Lecture No. 9 – Instance-Based Learning and SVM

- Overview of Instance-based Learning
- K-nearest Neighbours
- Case-Based Reasoning (CBR)
- Kernel-Based Methods
- Support Vector Machines (SVM)

Instance-Based Methods

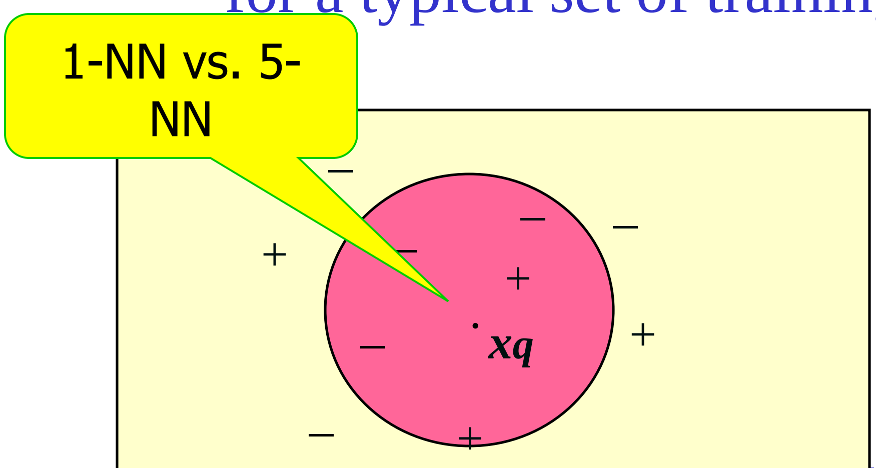
- Model-based ("eager") learning
  - Process training examples and store the model for classification of future instances
- Instance-based ("lazy") learning
  - Store training examples and delay the processing ("lazy evaluation") until a new instance must be classified
- Typical approaches of instance-based learning
  - *k*-nearest neighbor approach
    - Instances represented as points in a Euclidean space.
  - Kernel-based methods / Locally weighted regression
    - Construct local approximation
  - Case-based reasoning
    - Uses symbolic representations and knowledge-based inference

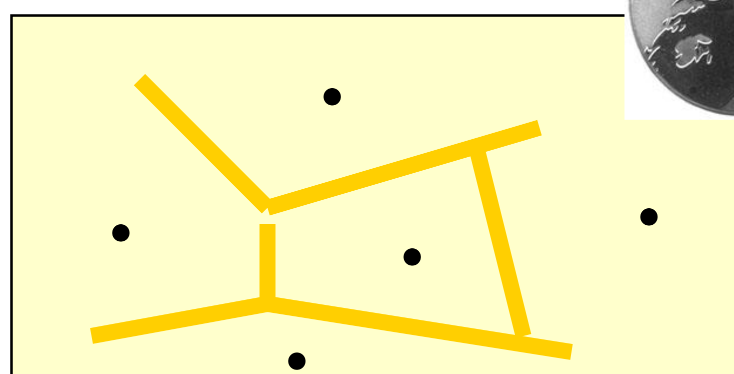
Lecture No. 9 – Instance-Based Learning and SVM

- Overview of Instance-based Learning
- K-nearest Neighbours
- Case-Based Reasoning (CBR)
- Kernel-Based Methods
- Overview of Support Vector Machines (SVM)


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.
- The target function could be discrete- or real- valued.
- For discrete-valued, the k-NN returns the most common value among the *k* training examples nearest to x_q.
- For continuous-valued target functions, calculate the mean values of the *k* nearest neighbors
- Voronoi diagram: the decision surface induced by 1-NN for a typical set of training examples.





K-nearest neighbor for discrete classes

Source: www2.cs.uh.edu/~vilalta/courses/ machinelearning/instancelearning1.ppt

Algorithm (parameter k)

Training

For each training example $(X,C(X))$

add the example to our training list.

Testing

When a new example X_q arrives, assign class:

$C(X_q)$ = majority voting on the k nearest neighbors of X_q

$$C(X_q) = \arg\max_{v} \sum_{i} \delta(v, C(X_i))$$

where $\delta(a,b) = 1$ if $a = b$ and 0 otherwise

K-nearest neighbor for real-valued functions

Source: www2.cs.uh.edu/~vilalta/courses/ machinelearning/instancelearning1.ppt

Algorithm (parameter k)

Training

For each training example $(X,C(X))$

add the example to our training list.

Testing

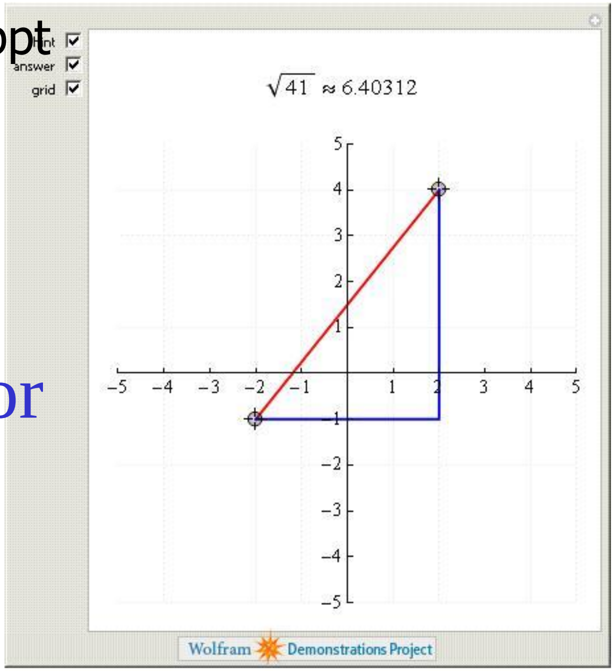
When a new example X_q arrives, assign class:

$C(X_q)$ = average value among k nearest neighbors of X_q

$$C(X_q) = \frac{\sum_i C(X_i)}{k}$$

The Distance Between Examples

Source; www.cs.bham.ac.uk/~axk/KNN_CBR.ppt



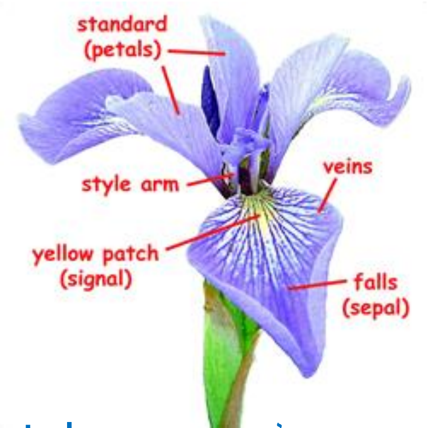
- We need a measure of distance in order to know who are the neighbours
- Assume that we have *T* attributes for the learning problem. Then one example point $\mathbf{x}$ has elements $x_t \in \Re$, $t=1,\ldots T$.
- The distance between two points $\mathbf{x_i}\ \mathbf{x_j}$ is often defined as the Euclidean distance:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{t=1}^{T} [x_{ti} - x_{tj}]^2}$$

K-NN Example: Iris Dataset

Source: Fisher (1936)

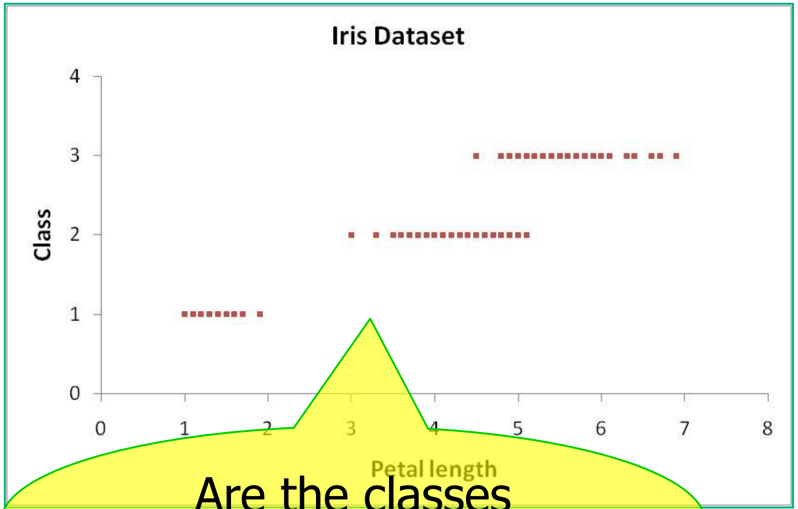
- Number of records: 150
- Number of attributes: 4
- Number of classes: 3 (iris setosa, iris versicolor, and iris virginica)



Petal - עלה-כותרת

Sepal - עלה-גביע

	sepal length in cm	sepal width in cm	petal length in cm	petal width in cm	Class
1	4.6	3.6	1	0.2	1
2	4.3	3	1.1	0.1	1
3	5	3.2	1.2	0.2	1
4	5.8	4	1.2	0.2	1
5	4.4	3	1.3	0.2	1
6	4.4	3.2	1.3	0.2	1
7	4.5	2.3	1.3	0.3	1
8	4.7	3.2	1.3	0.2	1
9	5	3.5	1.3	0.3	1
10	5.4	3.9	1.3	0.4	1



Are the classes linearly separable?

K-NN Example: Iris Dataset (cont.)

- Full size: 150 observations
- Testing set (query examples): q = 50, 100, 150

Nearest Neighbor

k	sepal length in cm	sepal width in cm	petal length in cm	petal width in cm	Actual Class	**Dist (k,50)**	Predicted Class
8	5	3.4	1.5	0.2	1		
50	5	3.3	1.4	0.2		0.1414	1

Nearest Neighbor

k	sepal length in cm	sepal width in cm	petal length in cm	petal width in cm	Actual Class	**Dist (k,100)**	Predicted Class
97	5.7	2.9	4.2	1.3	2		
100	5.7	2.8	4.1	1.3		0.1414	2

Nearest Neighbor

k	sepal length in cm	sepal width in cm	petal length in cm	petal width in cm	Actual Class	**Dist (k,150)**	Predicted Class
128	6.1	3	4.9	1.8	3		
150	5.9	3	5.1	1.8		0.2828	3

Distance-weighted k-NN

Source: faculty.cs.byu.edu/~cgc/Teaching/CS_478/ CS%20478%20-%20Instance%20Based%20Learning.ppt

- Replace $\hat{f}(q) = \arg\max_{v \in V} \sum_{i=1}^{k} \delta(v, f(x_i))$ by:

$$\boxed{\hat{f}(q) = \arg\max_{v \in V} \sum_{i=1}^{k} w_i \delta(v, f(x_i))}$$

- Where $\delta(a,b) = 1$ if $a = b$ and 0 otherwise

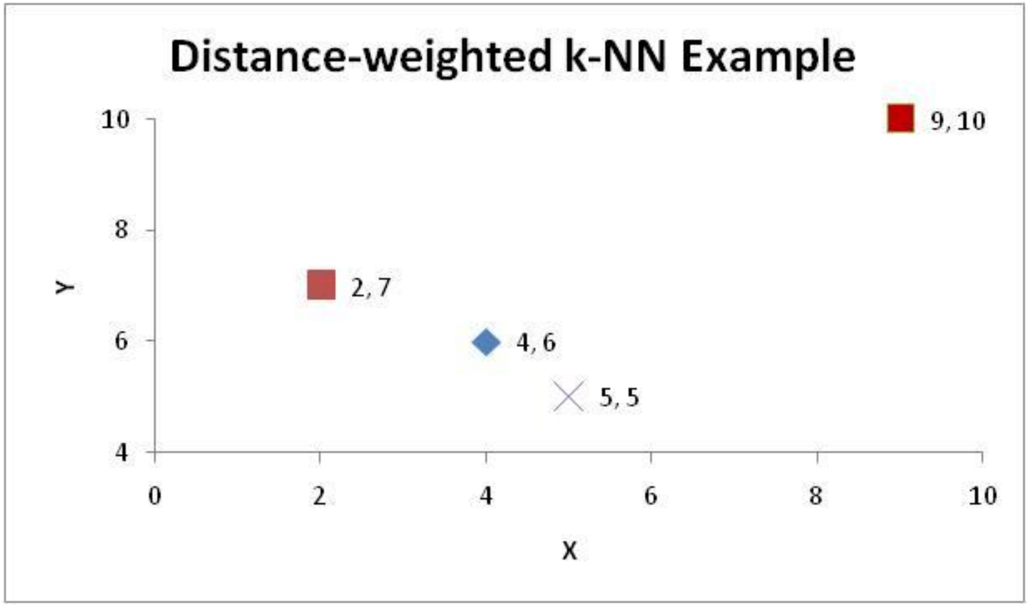
$$\boxed{w_i = \frac{1}{d(x_q - x_i)^2}}$$

Distance-weighted k-NN Example

k = 3 (number of nearest neighbors)

$$w_i = \frac{1}{d(x_q - x_i)^2}$$

i	X	Y	Class	DistX	DistY	Dist^2	Weight	w*delta(0)	w*delta(1)
1	4	6	0	-1	1	2	0.500	0.500	0.000
2	2	7	1	-3	2	13	0.077	0.000	0.077
3	9	10	1	4	5	41	0.024	0.000	0.024
Query	**5**	**5**	**?**					**0.500**	**0.101**



Predicted class

k-NN: ?

Distance-weighted k-NN:

$$\hat{f}(q) = \arg\max_{v \in V} \sum_{i=1}^{k} w_i \delta(v, f(x_i))$$

When To Consider Nearest Neighbor

Source: Lecture Slides for *Machine Learning* by T Mitchell, 1997

- Instances map to points in $\Re^n$
- Less than 20 attributes per instance
- Lots of training data
- Advantages
  - Training is very fast
  - Learn complex target functions
  - Don't lose information
- Disadvantages
  - Slow at query time
  - Limited interpretability
  - Curse of dimensionality
    - Distance between neighbors could be dominated by irrelevant attributes
    - Solutions?

Lecture No. 9 – Instance-Based Learning and SVM

- Overview of Instance-based Learning
- K-nearest Neighbours
- Case-Based Reasoning (CBR)
- Kernel-Based Methods
- Support Vector Machines (SVM)

One-dimensional Kernel Smoothers

(Based on Hastie et al.)

- *k*-nearest-neighbor average for real-valued functions
  - $$\hat{f}(x) = Ave(y_i | x_i \in N_k(x))$$
  - $N_k(x)$ – the set of k points nearest to x
  - assigns equal weight to all points in neighborhood
  - changes in a discrete way => discontinuous
  - Solution: points should have smooth decrease in weight according to their distance from the target point

One-dimensional Kernel Smoothers (cont.)

$f(x0)=?$

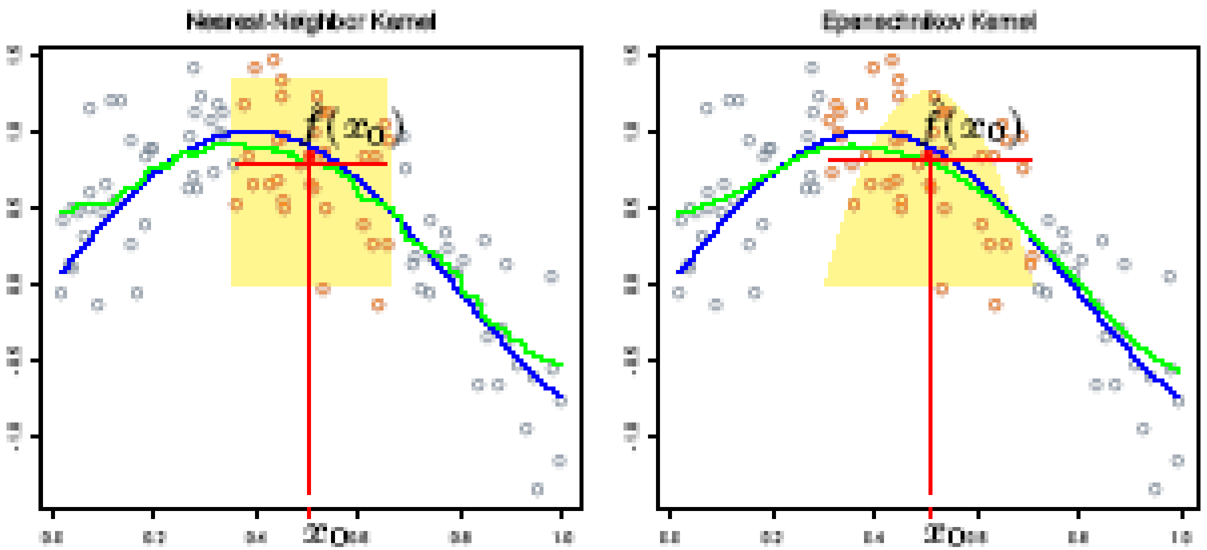


Figure 6.1: *In each panel* 100 *pairs* x_i, y_i *are generated at random from the blue curve with Gaussian errors:* $Y = \sin(4X) + \varepsilon$, $X \sim U[0, 1]$, $\varepsilon \sim N(0, 1/3)$. *In the left panel the green curve is the result of a* 30*-nearest-neighbor running-mean smoother. The red point is the fitted constant* $\hat{f}(x0)$, *and the red circles indicate those observations contributing to the fit at* $x0$. *The solid yellow region indicates the weights assigned to observations. In the right panel, the green curve is the kernel-weighted average, using an Epanechnikov kernel with (half)window width* $\lambda = 0.2$.

Nadaraya-Watson Kernel-weighted Average

$$\hat{f}(x) = \frac{\sum_{i=1}^{N} K_\lambda(x_0, x_i) y_i}{\sum_{i=1}^{N} K_\lambda(x_0, x_i)}$$

- Using the Epanechnikov quadratic kernel:

$$K_\lambda(x_0\, x_i) = D\left(\frac{\|x - x_0\|}{\lambda}\right)$$

$$D(t) = \begin{cases} \frac{3}{4}(1 - t^2) \text{ if } |\, t \,| \le 1 \\ 0 \text{ otherwise} \end{cases}$$

- λ is a smoothing parameter and determines width of kernel
- increase in λ means lower variance, higher bias (why?)
- more generally, it can be a function $h_m(x)$ (adaptive neighborhood)
- e.g. for *k*-nearest neighbors: $h_k(x) = |x - x_{[k]}|$ where $x_{[k]}$ is *k*-th nearest neighbor of *x*.

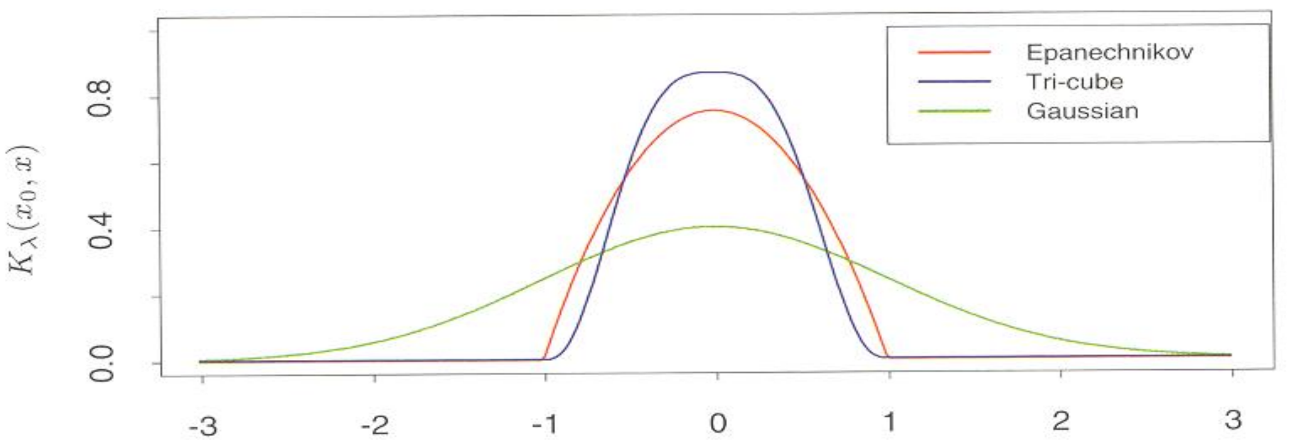
`

Other Kernels

- Other popular kernels:
  - Tri-cube function:
    - Differentiable
    - Flatter on top
  - Gaussian density function:
    - s.d. = window size
- Epanechnikov & tri-cube have compact support

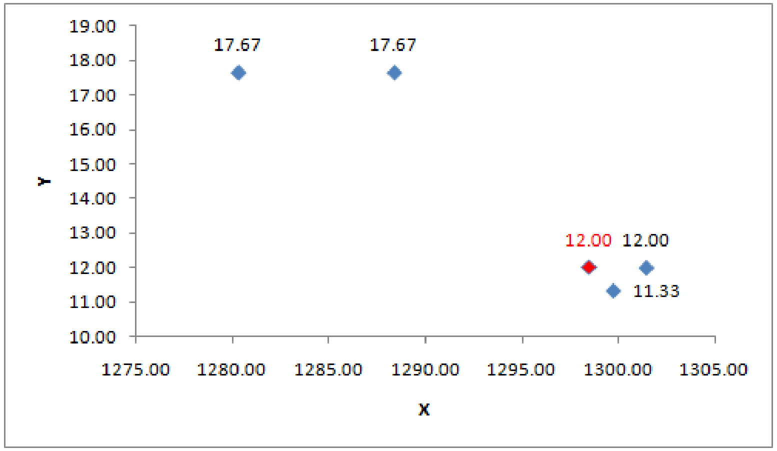
$$D(t) = \begin{cases} (1-|t|^3)^3 & \text{if } |t| \le 1; \\ 0 & \text{otherwise} \end{cases}$$

$$D(t) = z(t)$$



Epanechnikov Quadratic Kernel Example ($\lambda = 50$)

	X	Y	|X-X_0|	|X-X_0|/ lambda	K	f_est
	1280.29	17.67	18.19	0.3638	0.6507	
	1288.38	17.67	10.10	0.2021	0.7194	
X_0 →	**1298.48**	**12.00**	0.00	0.0000	**0.7500**	**14.01**
	1299.73	11.33	1.25	0.0250	0.7495	
	1301.44	12.00	2.96	0.0592	0.7474	
	Total				**3.6170**	



$$D(t) = \begin{cases} 0.75(1-t^2) \text{ if } |t| \le 1 \\ 0 \text{ otherwise} \end{cases}$$

$$K_\lambda(x_0\, x_i) = D\left(\frac{\|x - x_0\|}{\lambda}\right)$$

$$\hat{f}(x) = \frac{\sum_{i=1}^{N} K_\lambda(x_0, x_i) y_i}{\sum_{i=1}^{N} K_\lambda(x_0, x_i)}$$

Remarks on Lazy vs. Eager Learning

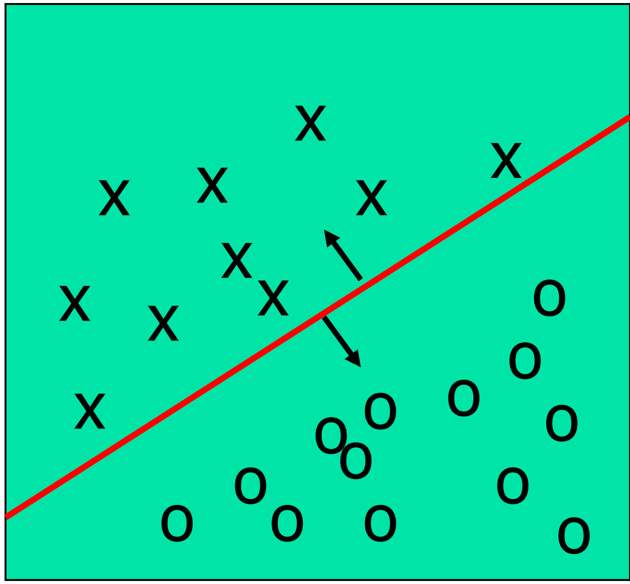
- Instance-based learning: lazy evaluation
- Decision-tree and Bayesian classification: eager evaluation
- Key differences
  - Lazy method uses a "local model" when querying instance x_q
  - Eager method uses a global approximation
- Efficiency: Lazy - less time training but more time predicting
- Accuracy
  - Lazy method effectively uses a richer hypothesis space
  - Eager: must commit to a single hypothesis (model)

Lecture No. 9 – Instance-Based Learning and SVM

- Overview of Instance-based Learning
- K-nearest Neighbours
- Case-Based Reasoning (CBR)
- Kernel-Based Methods
- Support Vector Machines (SVM)

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 words "homepage"
    - x_2 : # of words "welcome"
- Mathematically, $x \in X = \Re^n$, $y \in Y = \{+1, -1\}$
  - We want to derive a function f: $X \rightarrow Y$
- Linear Classification
  - Binary Classification problem
  - Data above the red line belongs to class 'x'
  - Data below red line belongs to class 'o'
  - Examples: SVM, Perceptron, Probabilistic Classifiers



SVM—Support Vector Machines

- A relatively new classification method for both linear and nonlinear data
- 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

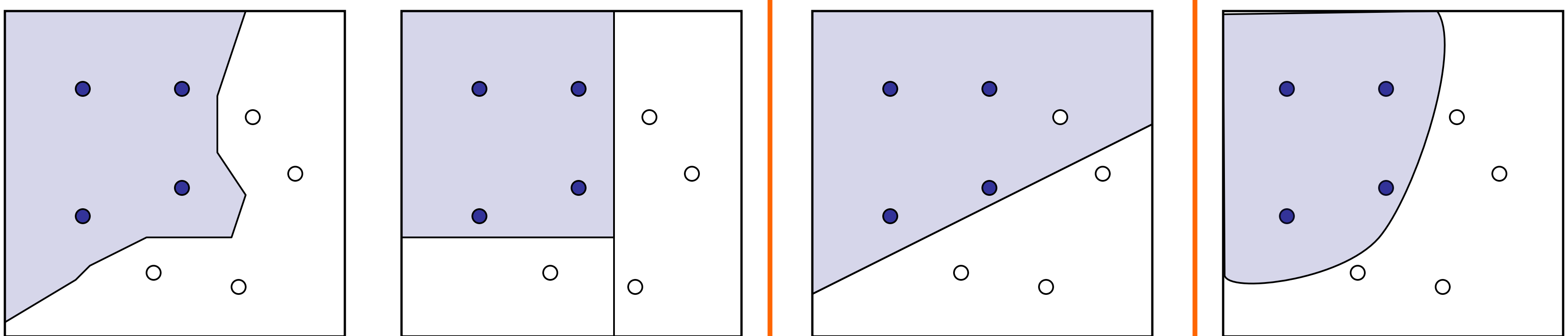
- Vapnik and colleagues (1992)—groundwork from Vapnik & Chervonenkis' statistical learning theory in 1960s
- Features: training can be slow but accuracy is high owing to their ability to model complex nonlinear decision boundaries (margin maximization)
- Used both for classification and numeric prediction
- Applications:
  - handwritten digit recognition, object recognition, speaker identification, benchmarking time-series prediction tests, text categorization

Vladimir Naumovich Vapnik

Владимир Наумович Вапник

A Discriminant Function

- It can be arbitrary functions of *x*, such as:



Nearest Neighbor | Decision Tree | Linear Functions | Nonlinear Functions

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

This slide is courtesy of *www1.cs.columbia.edu/~belhumeur/courses/biometrics/2009/svm.ppt*

Linear Discriminant Function

- g(x) is a linear function:

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

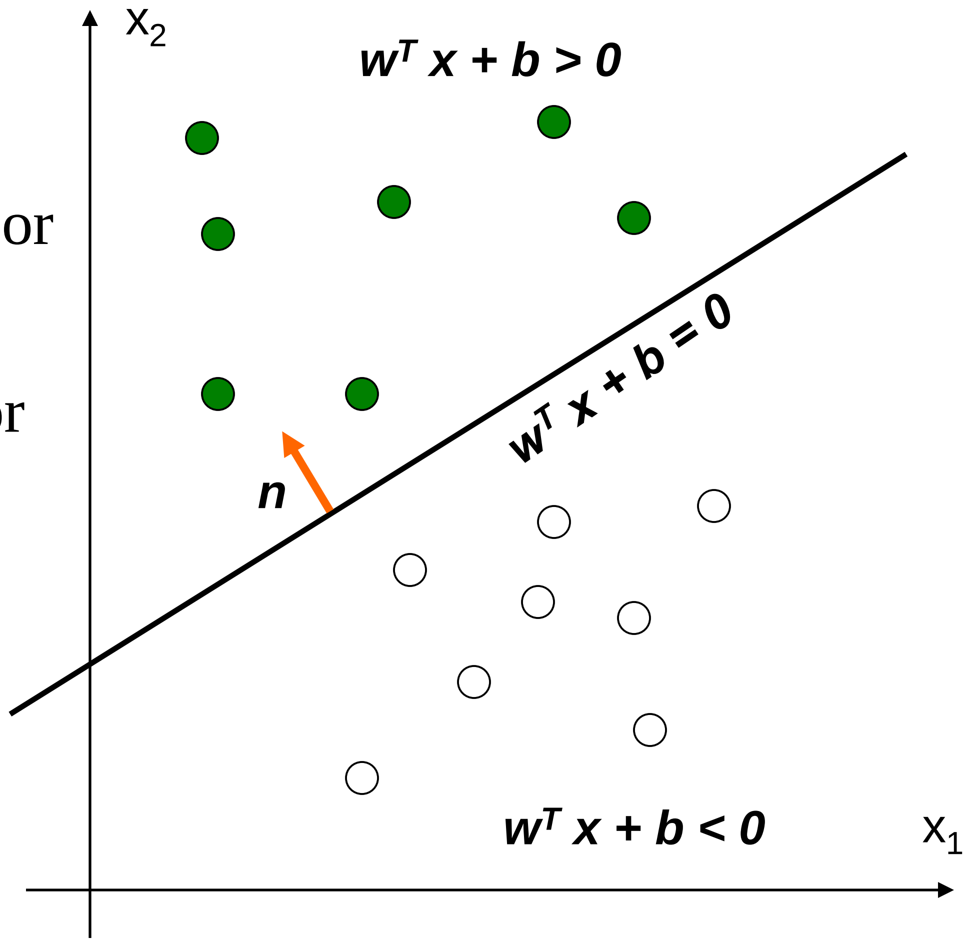
$w = (w_1, w_2, ..., w_m\} -$ weight vector

m - number of attributes

$x = (x_1, x_2, ..., x_m\}$ - training vector

b - bias

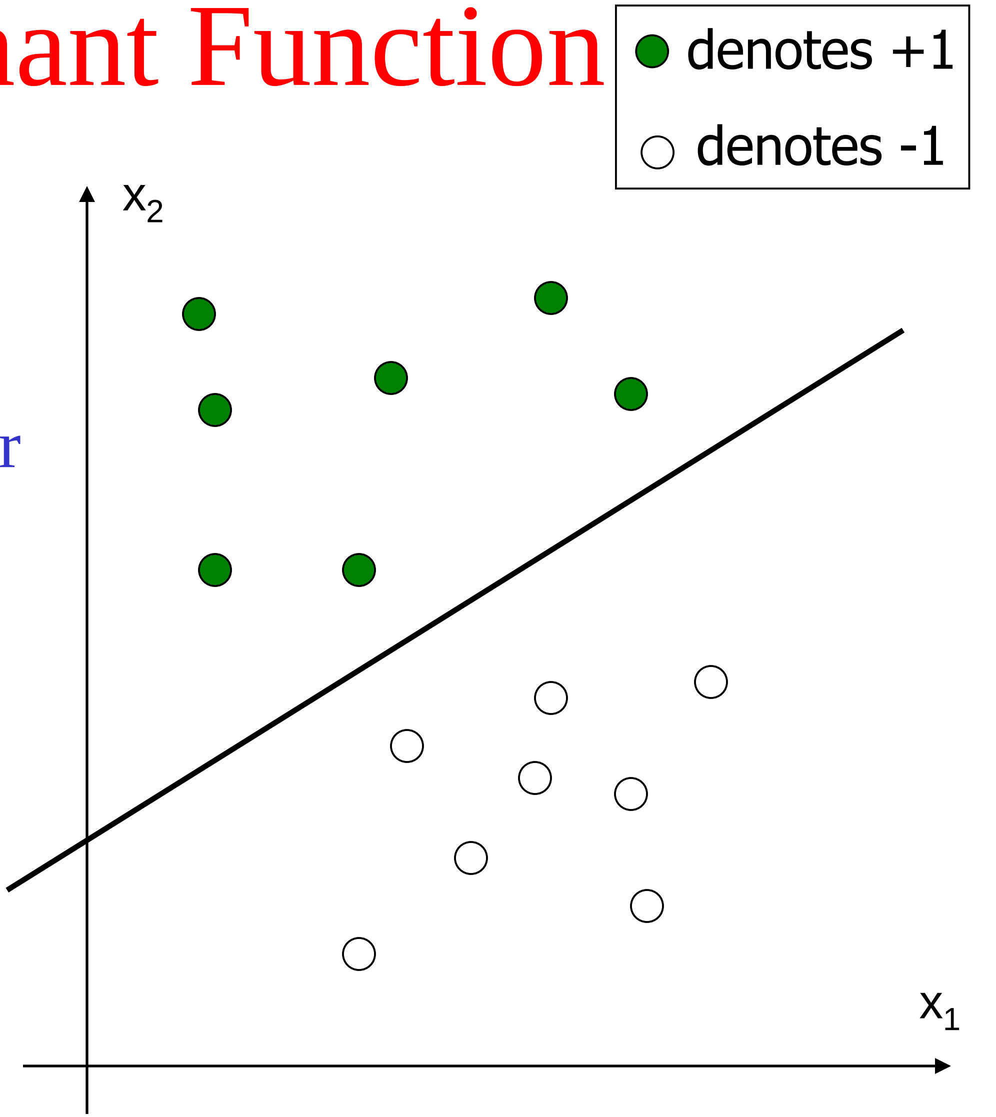
- A hyper-plane in the feature space (a straight line in 2-D)



This slide is courtesy of *www1.cs.columbia.edu/~belhumeur/courses/biometrics/2009/svm.ppt*

Linear Discriminant Function

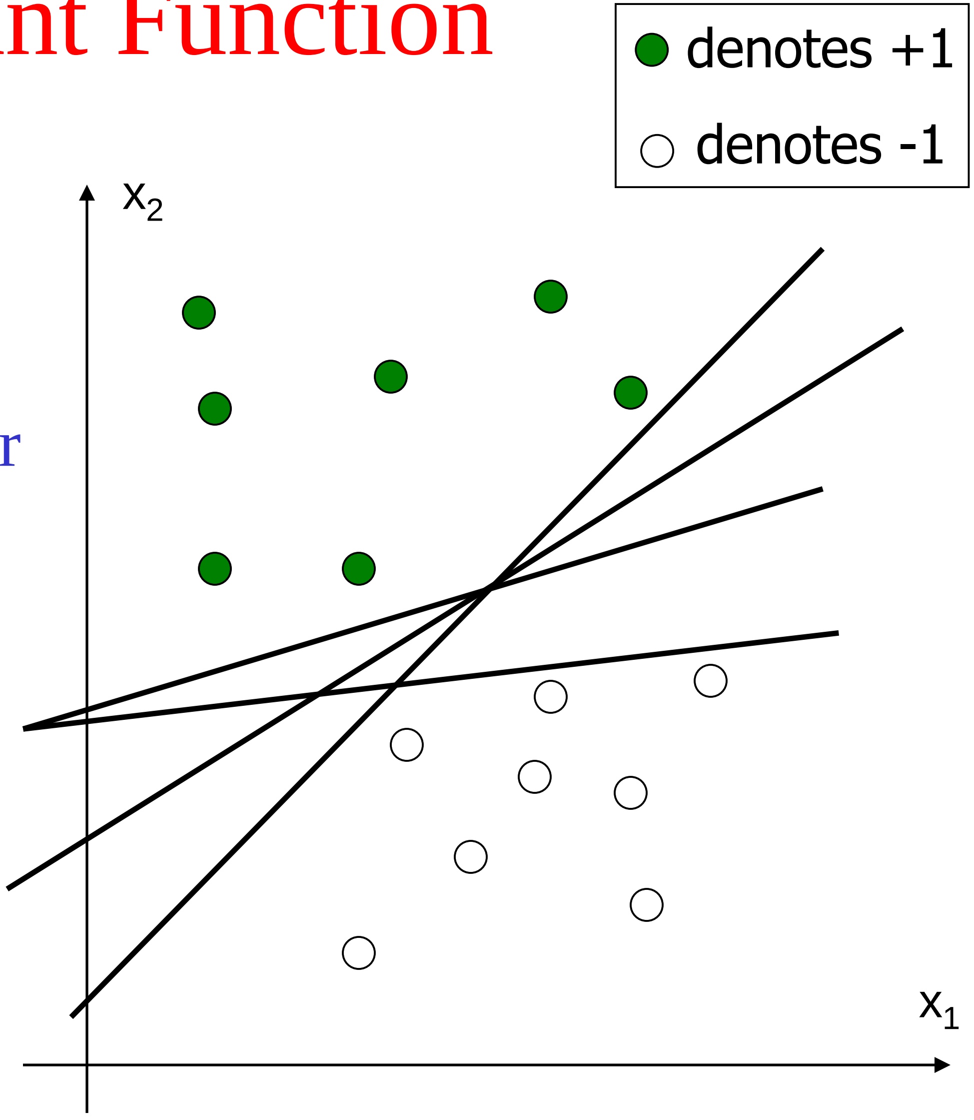
- How would you classify these points using a linear discriminant function in order to minimize the error rate?
- Infinite number of answers!



This slide is courtesy of *www1.cs.columbia.edu/~belhumeur/courses/biometrics/2009/svm.ppt*

Linear Discriminant Function

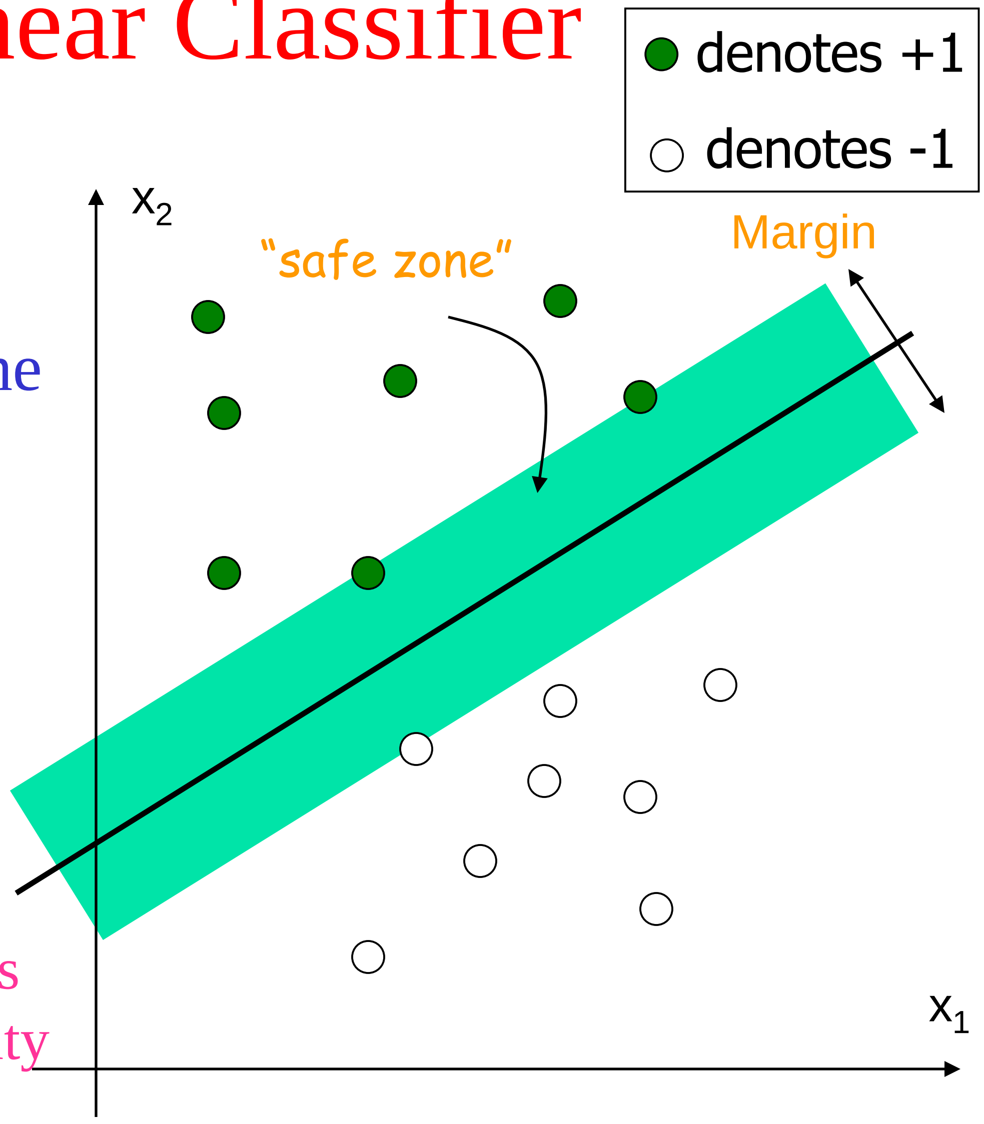
- How would you classify these points using a linear discriminant function in order to minimize the error rate?
- Infinite number of answers!
- Which one is the best?



This slide is courtesy of *www1.cs.columbia.edu/~belhumeur/courses/biometrics/2009/svm.ppt*

Large Margin Linear Classifier

- The linear discriminant function (classifier) with the maximum margin is the best
- Margin is defined as the width that the boundary could be increased by before hitting a data point
- Why it is the best?
  - Robust to outliers and thus strong generalization ability



This slide is courtesy of *www1.cs.columbia.edu/~belhumeur/courses/biometrics/2009/svm.ppt*

Large Margin Linear Classifier

- Given a set of data points:

$\{(\mathbf{x}_i, y_i)\},\ i = 1, 2, \cdots, n$, where

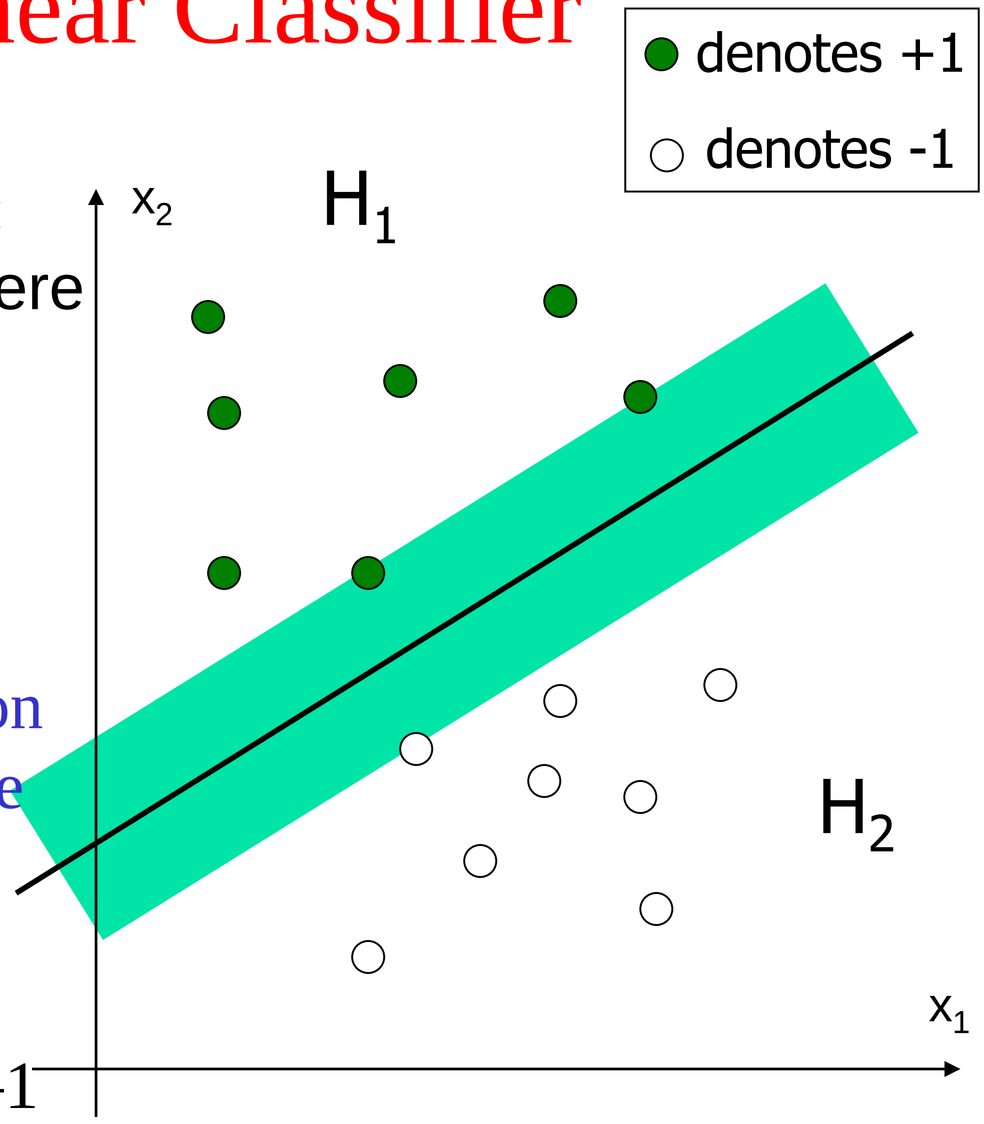
H$_1$: For $y_i = +1,\ \ \mathbf{w}^T\mathbf{x}_i + b > 0$

H$_2$: For $y_i = -1,\ \ \mathbf{w}^T\mathbf{x}_i + b < 0$

- With a scale transformation on both *w* and *b*, the above is equivalent to

For $y_i = +1,\ \ \mathbf{w}^T\mathbf{x}_i + b \geq 1$

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



This slide is courtesy of *www1.cs.columbia.edu/~belhumeur/courses/biometrics/2009/svm.ppt*

Large Margin Linear Classifier

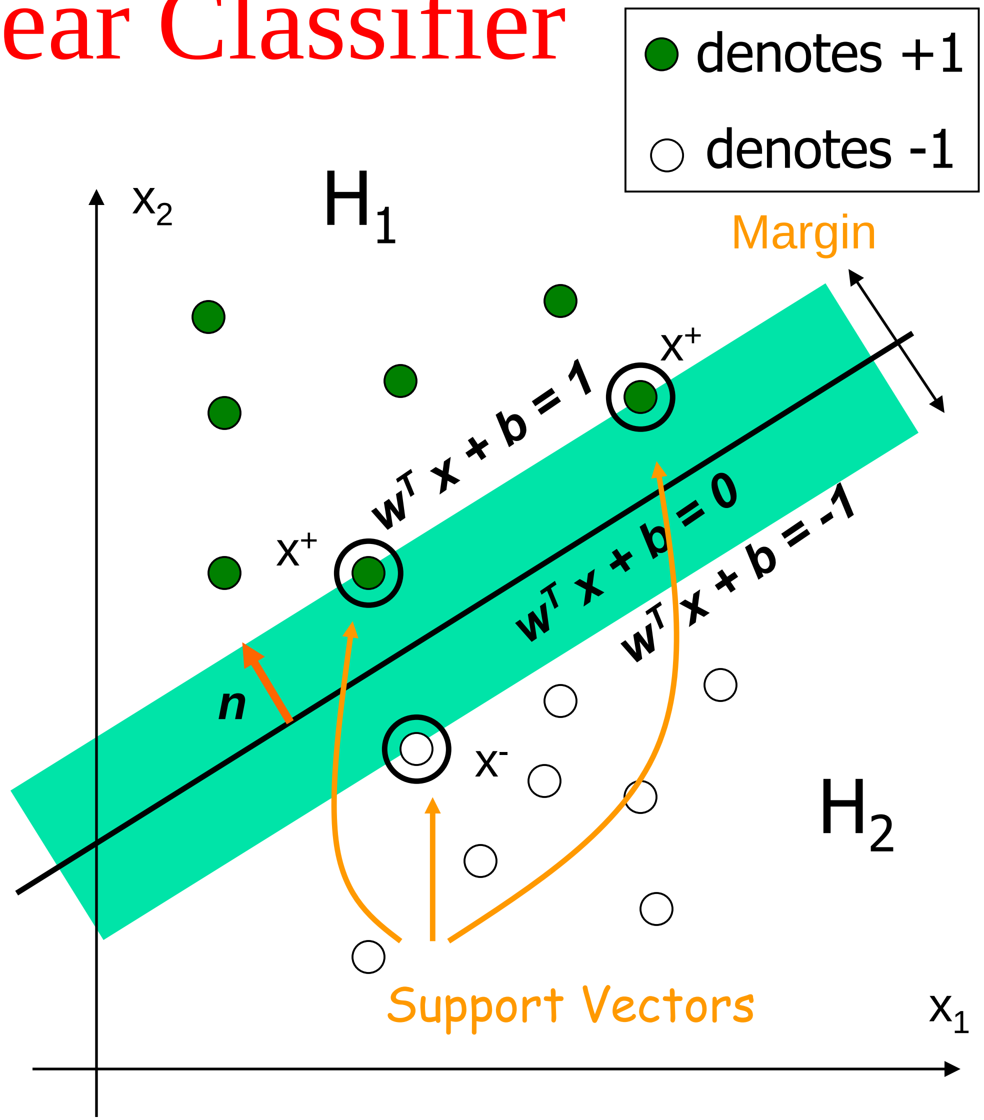
- We know that

$$\mathbf{w}^T \mathbf{x}^+ + b = 1$$
$$\mathbf{w}^T \mathbf{x}^- + b = -1$$

- The margin width is:

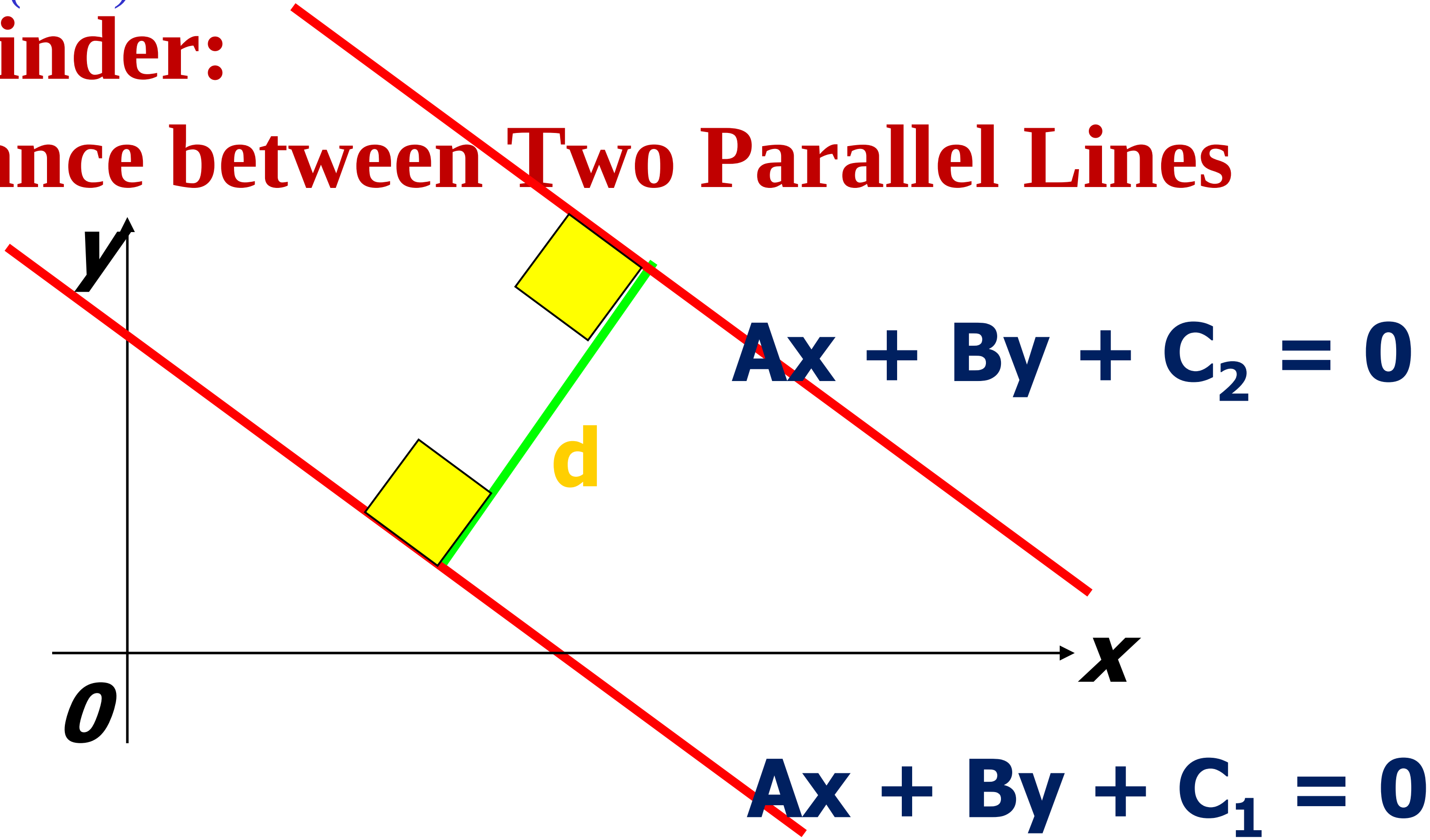
$$M = (\mathbf{x}^+ - \mathbf{x}^-) \cdot \mathbf{n}$$
$$= (\mathbf{x}^+ - \mathbf{x}^-) \cdot \frac{\mathbf{w}}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|}$$

$$\|w\| = \sqrt{w_1^2 + w_2^2 + \ldots + w_n^2}$$



This slide is courtesy of *www1.cs.columbia.edu/~belhumeur/courses/biometrics/2009/svm.ppt*

Reminder:
Distance between Two Parallel Lines



$$d = \left| \frac{C_2 - C_1}{\sqrt{A^2 + B^2}} \right|$$

Large Margin Linear Classifier

- Formulation:

$$\text{minimize} \quad \frac{1}{2}\|\mathbf{w}\|^2$$

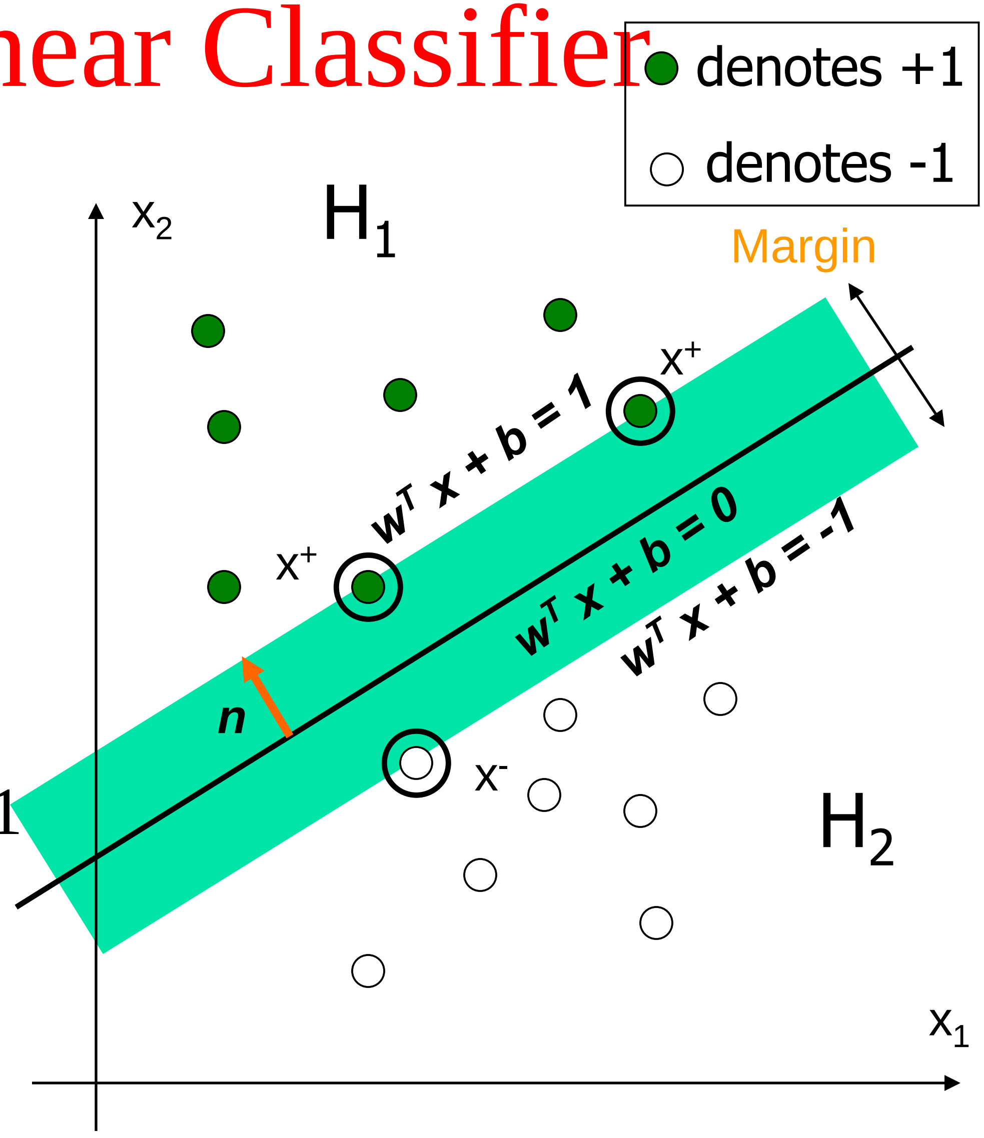
such that

$$\text{For } y_i = +1, \quad \mathbf{w}^T\mathbf{x}_i + b \geq 1$$

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

Which is the same as

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$$



This slide is courtesy of *www1.cs.columbia.edu/~belhumeur/courses/biometrics/2009/svm.ppt*

The Dual Problem

$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{subject to } \alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i y_i = 0$$

- n – number of training records
- This is a quadratic programming (QP) problem
  - A global maximum of α_i can always be found
- **w** can be recovered by

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

get b from $y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 = 0$,

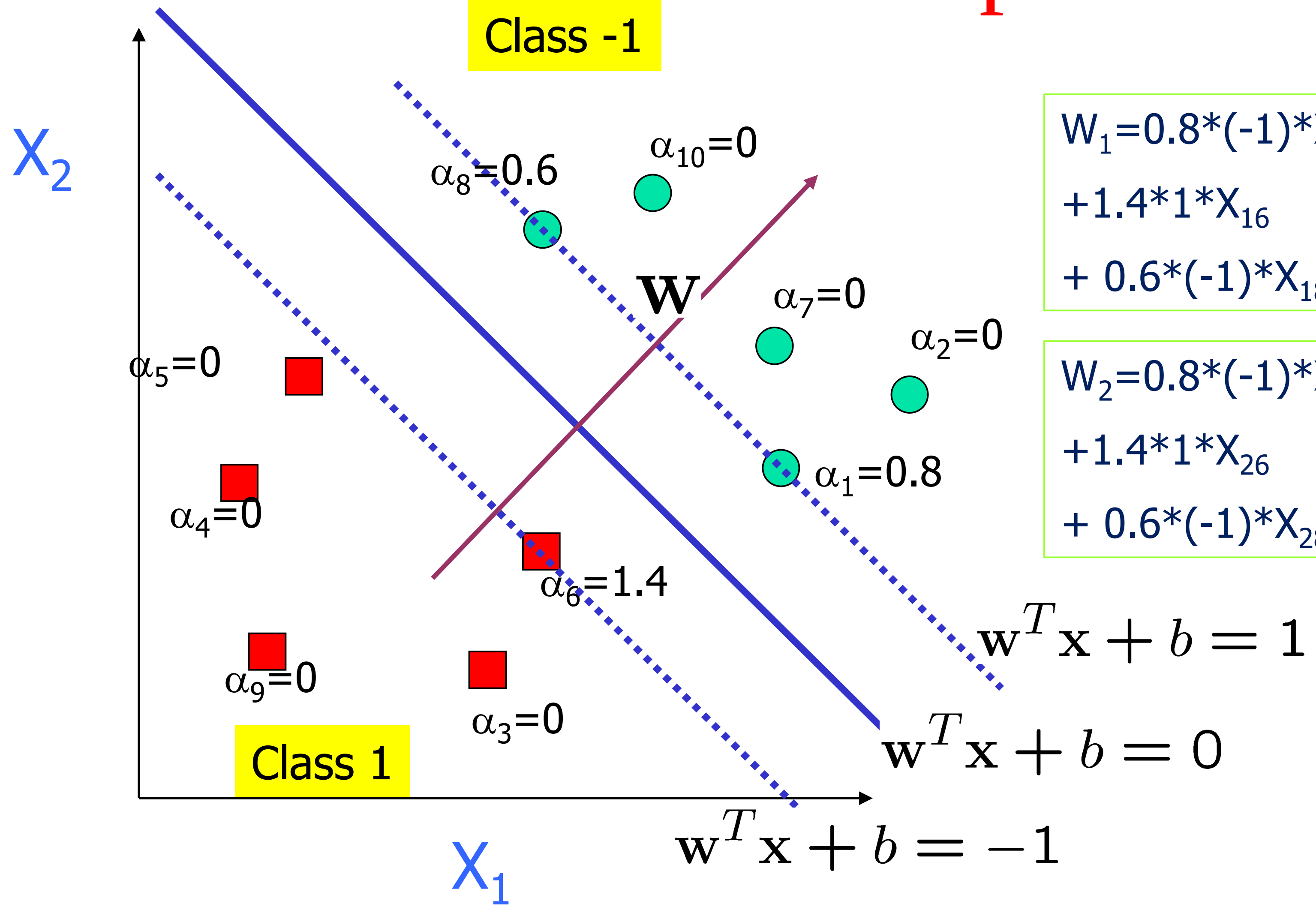
where $\mathbf{x}_i$ is support vector

Characteristics of the Solution

- Many of the α_i are zero
  - **w** is a linear combination of a small number of data points
  - This "sparse" representation can be viewed as data compression as in the construction of knn classifier
- x_i with non-zero α_i are called support vectors (SV)
  - The decision boundary is determined only by the SV
  - Let t_j (j=1, ..., s) be the indices of the s support vectors. We can write $\mathbf{w} = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$
- For testing with a new data **z**
  - Compute $\mathbf{w}^T\mathbf{z} + b = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} (\mathbf{x}_{t_j}^T \mathbf{z}) + b$ and classify **z** as class 1 if the sum is positive, and class -1 otherwise. *Note*: **w** need not be formed explicitly

$$\mathbf{w} = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$$

A Geometrical Interpretation



W_1=0.8*(-1)*X_{11}
+1.4*1*X_{16}
+ 0.6*(-1)*X_{18}

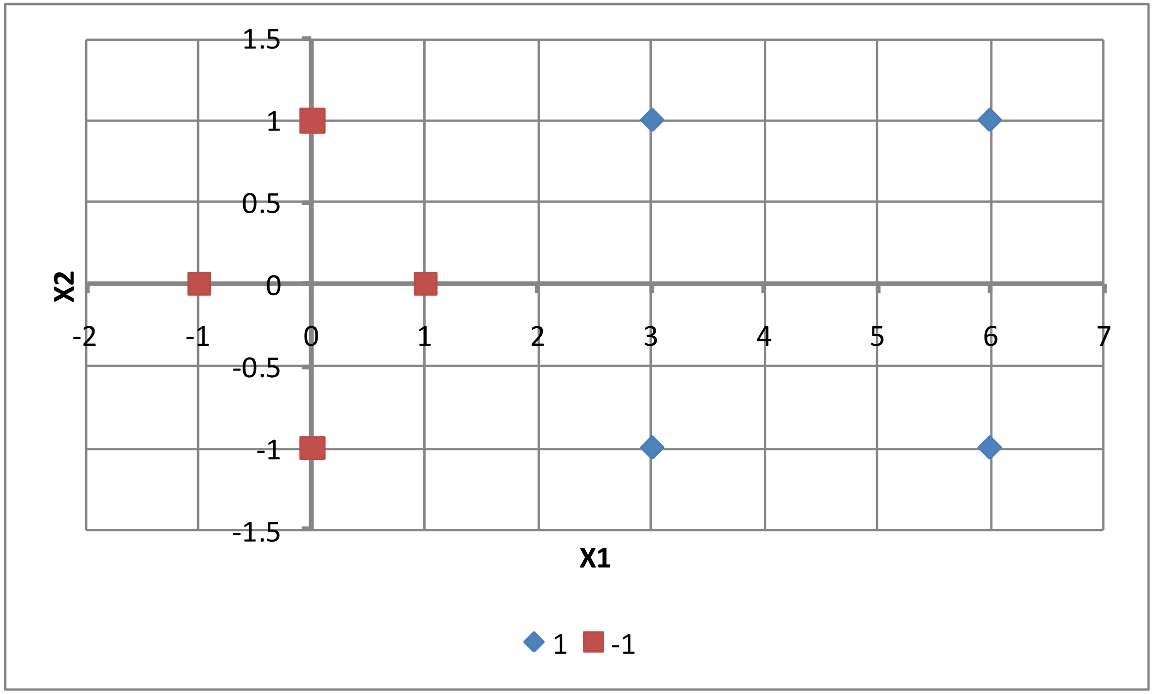
W_2=0.8*(-1)*X_{21}
+1.4*1*X_{26}
+ 0.6*(-1)*X_{28}

Linear SVM Example

Source : Dan Ventura, Brigham Young University, March 12, 2009

i	X1	X2	Y
1	3	1	1
2	3	-1	1
3	6	1	1
4	6	-1	1
5	1	0	-1
6	0	1	-1
7	0	-1	-1
8	-1	0	-1

Which line is the best discriminating function for this data?



How many unknown variables do we need to find?

Linear SVM Example (cont.)

$$Q = \sum_{i=1}^{n} \sum_{i=1}^{n} y_i y_j x_i^T x_j$$

i	X1	X2	Y
1	3	1	1
2	3	-1	1
3	6	1	1
4	6	-1	1
5	1	0	-1
6	0	1	-1
7	0	-1	-1
8	-1	0	-1

Example:

$i = 1, j = 2$: 1*1*(3*3 + 1*(-1)) = 8

i / j	1	2	3	4	5	6	7	8
1	10	8	19	17	-3	-1	1	3
2	8	10	17	19	-3	1	-1	3
3	19	17	37	35	-6	-1	1	6
4	17	19	35	37	-6	1	-1	6
5	-3	-3	-6	-6	1	0	0	-1
6	-1	1	-1	1	0	1	-1	0
7	1	-1	1	-1	0	-1	1	0
8	3	3	6	6	-1	0	0	1

Linear SVM Example: The Optimal Solution

$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{subject to } \alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i y_i = 0$$

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

i	X1	X2	Y	alpha	alpha*Y	
1	3	1	1	0.25	0.25	s2
2	3	-1	1	0.25	0.25	s3
3	6	1	1	0	0	
4	6	-1	1	0	0	
5	1	0	-1	0.5	-0.5	s1
6	0	1	-1	0	0	
7	0	-1	-1	0	0	
8	-1	0	-1	0	0	
Total				**1.00**	**0.00**	
W(alpha)					**0.5**	

$w_1 = 1$

$w_2 = 0$

$b = -2$

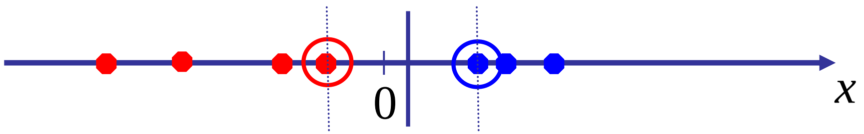
$$y_j(w^T x_j + b) = 1$$

Why Is SVM Effective on High Dimensional Data?

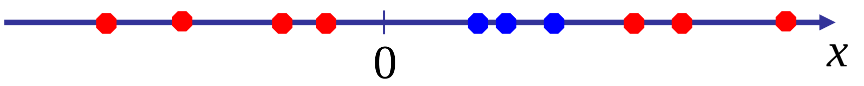
- The complexity of trained classifier is characterized by the # of support vectors rather than the dimensionality of the data
- The support vectors are the essential or critical training examples — they lie closest to the decision boundary (MMH)
- If all other training examples are removed and the training 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

Non-linear SVMs

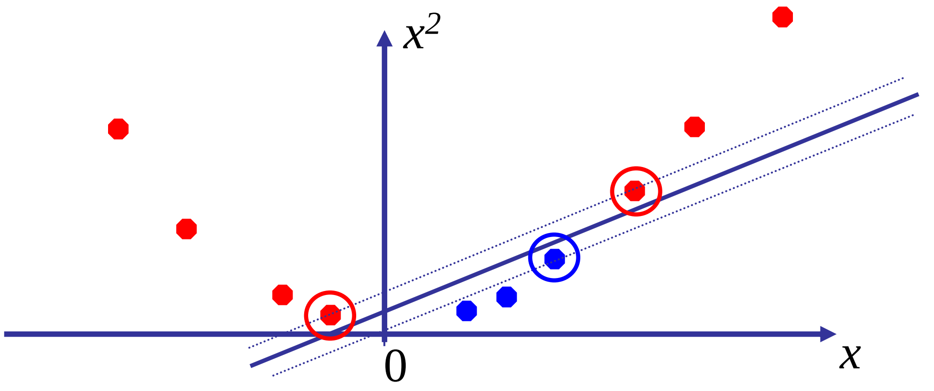
- Datasets that are linearly separable with noise work out great:



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



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

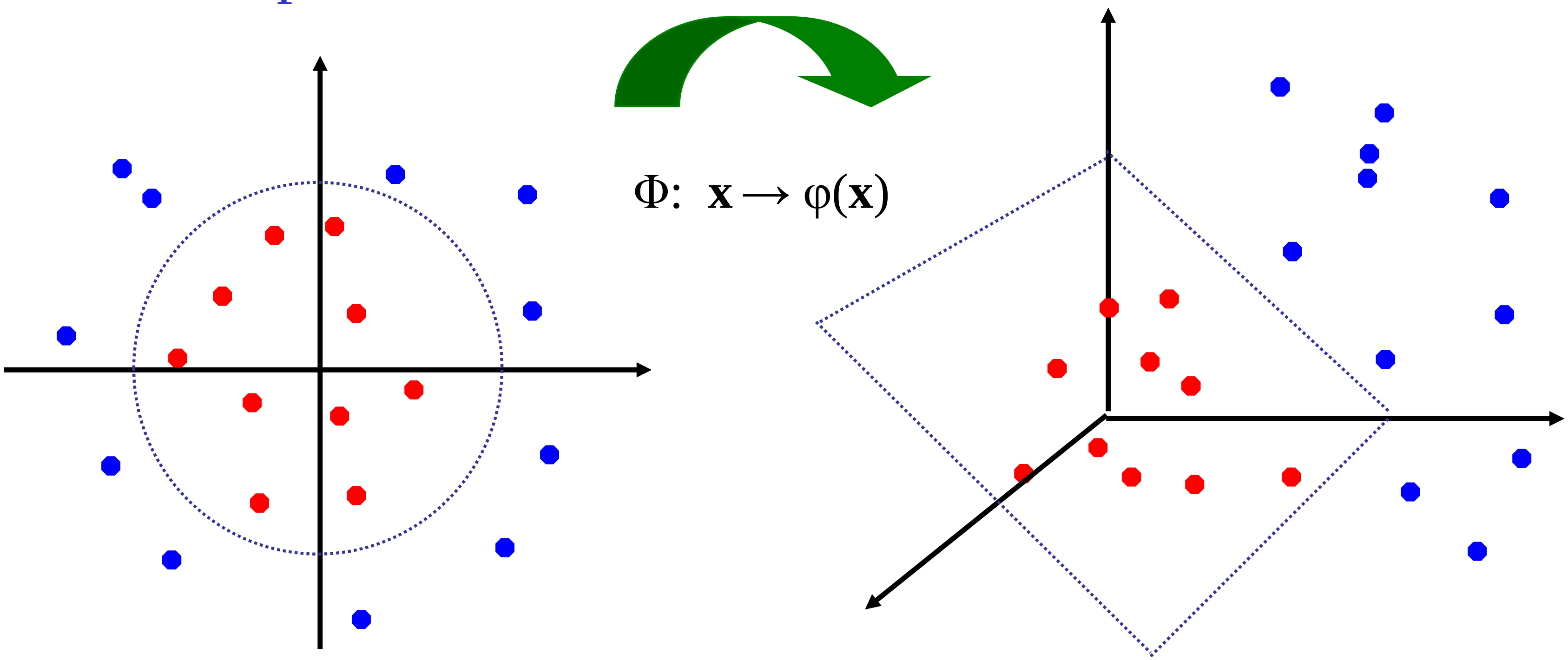


$$(x_k, x_k^2)$$

This slide is courtesy of *www.iro.umontreal.ca/~pift6080/documents/papers/**svm**_tutorial.**ppt***

Non-linear SVMs: Feature Space

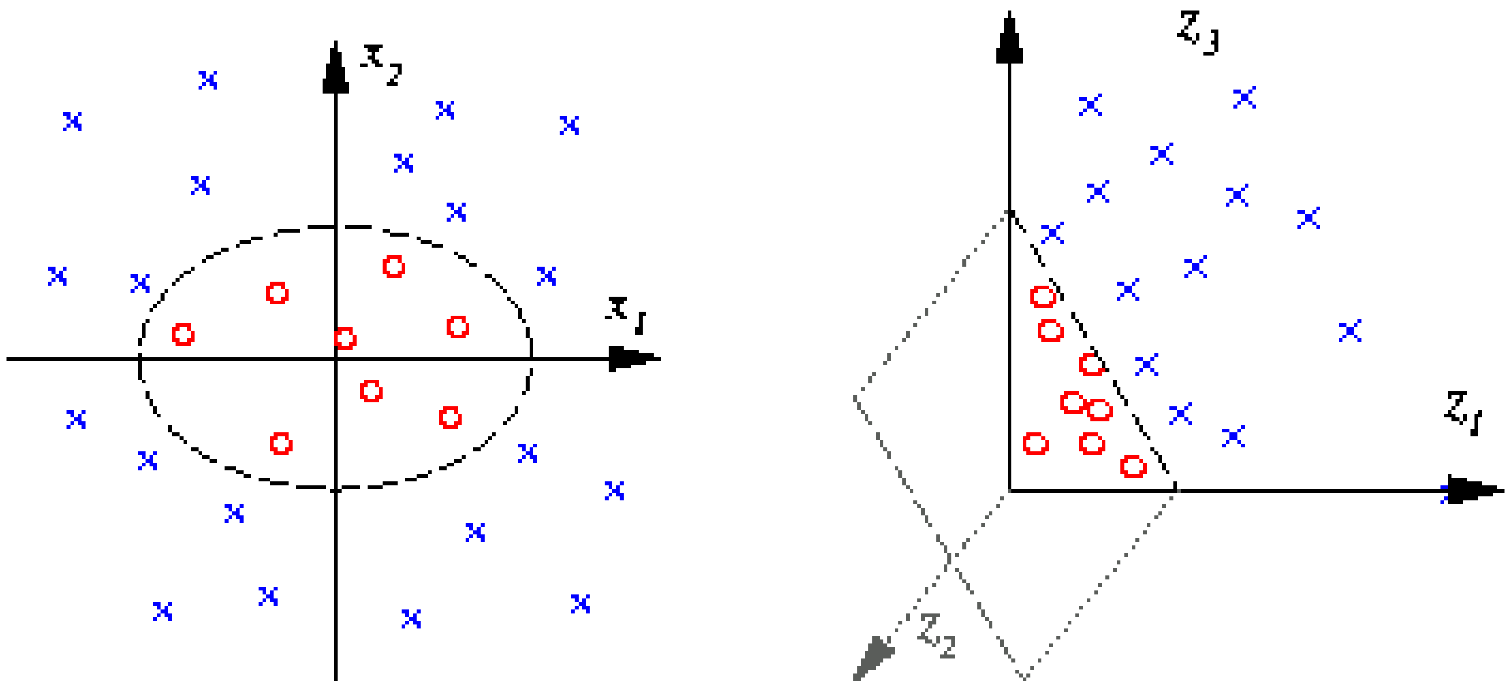
- General idea: the original input space can be mapped to some higher-dimensional feature space where the training set is separable:



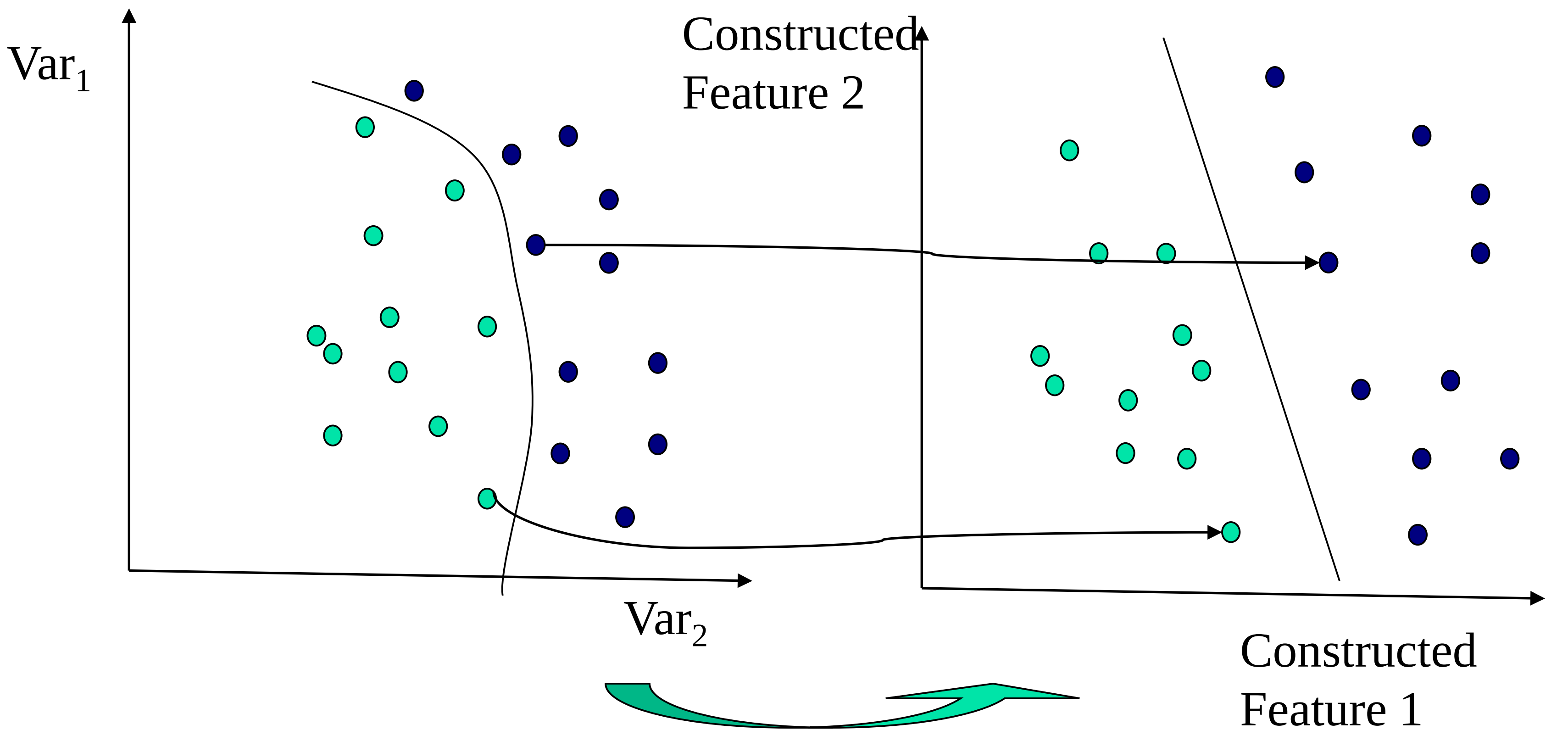
This slide is courtesy of *www.iro.umontreal.ca/~pift6080/documents/papers/**svm**_tutorial.**ppt***

Example: All Degree 2 Monomials

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$$

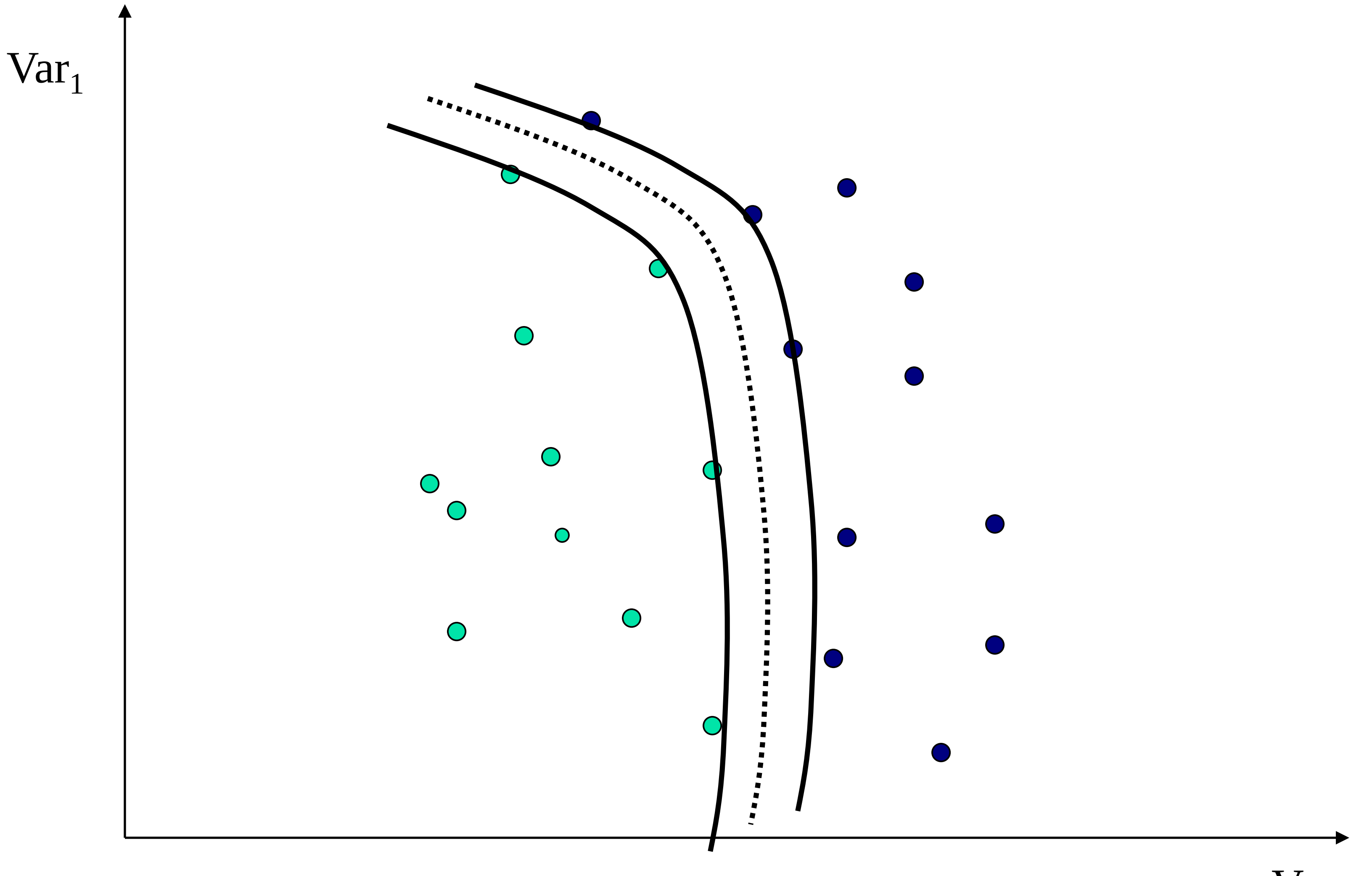


Linear Classifiers in High-Dimensional Spaces



Find function $\Phi(x)$ to map to a different space

Advantages of Non-Linear Surfaces



Nonlinear SVMs: The Kernel Trick

- With this mapping, our discriminant function is now:

$$g(x_j) = w^T \phi(x_j) + b = \sum_{i \in SV} \alpha_i y_i \boxed{\phi(x_i)^T \phi(x_j)} + b$$

- No need to know this mapping explicitly, because we only use the dot product of feature vectors in both the training and test.
- A *kernel function* is defined as a function that corresponds to a dot product of two feature vectors in some expanded feature space:

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

Nonlinear SVMs: The Kernel Trick

- An example:

2-dimensional vectors $\mathbf{x}=[x_1 \ \ x_2]$;

let $K(\mathbf{x_i},\mathbf{x_j})=(1 + \mathbf{x_i}^T\mathbf{x_j})^2$, (Polynomial kernel of degree 2)

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

$K(\mathbf{x_i},\mathbf{x_j})=(1 + \mathbf{x_i}^T\mathbf{x_j})^2$,

$= 1+ x_{i1}^2x_{j1}^2 + 2\, x_{i1}x_{j1}\, x_{i2}x_{j2}+ x_{i2}^2x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2}$

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

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

What is the number of new dimensions?

This slide is courtesy of *www.iro.umontreal.ca/~pift6080/documents/papers/**svm**_tutorial.**ppt***

Nonlinear SVMs: The Kernel Trick

- Examples of commonly-used kernel functions:
  - Linear kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
  - Polynomial kernel of degree *P*: $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^p$
  - Gaussian (Radial-Basis Function (RBF)) kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2}{2\sigma^2}\right)$$

  - Sigmoid:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta_0 \mathbf{x}_i^T \mathbf{x}_j + \beta_1)$$

- In general, functions that satisfy *Mercer's condition* can be kernel functions.

What Functions are Kernels?

- For some functions $K(\mathbf{x}_i,\mathbf{x}_j)$ checking that $K(\mathbf{x}_i,\mathbf{x}_j)= \varphi(\mathbf{x}_i)^{\mathbf{T}}\varphi(\mathbf{x}_j)$ can be cumbersome.
- Mercer's theorem:

  Every semi-positive definite symmetric function is a kernel

- Semi-positive definite symmetric functions correspond to a semi-positive definite symmetric Gram matrix:

K=

$K(\mathbf{x}_1,\mathbf{x}_1)$	$K(\mathbf{x}_1,\mathbf{x}_2)$	$K(\mathbf{x}_1,\mathbf{x}_3)$	…	$K(\mathbf{x}_1,\mathbf{x}_n)$
$K(\mathbf{x}_2,\mathbf{x}_1)$	$K(\mathbf{x}_2,\mathbf{x}_2)$	$K(\mathbf{x}_2,\mathbf{x}_3)$		$K(\mathbf{x}_2,\mathbf{x}_n)$
…	…	…	…	…
$K(\mathbf{x}_n,\mathbf{x}_1)$	$K(\mathbf{x}_n,\mathbf{x}_2)$	$K(\mathbf{x}_n,\mathbf{x}_3)$	…	$K(\mathbf{x}_n,\mathbf{x}_n)$

1) $K(x; y) = K(y; x)$

2) $\forall c \in R^n, c^T K c \geq 0$

For more details, see HTF, section 5.8.1

Nonlinear SVM: Optimization

- Original SVM formulation
  - *n* inequality constraints
  - *n positivity constraints*
  - *n* number of ξ variables

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_i \xi_i$$

$$s.t.\ \ y_i(w \cdot \Phi(x) + b) \geq 1 - \xi_i, \forall x_i$$

$$\xi_i \geq 0$$

- The solution of the discriminant function is

$$g(\mathbf{x}) = \sum_{i \in \mathrm{SV}} \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b$$

- The (Wolfe) dual of this problem
  - one equality constraint
  - *n* positivity constraints
  - *n* inequality constraints
  - *n* number of α variables (Lagrange multipliers)
  - Objective function more complicated

$$\min_{a_i} \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j (\Phi(x_i) \cdot \Phi(x_j)) - \sum_i y_i \alpha_i$$

$$s.t.\ \ \mathrm{C} \geq \alpha_{\mathrm{i}} y_i \geq 0, \forall i$$

$$\sum_i \alpha_i = 0$$

- NOTICE: Data only appear as $\Phi(x_i) \cdot \Phi(x_j)$

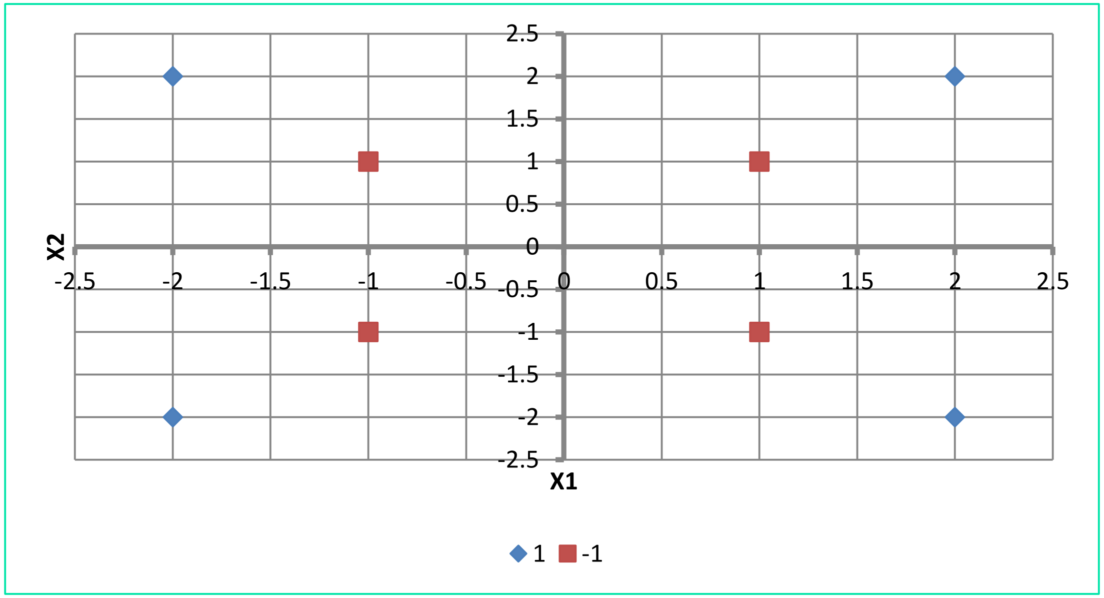
Support Vector Machine: Algorithm

- 1. Choose a kernel function
- 2. Choose a value for C
  - SVM complexity constant which sets the tolerance for misclassification, where higher *C* values allow for 'softer' boundaries and lower values create 'harder' boundaries. A complexity constant that is too large can lead to over-fitting, while values that are too small may result in over-generalization (RM)
- 3. Solve the quadratic programming problem (many software packages available)
- 4. Construct the discriminant function from the support vectors

Non-linear SVM Example

i	X1	X2	Y
1	2	2	1
2	2	-2	1
3	-2	-2	1
4	-2	2	1
5	1	1	-1
6	1	-1	-1
7	-1	-1	-1
8	-1	1	-1

Which kernel should we use?



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 (not that trivial)
  - Linear regression
    - Linear output neuron
  - Logistic regression
    - Sigmoid output neuron

SVM Related Links

- SVM Website
  - http://www.kernel-machines.org/
- Representative implementations
  - LIBSVM: an efficient implementation of SVM, multi-class classifications, nu-SVM, one-class SVM, including also various interfaces with java, python, etc.
  - SVM-light: simpler but performance is not better than LIBSVM, support only binary classification and only C language
  - SVM-torch: another recent implementation also written in C.

SVM—Introduction Literature

- "Statistical Learning Theory" by Vapnik: extremely hard to understand, containing many errors too.
- C. J. C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Knowledge Discovery and Data Mining*, 2(2), 1998.
- Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, New York, NY, USA, 1999.
- Hastie, T., Tibshirani, R., and Friedman, J., *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd Edition, Springer Verlag, 2009 (Chapters 4, 12)