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)

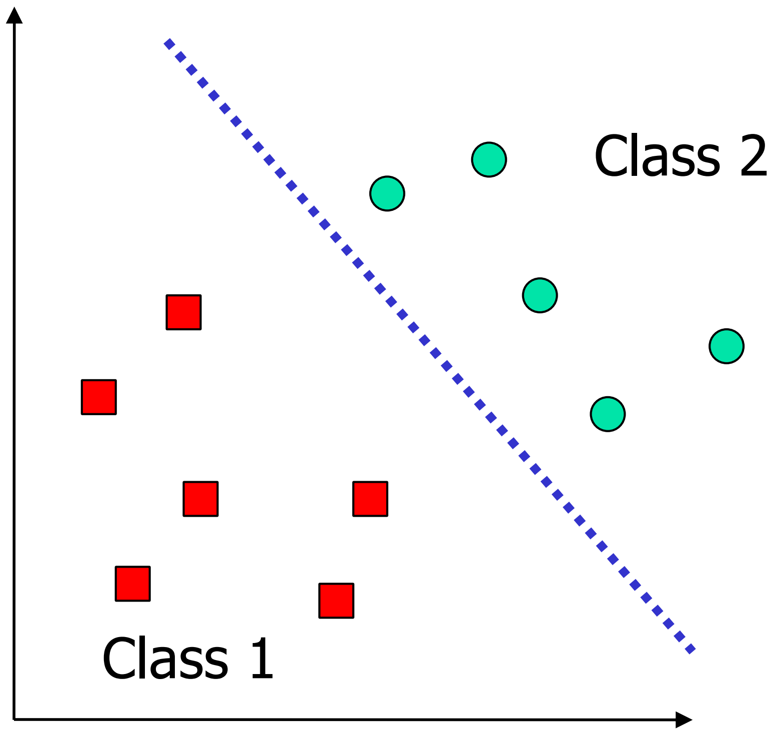
→ SVM got into mainstream because of their exceptional performance in Handwritten Digit Recognition

- 1.1% error rate which was comparable to a very carefully constructed (and complex) ANN

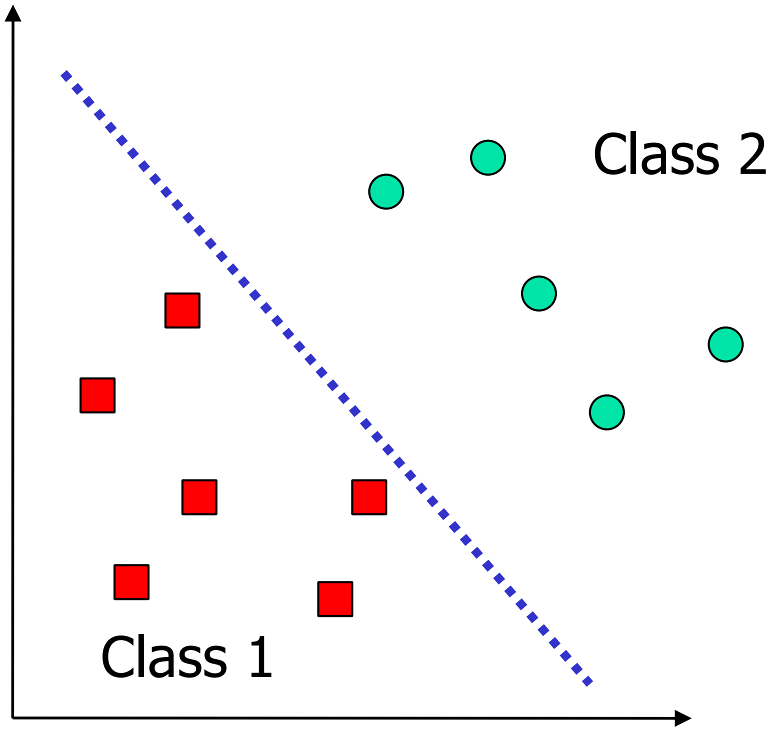
Two Class Problem: Linear Separable Case

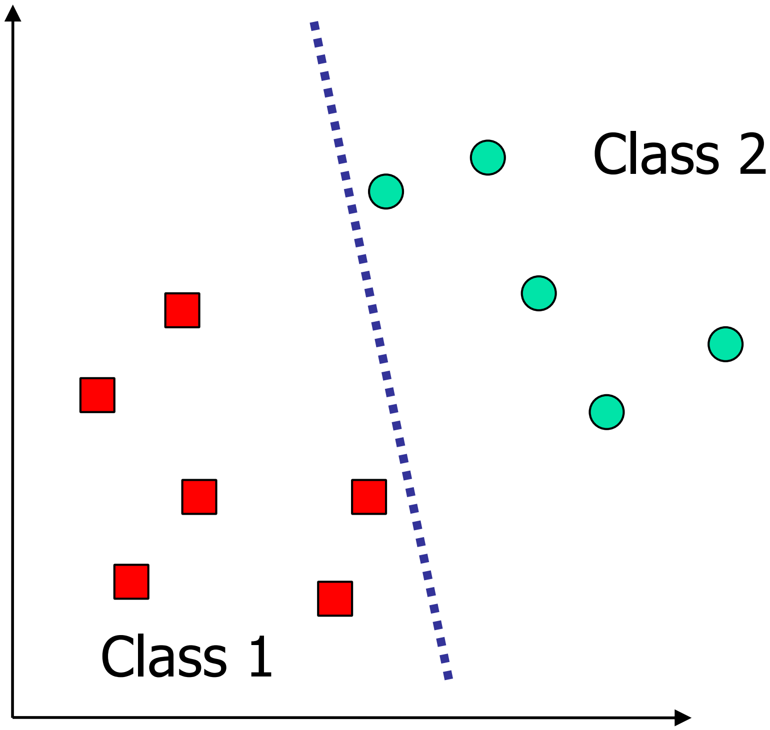
Many decision boundaries can separate these two classes

Which one should we choose?



Example of Bad Decision Boundaries

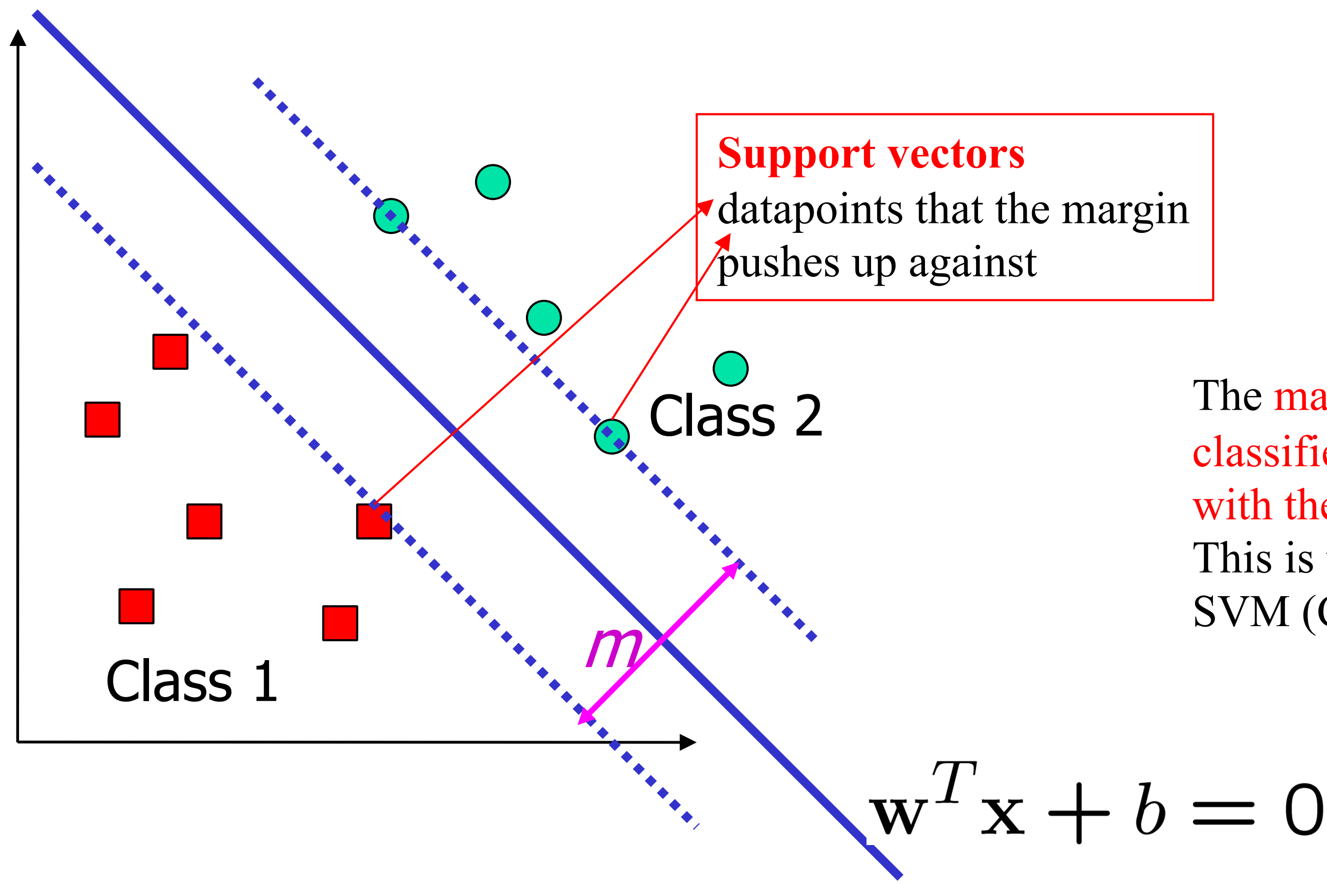




Good Decision Boundary: Margin Should Be Large

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

– We should maximize the margin, m

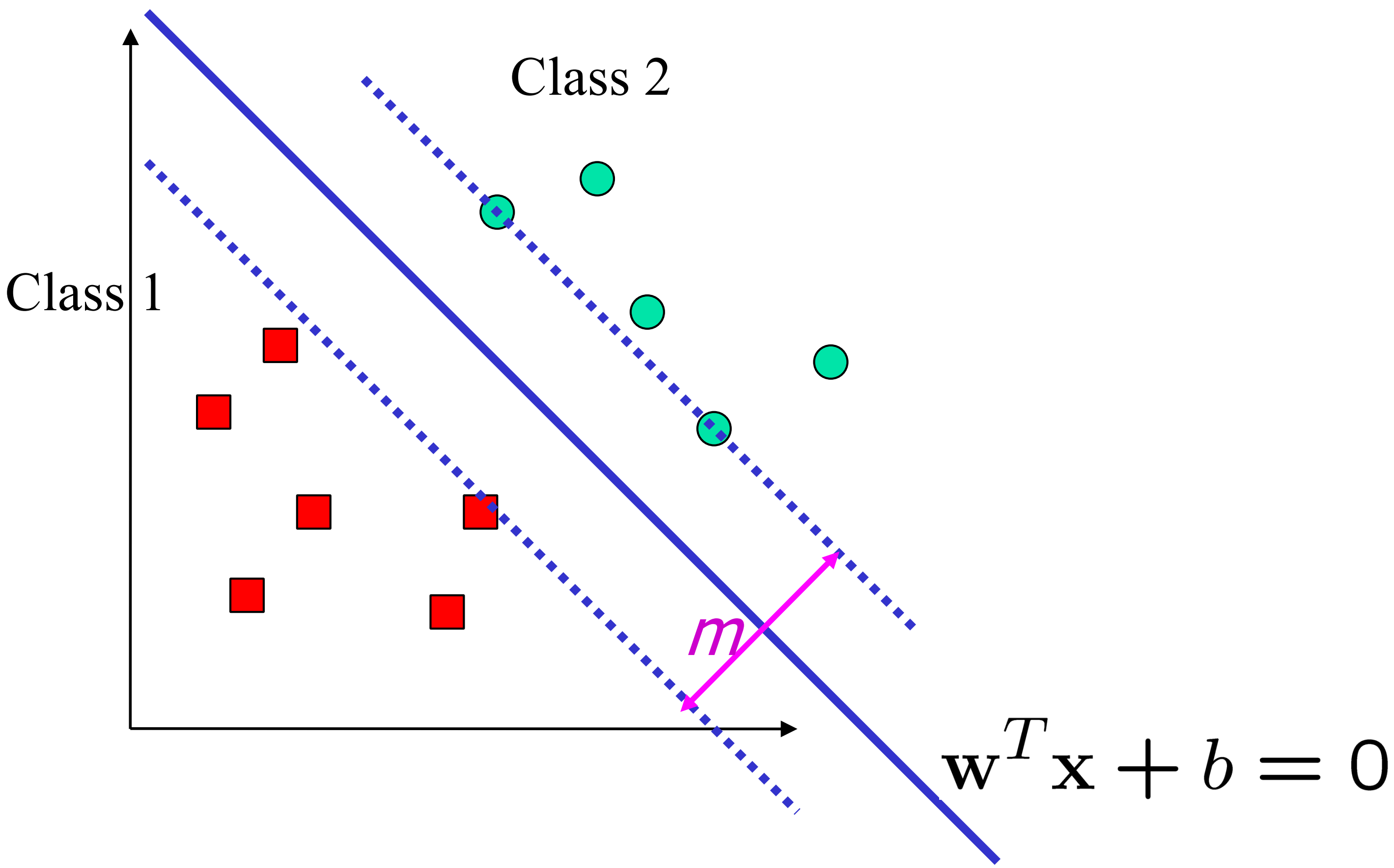


The maximum margin linear classifier is the linear classifier with the maximum margin. This is the simplest kind of SVM (Called a Linear SVM)

What is the value of length of m?

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

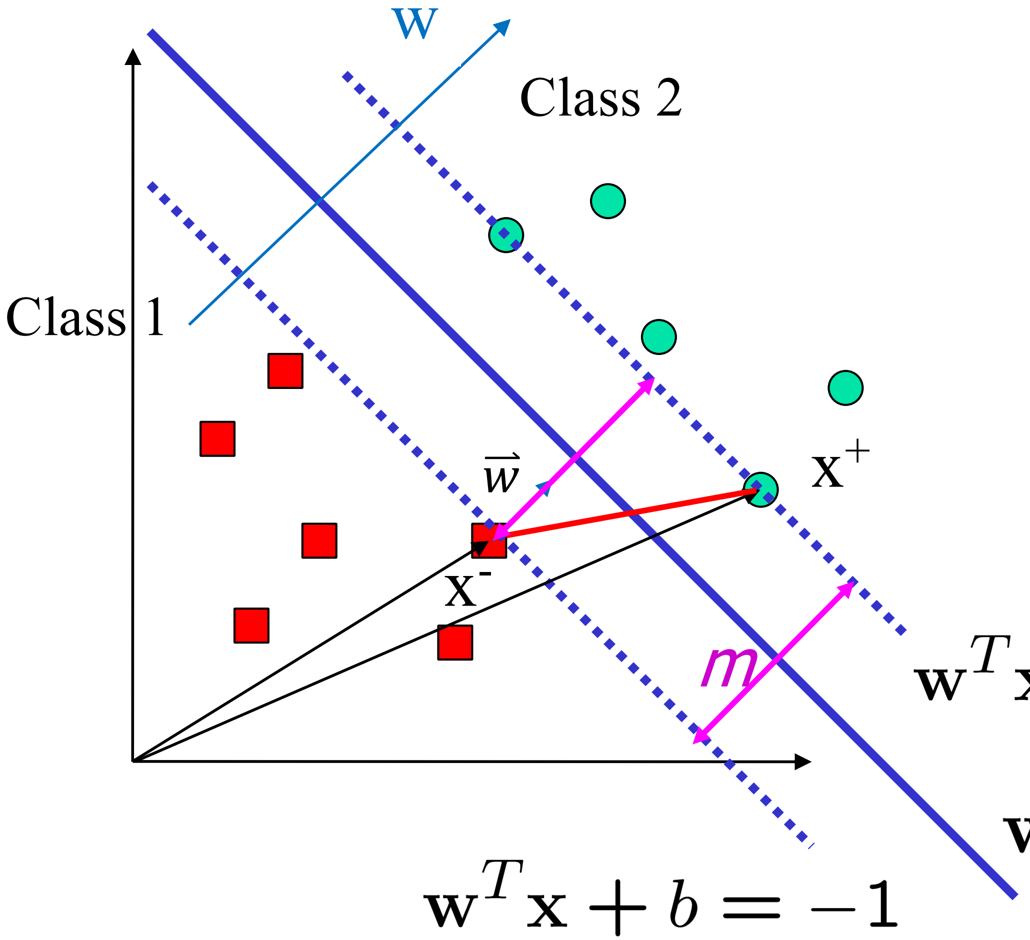
– We should maximize the margin, *m*



Compute the length of m

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

– We should maximize the margin, $\|m\|$



vector w perpendicular to w^T
vector m is the vector $u=x^+x^-$
projected on vector w :

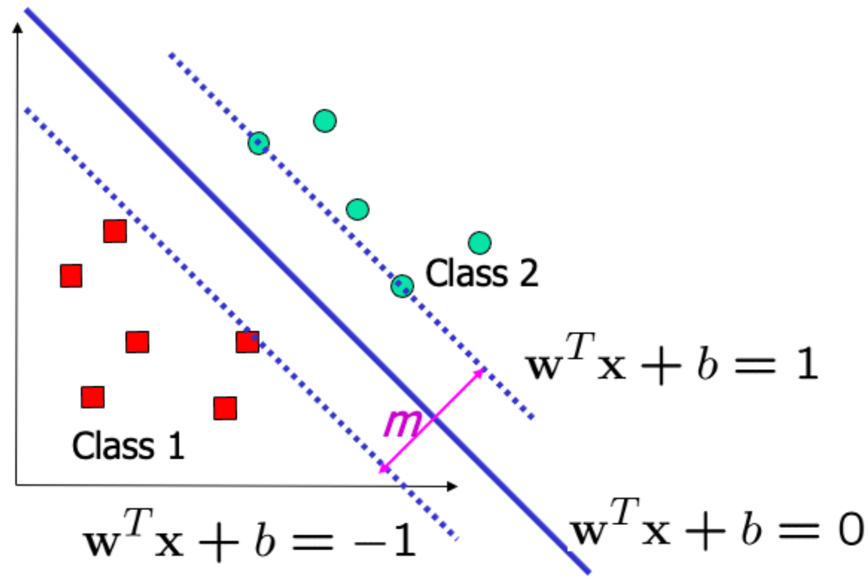
$$
\begin{aligned}
m &= u \cdot \widehat{w} \\
&= (x^+x^-) \cdot \frac{\vec{w}}{\|w\|} \\
&=(x^+ \vec{w} \; - x^- \vec{w}) \cdot \frac{1}{\|w\|} \\
&=((1\text{-}b)\text{-}(\text{-}1\text{-}b)) \cdot \frac{1}{\|w\|} \\
&= \frac{2}{\|w\|}
\end{aligned}
$$

The Optimization Problem

Let $\{x_1, ..., x_n\}$ 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:

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

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

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

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

Lagrangian of Original Problem

The Lagrangian is Minimize $\frac{1}{2}||\mathbf{w}||^2$

subject to $1-y_i(\mathbf{w}^T\mathbf{x}_i+b) \leq 0 \quad$ for $i = 1, \ldots, n$

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

Lagrangian multipliers

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

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

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

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

$\alpha_i \geq 0$

Change into the Dual Problem

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

Just obtained: $$\boxed{\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i} \qquad \boxed{\sum_{i=1}^{n} \alpha_i y_i = 0}$$

Transform the Lagrangian into into dual problem:

The Dual Optimization Problem

The problem now is to:

Dot product of X

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

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

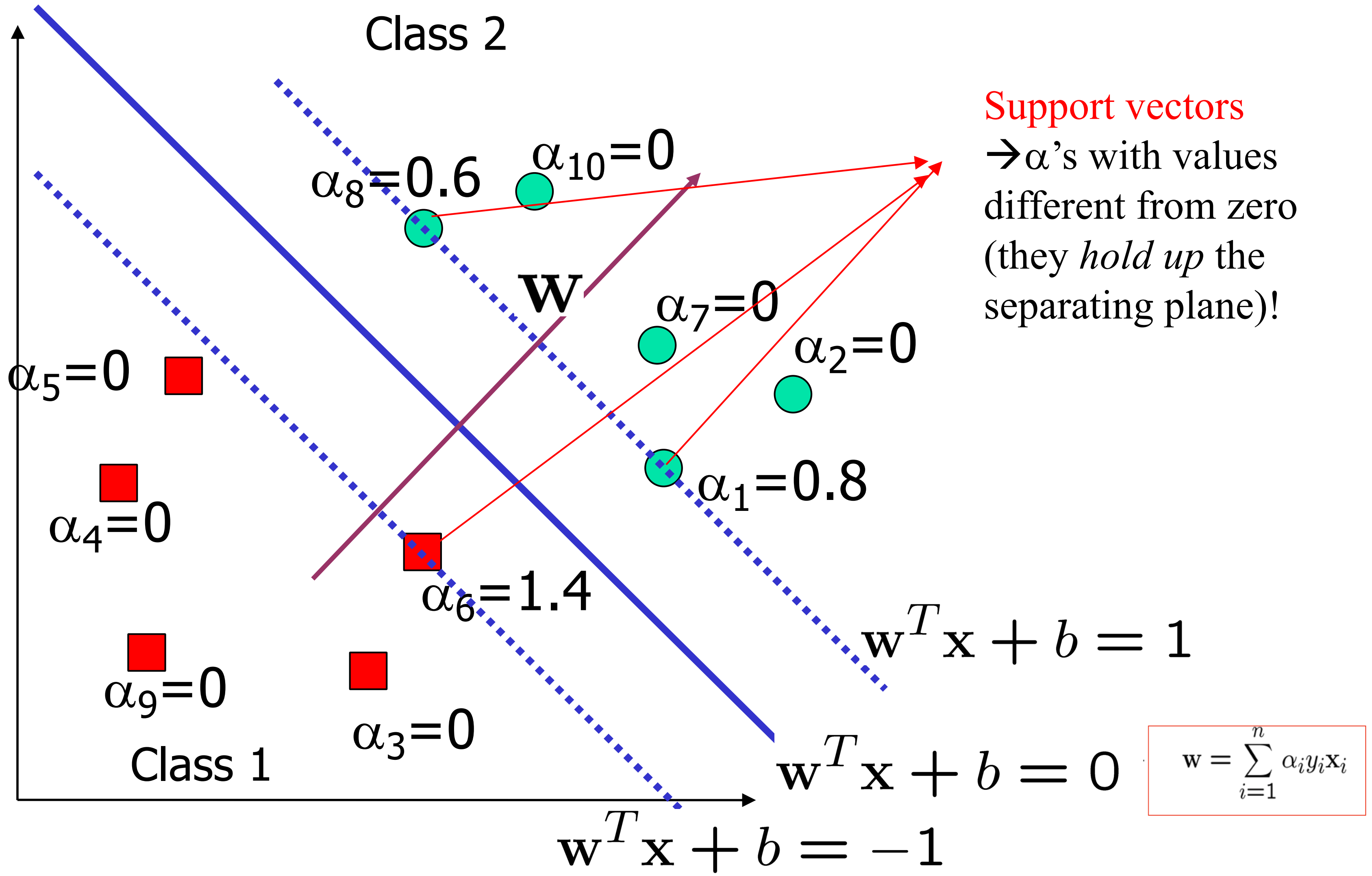
α's → New variables
(Lagrangian multipliers)

This is a convex quadratic programming (QP) problem

– Global maximum of α_i can always be found

→well established tools for solving this optimization problem

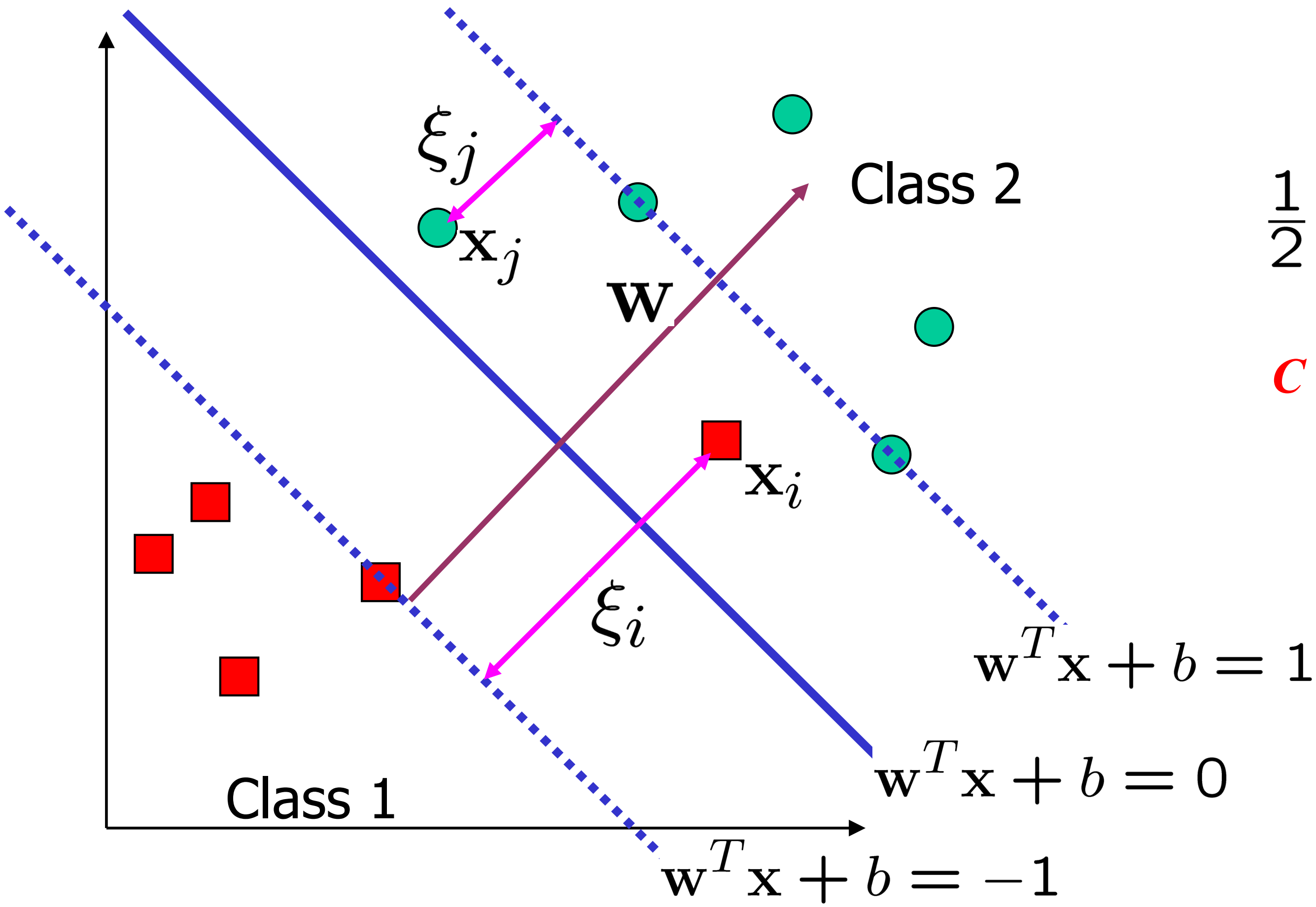
A Geometrical Interpretation



Non-linearly Separable Problems

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

ξ_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

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

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

w is also recovered as

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

The only difference with the linear separable case is that there is an upper bound *C* on α_i

Once again, a QP solver can be used to find α_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.

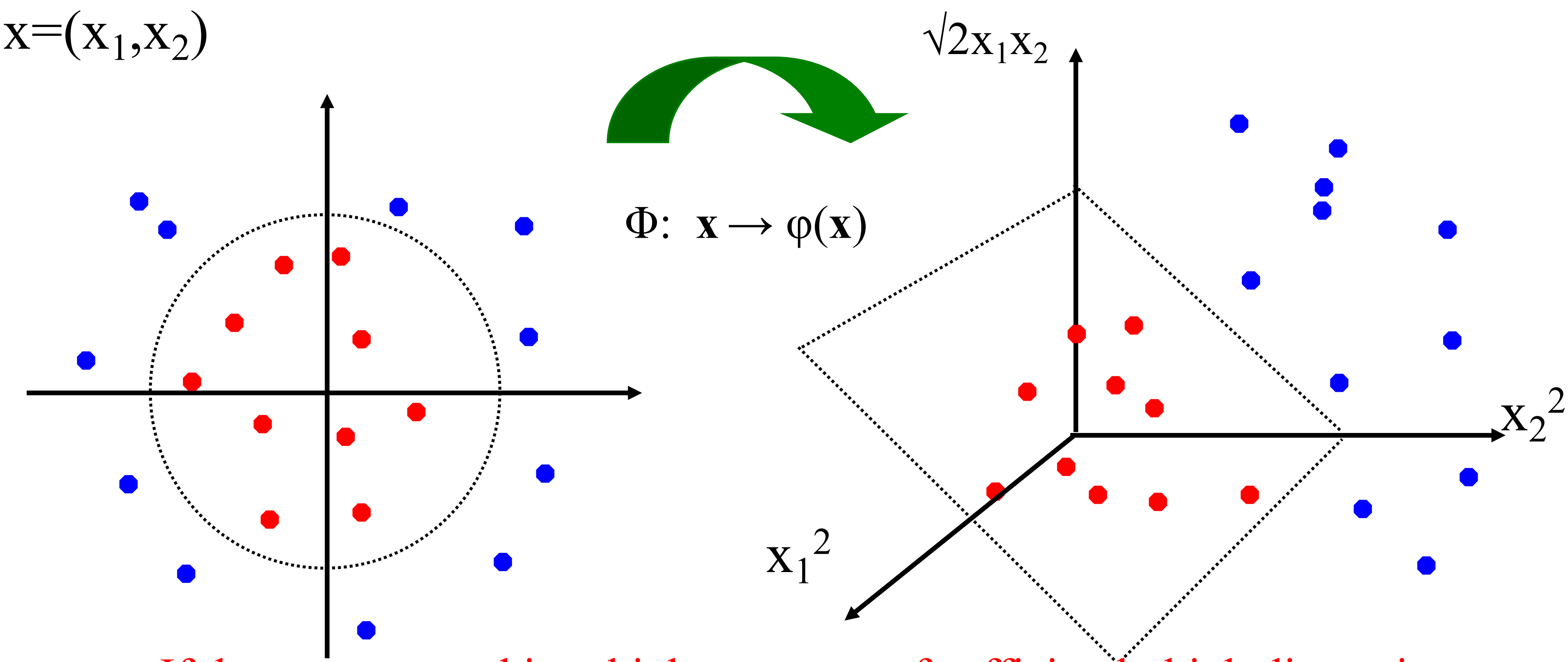
Similar to Hidden Layers in ANN

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

Why?

- Linear operation in **H** is equivalent to non-linear operation in input space.

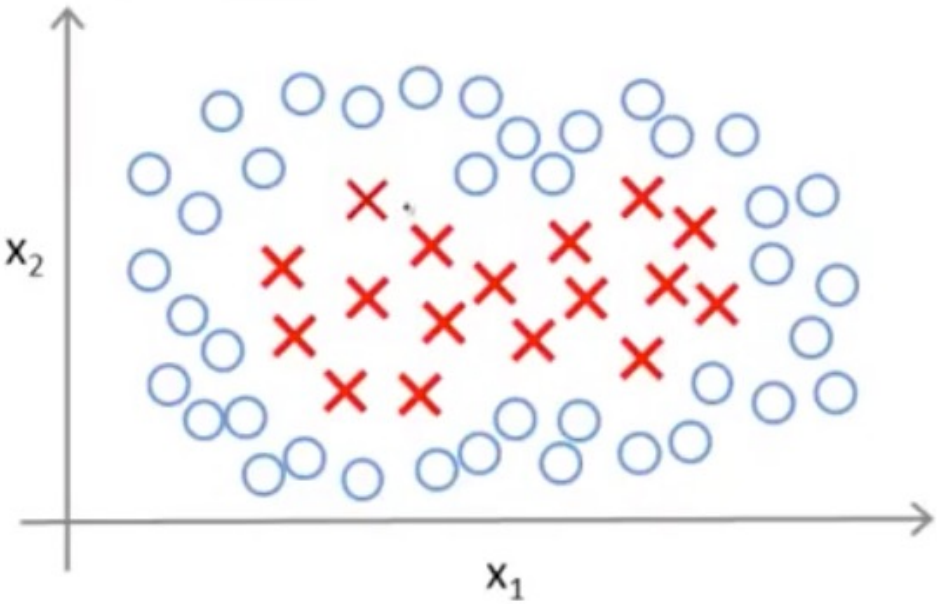
Non-linear SVMs: Feature Space



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

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

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

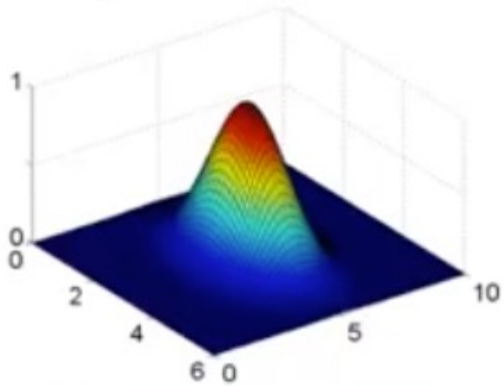
If x if far from $l^{(1)}$:

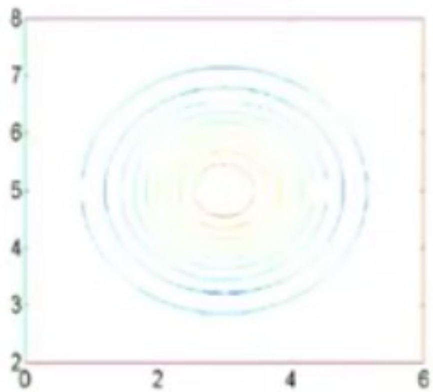
Gaussian Kernel

Example:

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

$$\sigma^2 = 1$$



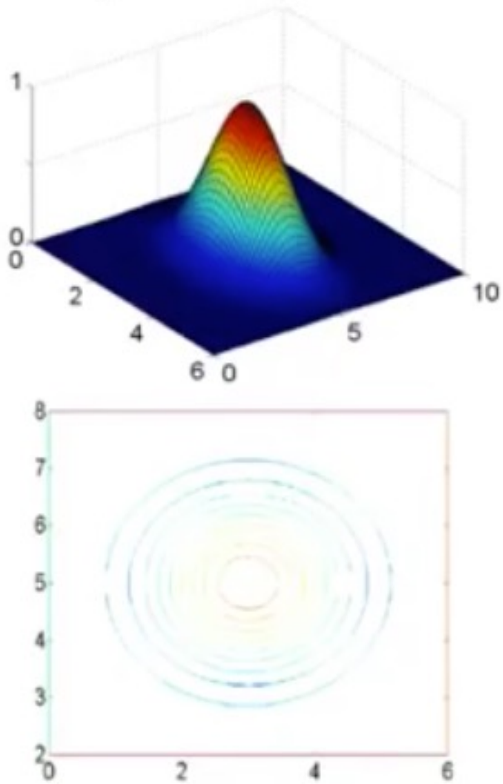


Gaussian Kernel

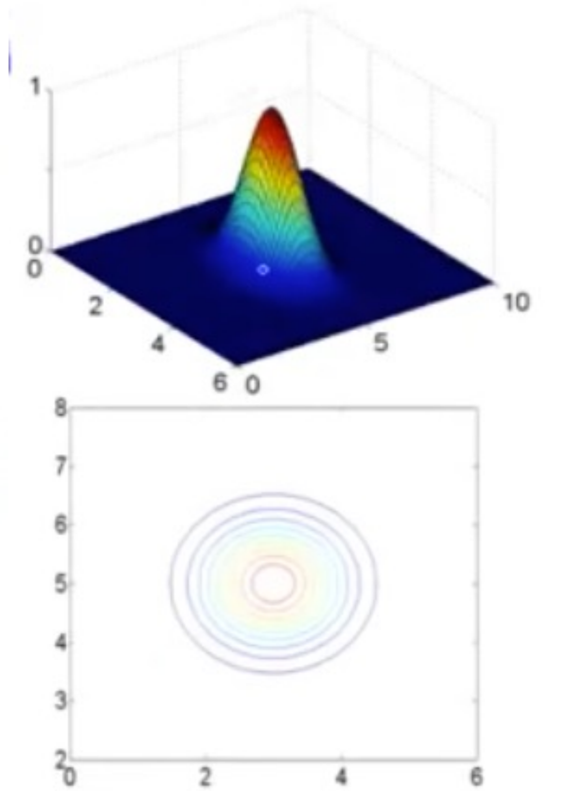
Example:

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

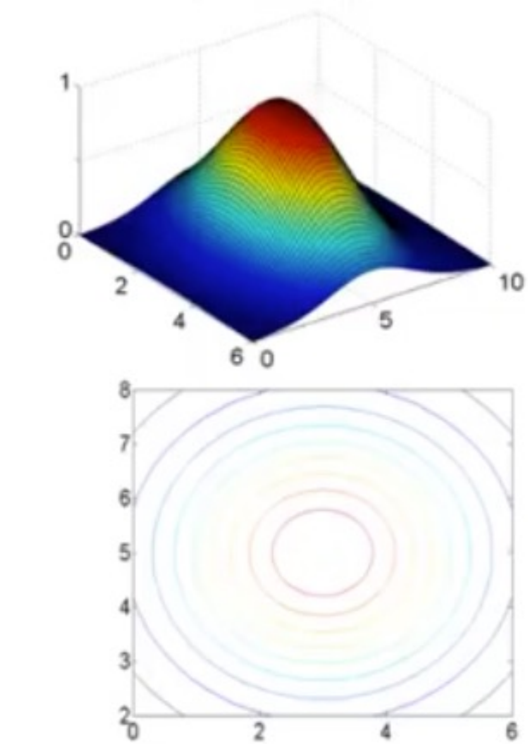
$\sigma^2 = 1$



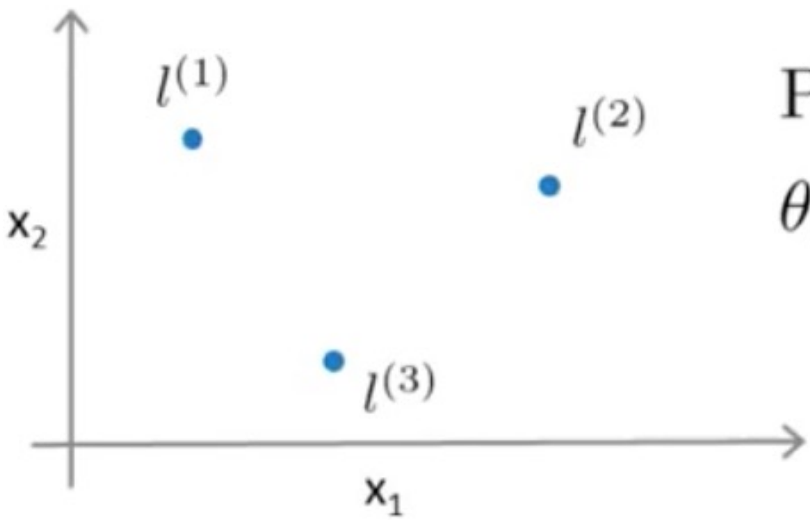
$\sigma^2 = 0.5$



$\sigma^2 = 3$



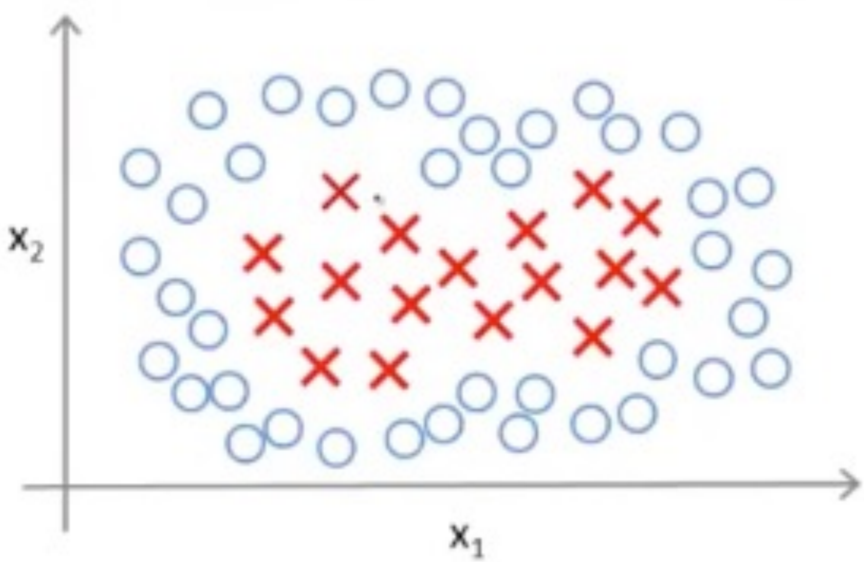
Classification with the Kernel



Predict "1" when

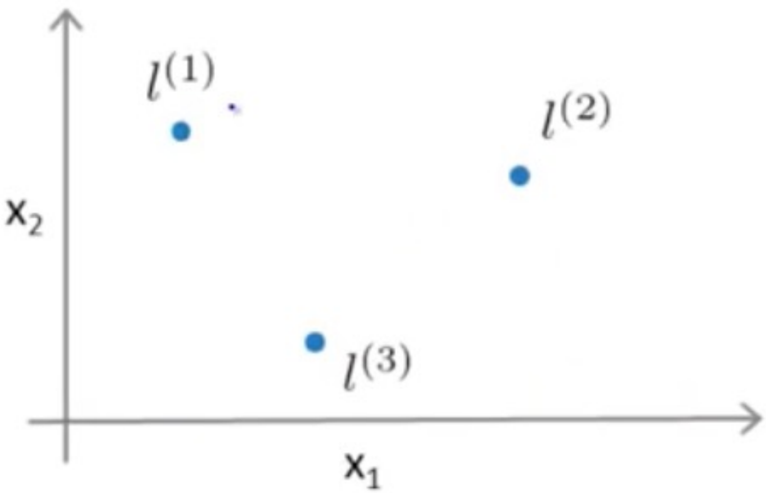
$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$$

Higher Dimension Illustration



How to choose landmarks

Choosing the landmarks

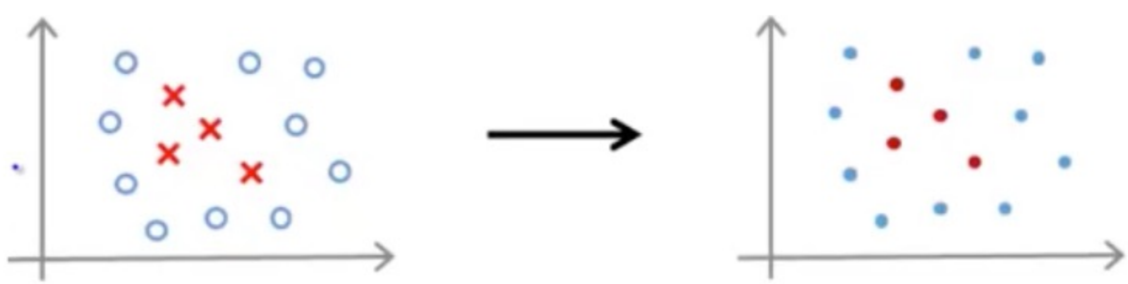


Given x:

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

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

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



Transformation to Feature Space

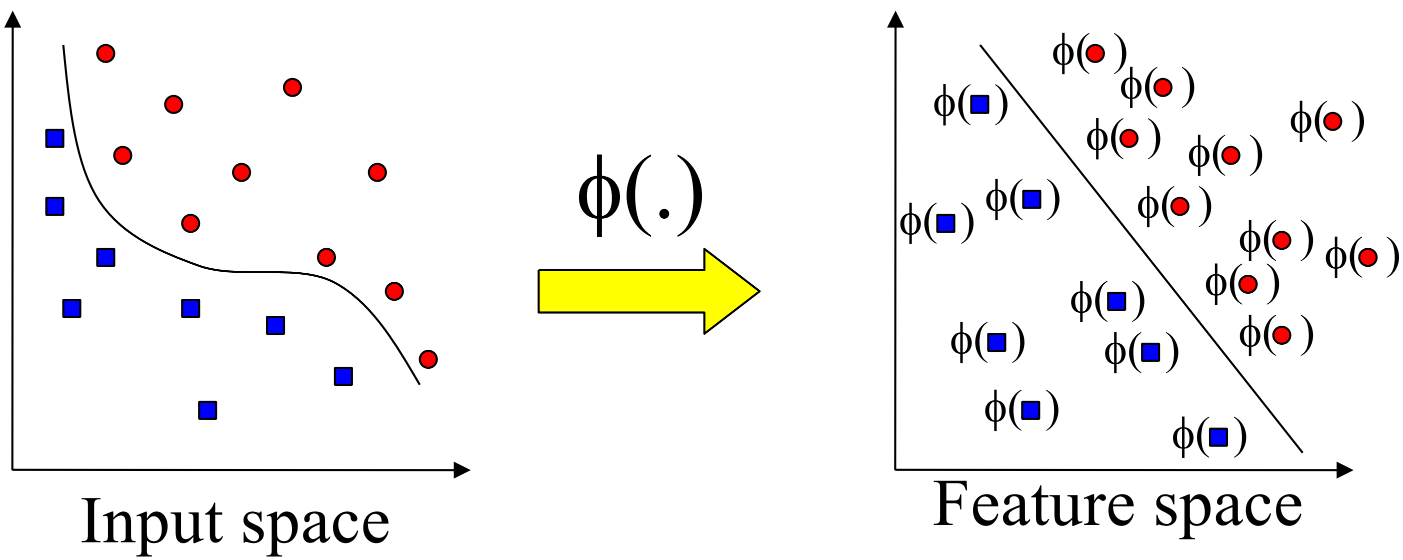
Possible problem of the transformation

– 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

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

Note that data only appears as dot products

$$C \geq \alpha_i \geq 0, \sum_{i=1}^{N} \alpha_i y_i = 0$$

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

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

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

Define the kernel function *K* (**x**,**y**) as

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

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

The inner product φ(.)φ(.) can be computed by *K* without going through the map φ(.) explicitly!!!

Modification Due to Kernel Function

Change all inner products to kernel functions

For training,

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

Original

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

With kernel function

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

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

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 σ

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

– 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 α_i ($i=1, \ldots, 5$) by

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

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

An Example

By using a QP solver, we get

α_1=0, α_2=2.5, α_3=0, α_4=7.333, α_5=4.833

– Verify that the constraints are indeed satisfied

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

$$\mathbf{w}^T\mathbf{x} + b = 0$$

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

The discriminant function is

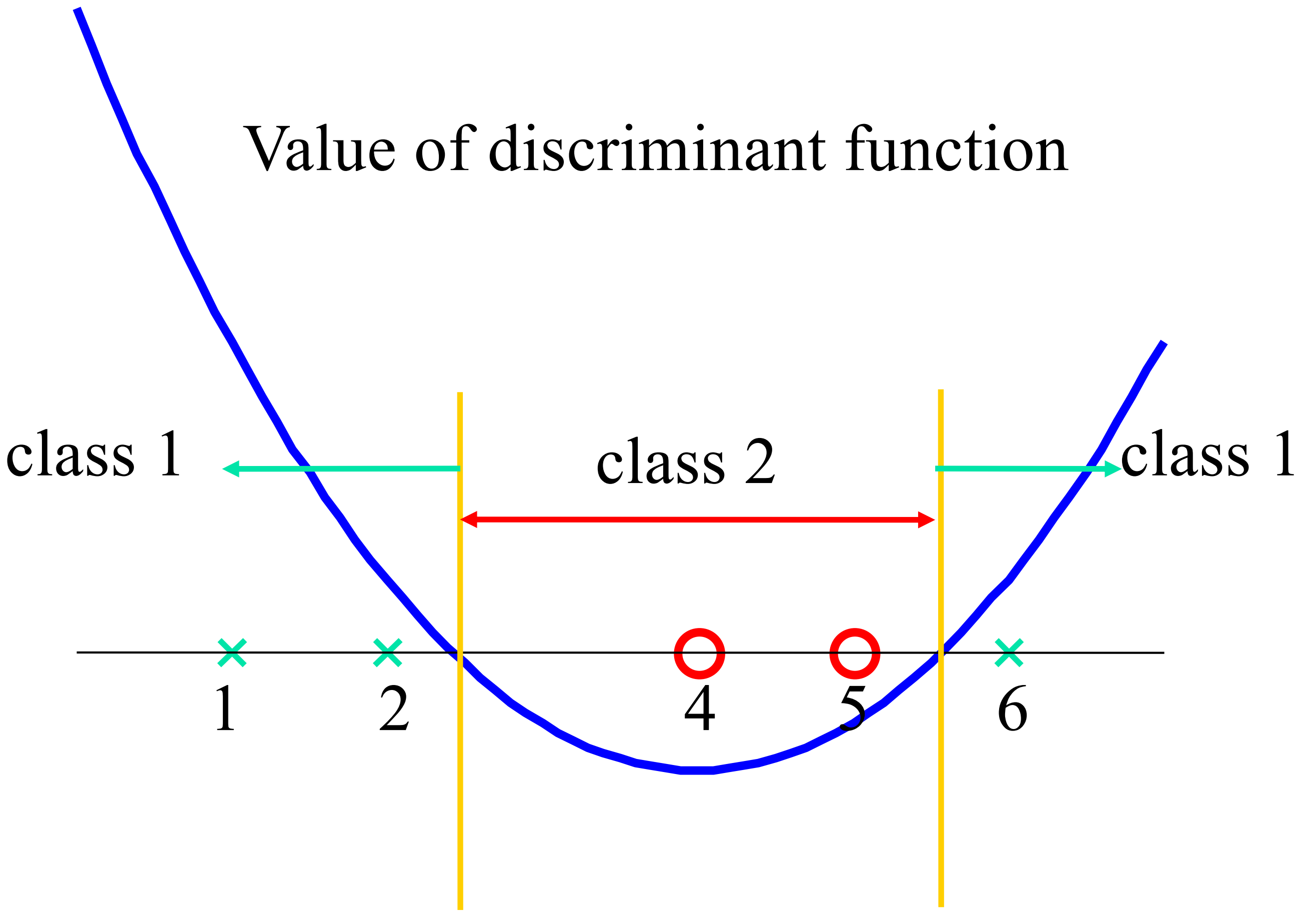
$$f(x) = 2.5(1)(2x+1)^2 + 7.333(-1)(5x+1)^2 + 4.833(1)(6x+1)^2 + \mathrm{b}$$
$$= 0.6667x^2 - 5.333x + b$$

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

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

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

An Example



Value of discriminant function
class 1
class 2
class 1
1
2
4
5
6

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 α_i)

New data can be classified using α_i and Support Vectors

SVM Strengths

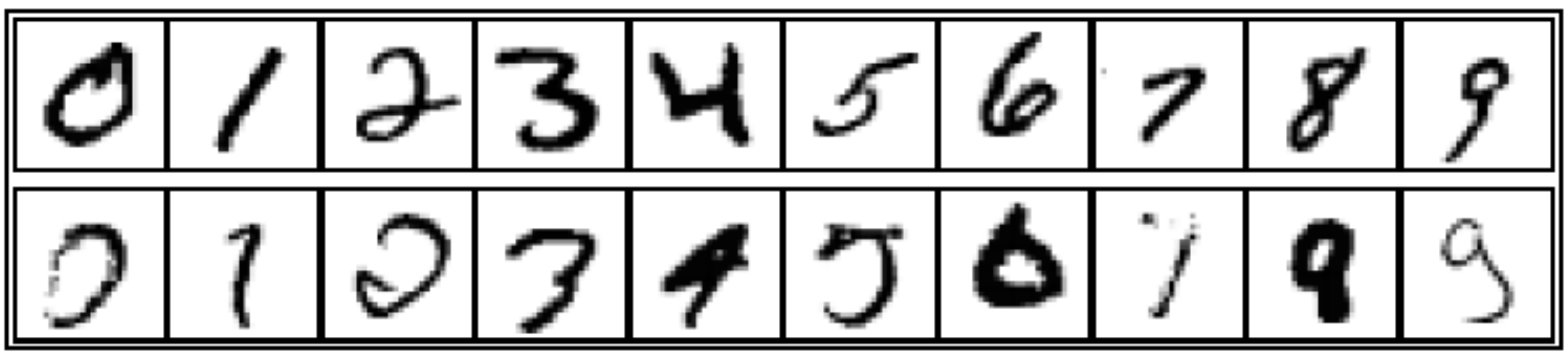
- Training is relatively easy
  - We don't have to deal with local minimum like in ANN
  - SVM solution is always global and unique (check "Burges" paper for proof and justification).
- Unlike ANN, doesn't suffer from "curse of dimensionality".
  - How? Why? We have infinite dimensions?!
  - Maximum 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