



## Support Vector Machine and Kernel Functions

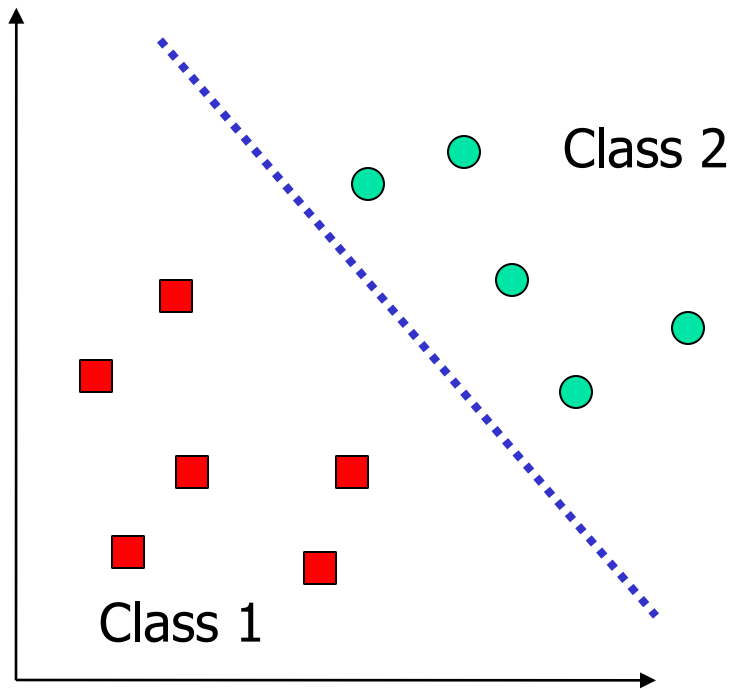
# Support Vector Machines (SVM)

- Supervised learning methods for classification and regression  
relatively **new class of successful learning methods** -
- they can represent **non-linear functions** and they have an **efficient training algorithm**
- derived from statistical learning theory by Vapnik and Chervonenkis (COLT-92)
- SVM got into mainstream because of their exceptional performance in Handwritten Digit Recognition
  - 1.1% error rate which was comparable to a very carefully constructed (and complex) ANN

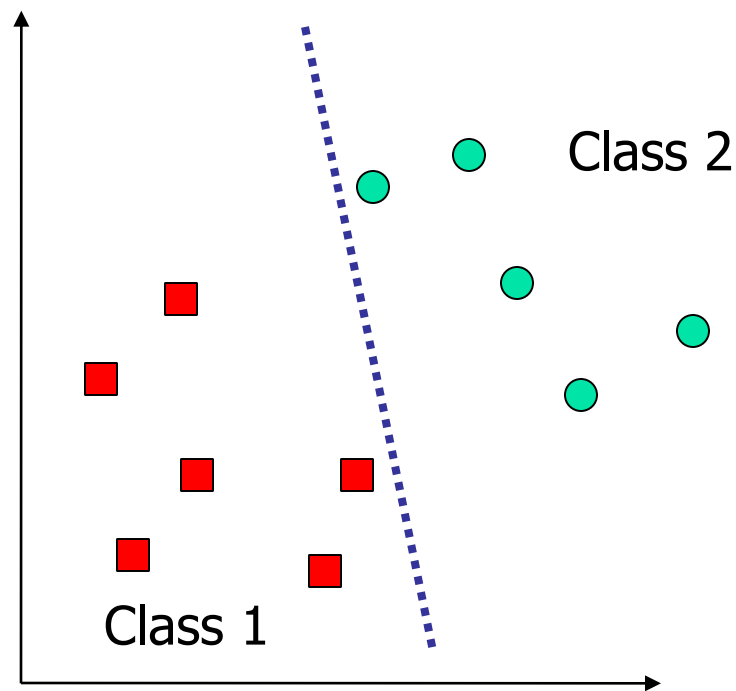
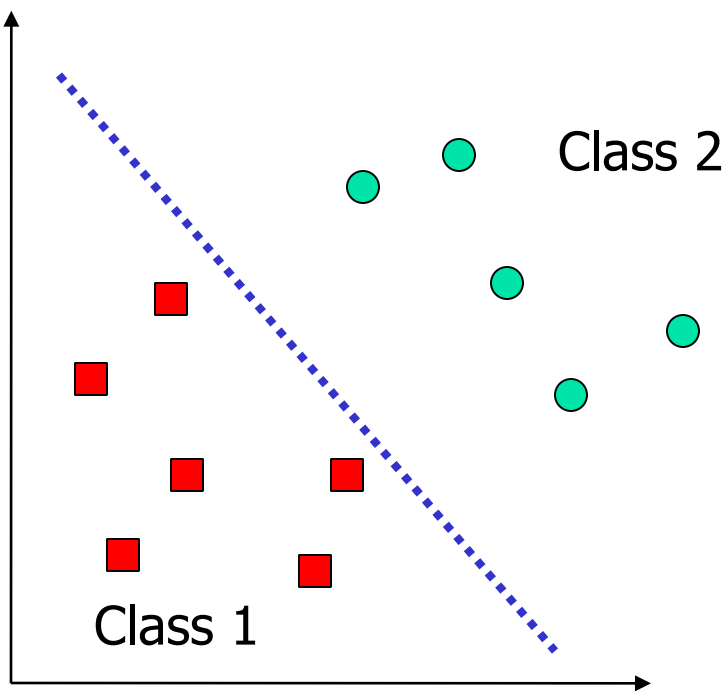
# Two Class Problem: Linear Separable Case

Many decision boundaries can separate these two classes

Which one should we choose?



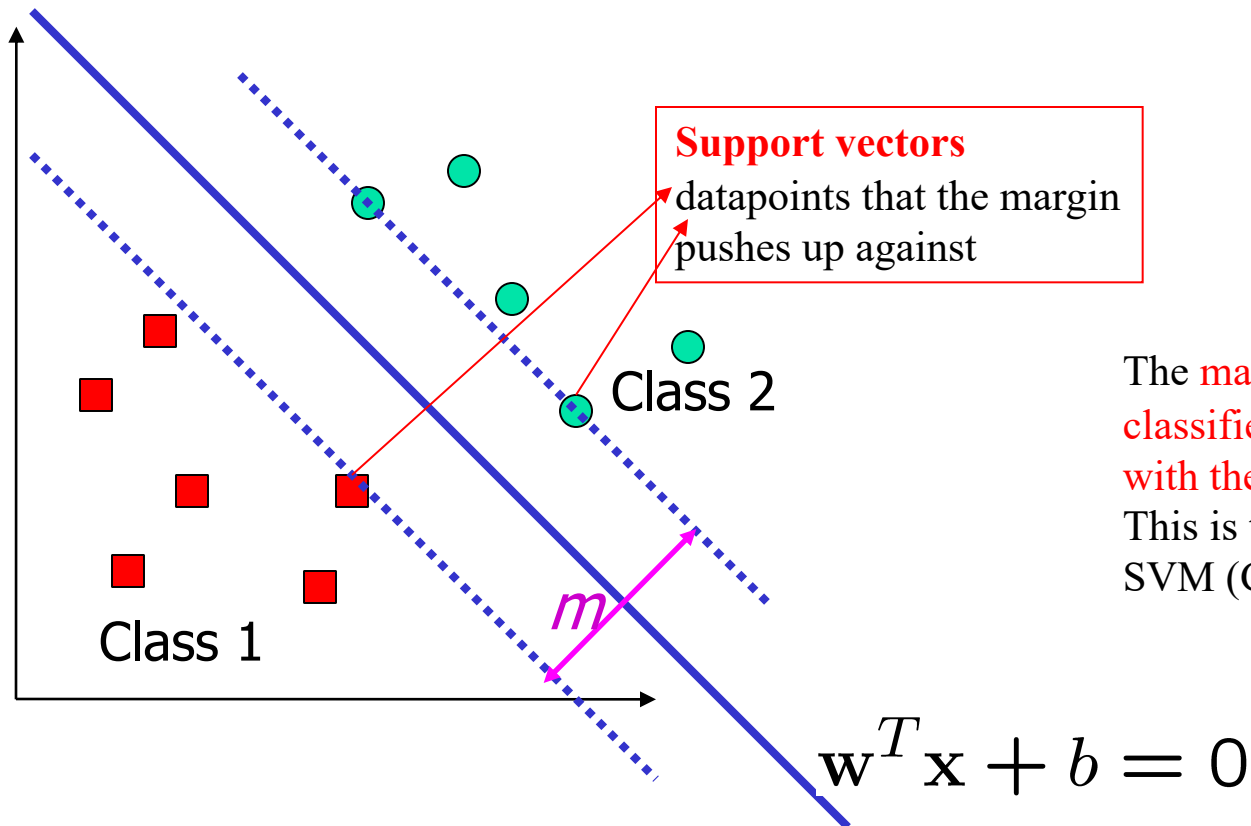
# Example of Bad Decision Boundaries



# Good Decision Boundary: Margin Should Be Large

The decision boundary should be as far away from the data of both classes as possible

- We should maximize the margin,  $m$

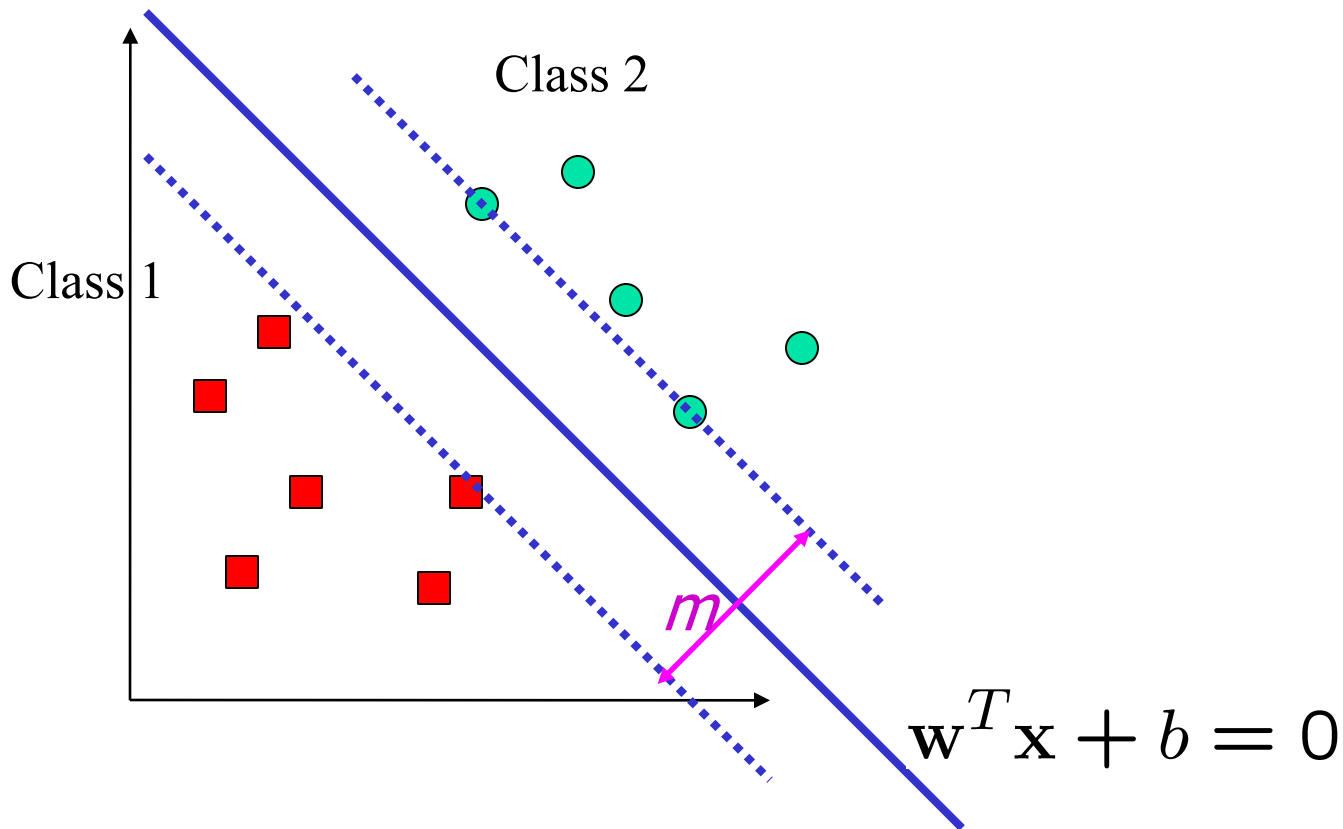


The maximum margin linear classifier is the linear classifier with the maximum margin. This is the simplest kind of SVM (Called a **Linear SVM**)

# What is the value of length of $m$ ?

The decision boundary should be as far away from the data of both classes as possible

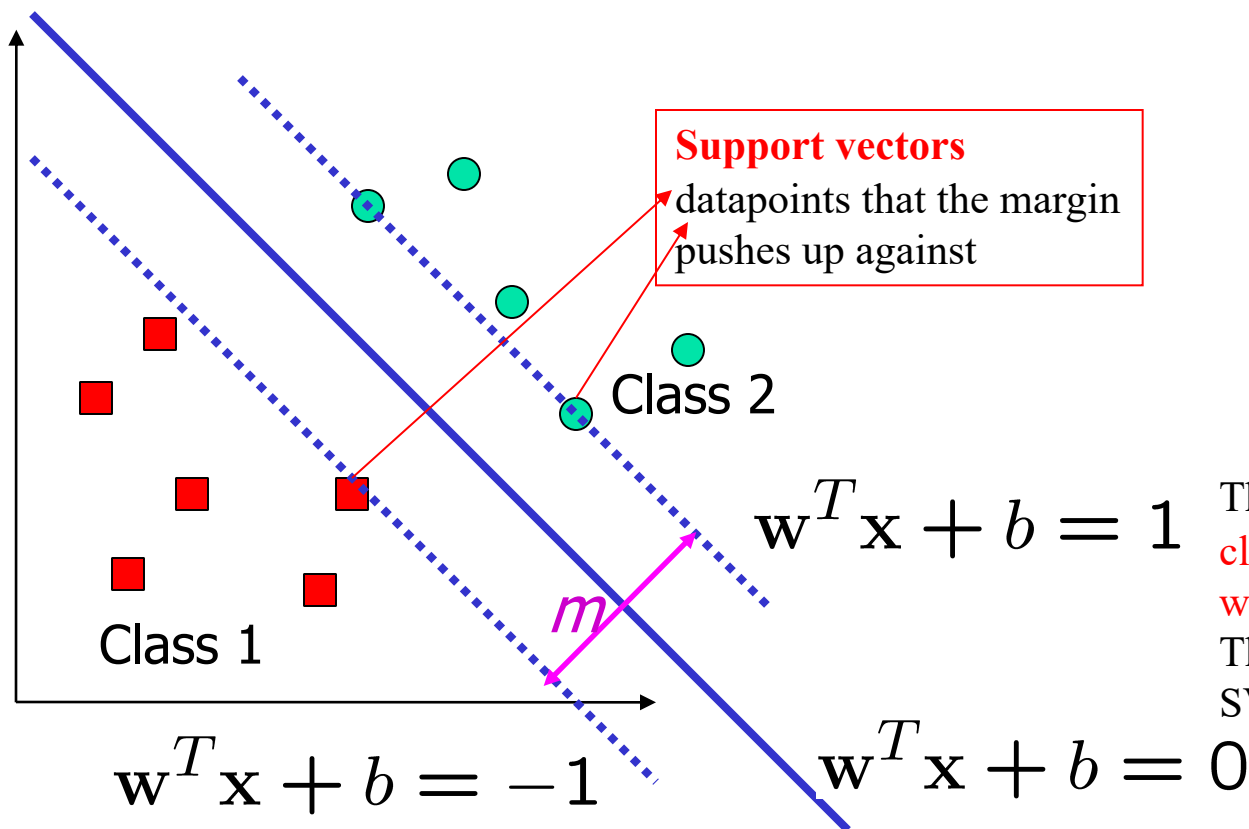
- We should maximize the margin,  $m$



# Good Decision Boundary: Margin Should Be Large

The decision boundary should be as far away from the data of both classes as possible

- We should maximize the margin,  $m$



$$m = \frac{2}{\|\mathbf{w}\|}$$

$$\|\mathbf{x}\| := \sqrt{x_1^2 + \cdots + x_n^2}.$$

The **maximum margin linear classifier** is the **linear classifier** with the **maximum margin**. This is the simplest kind of SVM (Called a **Linear SVM**)

# The Optimization Problem

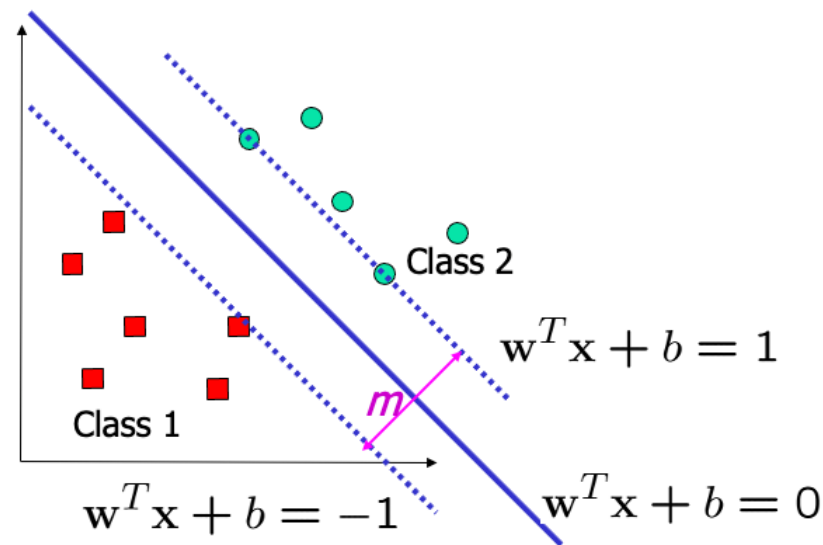
Let  $\{x_1, \dots, x_n\}$  be our data set and let  $y_i \in \{1, -1\}$  be the class label of  $x_i$

The decision boundary should **classify all points correctly**  $\Rightarrow$

A constrained optimization problem

To maximize:

$$m = \frac{2}{\|\mathbf{w}\|}$$



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

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

$$\|\mathbf{w}\|^2 = \mathbf{w}^T \mathbf{w}$$

$$\forall i$$



# Lagrangian of Original Problem

The Lagrangian is    Minimize  $\frac{1}{2}\|\mathbf{w}\|^2$   
subject to  $1 - y_i(\mathbf{w}^T \mathbf{x}_i + b) \leq 0$     for  $i = 1, \dots, n$

– Note that  $\|\mathbf{w}\|^2 = \mathbf{w}^T \mathbf{w}$

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

Lagrangian multipliers

Setting the **gradient of  $\mathcal{L}$  w.r.t.  $\mathbf{w}$  and  $b$  to zero**, we have

$$\mathbf{w} + \sum_{i=1}^n \alpha_i (-y_i) \mathbf{x}_i = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i$$
$$\sum_{i=1}^n \alpha_i y_i = 0$$
$$\alpha_i \geq 0$$

# Change into the Dual Problem

The Lagrangian is  $\mathcal{L} = \frac{1}{2} \mathbf{w}^T \mathbf{w} + \sum_{i=1}^n \alpha_i (1 - y_i (\mathbf{w}^T \mathbf{x}_i + b))$

Just obtained:

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

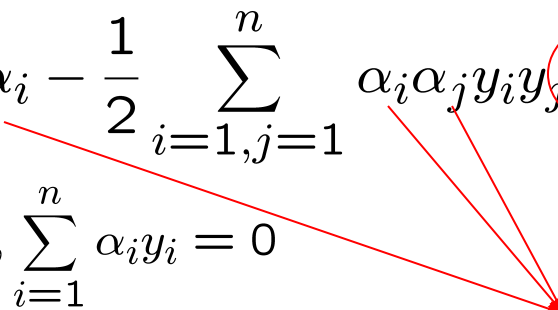
$$\sum_{i=1}^n \alpha_i y_i = 0$$

Transform the Lagrangian into into dual problem:

# The Dual Optimization Problem

The problem now is to:

Dot product of X

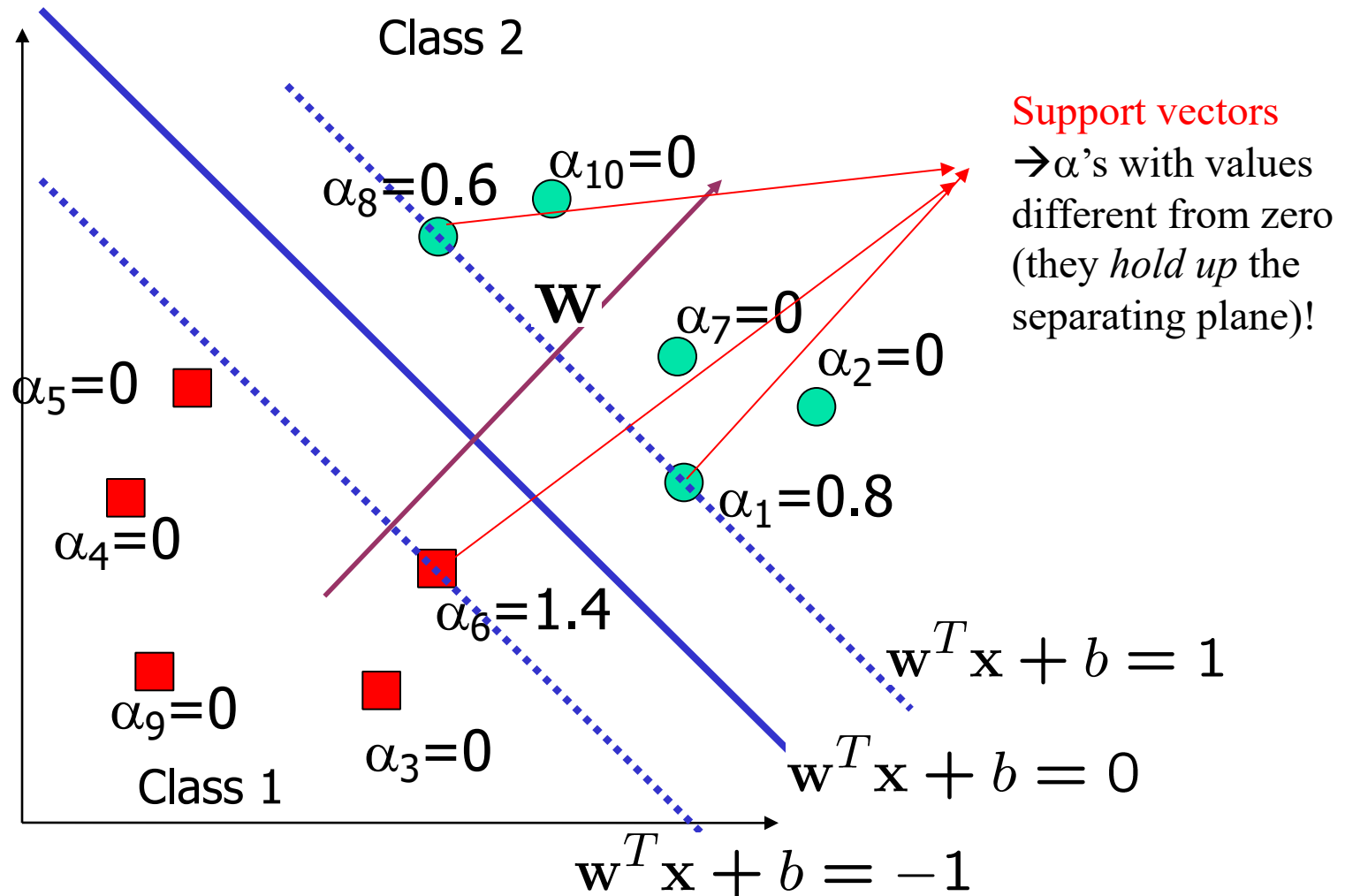
$$\begin{aligned} \max. \quad W(\alpha) &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1, j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \\ \text{subject to } \alpha_i &\geq 0, \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned}$$


$\alpha$ 's  $\rightarrow$  New variables  
(Lagrangian multipliers)

This is a convex quadratic programming (QP) problem

- Global maximum of  $\alpha_i$  can always be found
- $\rightarrow$  well established tools for solving this optimization problem

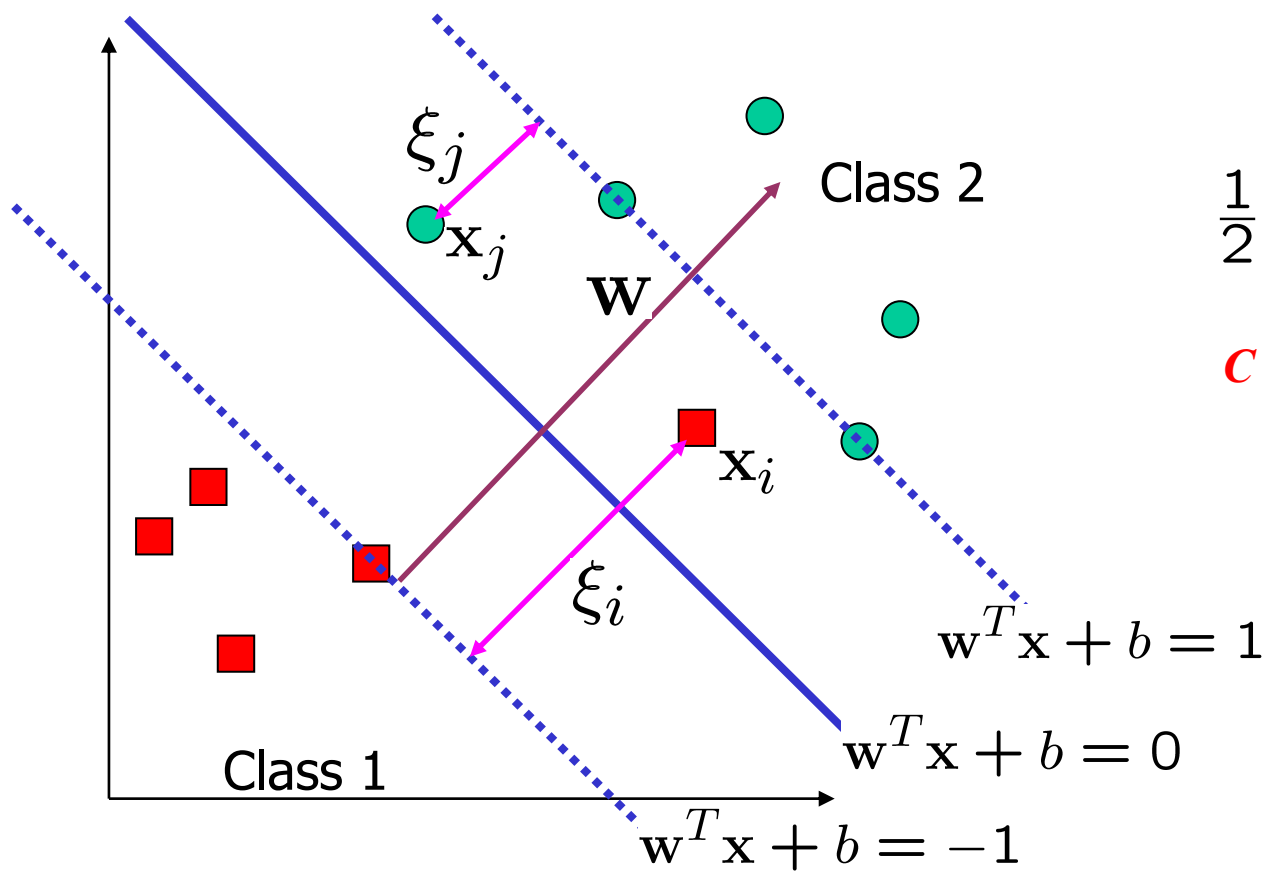
# A Geometrical Interpretation



# Non-linearly Separable Problems

We allow “error”  $\xi_i$  in classification; it is based on the output of the discriminant function  $\mathbf{w}^T \mathbf{x} + b$

$\xi_i$  approximates the number of misclassified samples



New objective function:

$$\frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i$$

$C$  : tradeoff parameter between error and margin; chosen by the user; large  $C$  means a higher penalty to errors

# The Optimization Problem

The dual of the problem is

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

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

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

$\mathbf{w}$  is also recovered as

The only difference with the linear separable case is that there is an upper bound  $C$  on  $\alpha_i$

Once again, a **QP solver** can be used to find  $\alpha_i$  efficiently!!!



## Extension to Non-linear SVMs (Kernel Machines)

# Non-Linear SVM

How could we generalize this procedure to non-linear data?

Vapnik in 1992 showed that transforming input data  $\mathbf{x}_i$  into a higher dimensional makes the problem easier.

Similar to Hidden Layers in ANN

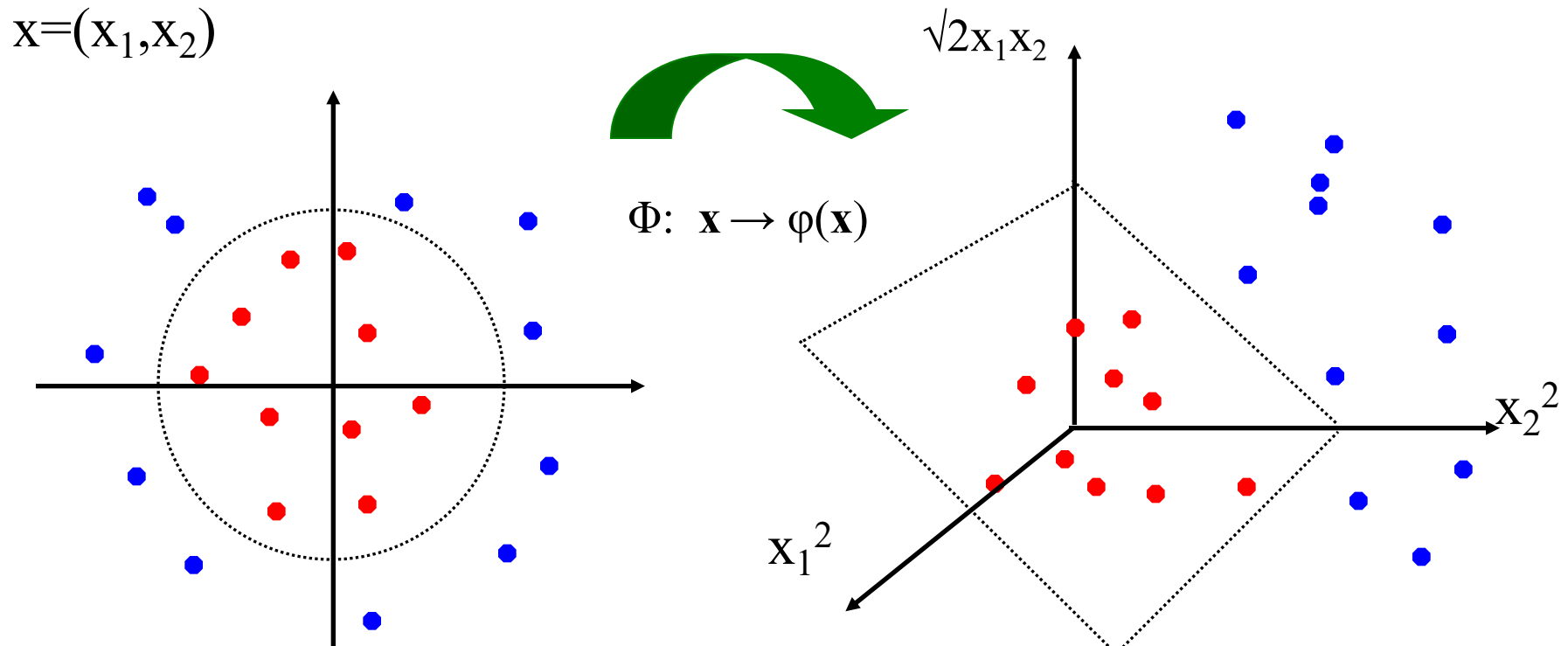
- We know that data appears only as dot products  $(\mathbf{x}_i \cdot \mathbf{x}_j)$
- Suppose we transform the data to some (possibly infinite dimensional) space  $\mathbf{H}$  via a mapping function  $\Phi$  such that the data appears of the form  $\Phi(\mathbf{x}_i)\Phi(\mathbf{x}_j)$

Why?

- Linear operation in  $\mathbf{H}$  is equivalent to non-linear operation in input space.



# Non-linear SVMs: Feature Space

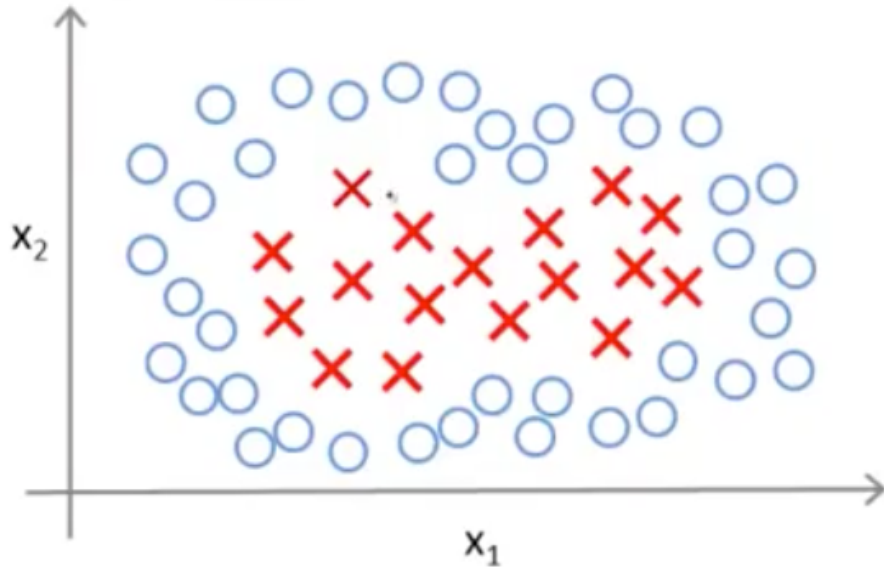


If data are mapped into higher a space of sufficiently high dimension,  
then they will in general be linearly separable;

N data points are in general separable in a space of N-1 dimensions or more!!!

# Non-linear Boundary

## Non-linear Decision Boundary

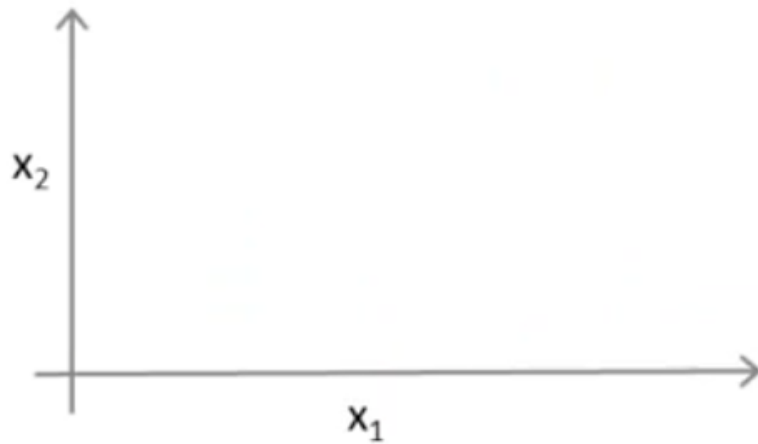


Predict  $y = 1$  if

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \dots \geq 0$$

# Kernel Function

Kernel



Given  $x$ , compute new feature depending on proximity to landmarks  $l^{(1)}, l^{(2)}, l^{(3)}$

# Kernel Function

## Kernels and Similarity

$$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$$

If  $x \approx l^{(1)}$  :

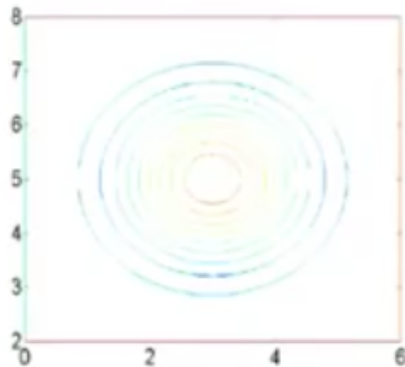
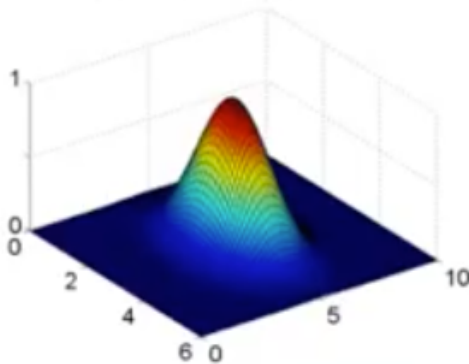
If  $x$  is far from  $l^{(1)}$  :

# Gaussian Kernel

**Example:**

$$l^{(1)} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, \quad f_1 = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$$

$$\sigma^2 = 1$$

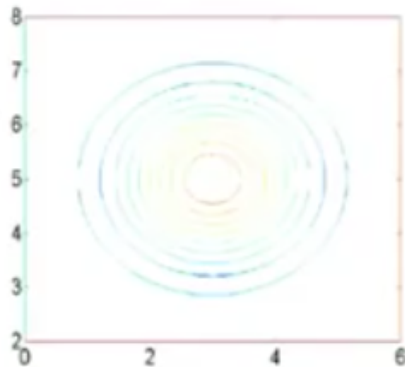
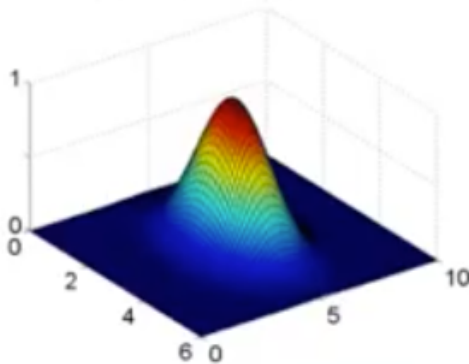


# Gaussian Kernel

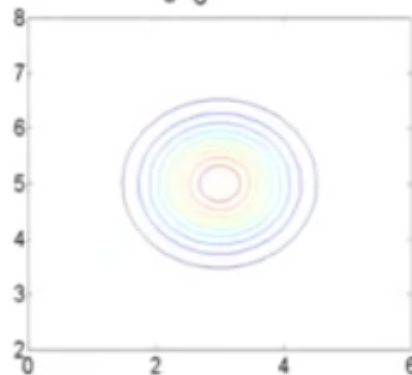
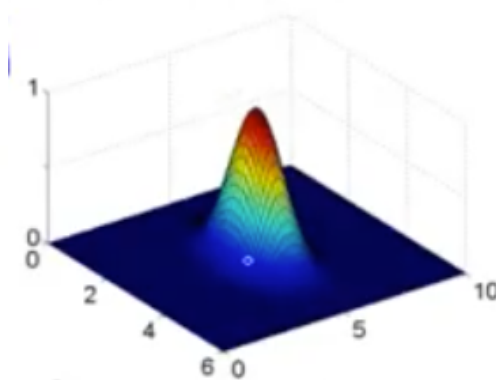
**Example:**

$$l^{(1)} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, \quad f_1 = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$$

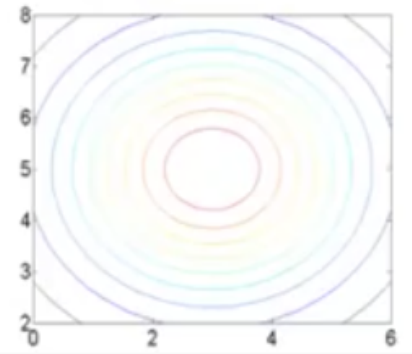
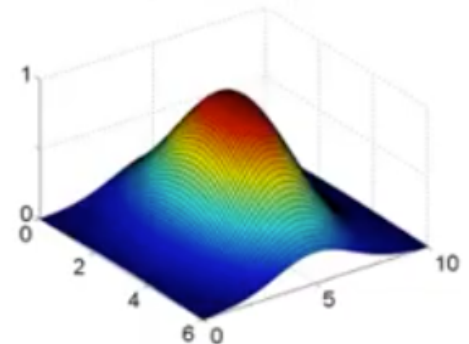
$$\sigma^2 = 1$$



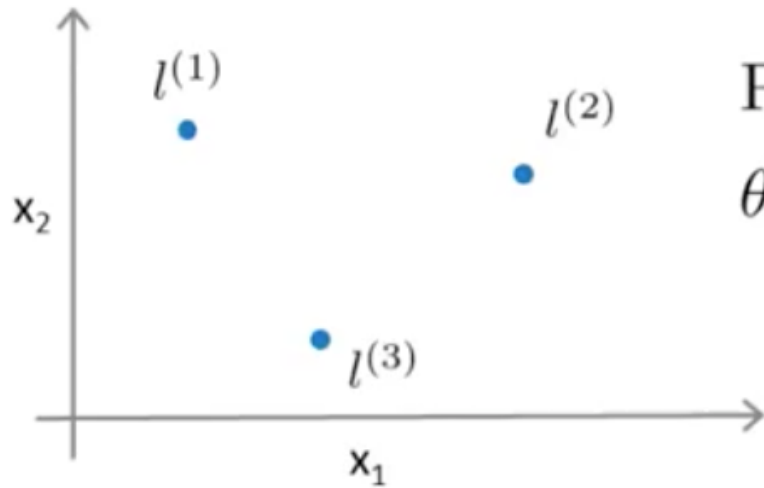
$$\sigma^2 = 0.5$$



$$\sigma^2 = 3$$



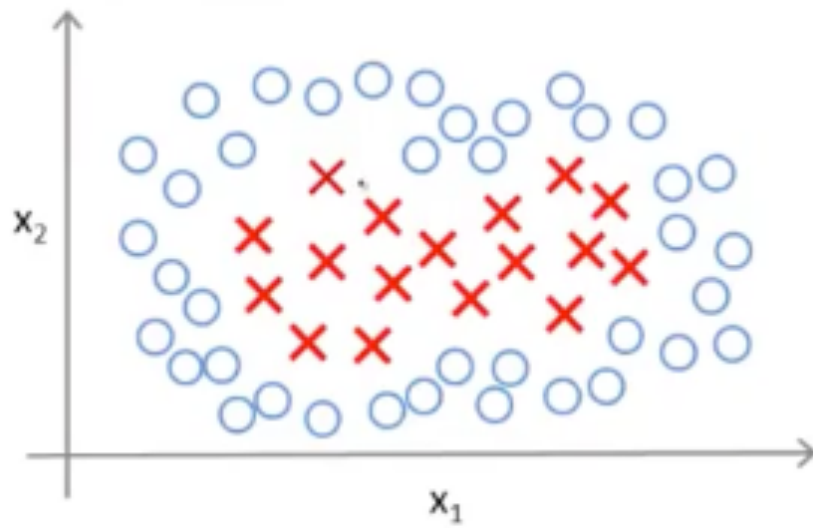
# Classification with the Kernel



Predict “1” when

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$$

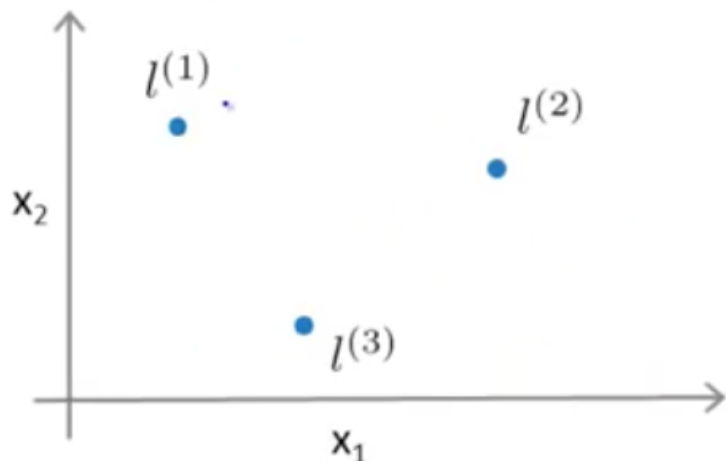
# Higher Dimension Illustration





# How to choose landmarks

## Choosing the landmarks

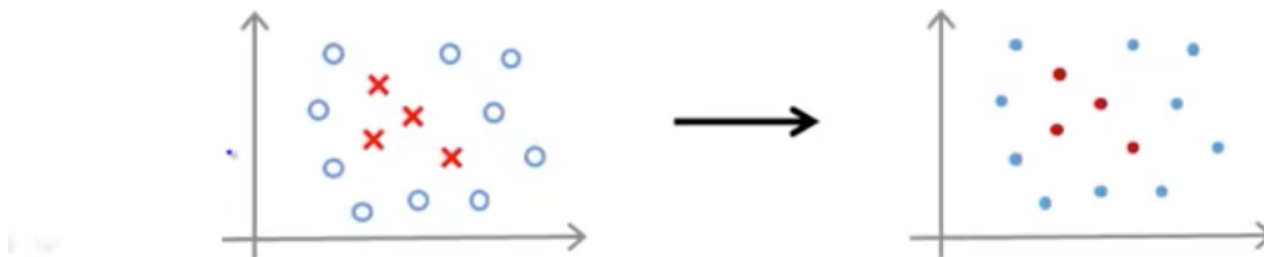


Given  $x$ :

$$f_i = \text{similarity}(x, l^{(i)}) \\ = \exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$$

Predict  $y = 1$  if  $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

Where to get  $l^{(1)}, l^{(2)}, l^{(3)}, \dots$ ?



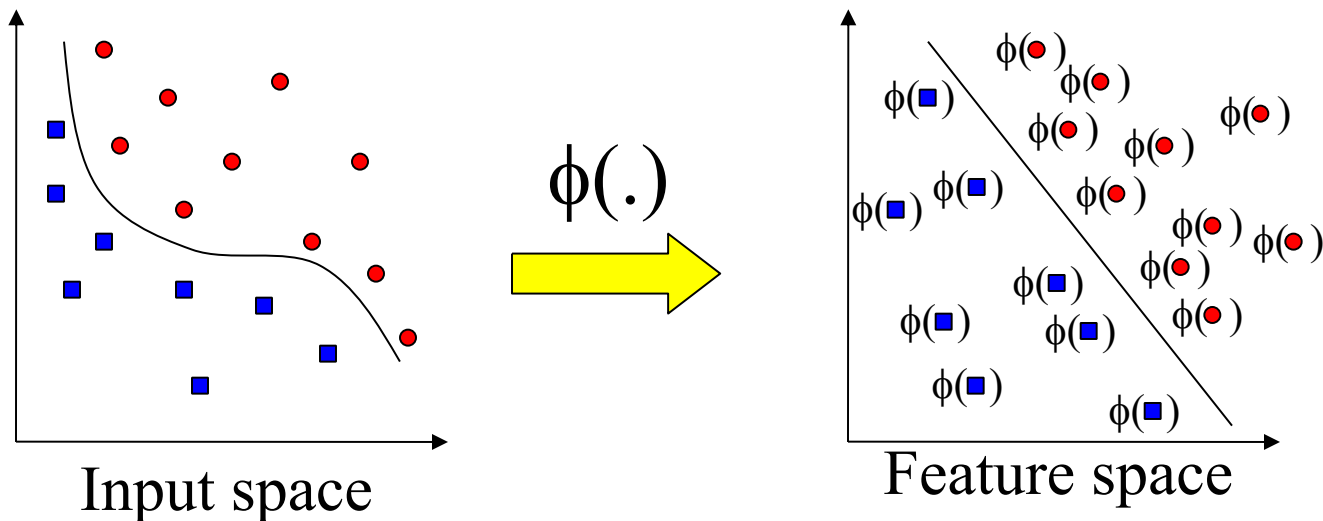
# Transformation to Feature Space

Possible problem of the transformation

- High computation burden due to high-dimensionality and hard to get a good estimate

SVM solves these two issues simultaneously

- “Kernel tricks” for efficient computation
- Minimize  $\|\mathbf{w}\|^2$  can lead to a “good” classifier



# Kernel Trick

Recall:

maximize

subject to

$$\sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j y_i y_j x_i x_j$$
$$C \geq \alpha_i \geq 0, \sum_{i=1}^N \alpha_i y_i = 0$$

Note that data only appears as dot products

Since data is only represented as **dot products**, we need **not do the mapping explicitly**.

Introduce a Kernel Function (\*)  $K$  such that:

$$K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$$

(\*)Kernel function – a function that can be applied **to pairs of input data to evaluate dot products in some corresponding feature space**

# Example Transformation

Consider the following transformation

$$\phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

$$\phi\left(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}\right) = (1, \sqrt{2}y_1, \sqrt{2}y_2, y_1^2, y_2^2, \sqrt{2}y_1y_2)$$

Define the kernel function  $K(\mathbf{x}, \mathbf{y})$  as

$$\begin{aligned} \langle \phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right), \phi\left(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}\right) \rangle &= (1 + x_1y_1 + x_2y_2)^2 \\ &= K(\mathbf{x}, \mathbf{y}) \end{aligned}$$

$$K(\mathbf{x}, \mathbf{y}) = (1 + x_1y_1 + x_2y_2)^2$$

The inner product  $\phi(.)\phi(.)$  can be computed by  $K$  without going through the map  $\phi(.)$  explicitly!!!

# Modification Due to Kernel Function

Change all inner products to kernel functions

For training,

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

Original

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

With kernel  
function

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

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

# Examples of Kernel Functions

Polynomial kernel with degree  $d$        $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + 1)^d$

Radial basis function kernel with width  $\sigma$

$$K(\mathbf{x}, \mathbf{y}) = \exp(-\|\mathbf{x} - \mathbf{y}\|^2 / (2\sigma^2))$$

– Closely related to radial basis function neural networks

Research on different kernel functions in different applications is very active

# Example

Suppose we have 5 1D data points

–  $x_1=1, x_2=2, x_3=4, x_4=5, x_5=6,$

with 1, 2, 6 as class 1 and 4, 5 as class 2

$\Rightarrow y_1=1, y_2=1, y_3=-1, y_4=-1, y_5=1$

Use the polynomial kernel of degree 2

–  $K(x,y) = (xy+1)^2$

– C is set to 100

First find  $\alpha_i$  ( $i=1, \dots, 5$ ) by

$$\max. \sum_{i=1}^5 \alpha_i - \frac{1}{2} \sum_{i=1}^5 \sum_{j=1}^5 \alpha_i \alpha_j y_i y_j (x_i x_j + 1)^2$$

$$\text{subject to } 100 \geq \alpha_i \geq 0, \sum_{i=1}^5 \alpha_i y_i = 0$$

# An Example

By using a QP solver, we get

$$\alpha_1=0, \alpha_2=2.5, \alpha_3=0, \alpha_4=7.333, \alpha_5=4.833$$

– Verify that the constraints are indeed satisfied

– The support vectors are  $\{x_2=2, x_4=5, x_5=6\}$

The discriminant function is

$$\begin{aligned} f(y) &= 2.5(1)(2y + 1)^2 + 7.333(-1)(5y + 1)^2 + 4.833(1)(6y + 1)^2 + b \\ &= 0.6667x^2 - 5.333x + b \end{aligned}$$

$b$  is recovered by solving  $f(2)=1$  or by  $f(5)=-1$  or by  $f(6)=1$ , as  $x_2, x_4, x_5$  lie on and all give  $b=9$

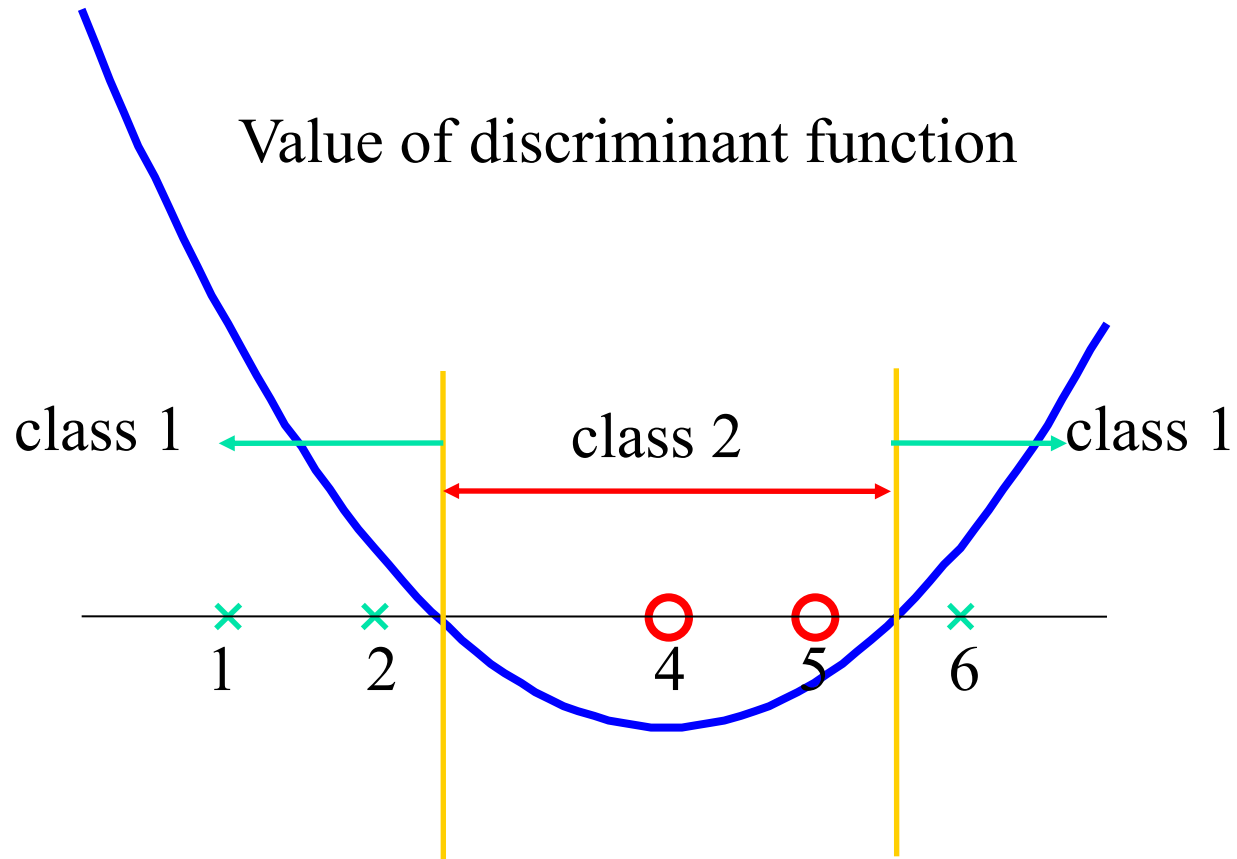
$$y_i(\mathbf{w}^T \phi(z) + b) = 1$$



$$f(y) = 0.6667x^2 - 5.333x + 9$$



# An Example



# Steps in SVM

1. Prepare data matrix  $\{(x_i, y_i)\}$
2. Select a Kernel function
3. Select the error parameter  $C$
4. “Train” the system (to find all  $\alpha_i$ )

New data can be classified using  $\alpha_i$  and Support Vectors

# SVM Strengths

- Training is relatively easy
  - We don't have to deal with local minimum like in ANN
  - SVM solution is always global and unique (check “Burges” paper for proof and justification).
- Unlike ANN, doesn't suffer from “curse of dimensionality”.
  - How? Why? We have infinite dimensions?!
  - Maximum Margin Constraint: DOT-PRODUCTS!
- Less prone to overfitting
- Simple, easy to understand geometric interpretation.
  - No large networks to mess around with.

# SVM Weakness

- Training (and Testing) is quite slow compared to ANN
  - Because of Constrained Quadratic Programming
- Essentially a binary classifier
  - However, there are tricks to evade this.
- Very sensitive to noise
  - A few off data points can completely throw off the algorithm
- Biggest Drawback: The choice of Kernel function.
  - There is no “set-in-stone” theory for choosing a kernel function for any given problem (still in research...)
  - Once a kernel function is chosen, there is only ONE modifiable parameter, the error penalty  $C$ .

# Applications of SVMs

- Lots of very successful applications!
  - Bioinformatics
  - Machine Vision
  - Text Categorization
  - Ranking (e.g., Google searches)
  - Time series analysis
  - Handwritten Character Recognition



3-nearest-neighbor = 2.4% error

400–300–10 unit MLP = 1.6% error

LeNet: 768–192–30–10 unit MLP = 0.9% error

Current best (kernel machines, vision algorithms)  $\approx$  0.6% error