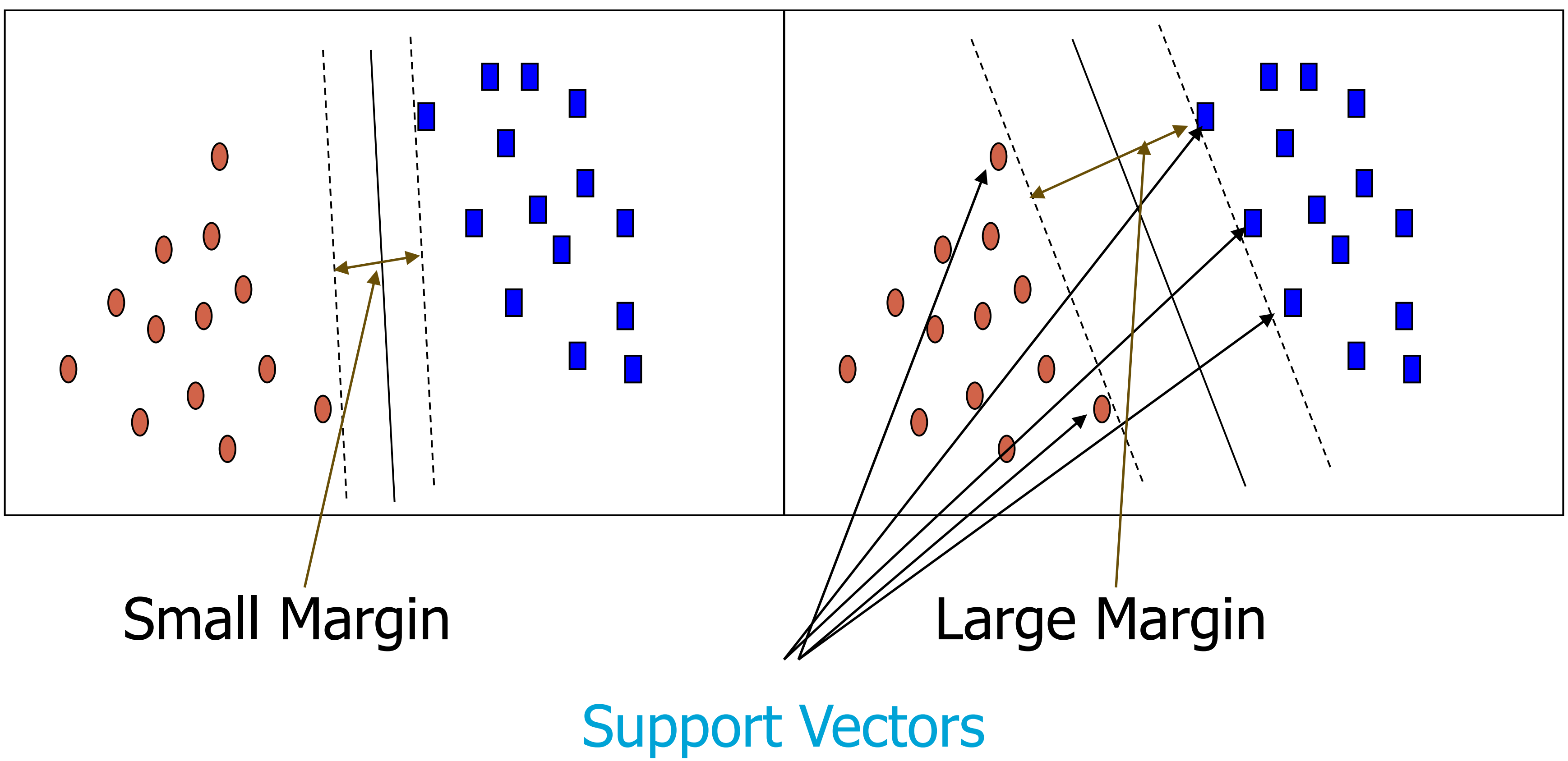
SVM—Support Vector Machines

- 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)

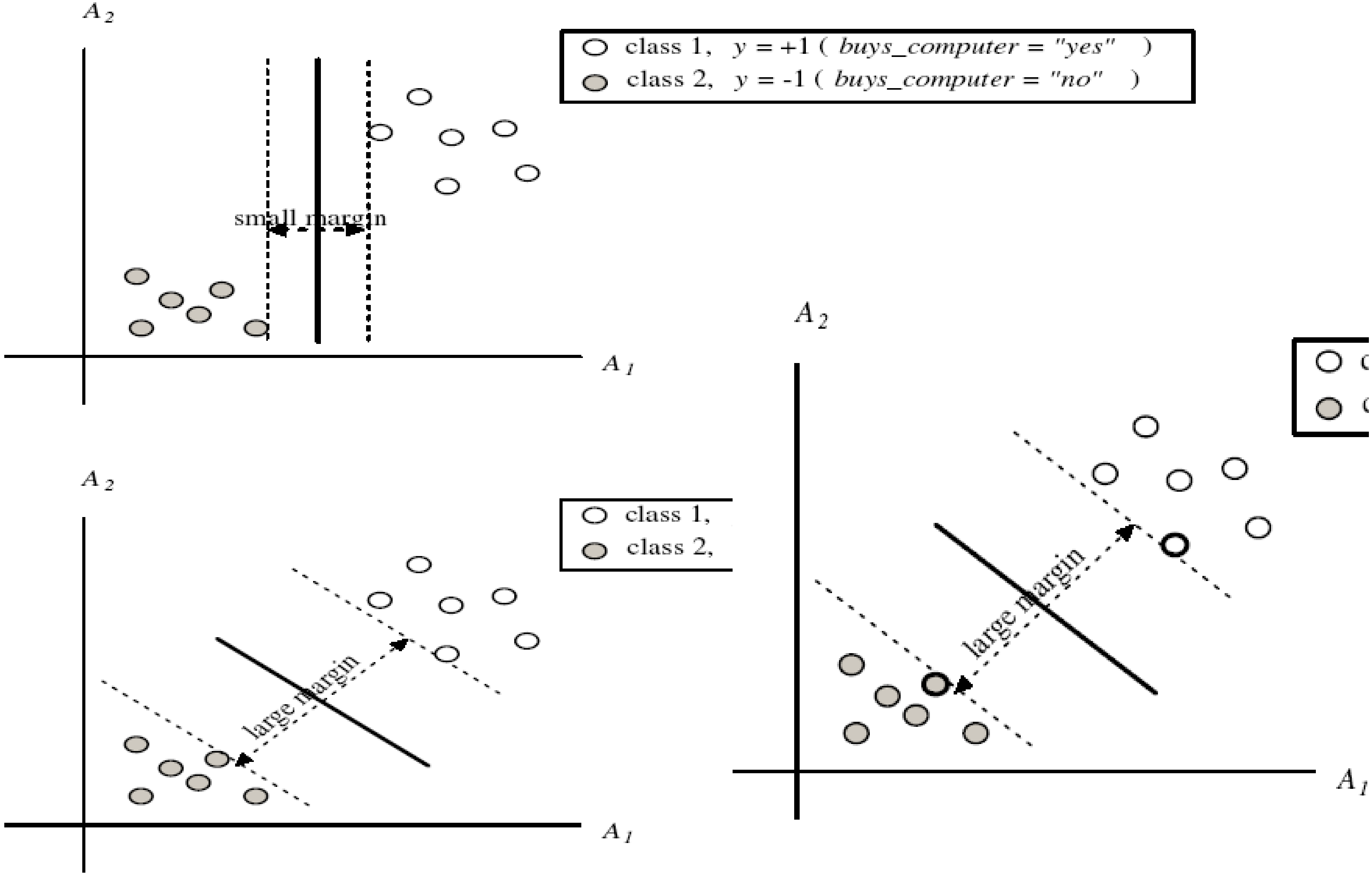
SVM—History and Applications

- 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

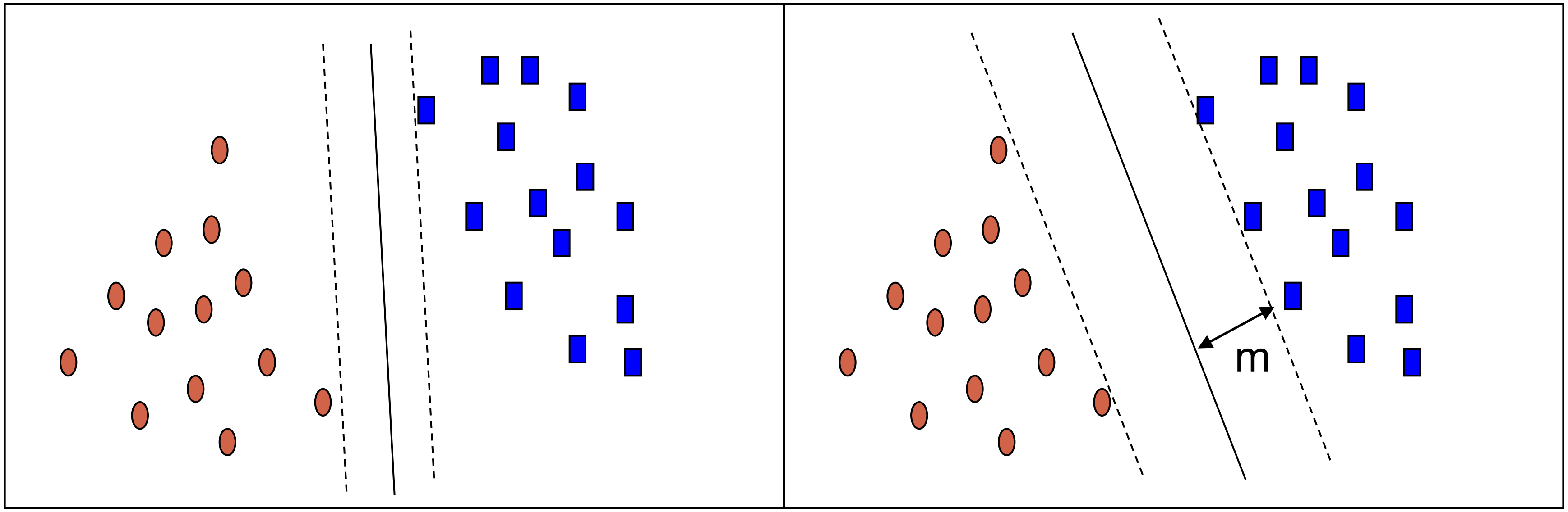
SVM—General Philosophy



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

  $$w_0 + w_1 x_1 + w_2 x_2 = 0$$

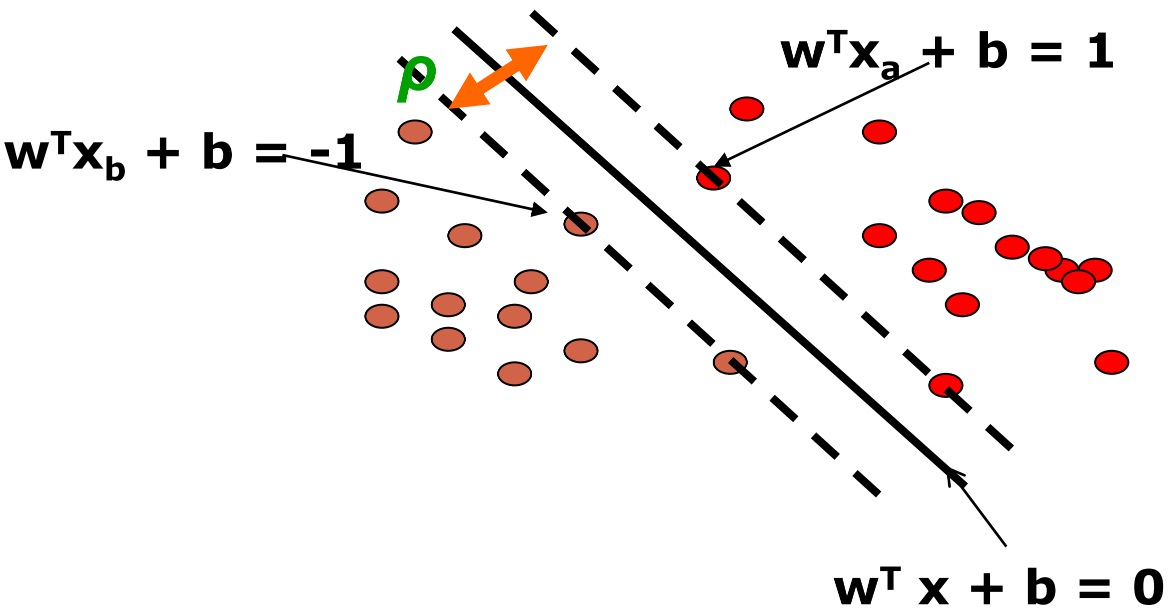
- The hyperplane defining the sides of the margin:

  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

Maximum Margin Calculation

- **w**: decision hyperplane normal vector
- **x**$_i$: data point *i*
- y$_i$: class of data point *i* (+1 or -1)



$$\rho = \frac{2}{||w||}$$

SVM as a Quadratic Programming

- QP

Objective: Find $\mathbf{w}$ and b such that $\rho = \frac{2}{||\boldsymbol{w}||}$ is maximized;

Constraints: For all $\{(\mathbf{x_i}, y_i)\}$

$\mathbf{w}^T\mathbf{x_i} + b \geq 1$ if y_i=1;

$\mathbf{w}^T\mathbf{x_i} + b \leq -1$ if $y_i = -1$

- A better form

Objective: Find $\mathbf{w}$ and b such that $\boldsymbol{\Phi}(\mathbf{w}) = ½\, \mathbf{w}^T\mathbf{w}$ is minimized;

Constraints: for all $\{(\mathbf{x_i}, y_i)\}$: $y_i (\mathbf{w}^T\mathbf{x_i} + b) \geq 1$

Solve QP

- This is now optimizing a *quadratic* function subject to *linear* constraints
- Quadratic optimization problems are a well-known class of mathematical programming problem, and many (intricate) algorithms exist for solving them (with many special ones built for SVMs)
- The solution involves constructing a *dual problem* where a *Lagrange multiplier* α_i is associated with every constraint in the primary problem:

Primal Form and Dual Form

Primal

Objective: Find $\mathbf{w}$ and b such that $\mathbf{\Phi}(\mathbf{w}) = ½\, \mathbf{w}^T\mathbf{w}$ is minimized;

Constraints: for all $\{(\mathbf{x_i}, y_i)\}$: $y_i(\mathbf{w}^T\mathbf{x_i} + b) \geq 1$

Equivalent under some conditions: KKT conditions

Dual

Objective: Find $\alpha_1 \ldots \alpha_n$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - ½\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$ is maximized and

Constraints
(1) $\Sigma\alpha_i y_i = 0$
(2) $\alpha_i \geq 0$ for all α_i

- More derivations: http://cs229.stanford.edu/notes/cs229-notes3.pdf

The Optimization Problem Solution

- The solution has the form:

$$\mathbf{w} = \Sigma \alpha_i y_i \mathbf{x_i} \qquad b = y_k - \mathbf{w}^{\mathbf{T}}\mathbf{x_k} \text{ for any } \mathbf{x_k} \text{ such that } \alpha_k \neq 0$$

- Each non-zero α_i indicates that corresponding $\mathbf{x_i}$ is a support vector.
- Then the classifying function will have the form:

$$f(\mathbf{x}) = \Sigma \alpha_i y_i \mathbf{x_i}^{\mathbf{T}}\mathbf{x} + b$$

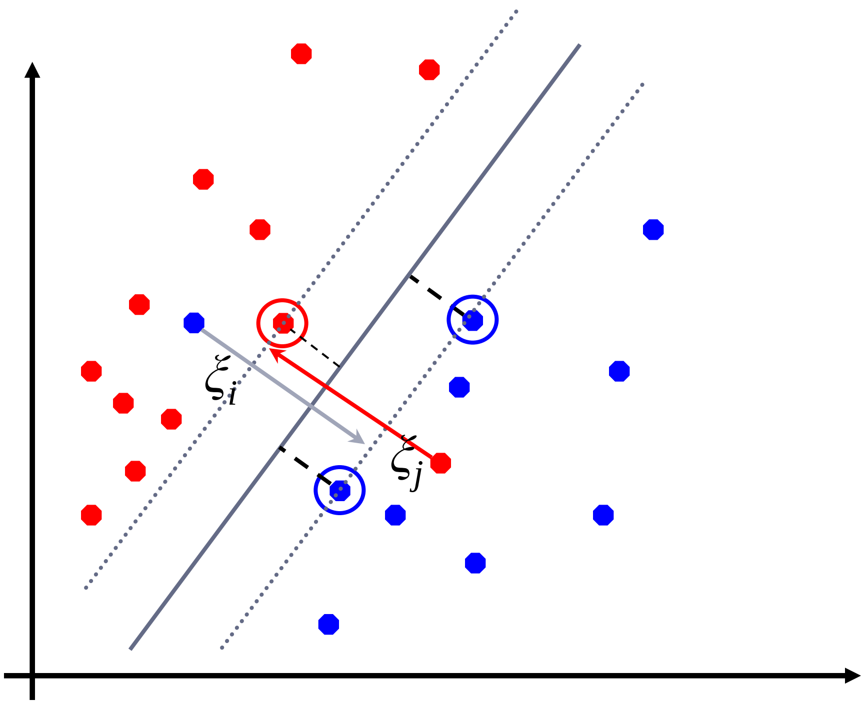
- Notice that it relies on an *inner product* between the test point **x** and the support vectors $\mathbf{x_i}$
  - We will return to this later.
- Also keep in mind that solving the optimization problem involved computing the inner products $\mathbf{x_i}^{\mathbf{T}}\mathbf{x_j}$ between all pairs of training points.

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

Soft Margin Classification

- If the training data is not linearly separable, *slack variables* ξ_i can be added to allow misclassification of difficult or noisy examples.
- Allow some errors
  - Let some points be moved to where they belong, at a cost
- Still, try to minimize training set errors, and to place hyperplane "far" from each class (large margin)



Soft Margin Classification Mathematically

- The old formulation:

Find **w** and b such that

$\mathbf{\Phi(w)} = ½\ \mathbf{w}^T\mathbf{w}$ is minimized and for all $\{(\mathbf{x_i}, y_i)\}$

$y_i\ (\mathbf{w^T x_i} + \mathrm{b}) \geq 1$

- The new formulation incorporating slack variables:

Find **w** and b such that

$\mathbf{\Phi(w)} = ½\ \mathbf{w}^T\mathbf{w} + C\Sigma\xi_i$ is minimized and for all $\{(\mathbf{x_i}, y_i)\}$

$y_i\ (\mathbf{w^T x_i} + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for all i

- Parameter *C* can be viewed as a way to control overfitting
  - A regularization term (L1 regularization)

Soft Margin Classification – Solution

- The dual problem for soft margin classification:

$$
\begin{aligned}
&\text{Find } \alpha_1 \ldots \alpha_N \text{ such that} \\
&\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \tfrac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x_i}^{\mathbf{T}}\mathbf{x_j} \text{ is maximized and} \\
&(1)\ \ \Sigma\alpha_i y_i = 0 \\
&(2)\ \ 0 \leq \alpha_i \leq C \text{ for all } \alpha_i
\end{aligned}
$$

- Neither slack variables ξ_i nor their Lagrange multipliers appear in the dual problem!
- Again, $\mathbf{x_i}$ with non-zero α_i will be support vectors.
- Solution to the dual problem is:

$$
\begin{aligned}
&\mathbf{w} = \Sigma\alpha_i y_i \mathbf{x_i} \\
&b = y_k(1 - \xi_k) - \mathbf{w}^{\mathbf{T}}\mathbf{x}_k \text{ where } k = \underset{k'}{\operatorname{argmax}}\ \alpha_{k'}
\end{aligned}
$$

w is not needed explicitly for classification!

$$
f(\mathbf{x}) = \Sigma\alpha_i y_i \mathbf{x_i}^{\mathbf{T}}\mathbf{x} + b
$$

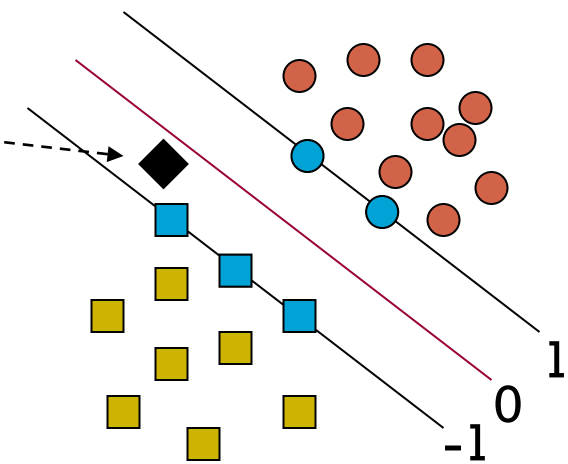
Classification with SVMs

- Given a new point **x**, we can score its projection onto the hyperplane normal:
  - I.e., compute score: $\mathbf{w}^{\mathrm{T}}\mathbf{x} + b = \Sigma \alpha_i y_i \mathbf{x}_{\mathrm{i}}^{\mathrm{T}}\mathbf{x} + b$
    - Decide class based on whether < or > 0
  - Can set confidence threshold *t*.

Score > *t*: yes

Score < -*t*: no

Else: don't know



Linear SVMs: Summary

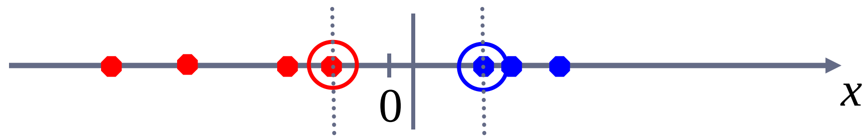
- The classifier is a *separating hyperplane.*
- The most "important" training points are the support vectors; they define the hyperplane.
- Quadratic optimization algorithms can identify which training points $\mathbf{x_i}$ are support vectors with non-zero Lagrangian multipliers α_i.
- Both in the dual formulation of the problem and in the solution, training points appear only inside inner products:

Find $\alpha_1 \ldots \alpha_N$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \boxed{\mathbf{x_i}^T\mathbf{x_j}}$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $0 \leq \alpha_i \leq C$ for all α_i

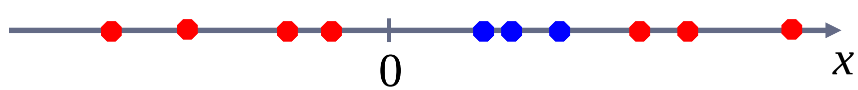
$$f(\mathbf{x}) = \Sigma\alpha_i y_i \boxed{\mathbf{x_i}^T\mathbf{x}} + b$$

Non-linear SVMs

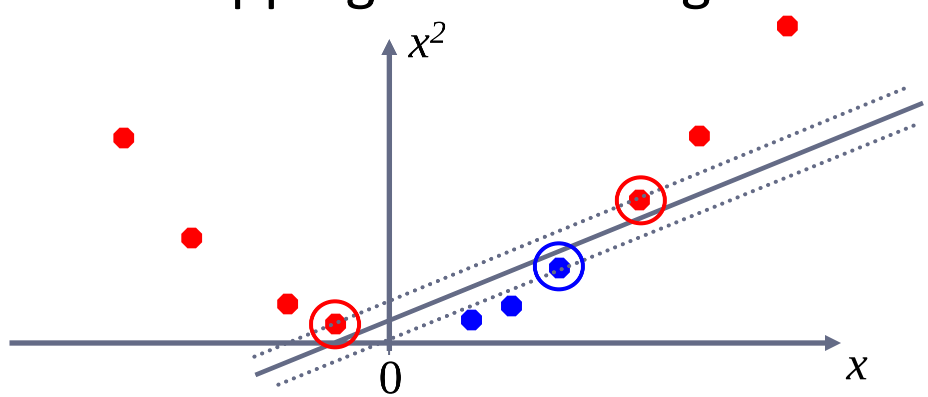
- Datasets that are linearly separable (with some noise) work out great:



- But what are we going to do if the dataset is just too hard?

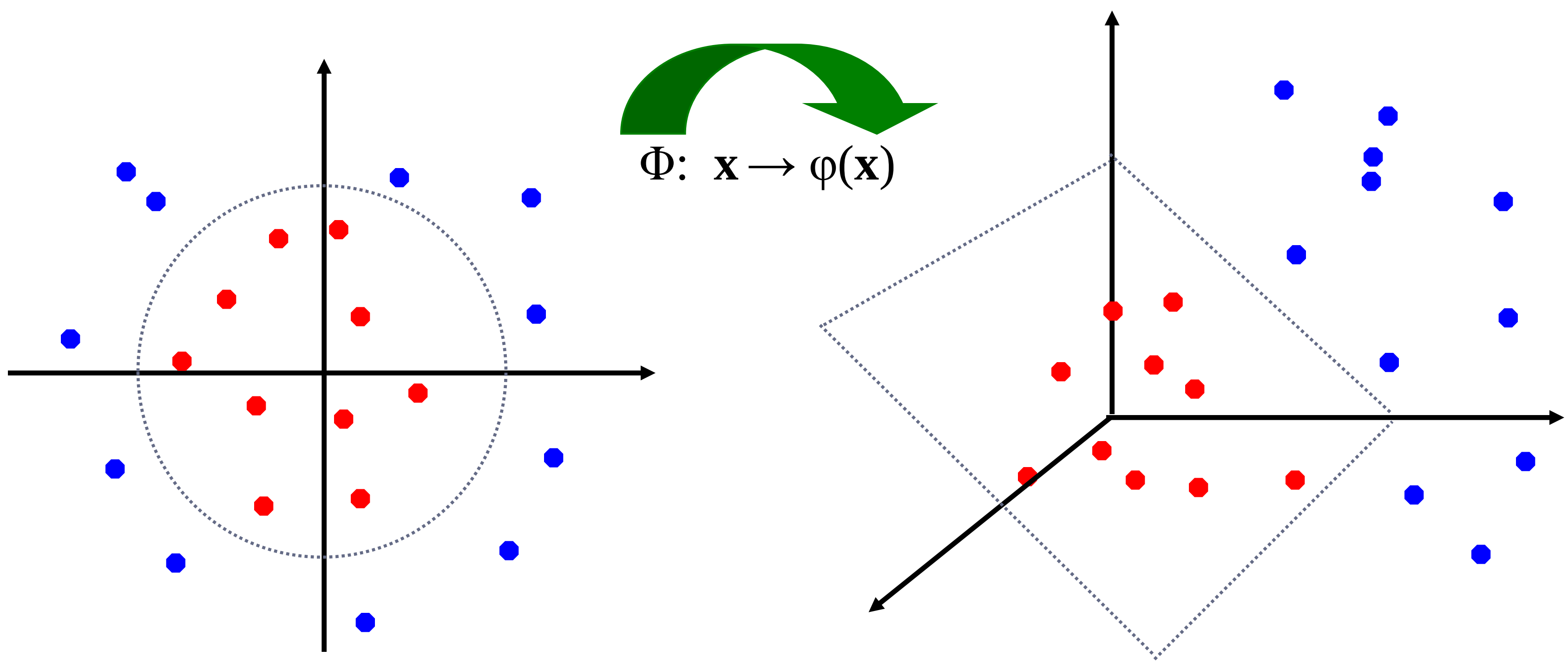


- How about … mapping data to a higher-dimensional space:



Non-linear SVMs: Feature spaces

- General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:



The "Kernel Trick"

- The linear classifier relies on an inner product between vectors $K(\mathbf{x_i},\mathbf{x_j})=\mathbf{x_i}^\mathsf{T}\mathbf{x_j}$
- If every data point is mapped into high-dimensional space via some transformation Φ: $\mathbf{x} \rightarrow \phi(\mathbf{x})$, the inner product becomes:

$$K(\mathbf{x_i},\mathbf{x_j}) = \phi(\mathbf{x_i})^\mathsf{T}\phi(\mathbf{x_j})$$

- A *kernel function* is some function that corresponds to an inner product in some expanded feature space.
- Example:

2-dimensional vectors $\mathbf{x}=[x_1 \;\; x_2]$; let $K(\mathbf{x_i},\mathbf{x_j})=(1 + \mathbf{x_i}^\mathsf{T}\mathbf{x_j})^2$,

Need to show that $K(\mathbf{x_i},\mathbf{x_j}) = \phi(\mathbf{x_i})^\mathsf{T}\phi(\mathbf{x_j})$:

$K(\mathbf{x_i},\mathbf{x_j})=(1 + \mathbf{x_i}^\mathsf{T}\mathbf{x_j})^2 = 1 + x_{i1}^2 x_{j1}^2 + 2\, x_{i1}x_{j1}\, x_{i2}x_{j2} + x_{i2}^2 x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2} =$

$= [1 \;\; x_{i1}^2 \;\; \sqrt{2}\, x_{i1}x_{i2} \;\; x_{i2}^2 \;\; \sqrt{2}x_{i1} \;\; \sqrt{2}x_{i2}]^\mathsf{T} [1 \;\; x_{j1}^2 \;\; \sqrt{2}\, x_{j1}x_{j2} \;\; x_{j2}^2 \;\; \sqrt{2}x_{j1} \;\; \sqrt{2}x_{j2}]$

$= \phi(\mathbf{x_i})^\mathsf{T}\phi(\mathbf{x_j})$ where $\phi(\mathbf{x}) = [1 \;\; x_1^2 \;\; \sqrt{2}\, x_1x_2 \;\; x_2^2 \;\; \sqrt{2}x_1 \;\; \sqrt{2}x_2]$

SVM: Different Kernel functions

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

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

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

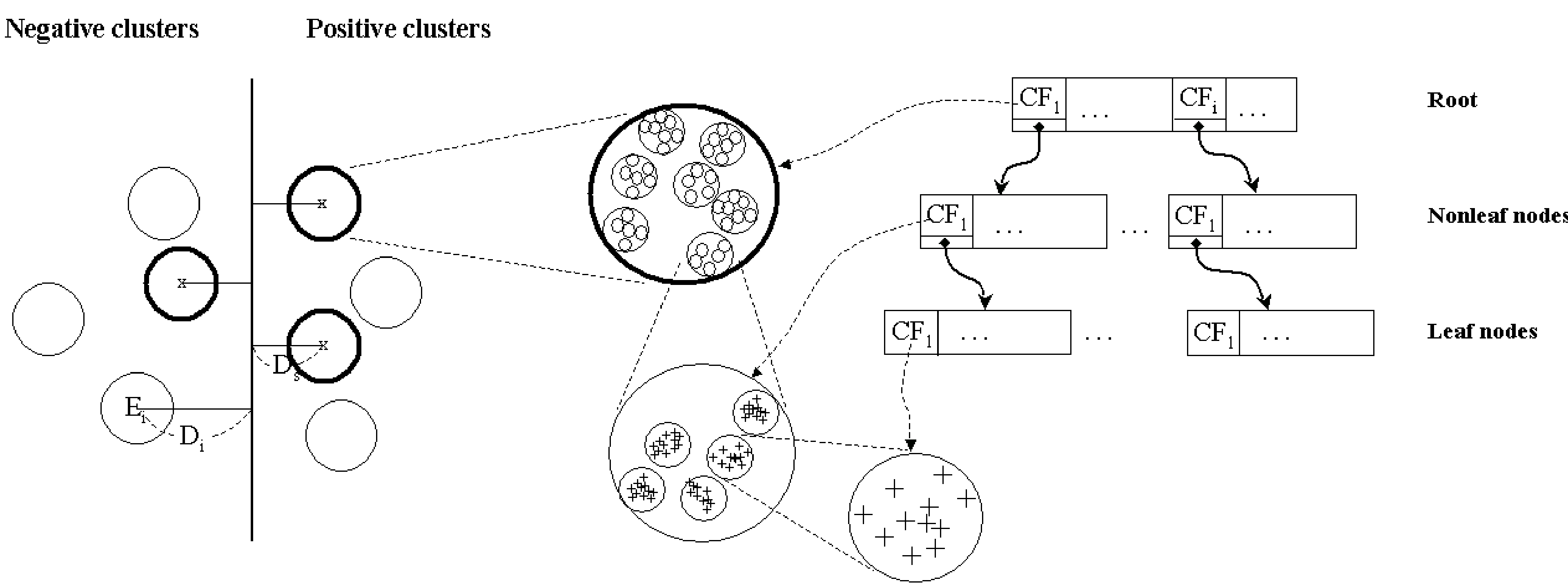
$$\textbf{Sigmoid kernel}: \quad K(X_i, X_j) = \tanh(\kappa X_i \cdot X_j - \delta)$$

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

Scaling SVM by Hierarchical Micro-Clustering

- SVM is not scalable to the number of data objects in terms of training time and memory usage
- H. Yu, J. Yang, and J. Han, "Classifying Large Data Sets Using SVM with Hierarchical Clusters", KDD'03)
- CB-SVM (Clustering-Based SVM)
  - Given limited amount of system resources (e.g., memory), maximize the SVM performance in terms of accuracy and the training speed
  - Use micro-clustering to effectively reduce the number of points to be considered
  - At deriving support vectors, de-cluster micro-clusters near "candidate vector" to ensure high classification accuracy

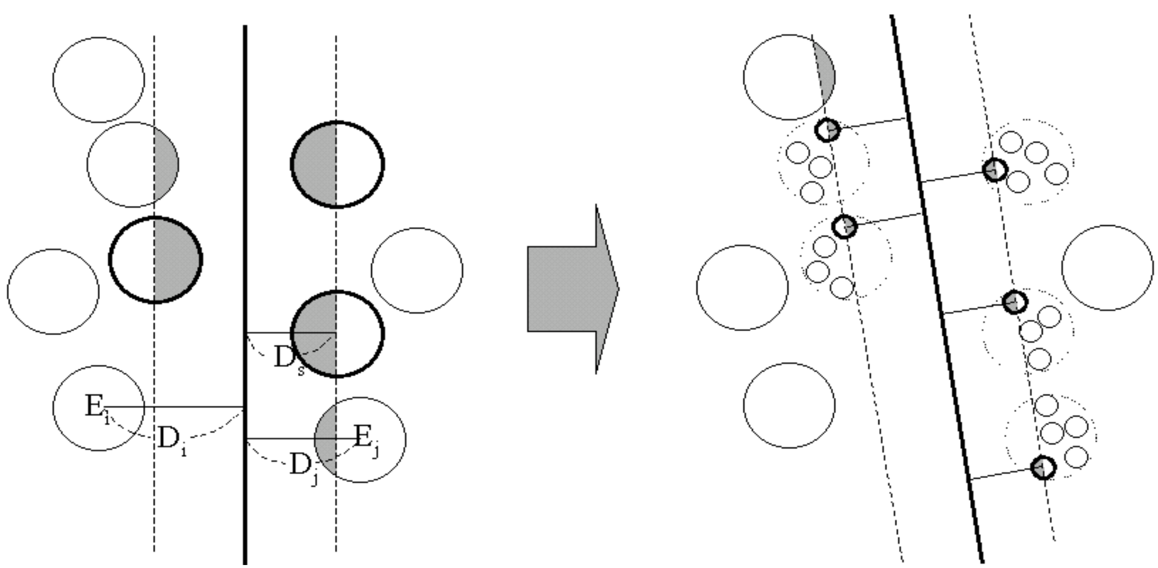
CF-Tree: Hierarchical Micro-cluster



- Read the data set once, construct a statistical summary of the data (i.e., hierarchical clusters) given a limited amount of memory
- Micro-clustering: Hierarchical indexing structure
  - provide finer samples closer to the boundary and coarser samples farther from the boundary

Selective Declustering: Ensure High Accuracy

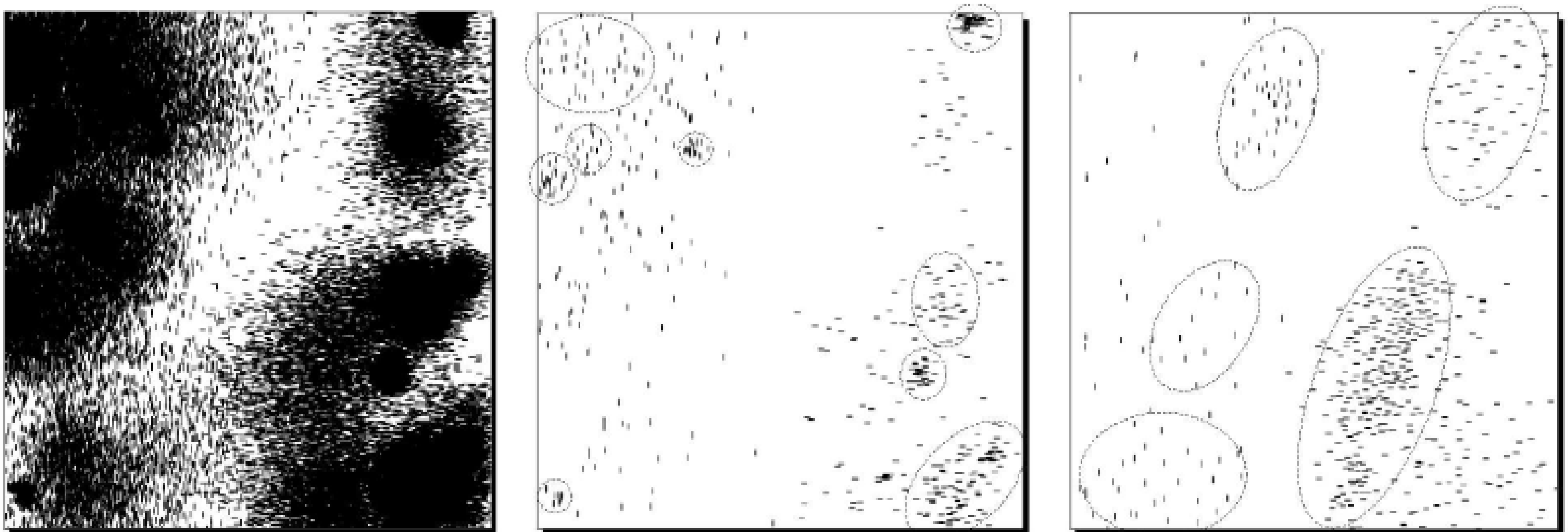
- CF tree is a suitable base structure for selective declustering
- De-cluster only the cluster E_i such that
  - $D_i - R_i < D_s$, where D_i is the distance from the boundary to the center point of E_i and R_i is the radius of E_i
  - Decluster only the cluster whose subclusters have possibilities to be the support cluster of the boundary
    - "Support cluster": The cluster whose centroid is a support vector



CB-SVM Algorithm: Outline

- Construct two CF-trees from positive and negative data sets independently
  - Need one scan of the data set
- Train an SVM from the centroids of the root entries
- De-cluster the entries near the boundary into the next level
  - The children entries de-clustered from the parent entries are accumulated into the training set with the non-declustered parent entries
- Train an SVM again from the centroids of the entries in the training set
- Repeat until nothing is accumulated

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 8&9. Classification: Part 2

- Neural Networks
- Support Vector Machine
- Summary

- Artificial Neural Networks
  - Single unit
  - Multilayer neural networks
  - Backpropagation algorithm
- Support Vector Machine
  - Support vectors
  - Maximum marginal hyperplane
  - Linear separable
  - Linear inseparable
  - Kernel tricks

Announcement

- Homework 1
  - Apriori algorithm: some length-l candidate can be pruned by checking whether all its sub-patterns with length-(l-1) are in frequent
  - FP-growth: Need to scan DB twice
- Proposal due tomorrow
  - Report
  - Sign up meeting time with me on Wednesday afternoon or Thursday morning
- Homework 2 will be out tomorrow
- No class next week
  - The make-up class is canceled
  - With homework 2 due on next Friday
- Midterm exam (Feb. 25, the same location, same time as classes, 6-8pm)
  - You can take a A4 “cheating sheet”
  - Covers to the content taught by today

References (1)

- C. M. Bishop, Neural Networks for Pattern Recognition. Oxford University Press, 1995
- C. J. C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2): 121-168, 1998
- H. Cheng, X. Yan, J. Han, and C.-W. Hsu, Discriminative Frequent pattern Analysis for Effective Classification, ICDE'07
- H. Cheng, X. Yan, J. Han, and P. S. Yu, Direct Discriminative Pattern Mining for Effective Classification, ICDE'08
- N. Cristianini and J. Shawe-Taylor, Introduction to Support Vector Machines and Other Kernel-Based Learning Methods, Cambridge University Press, 2000
- A. J. Dobson. An Introduction to Generalized Linear Models. Chapman & Hall, 1990
- G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. KDD'99

References (2)

- 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.
- V. Kecman, Learning and Soft Computing: Support Vector Machines, Neural Networks, and Fuzzy Logic, MIT Press, 2001
- W. Li, J. Han, and J. Pei, CMAR: Accurate and Efficient Classification Based on Multiple Class-Association Rules, ICDM'01
- T.-S. Lim, W.-Y. Loh, and Y.-S. Shih. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. Machine Learning, 2000

References (3)

- B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining, p. 80-86, KDD'98.
- T. M. Mitchell. Machine Learning. McGraw Hill, 1997.
- D.E. Rumelhart, and J.L. McClelland, editors, Parallel Distributed Processing, MIT Press, 1986.
- P. Tan, M. Steinbach, and V. Kumar. Introduction to Data Mining. Addison Wesley, 2005.
- S. M. Weiss and N. Indurkhya. Predictive Data Mining. Morgan Kaufmann, 1997.
- I. H. Witten and E. Frank. Data Mining: Practical Machine Learning Tools and Techniques, 2ed. Morgan Kaufmann, 2005.
- X. Yin and J. Han. CPAR: Classification based on predictive association rules. SDM'03
- H. Yu, J. Yang, and J. Han. Classifying large data sets using SVM with hierarchical clusters. KDD'03.