

CS6220: DATA MINING TECHNIQUES

Matrix Data: Classification: Part 3

Instructor: Yizhou Sun

yzsun@ccs.neu.edu

February 3, 2016


Announcements

- Homework 2 will be out tomorrow

Methods to Learn

	Matrix Data	Text Data	Set Data	Sequence Data	Time Series	Graph & Network	Images
Classification	Decision Tree; Naïve Bayes; Logistic Regression SVM; kNN			HMM		Label Propagation	Neural Network
Clustering	K-means; hierarchical clustering; DBSCAN; Mixture Models; kernel k-means*	PLSA				SCAN; Spectral Clustering	
Frequent Pattern Mining			Apriori; FP-growth	GSP; PrefixSpan			
Prediction	Linear Regression				Autoregression	Collaborative Filtering	
Similarity Search					DTW	P-PageRank	
Ranking						PageRank	

Matrix Data: Classification: Part 3

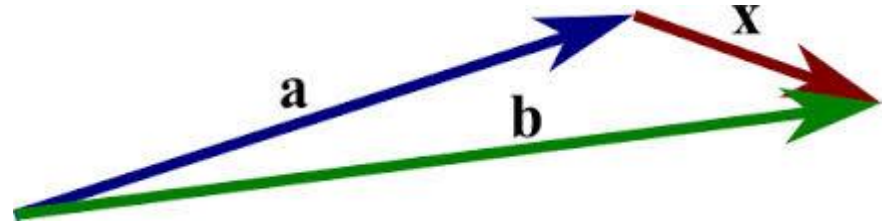
- SVM (Support Vector Machine) 
- kNN (k Nearest Neighbor)
- Other Issues
- Summary

Math Review

- Vector

- $\mathbf{x} = (x_1, x_2, \dots, x_n)$

- Subtracting two vectors: $\mathbf{x} = \mathbf{b} - \mathbf{a}$

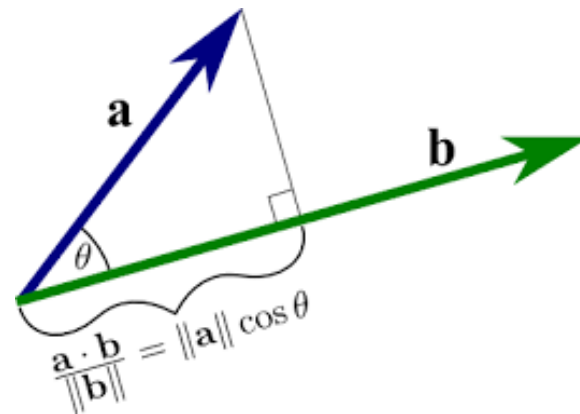


- Dot product

- $\mathbf{a} \cdot \mathbf{b} = \sum a_i b_i$

- Geometric interpretation: projection

- If \mathbf{a} and \mathbf{b} are orthogonal, $\mathbf{a} \cdot \mathbf{b} = 0$



Math Review (Cont.)

- Plane/Hyperplane

- $a_1x_1 + a_2x_2 + \cdots + a_nx_n = c$

- Line (n=2), plane (n=3), hyperplane (higher dimensions)

- Normal of a plane

- $\mathbf{n} = (a_1, a_2, \dots, a_n)$

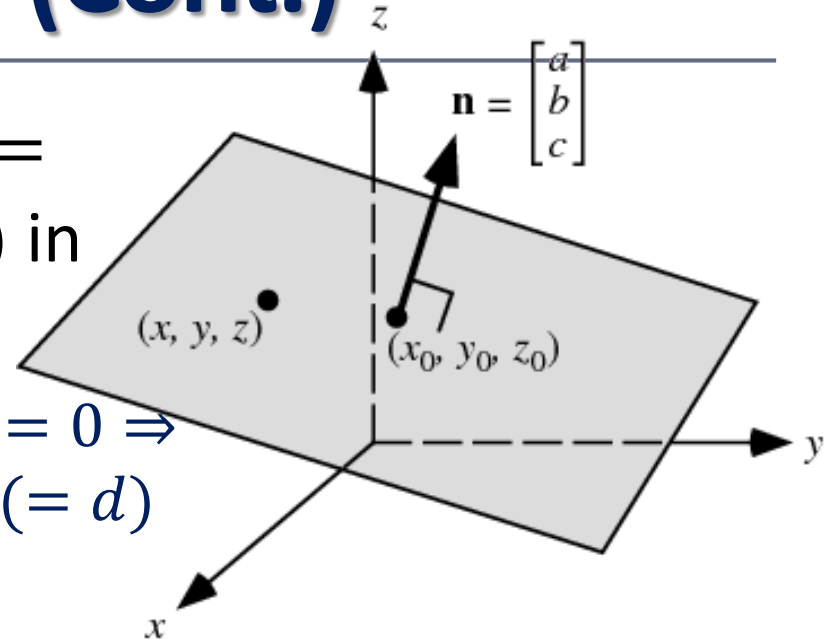
- a vector which is perpendicular to the surface

Math Review (Cont.)

- Define a plane using normal $\mathbf{n} = (a, b, c)$ and a point (x_0, y_0, z_0) in the plane:

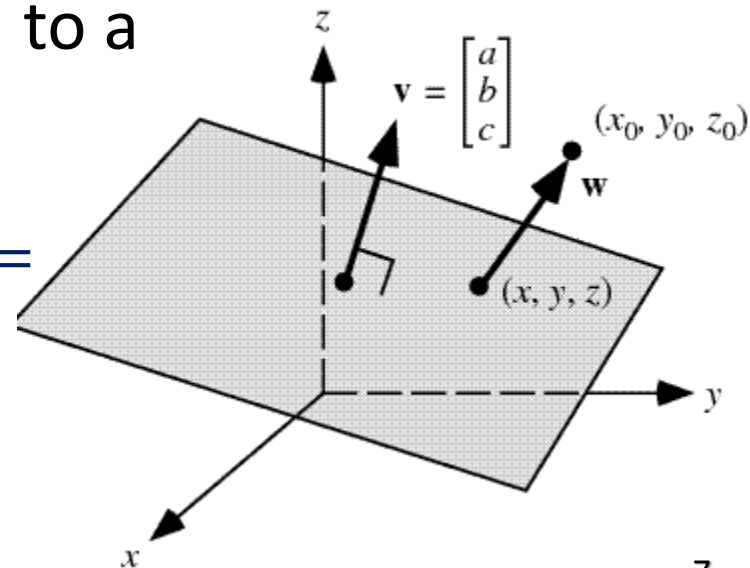
- $$(a, b, c) \cdot (x_0 - x, y_0 - y, z_0 - z) = 0 \Rightarrow$$

$$ax + by + cz = ax_0 + by_0 + cz_0 (= d)$$



- Distance from a point (x_0, y_0, z_0) to a plane $ax + by + cz = d$

- $$\frac{\left| (x_0 - x, y_0 - y, z_0 - z) \cdot \frac{(a, b, c)}{\|(a, b, c)\|} \right|}{\frac{|ax_0 + by_0 + cz_0 - d|}{\sqrt{a^2 + b^2 + c^2}}} =$$



Linear Classifier

- Given a training dataset $\{\mathbf{x}_i, y_i\}_{i=1}^N$
 - A separating hyperplane can be written as a linear combination of attributes

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

where $\mathbf{W} = \{w_1, w_2, \dots, w_n\}$ is a weight vector and b a scalar (bias)

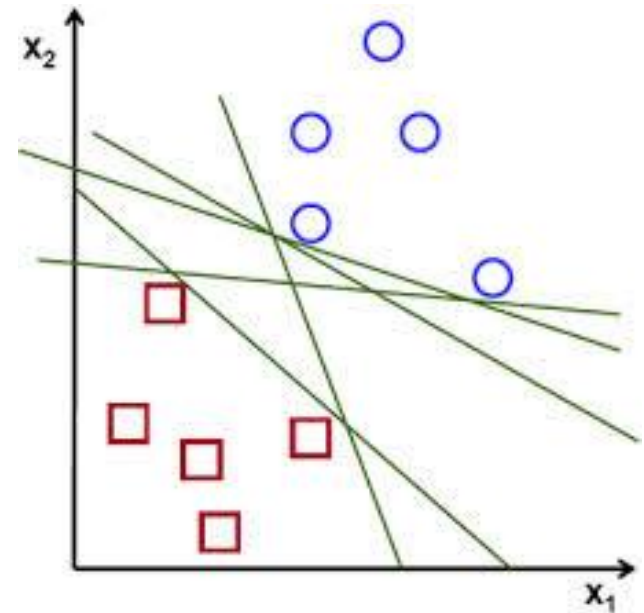
- For 2-D it can be written as

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

- Classification:

$$w_0 + w_1 x_1 + w_2 x_2 > 0 \Rightarrow y_i = +1$$

$$w_0 + w_1 x_1 + w_2 x_2 \leq 0 \Rightarrow y_i = -1$$



Perceptron

$$\mathbf{x} = (\mathbf{1}, x_1, x_2, \dots, x_d)^T \quad \mathbf{w} = (\omega_0, \omega_1, \omega_2, \dots, \omega_d)^T$$
$$y = \{1, -1\} \quad \alpha \in (0, 1] \text{ (learning rate)}$$

Initialize $\mathbf{w} = \mathbf{0}$ (can be any vector)

Repeat:

- For each training example (\mathbf{x}_i, y_i) :
 - Compute $\hat{y}_i = \text{sign}(\mathbf{w}^T \mathbf{x}_i)$
 - if $(y_i \neq \hat{y}_i)$ $\mathbf{w} = \mathbf{w} + \alpha(y_i \mathbf{x}_i)$

Until $(y_i = \hat{y}_i \quad \forall i = 1 \dots N)$

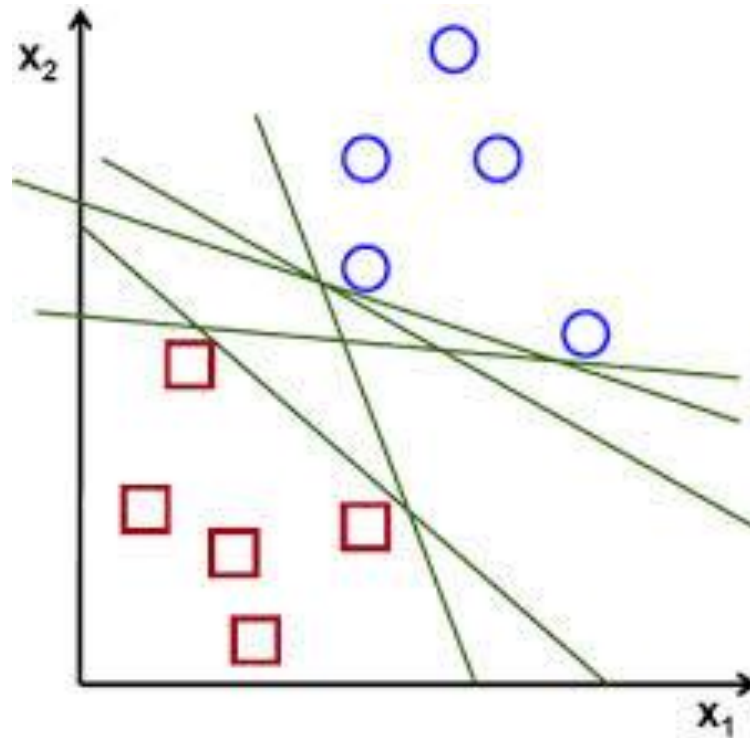
Return \mathbf{w}

Example

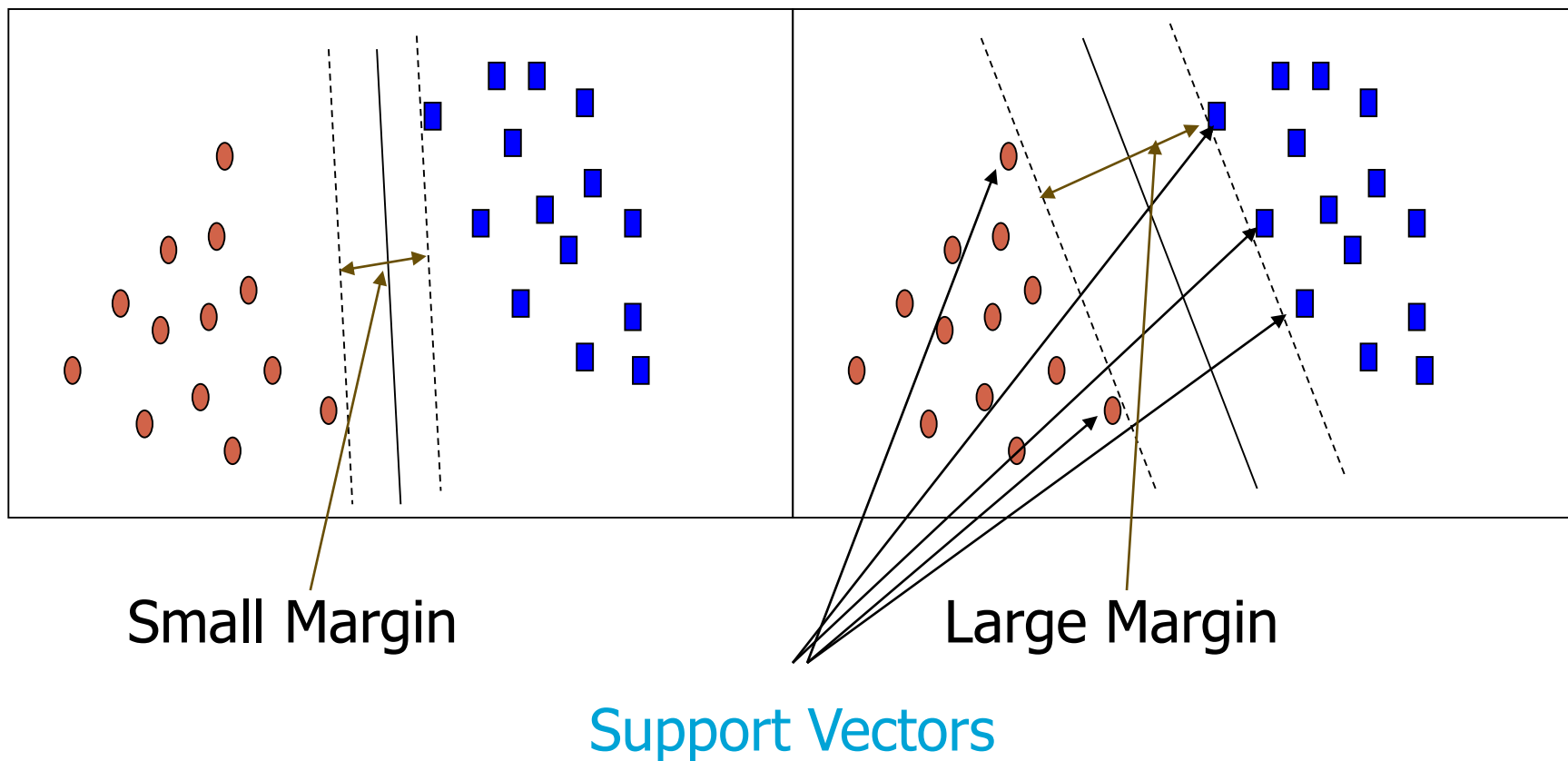
x0	x1	x2	true label	w before update	predicted label	w after update
1	0	1	Y	(0.0, 0.0, 0.0)	N	(0.9, 0.0, 0.9)
1	1	1	N	(0.9, 0.0, 0.9)	Y	(0.0, -0.9, 0.0)
1	0	0	Y	(0.0, -0.9, 0.0)	N	(0.9, -0.9, 0.0)
1	1	0	Y	(0.9, -0.9, 0.0)	N	(1.8, 0.0, 0.0)
1	0	1	Y	(1.8, 0.0, 0.0)	Y	(1.8, 0.0, 0.0)
1	1	1	N	(1.8, 0.0, 0.0)	Y	(0.9, -0.9, -0.9)
1	0	0	Y	(0.9, -0.9, -0.9)	Y	(0.9, -0.9, -0.9)
1	1	0	Y	(0.9, -0.9, -0.9)	N	(1.8, 0.0, -0.9)
1	0	1	Y	(1.8, 0.0, -0.9)	Y	(1.8, 0.0, -0.9)
1	1	1	N	(1.8, 0.0, -0.9)	Y	(0.9, -0.9, -1.8)
1	0	0	Y	(0.9, -0.9, -1.8)	Y	(0.9, -0.9, -1.8)
1	1	0	Y	(0.9, -0.9, -1.8)	N	(1.8, 0.0, -1.8)

Can we do better?

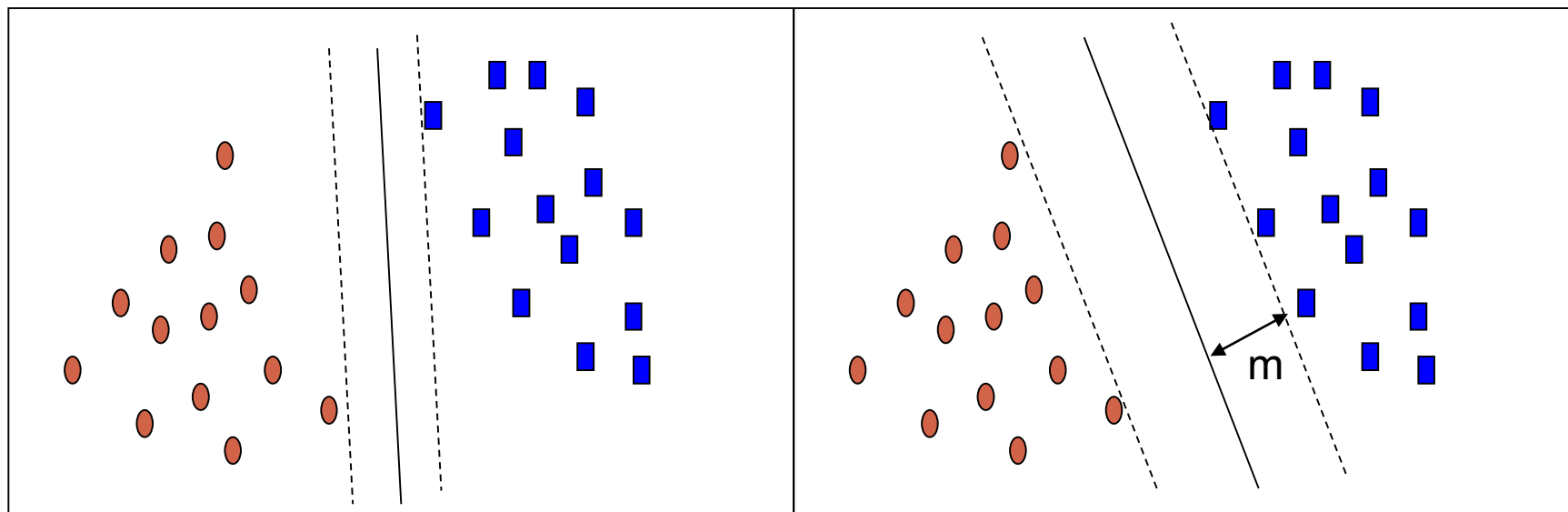
- Which hyperplane to choose?



SVM—Margins and Support Vectors



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

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

- The hyperplane defining the sides of the margin, e.g.,:

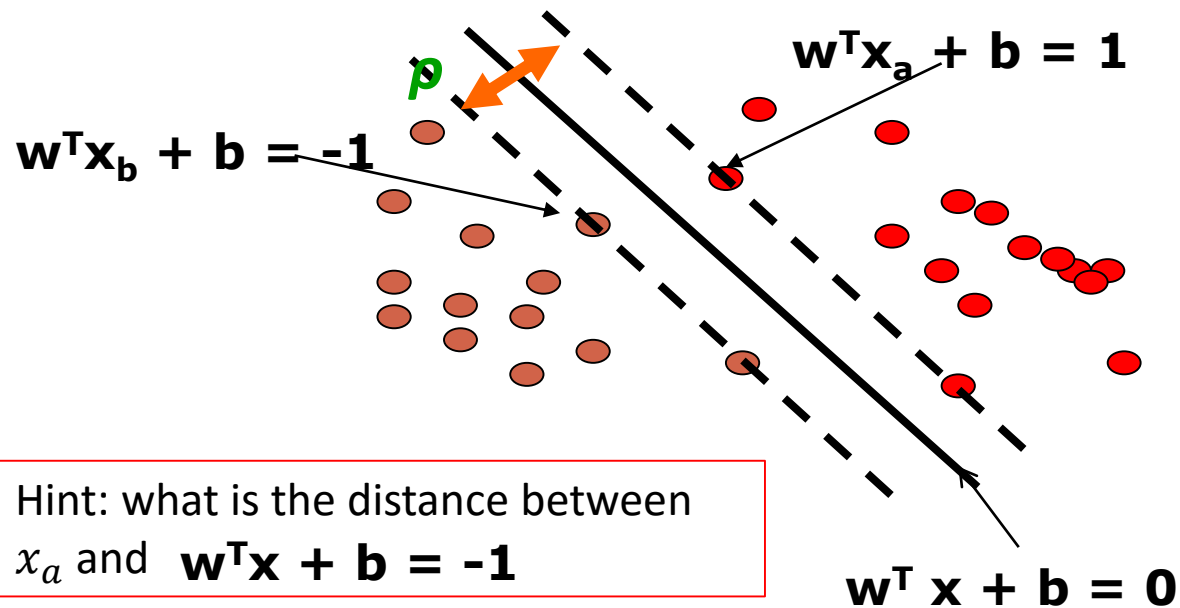
$$H_1: w_0 + w_1 x_1 + w_2 x_2 \geq 1 \quad \text{for } y_i = +1, \text{ and}$$

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

- 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 \rightarrow *Quadratic Programming (QP)* \rightarrow Lagrangian multipliers

Maximum Margin Calculation

- \mathbf{w} : decision hyperplane normal vector
- \mathbf{x}_i : data point i
- y_i : class of data point i (+1 or -1)



$$\text{margin: } \rho = \frac{2}{\|\mathbf{w}\|}$$

SVM as a Quadratic Programming

- QP

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

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

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

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

- A better form

Objective: Find \mathbf{w} and b such that $\Phi(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{w}$ 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_j is associated with every constraint in the primary problem:

Lagrange Formulation

Minimize

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^N \alpha_i (y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1)$$

Take the partial derivatives w.r.t \mathbf{w} , b :

$$\nabla_{\mathbf{w}} L = \mathbf{w} - \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i = 0 \implies \mathbf{w} = \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i$$

$$\frac{\partial L}{\partial b} = - \sum_{i=1}^N \alpha_i y_i = 0$$

Primal Form and Dual Form

Primal

Objective: Find \mathbf{w} and b such that $\Phi(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{w}$ 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 $\alpha_1 \dots \alpha_n$ such that $Q(\alpha) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$ is maximized and

Constraints

(1) $\sum \alpha_i y_i = 0$

(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:

$$\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i \quad b = y_k - \mathbf{w}^T \mathbf{x}_k \text{ for any } \mathbf{x}_k \text{ such that } \alpha_k \neq 0$$

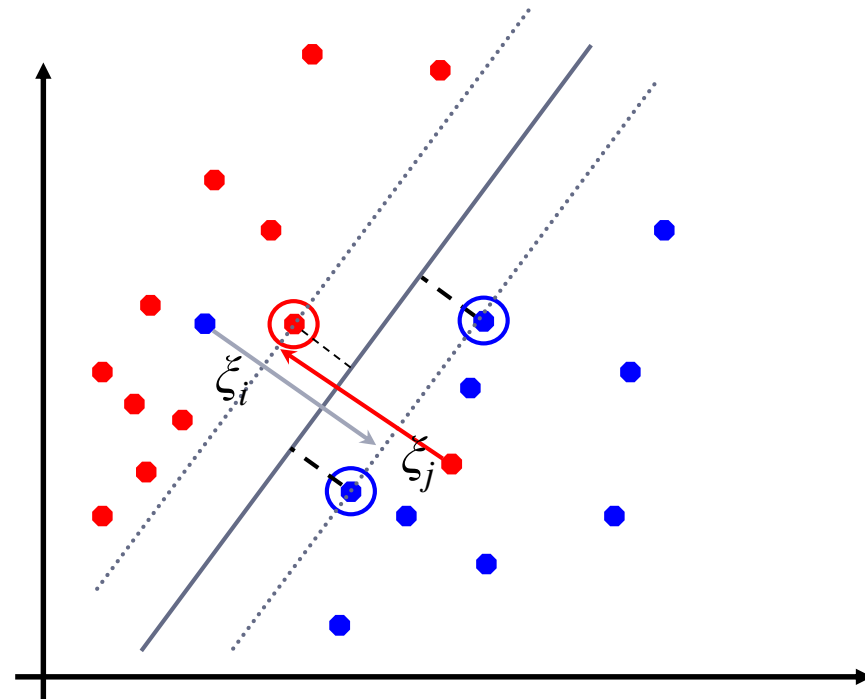
- Each non-zero α_i indicates that corresponding \mathbf{x}_i is a **support vector**.
- Then the classifying function will have the form:

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

- Notice that it relies on an *inner product* between the test point \mathbf{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 $\mathbf{x}_i^T \mathbf{x}_j$ between all pairs of training points.

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 \mathbf{w} and b such that

$$\Phi(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{w} \text{ is minimized and for all } \{(\mathbf{x}_i, y_i)\}$$
$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1$$

- The new formulation incorporating slack variables:

Find \mathbf{w} and b such that

$$\Phi(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum \xi_i \text{ is minimized and for all } \{(\mathbf{x}_i, y_i)\}$$
$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i \quad \text{and} \quad \xi_i \geq 0 \text{ for all } i$$

- 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:

Find $\alpha_1 \dots \alpha_N$ such that

$Q(\boldsymbol{\alpha}) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$ is maximized and

(1) $\sum \alpha_i y_i = 0$

(2) $0 \leq \alpha_i \leq C$ for all α_i

- 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:

$$\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i$$

$$b = y_k (1 - \xi_k) - \mathbf{w}^T \mathbf{x}_k \text{ where } k = \operatorname{argmax}_{k'} \alpha_{k'}$$

\mathbf{w} is not needed explicitly for classification!

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

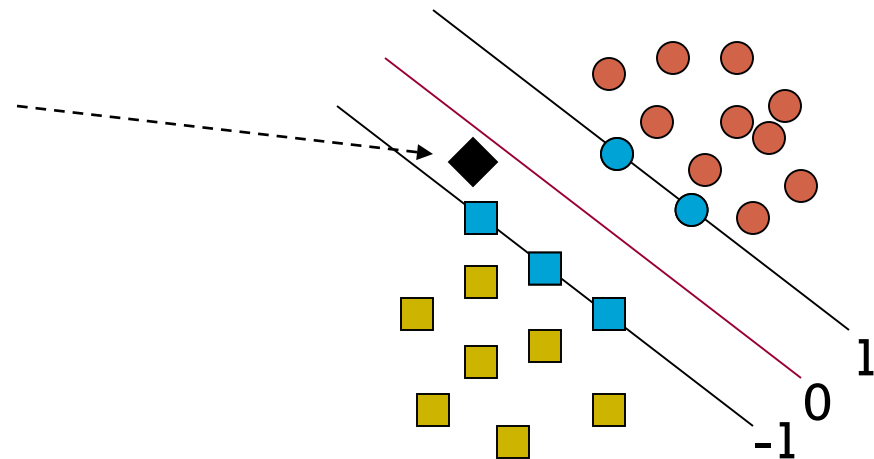
Classification with SVMs

- Given a new point \mathbf{x} , we can score its projection onto the hyperplane normal:
 - I.e., compute score: $\mathbf{w}^T \mathbf{x} + b = \sum \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b$
 - 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 $\alpha_1 \dots \alpha_N$ such that

$\mathbf{Q}(\boldsymbol{\alpha}) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$ is maximized and

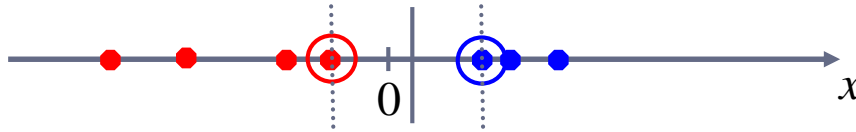
(1) $\sum \alpha_i y_i = 0$

(2) $0 \leq \alpha_i \leq C$ for all α_i

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

Non-linear SVMs

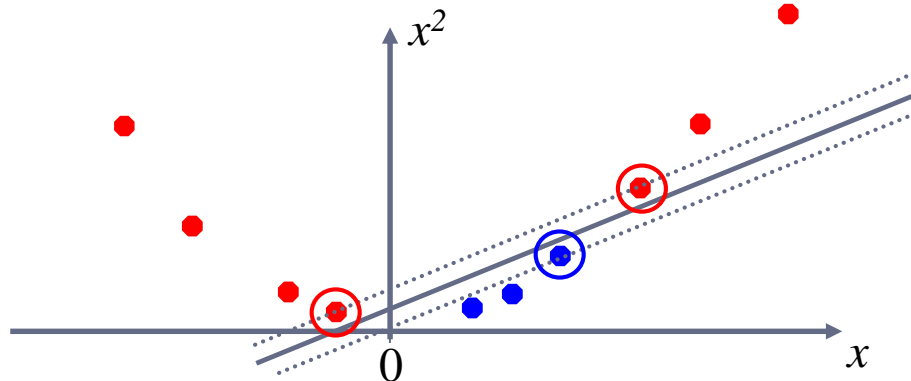
- Datasets that are linearly separable (with some noise) work out great:



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

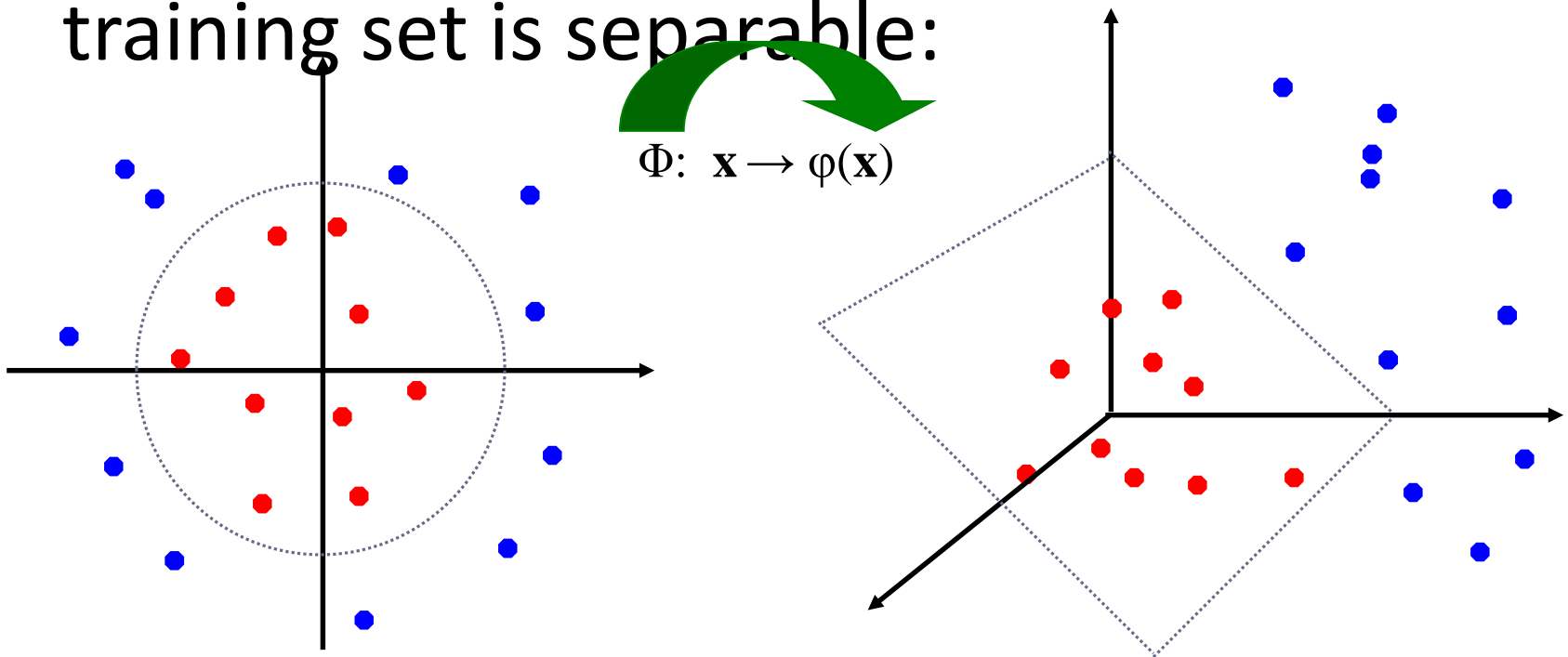


- How about ... mapping data to a higher-dimensional space:



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 $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
- If every data point is mapped into high-dimensional space via some transformation $\Phi: \mathbf{x} \rightarrow \phi(\mathbf{x})$, the inner product becomes:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

- 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 $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^2$,

Need to show that $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$:

$$\begin{aligned} K(\mathbf{x}_i, \mathbf{x}_j) &= (1 + \mathbf{x}_i^T \mathbf{x}_j)^2 = 1 + x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{j1} x_{i2} x_{j2} + x_{i2}^2 x_{j2}^2 + 2 x_{i1} x_{j1} + 2 x_{i2} x_{j2} = \\ &= [1 \ x_{i1}^2 \ \sqrt{2} x_{i1} x_{i2} \ x_{i2}^2 \ \sqrt{2} x_{i1} \ \sqrt{2} x_{i2}]^T [1 \ x_{j1}^2 \ \sqrt{2} x_{j1} x_{j2} \ x_{j2}^2 \ \sqrt{2} x_{j1} \ \sqrt{2} x_{j2}] \\ &= \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) \quad \text{where } \phi(\mathbf{x}) = [1 \ x_1^2 \ \sqrt{2} x_1 x_2 \ x_2^2 \ \sqrt{2} x_1 \ \sqrt{2} x_2] \end{aligned}$$

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., $K(\mathbf{X}_i, \mathbf{X}_j) = \Phi(\mathbf{X}_i)^\top \Phi(\mathbf{X}_j)$
- 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 : $K(\mathbf{X}_i, \mathbf{X}_j) = \tanh(\kappa \mathbf{X}_i \cdot \mathbf{X}_j - \delta)$

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

Non-linear SVM

- Replace inner-product with kernel functions
 - Optimization problem

Find $\alpha_1 \dots \alpha_N$ such that

$\mathbf{Q}(\boldsymbol{\alpha}) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j \mathbf{K}(\mathbf{x}_i, \mathbf{x}_j)$ is maximized and

(1) $\sum \alpha_i y_i = 0$

(2) $0 \leq \alpha_i \leq C$ for all α_i

- Decision boundary

$$f(\mathbf{x}) = \sum \alpha_i y_i \mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) + b$$

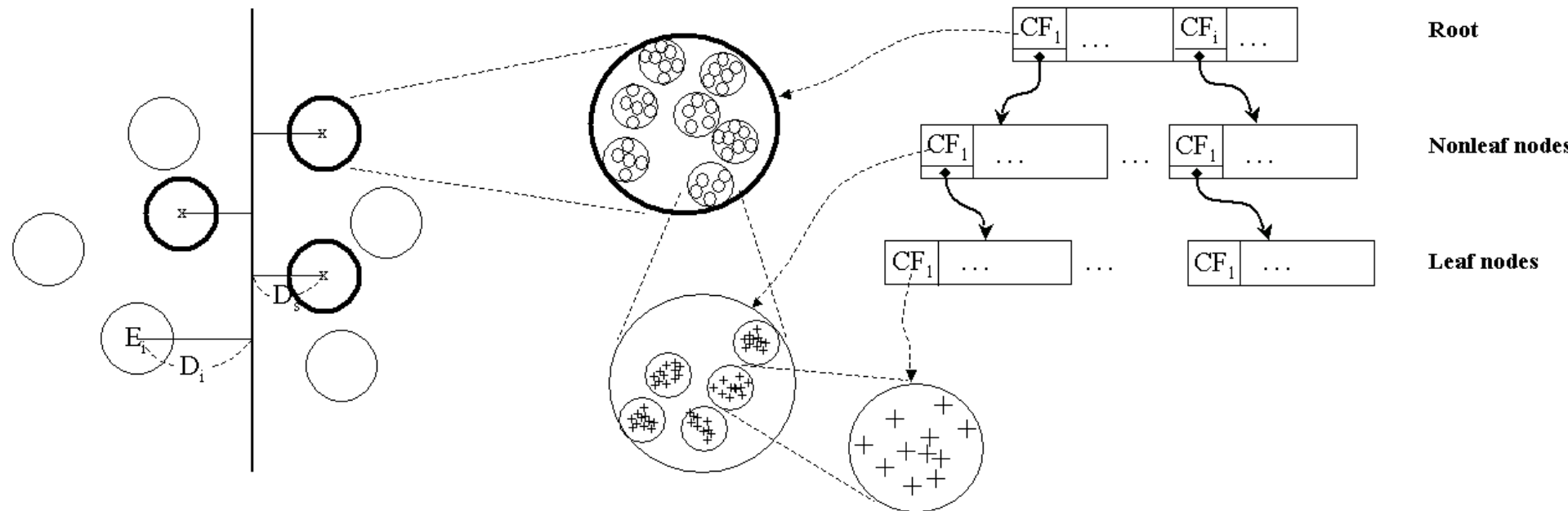
*Scaling SVM by Hierarchical Micro-Clustering

- 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

Negative clusters

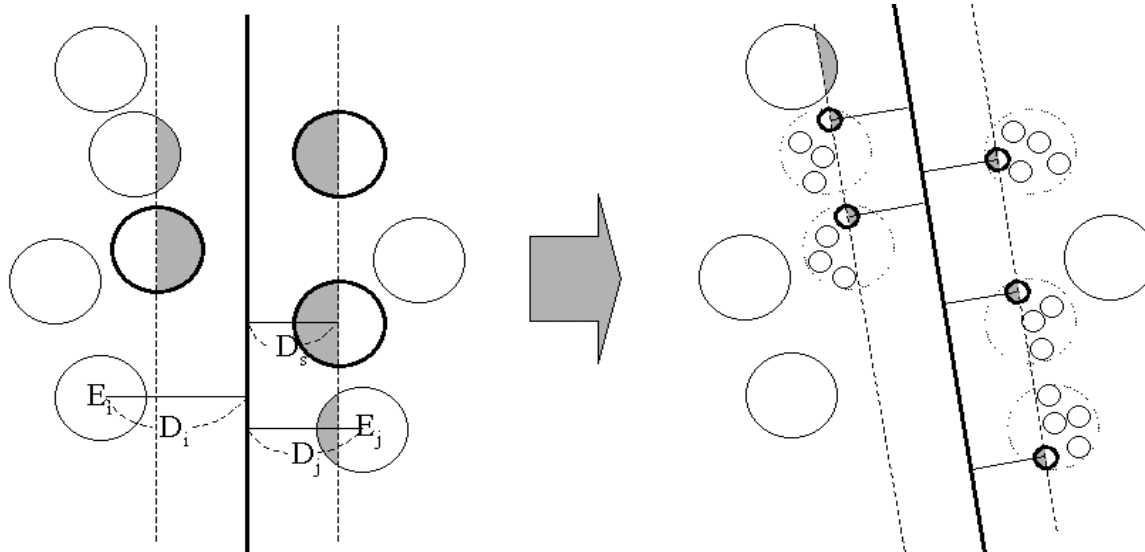
Positive clusters



- 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

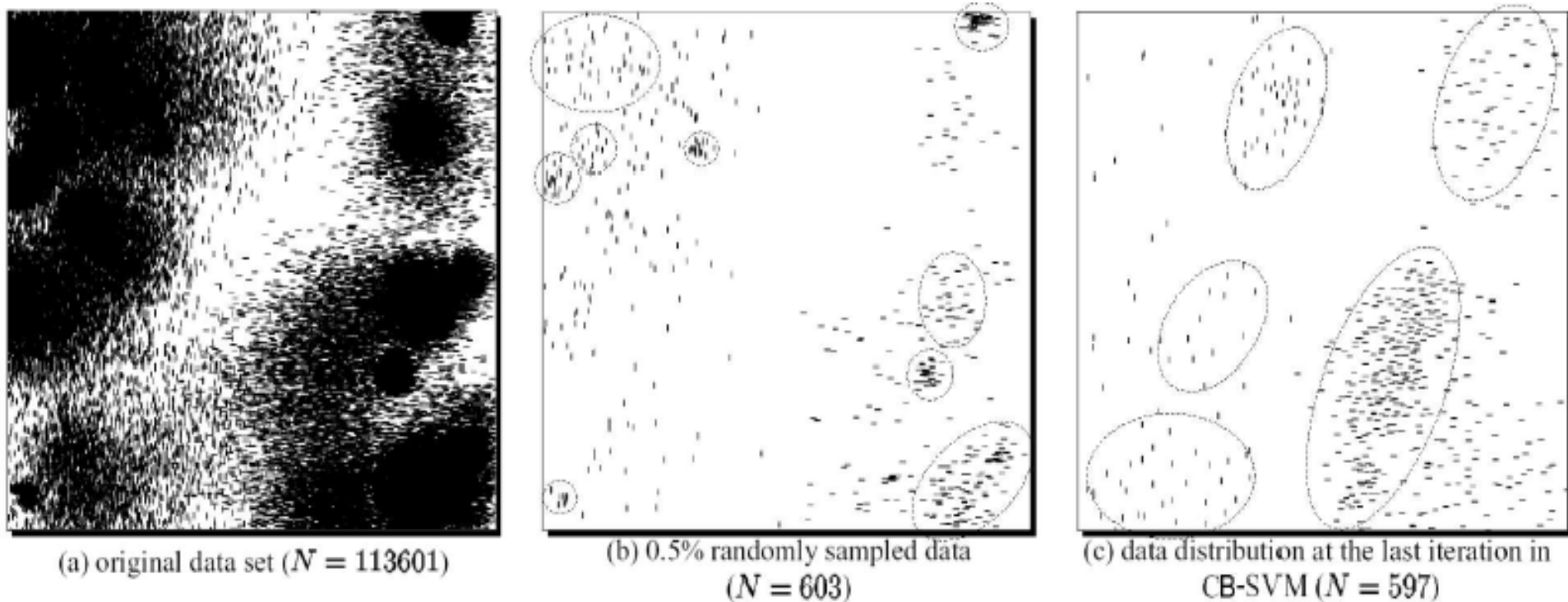



Figure 6: Synthetic data set in a two-dimensional space. ‘|’: positive data; ‘-’: negative data

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

SVM 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
- From classification to regression and ranking:
 - <http://www.dainf.ct.utfpr.edu.br/~kaestner/Mineracao/hwanjoyu-svmtutorial.pdf>

Matrix Data: Classification: Part 3

- SVM (Support Vector Machine)
- kNN (k Nearest Neighbor) 
- Other Issues
- Summary

Lazy vs. Eager Learning

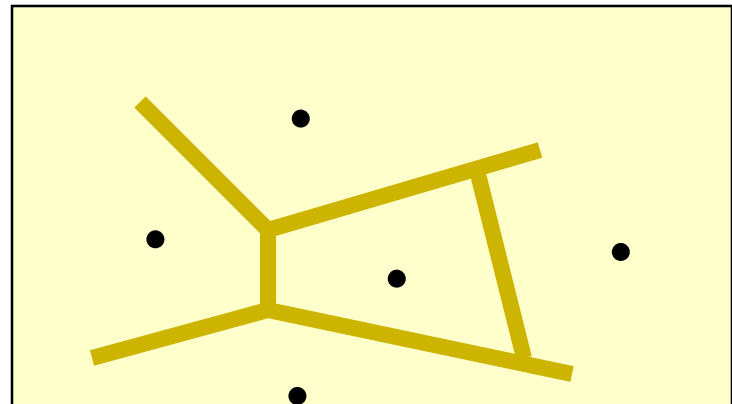
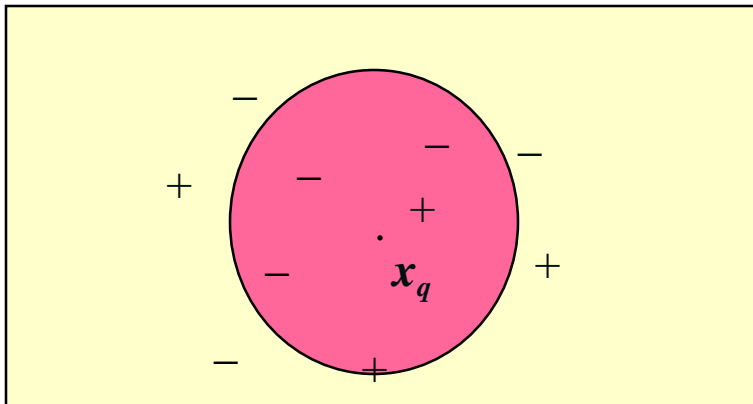
- Lazy vs. eager learning
 - **Lazy learning** (e.g., instance-based learning): Simply stores training data (or only minor processing) and waits until it is given a test tuple
 - **Eager learning** (the above discussed methods): Given a set of training tuples, constructs a classification model before 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

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, $\text{dist}(\mathbf{X}_1, \mathbf{X}_2)$
- 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
- Voronoi diagram: the decision surface induced by 1-NN for a typical set of training examples

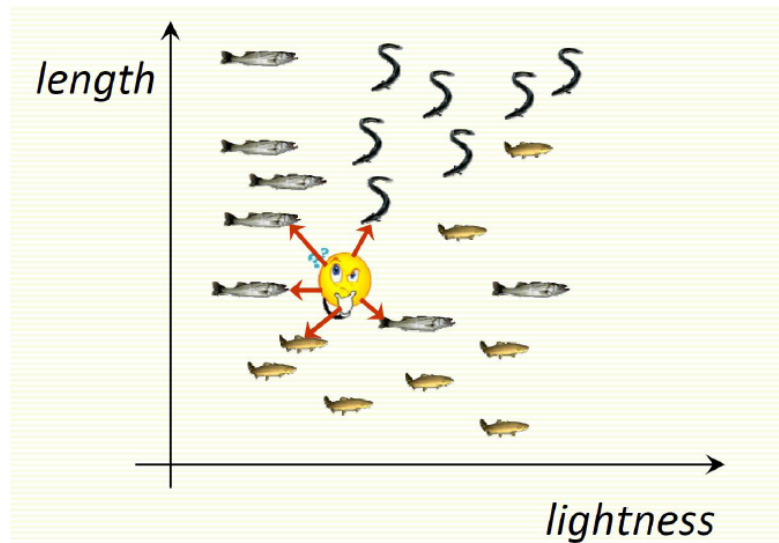


kNN Example

$X = (\text{length}, \text{lightness})$

Classes = {salmon, sea bass, eel}

Task: Identify fish given its (length, lightness)



$K = 5 : 3 \text{ sea bass}, 1 \text{ eel}, 1 \text{ salmon} \Rightarrow \text{sea bass}$

kNN Algorithm Summary

- Choose K
- For a given new instance X_{new} , find K closest training points w.r.t. a distance measure
- Classify X_{new} = majority vote among the K points

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
 - $y_q = \frac{\sum w_i y_i}{\sum w_i}$, where x_i 's are x_q 's nearest 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

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

Similarity and Dissimilarity

- **Similarity**
 - Numerical measure of how alike two data objects are
 - Value is higher when objects are more alike
 - Often falls in the range $[0,1]$
- **Dissimilarity** (e.g., distance)
 - Numerical measure of how different two data objects are
 - Lower when objects are more alike
 - Minimum dissimilarity is often 0
 - Upper limit varies
- **Proximity** refers to a similarity or dissimilarity

Proximity Measure for Nominal Attributes

- Can take 2 or more states, e.g., red, yellow, blue, green (generalization of a binary attribute)
- Method 1: Simple matching
 - m : # of matches, p : total # of variables

$$d(i, j) = \frac{p - m}{p}$$

- Method 2: Use a large number of binary attributes
 - creating a new binary attribute for each of the M nominal states

Proximity Measure for Binary Attributes

- A contingency table for binary data
- Distance measure for symmetric binary variables:
- Distance measure for asymmetric binary variables:
- Jaccard coefficient (*similarity* measure for *asymmetric* binary variables):

		Object j		
		1	0	sum
Object i	1	q	r	$q+r$
	0	s	t	$s+t$
sum		$q+s$	$r+t$	p

$$d(i, j) = \frac{r + s}{q + r + s + t}$$

$$d(i, j) = \frac{r + s}{q + r + s}$$

$$sim_{Jaccard}(i, j) = \frac{q}{q + r + s}$$

Dissimilarity between Binary Variables

- Example

Name	Gender	Fever	Cough	Test-1	Test-2	Test-3	Test-4
Jack	M	Y	N	P	N	N	N
Mary	F	Y	N	P	N	P	N
Jim	M	Y	P	N	N	N	N

- Gender is a symmetric attribute
- The remaining attributes are asymmetric binary
- Let the values Y and P be 1, and the value N 0

$$d(\mathit{jack}, \mathit{mary}) = \frac{\mathbf{0 + 1}}{\mathbf{2 + 0 + 1}} = \mathbf{0.33}$$

$$d(\mathit{jack}, \mathit{jim}) = \frac{\mathbf{1 + 1}}{\mathbf{1 + 1 + 1}} = \mathbf{0.67}$$

$$d(\mathit{jim}, \mathit{mary}) = \frac{\mathbf{1 + 2}}{\mathbf{1 + 1 + 2}} = \mathbf{0.75}$$

Standardizing Numeric Data

- Z-score: $z = \frac{x - \mu}{\sigma}$
 - X: raw score to be standardized, μ : mean of the population, σ : standard deviation
 - the distance between the raw score and the population mean in units of the standard deviation
 - negative when the raw score is below the mean, “+” when above
- An alternative way: Calculate the mean absolute deviation

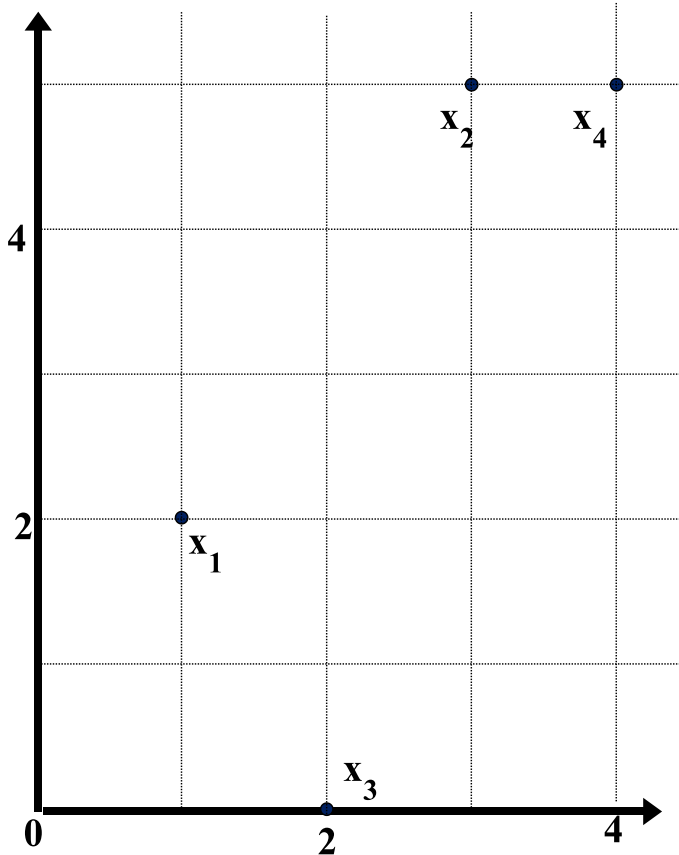
where

$$s_f = \frac{1}{n} (|x_{1f} - m_f| + |x_{2f} - m_f| + \dots + |x_{nf} - m_f|)$$
$$m_f = \frac{1}{n} (x_{1f} + x_{2f} + \dots + x_{nf}).$$

- standardized measure (z-score): $z_{if} = \frac{x_{if} - m_f}{s_f}$
- Using mean absolute deviation is more robust than using standard deviation

Example:

Data Matrix and Dissimilarity Matrix



Data Matrix

point	attribute1	attribute2
$x1$	1	2
$x2$	3	5
$x3$	2	0
$x4$	4	5

Dissimilarity Matrix
(with Euclidean Distance)

	$x1$	$x2$	$x3$	$x4$
$x1$	0			
$x2$	3.61	0		
$x3$	2.24	5.1	0	
$x4$	4.24	1	5.39	0

Distance on Numeric Data: Minkowski Distance

- *Minkowski distance*: A popular distance measure

$$d(i, j) = \sqrt[h]{|x_{i1} - x_{j1}|^h + |x_{i2} - x_{j2}|^h + \dots + |x_{ip} - x_{jp}|^h}$$

where $i = (x_{i1}, x_{i2}, \dots, x_{ip})$ and $j = (x_{j1}, x_{j2}, \dots, x_{jp})$ are two p -dimensional data objects, and h is the order (the distance so defined is also called L - h norm)

- Properties
 - $d(i, j) > 0$ if $i \neq j$, and $d(i, i) = 0$ (Positive definiteness)
 - $d(i, j) = d(j, i)$ (Symmetry)
 - $d(i, j) \leq d(i, k) + d(k, j)$ (Triangle Inequality)
- A distance that satisfies these properties is a **metric**

Special Cases of Minkowski Distance

- $h = 1$: **Manhattan** (city block, L_1 norm) **distance**
 - E.g., the Hamming distance: the number of bits that are different between two binary vectors

$$d(i, j) = |x_{i_1} - x_{j_1}| + |x_{i_2} - x_{j_2}| + \dots + |x_{i_p} - x_{j_p}|$$

- $h = 2$: (L_2 norm) **Euclidean** distance

$$d(i, j) = \sqrt{(|x_{i_1} - x_{j_1}|^2 + |x_{i_2} - x_{j_2}|^2 + \dots + |x_{i_p} - x_{j_p}|^2)}$$

- $h \rightarrow \infty$. **“supremum”** (L_{\max} norm, L_{∞} norm) distance.
 - This is the maximum difference between any component (attribute) of the vectors

$$d(i, j) = \lim_{h \rightarrow \infty} \left(\sum_{f=1}^p |x_{if} - x_{jf}|^h \right)^{\frac{1}{h}} = \max_f |x_{if} - x_{jf}|$$

Example: Minkowski Distance

Dissimilarity Matrices

point	attribute 1	attribute 2
x1	1	2
x2	3	5
x3	2	0
x4	4	5

Manhattan (L_1)

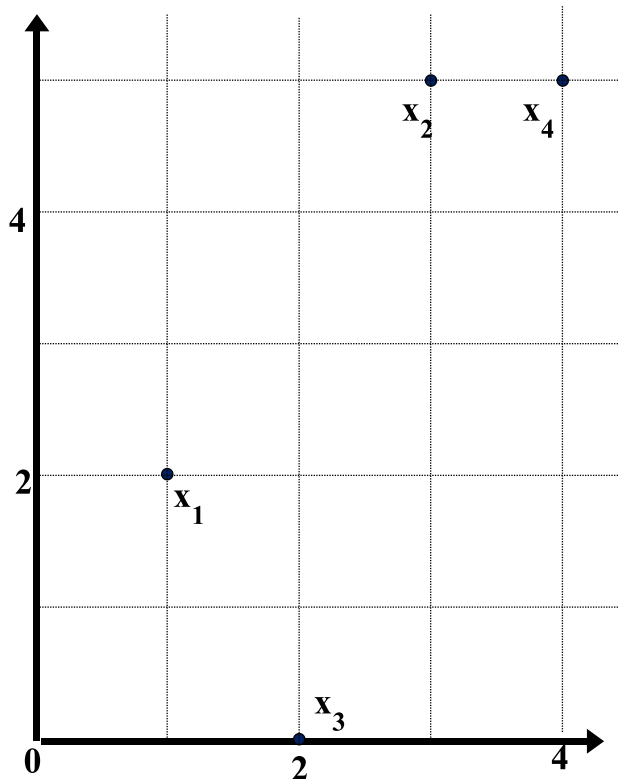
L	x1	x2	x3	x4
x1	0			
x2	5	0		
x3	3	6	0	
x4	6	1	7	0

Euclidean (L_2)

L2	x1	x2	x3	x4
x1	0			
x2	3.61	0		
x3	2.24	5.1	0	
x4	4.24	1	5.39	0

Supremum

L_∞	x1	x2	x3	x4
x1	0			
x2	3	0		
x3	2	5	0	
x4	3	1	5	0



Ordinal Variables

- Order is important, e.g., rank
- Can be treated like interval-scaled
 - replace x_{if} by their rank $r_{if} \in \{1, \dots, M_f\}$
 - map the range of each variable onto $[0, 1]$ by replacing i -th object in the f -th variable by

$$z_{if} = \frac{r_{if} - 1}{M_f - 1}$$

- compute the dissimilarity using methods for interval-scaled variables

Attributes of Mixed Type

- A database may contain all attribute types
 - Nominal, symmetric binary, asymmetric binary, numeric, ordinal
- One may use a weighted formula to combine their effects

$$d(i, j) = \frac{\sum_{f=1}^p \delta_{ij}^{(f)} d_{ij}^{(f)}}{\sum_{f=1}^p \delta_{ij}^{(f)}}$$

- f is binary or nominal:

$d_{ij}^{(f)} = 0$ if $x_{if} = x_{jf}$, or $d_{ij}^{(f)} = 1$ otherwise

- f is numeric: use the normalized distance
- f is ordinal

- Compute ranks r_{if} and $z_{if} = \frac{r_{if} - 1}{M_f - 1}$

- Treat z_{if} as interval-scaled

Cosine Similarity

- A **document** can be represented by thousands of attributes, each recording the *frequency* of a particular word (such as keywords) or phrase in the document.

<i>Document</i>	<i>teamcoach</i>	<i>hockey</i>	<i>baseball</i>	<i>soccer</i>	<i>penalty</i>	<i>score</i>	<i>win</i>	<i>loss</i>	<i>season</i>	
Document1	5	0	3	0	2	0	0	2	0	0
Document2	3	0	2	0	1	1	0	1	0	1
Document3	0	7	0	2	1	0	0	3	0	0
Document4	0	1	0	0	1	2	2	0	3	0

- Other vector objects: gene features in micro-arrays, ...
- Applications: information retrieval, biologic taxonomy, gene feature mapping, ...
- Cosine measure: If d_1 and d_2 are two vectors (e.g., term-frequency vectors), then

$$\cos(d_1, d_2) = (d_1 \bullet d_2) / (||d_1|| ||d_2||),$$

where \bullet indicates vector dot product, $||d||$: the length of vector d

Example: Cosine Similarity

- $\cos(d_1, d_2) = (d_1 \bullet d_2) / (||d_1|| ||d_2||)$,
where \bullet indicates vector dot product, $||d||$: the length of vector d
- Ex: Find the **similarity** between documents 1 and 2.

$$d_1 = (5, 0, 3, 0, 2, 0, 0, 2, 0, 0)$$

$$d_2 = (3, 0, 2, 0, 1, 1, 0, 1, 0, 1)$$

$$d_1 \bullet d_2 = 5 \cdot 3 + 0 \cdot 0 + 3 \cdot 2 + 0 \cdot 0 + 2 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 + 2 \cdot 1 + 0 \cdot 0 + 0 \cdot 1 = 25$$

$$||d_1|| = (5^2 + 0^2 + 3^2 + 0^2 + 2^2 + 0^2 + 0^2 + 2^2 + 0^2 + 0^2)^{0.5} = (42)^{0.5} = 6.481$$

$$||d_2|| = (3^2 + 0^2 + 2^2 + 0^2 + 1^2 + 1^2 + 0^2 + 1^2 + 0^2 + 1^2)^{0.5} = (17)^{0.5} = 4.12$$

$$\cos(d_1, d_2) = 0.94$$


Model Selection for kNN

- The number of neighbors k
 - Small k : overfitting (high variance)
 - Big k : bringing too many irrelevant points (high bias)
 - More discussions:

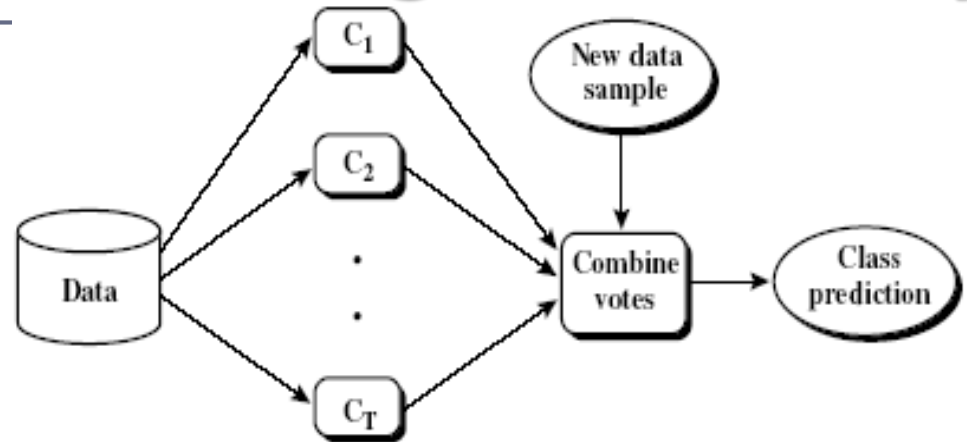
<http://scott.fortmann-roe.com/docs/BiasVariance.html>

- The distance function

Matrix Data: Classification: Part 3

- SVM (Support Vector Machine)
- kNN (k Nearest Neighbor)
- Other Issues 
- Summary

Ensemble Methods: Increasing the Accuracy



- Ensemble methods
 - Use a combination of models to increase accuracy
 - Combine a series of k learned models, M_1, M_2, \dots, M_k , with the aim of creating an improved model M^*
- Popular ensemble methods
 - **Bagging:** averaging the prediction over a collection of classifiers
 - **Boosting:** weighted vote with a collection of classifiers

Bagging: Bootstrap Aggregation

- Analogy: Diagnosis based on multiple doctors' majority vote
- Training
 - Given a set D of d tuples, at each iteration i , a training set D_i of d tuples is sampled with replacement from D (i.e., bootstrap)
 - A classifier model M_i is learned for each training set D_i
- Classification: classify an unknown sample X
 - Each classifier M_i returns its class prediction
 - The bagged classifier M^* counts the votes and assigns the class with the most votes to X
- Prediction: can be applied to the prediction of continuous values by taking the average value of each prediction for a given test tuple

Performance of Bagging

- Accuracy
 - Often significantly better than a single classifier derived from D
 - For noise data: not considerably worse, more robust
 - Proved improved accuracy in prediction
- Example
 - Suppose we have 5 completely independent classifiers...
 - If accuracy is 70% for each
 - The final prediction is correct, if at least 3 classifiers make the correct prediction
 - 3 are correct: $\binom{5}{3} \times (.7^3)(.3^2)$
 - 4 are correct: $\binom{5}{4} \times (.7^4)(.3^1)$
 - 5 are correct: $\binom{5}{5} \times (.7^5)(.3^0)$
 - In all, $10 (.7^3)(.3^2)+5(.7^4)(.3)+(.7^5)$
 - **83.7% majority vote accuracy**
 - 101 Such classifiers
 - **99.9% majority vote accuracy**

Boosting

- Analogy: Consult several doctors, based on a combination of weighted diagnoses—weight assigned based on the previous diagnosis accuracy
- How boosting works?
 - **Weights** are assigned to each training tuple
 - A series of k classifiers is iteratively learned
 - After a classifier M_t is learned, the weights are updated to allow the subsequent classifier, M_{t+1} , to **pay more attention to the training tuples that were misclassified by M_t**
 - The final **M^*** combines the votes of each individual classifier, where the weight of each classifier's vote is a function of its accuracy
- Boosting algorithm can be extended for numeric prediction
- Comparing with bagging: Boosting tends to have greater accuracy, but it also risks overfitting the model to misclassified data

*Adaboost (Freund and Schapire, 1997)

- Given a set of d class-labeled tuples, $(\mathbf{X}_1, y_1), \dots, (\mathbf{X}_d, y_d)$
- Initially, all the weights of tuples are set the same ($1/d$)
- Generate k classifiers in k rounds. At round t ,
 - Tuples from D are sampled (with replacement) to form a training set D_t of the same size based on its weight
 - A classification model M_t is derived from D_t
 - If a tuple is misclassified, its weight is increased, o.w. it is decreased
 - $w_{t+1,j} \propto w_{t,j} \times \exp(-\alpha_t)$ if j is correctly classified
 - $w_{t+1,j} \propto w_{t,j} \times \exp(\alpha_t)$ if j is incorrectly classified

α_t : weight for classifier t , the higher the better

AdaBoost

- Error rate: $\text{err}(\mathbf{X}_j)$ is the misclassification error of tuple \mathbf{X}_j . Classifier M_t error rate ($\epsilon_t = \text{error}(M_t)$) is the sum of the weights of the misclassified tuples:

$$\text{error}(M_t) = \sum_j^d w_{tj} \times \text{err}(\mathbf{X}_{tj})$$

- The weight of classifier M_t 's vote is

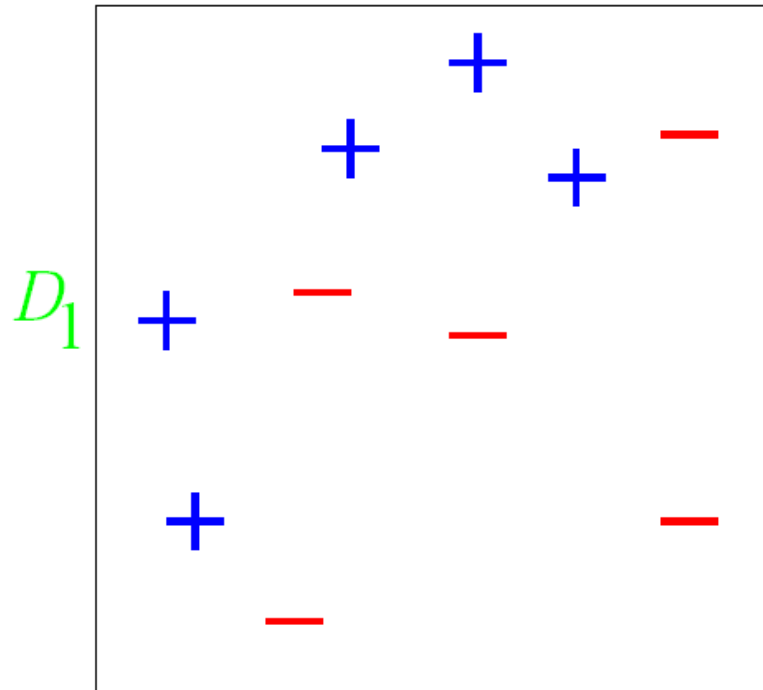
$$\alpha_t = \frac{1}{2} \ln \frac{1 - \text{error}(M_t)}{\text{error}(M_t)}$$

- Final classifier M^*

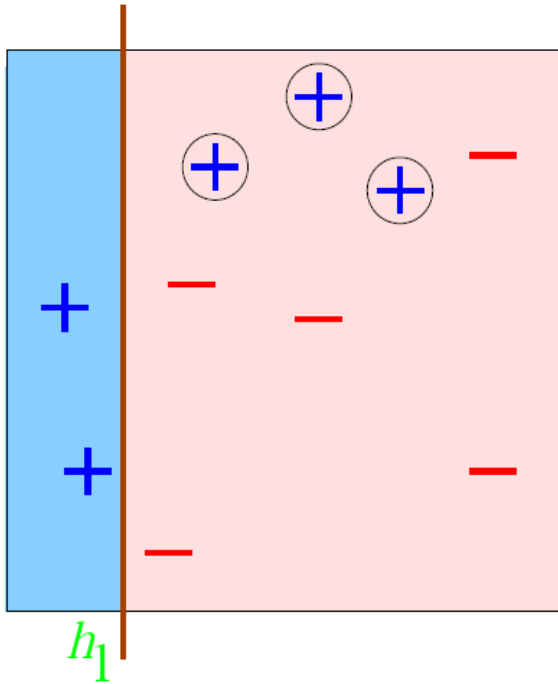
$$M^*(x) = \text{sign}\left(\sum_t \alpha_t M_t(x)\right)$$

AdaBoost Example

- From “A Tutorial on Boosting”
 - By Yoav Freund and Rob Schapire
- Note they use h_t to represent classifier instead of M_t



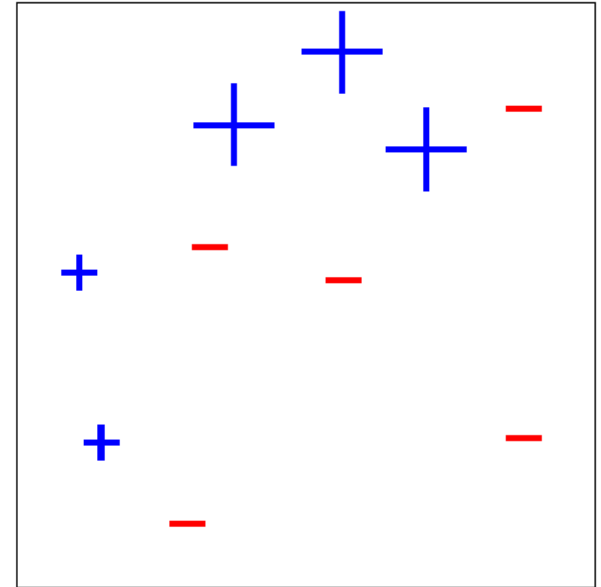
Round 1



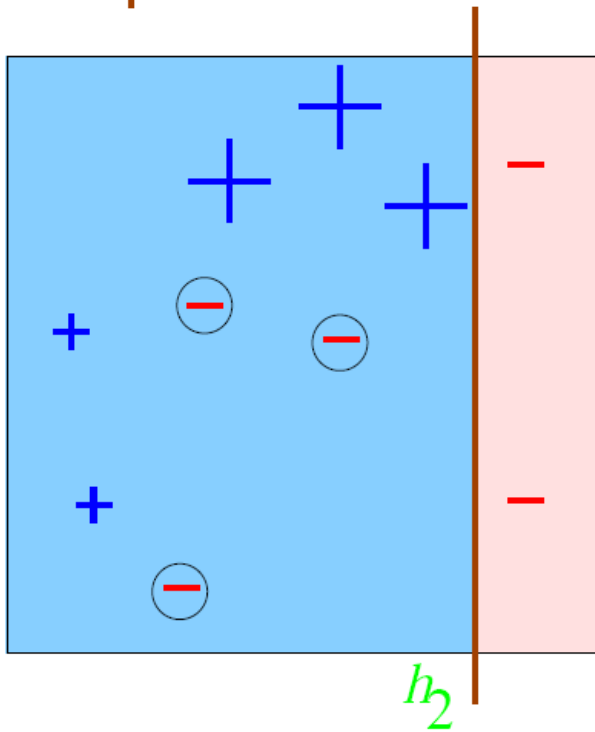
$$\begin{aligned}\epsilon_1 &= 0.30 \\ \alpha_1 &= 0.42\end{aligned}$$



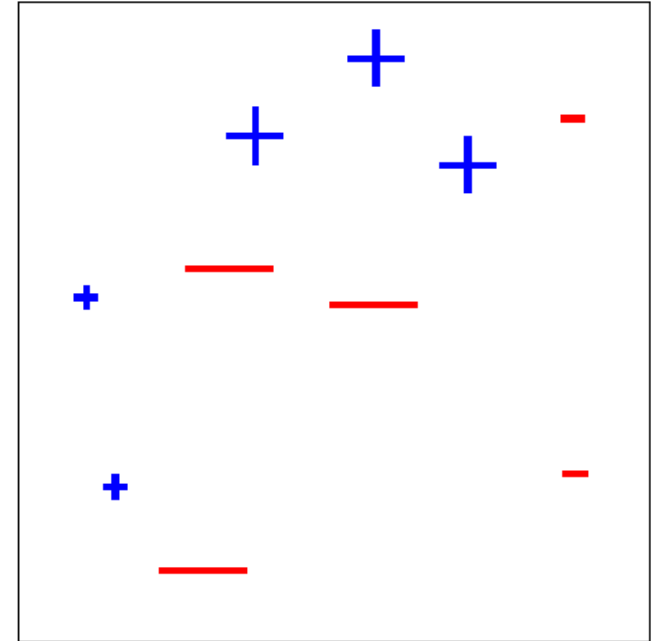
D_2



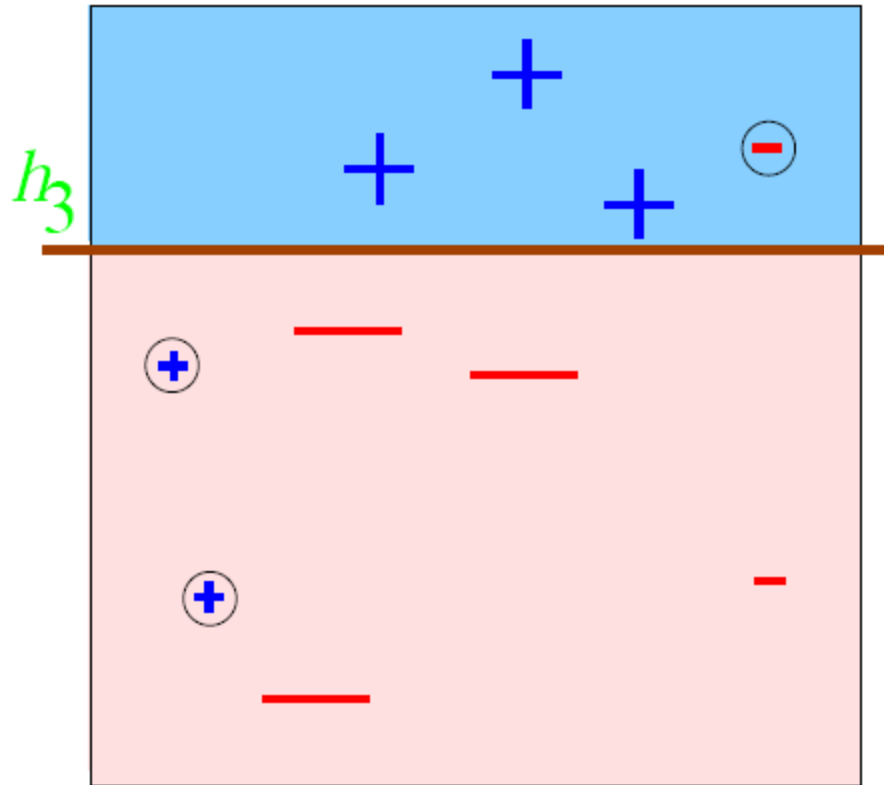
Round 2



$$\begin{aligned} \epsilon_2 &= 0.21 \\ \alpha_2 &= 0.65 \end{aligned} \Rightarrow D_3$$



Round 3

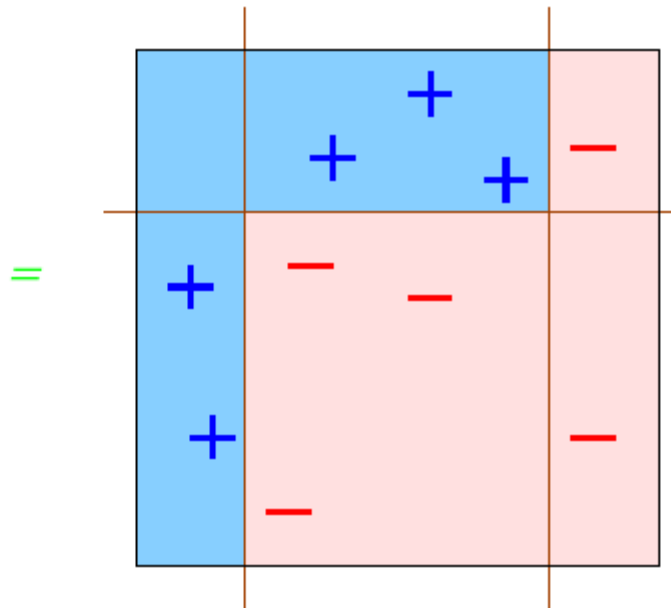


$$\varepsilon_3 = 0.14$$

$$\alpha_3 = 0.92$$

Final Model

$$M^* = \text{sign} \left(0.42 \begin{array}{|c|c|} \hline \text{blue} & \text{red} \\ \hline \end{array} + 0.65 \begin{array}{|c|c|} \hline \text{blue} & \text{red} \\ \hline \end{array} + 0.92 \begin{array}{|c|c|} \hline \text{blue} & \text{red} \\ \hline \end{array} \right)$$



*Random Forest (Breiman 2001)

- Random Forest:
 - Each classifier in the ensemble is a *decision tree* classifier and is generated using a random selection of attributes at each node to determine the split
 - During classification, each tree votes and the most popular class is returned
- Two Methods to construct Random Forest:
 - Forest-RI (*random input selection*): Randomly select, at each node, F attributes as candidates for the split at the node. The CART methodology is used to grow the trees to maximum size
 - Forest-RC (*random linear combinations*): Creates new attributes (or features) that are a linear combination of the existing attributes (reduces the correlation between individual classifiers)
- Comparable in accuracy to Adaboost, but more robust to errors and outliers
- Insensitive to the number of attributes selected for consideration at each split, and faster than bagging or boosting

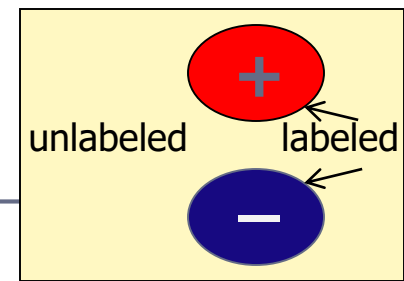
Classification of Class-Imbalanced Data Sets

- Class-imbalance problem: Rare positive example but numerous negative ones, e.g., medical diagnosis, fraud, oil-spill, fault, etc.
- Traditional methods assume a balanced distribution of classes and equal error costs: not suitable for class-imbalanced data
- Typical methods for imbalance data in 2-class classification:
 - **Oversampling:** re-sampling of data from positive class
 - **Under-sampling:** randomly eliminate tuples from negative class
 - **Threshold-moving:** moves the decision threshold, t , so that the rare class tuples are easier to classify, and hence, less chance of costly false negative errors
 - **Ensemble techniques:** Ensemble multiple classifiers introduced above
- Still difficult for class imbalance problem on multiclass tasks

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 \mathbf{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 \mathbf{X} , each classifier votes. \mathbf{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

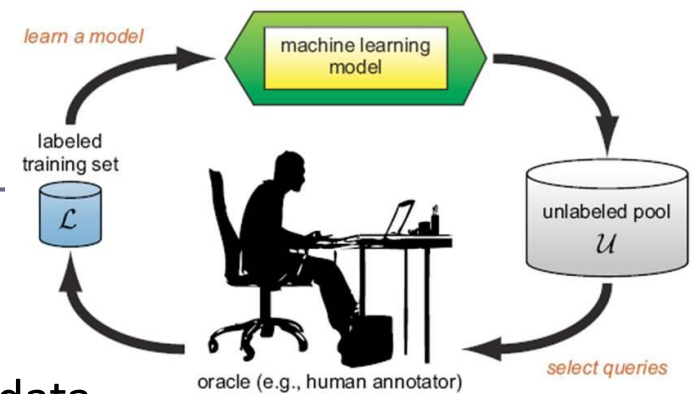
*Semi-Supervised Classification



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

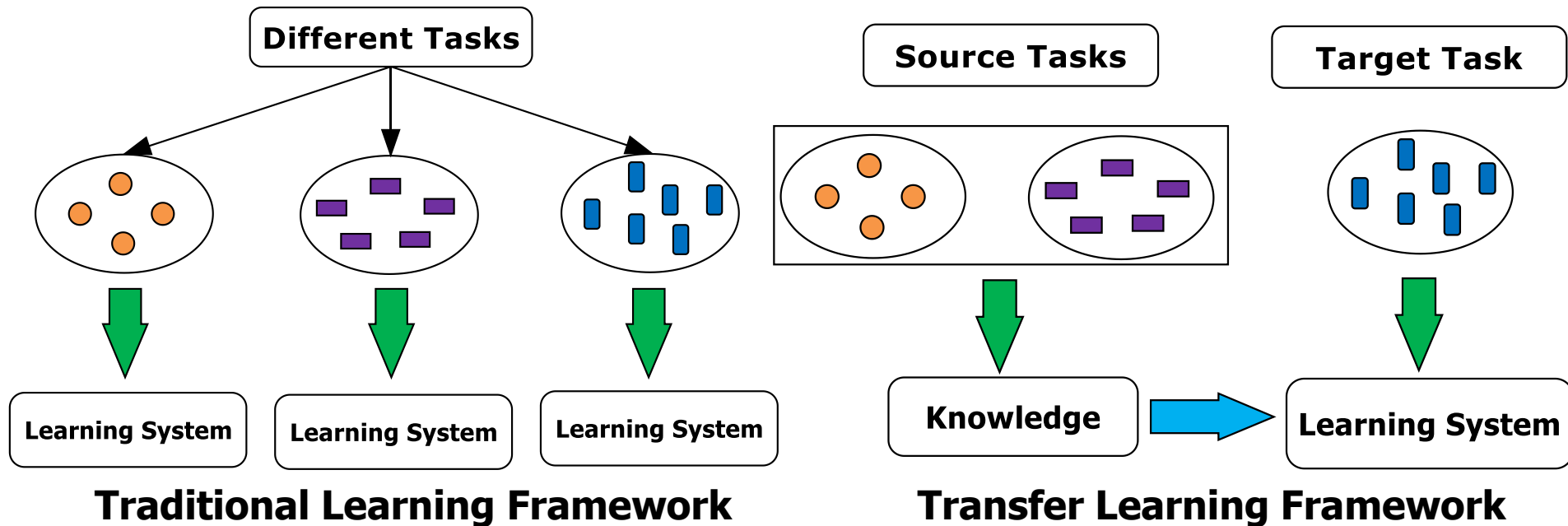
*Active Learning

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



*Transfer Learning: Conceptual Framework


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



Transfer Learning: Methods and Applications

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

Matrix Data: Classification: Part 3

- SVM (Support Vector Machine)
- kNN (k Nearest Neighbor)
- Other Issues
- Summary 

-
- **Support Vector Machine**
 - Support vectors; Maximum marginal hyperplane; Linear separable; Linear inseparable; Kernel tricks
 - **Instance-Based Learning**
 - Lazy learning vs. eager learning; K-nearest neighbor algorithm; Similarity / dissimilarity measures
 - ***Other Topics**
 - Ensemble; Class imbalanced data; multi-class classification; semi-supervised learning; active learning; transfer learning