

# Kernels Methods in Machine Learning

- Perceptron. Geometric Margins.
- Support Vector Machines (SVMs).

Maria-Florina Balcan

03/23/2015

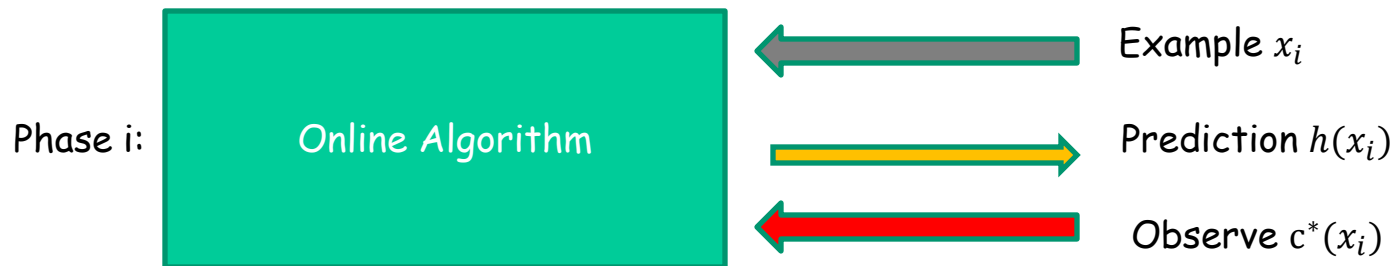
# Quick Recap about Perceptron and Margins

# The Online Learning Model

- Example arrive **sequentially**.
- We need to make a prediction.

Afterwards observe the outcome.

For  $i=1, 2, \dots$ :



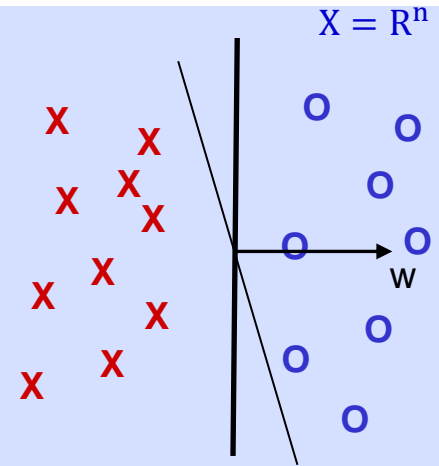
## Mistake bound model

- Analysis wise, make **no distributional** assumptions.
- **Goal:** **Minimize** the number of **mistakes**.

# Perceptron Algorithm in Online Model

WLOG homogeneous linear separators [ $w_0 = 0$ ].

- Set  $t=1$ , start with the all zero vector  $w_1$ .
- Given example  $x$ , predict + iff  $w_t \cdot x \geq 0$
- On a mistake, update as follows:
  - Mistake on positive,  $w_{t+1} \leftarrow w_t + x$
  - Mistake on negative,  $w_{t+1} \leftarrow w_t - x$



**Note 1:**  $w_t$  is weighted sum of incorrectly classified examples

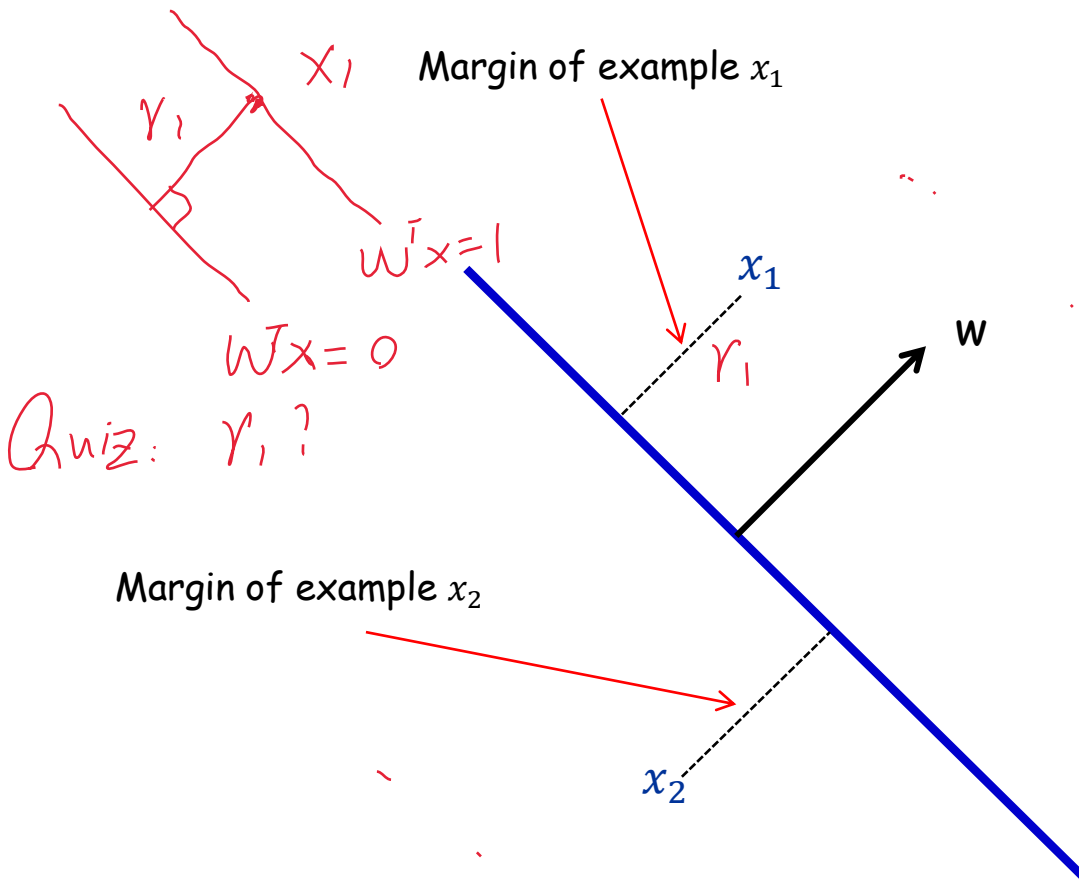
$$w_t = a_{i_1}x_{i_1} + \dots + a_{i_k}x_{i_k} \quad \text{So, } w_t \cdot x = a_{i_1}x_{i_1} \cdot x + \dots + a_{i_k}x_{i_k} \cdot x$$

**Note 2:** Number of mistakes ever made depends only on the geometric margin of examples seen.  $M \leq \left(\frac{R}{\gamma}\right)^2$   
 $\Sigma \leq \frac{1}{\gamma} (\ln(H) + \ln \frac{1}{\delta})$

- No matter how long the sequence is or how high dimension  $n$  is!

# Geometric Margin

**Definition:** The **margin** of example  $x$  w.r.t. a linear sep.  $w$  is the distance from  $x$  to the plane  $w \cdot x = 0$ .



If  $\|w\| = 1$ , margin of  $x$  w.r.t.  $w$  is  $|x \cdot w|$ .

$$\gamma_i = \frac{y_i w^T x_i}{\|w\|} \geq 0$$

# Geometric Margin

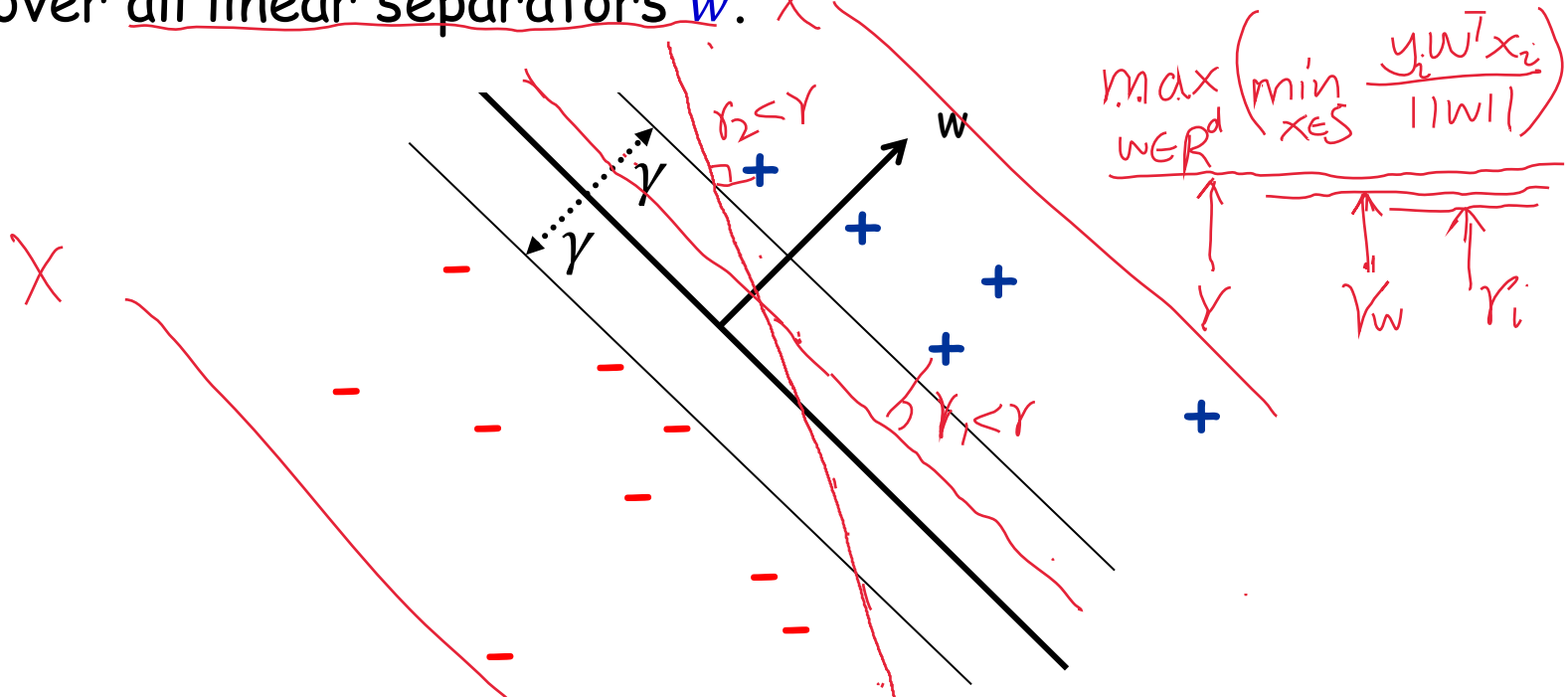
$$x \in \mathbb{R}^d$$

**Definition:** The margin of example  $x$  w.r.t. a linear sep.  $w$  is the distance from  $x$  to the plane  $w \cdot x = 0$ .

$$w \cdot x = w^T x = \langle w, x \rangle$$

**Definition:** The margin  $\gamma_w$  of a set of examples  $S$  wrt a linear separator  $w$  is the smallest margin over points  $x \in S$ .

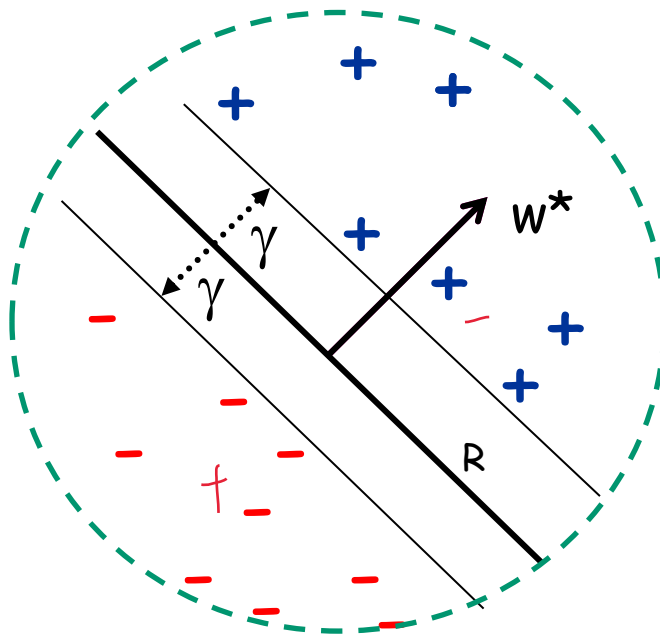
**Definition:** The margin  $\gamma$  of a set of examples  $S$  is the maximum  $\gamma_w$  over all linear separators  $w$ .



# Perceptron: Mistake Bound

**Theorem:** If data linearly separable by margin  $\gamma$  and points inside a ball of radius  $R$ , then Perceptron makes  $\leq (R/\gamma)^2$  mistakes.

- No matter how long the sequence is how high dimension  $n$  is!



Margin: the amount of wiggle-room available for a solution.

$$\left(\frac{R}{\gamma}\right)^2$$

# examples

# features

# VCdim(H)

(Normalized margin: multiplying all points by 100, or dividing all points by 100, doesn't change the number of mistakes; algo is invariant to scaling.)

# Perceptron Extensions

$$\left(\frac{R}{\gamma}\right)^2 + 1$$

- Can use it to find a consistent separator with a given set  $S$  linearly separable by margin  $\gamma$  (by cycling through the data).
- Can convert the mistake bound guarantee into a distributional guarantee too (for the case where the  $x_i$ s come from a fixed distribution).
- Can be adapted to the case where there is no perfect separator as long as the so called hinge loss (i.e., the total distance needed to move the points to classify them correctly large margin) is small.
- Can be kernelized to handle non-linear decision boundaries!



**Theorem:** If data linearly separable by margin  $\gamma$  and points inside a ball of radius  $R$ , then Perceptron makes  $\leq (R/\gamma)^2$  mistakes.

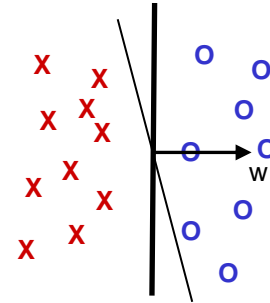
Implies that large margin classifiers have smaller complexity!

# Complexity of Large Margin Linear Sep.

- Know that in  $\mathbb{R}^n$  we can shatter  $n+1$  points with linear separators, but not  $n+2$  points (VC-dim of linear sep is  $n+1$ ).



What if we require that the points be linearly separated by margin  $\gamma$ ?



Can have at most  $\left(\frac{R}{\gamma}\right)^2$  points inside ball of radius  $R$  that can be shattered at margin  $\gamma$  (meaning that every labeling is achievable by a separator of margin  $\gamma$ ).

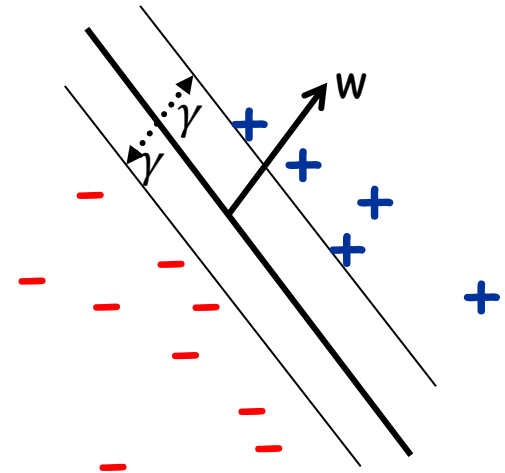
- So, large margin classifiers have smaller complexity!
  - Nice implications for usual distributional learning setting.  $\times \sim D$
  - Less classifiers to worry about that will look good over the sample, but bad over all....
- Less prone to overfitting!!!!

# Margin Important Theme in ML.

Both sample complexity and algorithmic implications.

## Sample/Mistake Bound complexity:

- If **large** margin, # mistakes Perceptron makes is small (**independent** on the dim of the space)!
- If **large** margin  $\gamma$  and if alg. produces a large margin classifier, then amount of data needed depends only on  $R/\gamma$  [Bartlett & Shawe-Taylor '99].
  - Suggests searching for a large margin classifier...



## Algorithmic Implications:

- Perceptron, Kernels, SVMs...

So far, talked about margins in the context of (nearly) linearly separable datasets

# What if Not Linearly Separable

**Problem:** data not linearly separable in the most natural feature representation.

Example:



vs



No good linear separator in pixel representation.

## Solutions:

- "Learn a more complex class of functions"
  - (e.g., decision trees, neural networks, boosting).
- "Use a Kernel" (a neat solution that attracted a lot of attention)
- "Use a Deep Network"
- "Combine Kernels and Deep Networks"

$$K(x, y) = \langle x, y \rangle$$

# Overview of Kernel Methods

$$\mathcal{X} = \mathbb{R}^n$$

$$\mathbb{R}$$

$$x_i, x_j$$

$$\langle x_i, x_j \rangle$$

## What is a Kernel?

$$K: \mathcal{X} \times \mathcal{X} \Rightarrow \mathbb{R}$$

A kernel  $K$  is a **legal def of dot-product**: i.e. there exists an implicit mapping  $\Phi$  s.t.  $K(\text{img1}, \text{img2}) = \Phi(\text{img1}) \cdot \Phi(\text{img2})$



$$\phi: \mathcal{X} \rightarrow \mathbb{R}^N$$

$$\text{E.g., } K(x, y) = (x \cdot y + 1)^d = \phi(x) \cdot \phi(y) = \langle \phi(x), \phi(y) \rangle \quad (\mathbb{R}^n) \quad (N \gg n)$$

$\phi: (n\text{-dimensional space}) \rightarrow n^d\text{-dimensional space}$

## Why Kernels matter?

$$\mathbb{R}^n$$

$$\mathbb{R}^{nd}$$

$$w = \sum_i a_i x_i$$

$$w \cdot x = \sum_i a_i \langle x_i, x \rangle$$

- Many algorithms interact with data only via dot-products.
- So, if replace  $x \cdot z$  with  $K(x, z)$  they act implicitly as if data was in the higher-dimensional  $\Phi$ -space.
- If data is linearly separable by large margin in the  $\Phi$ -space, then good sample complexity.

$$\mathbb{R}^N$$

[Or other regularity properties for controlling the capacity.]

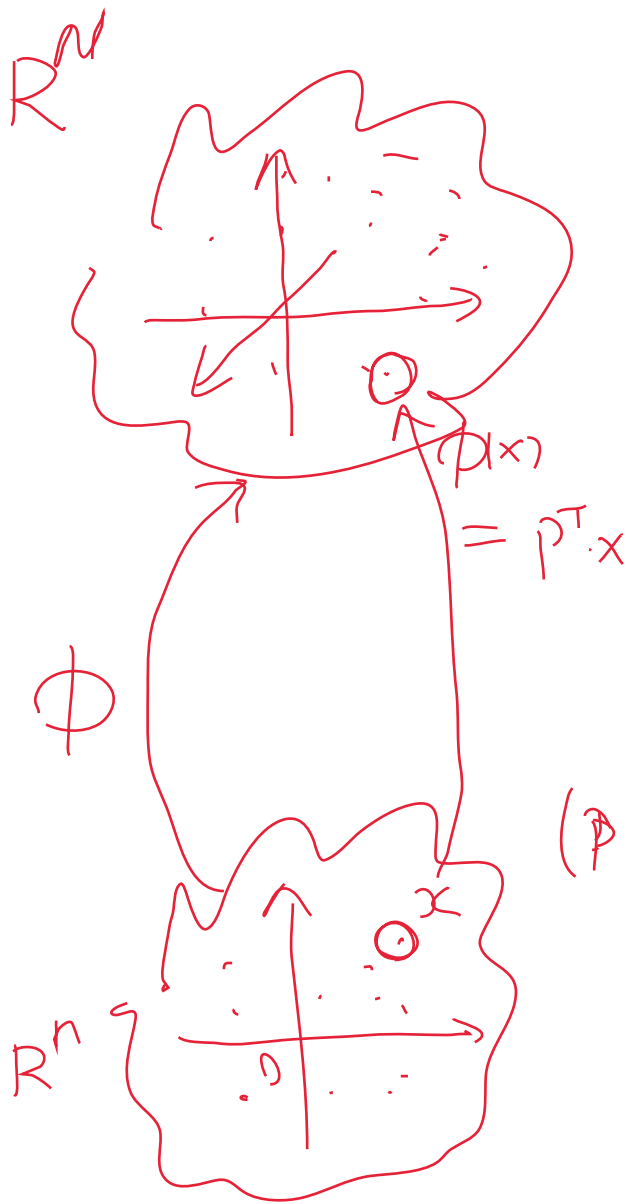
# Kernels

## Definition

$K(\cdot, \cdot)$  is a kernel if it can be viewed as a legal definition of inner product:

- $\exists \phi: X \rightarrow \mathbb{R}^N$  s.t.  $\underbrace{K(x, z)}_{\text{sample similarity}} = \phi(x) \cdot \phi(z)$ 
  - Range of  $\phi$  is called the  $\Phi$ -space.
  - $N$  can be very large. *infinite*
- But think of  $\phi$  as **implicit**, not explicit!!!!

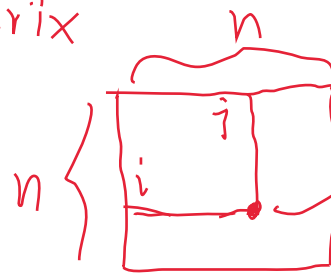
$$\begin{array}{ccc}
 o(n) & & \\
 \langle x, z \rangle & \xrightarrow{K} & K(x, z) \\
 \phi \downarrow & & = (x, z + 1)^d \\
 o(N) \langle \phi(x), \phi(z) \rangle & & \parallel o(n)
 \end{array}$$



Gram matrix

$$K \in \mathbb{R}^{n \times n}$$

$$O(n^2)$$

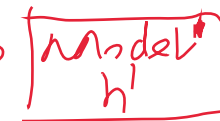


$$K(x_i, x_j)$$

$$= \langle \phi(x_i), \phi(x_j) \rangle$$

$$S' = \{(\phi(x_i), y_i)\}_{i=1}^m$$

A



$O(N)$

$$h': \mathbb{R}^N \rightarrow \mathbb{R}$$

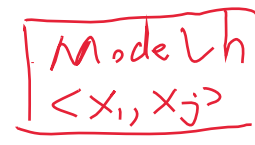
$$\phi(x) \rightarrow y$$

$K$

$O(n^2)$

$$S = \{(x_i, y_i)\}_{i=1}^m$$

A



$O(n)$

$$h: \mathbb{R}^n \rightarrow \mathbb{R}$$

$$x \rightarrow y$$

$$K(x_i, x_j)$$

$$= \langle \phi(x_i), \phi(x_j) \rangle$$

$$O(n)$$

$$\hat{y} \leftarrow h(x)$$

$$O(n)$$

$$x \rightarrow \phi(x)$$

$$P^T x$$

$$y \leftarrow h'(\phi(x))$$

$$O(N)$$



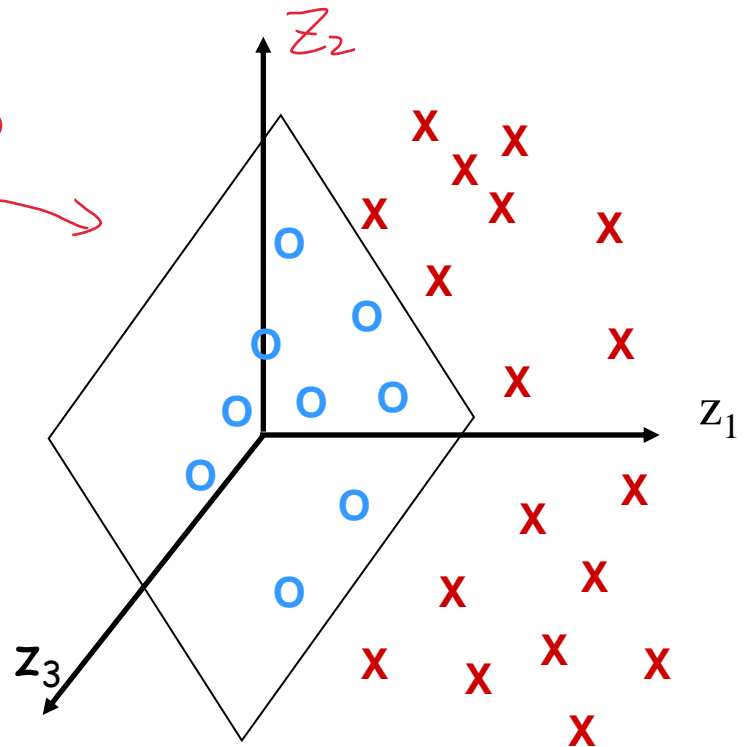
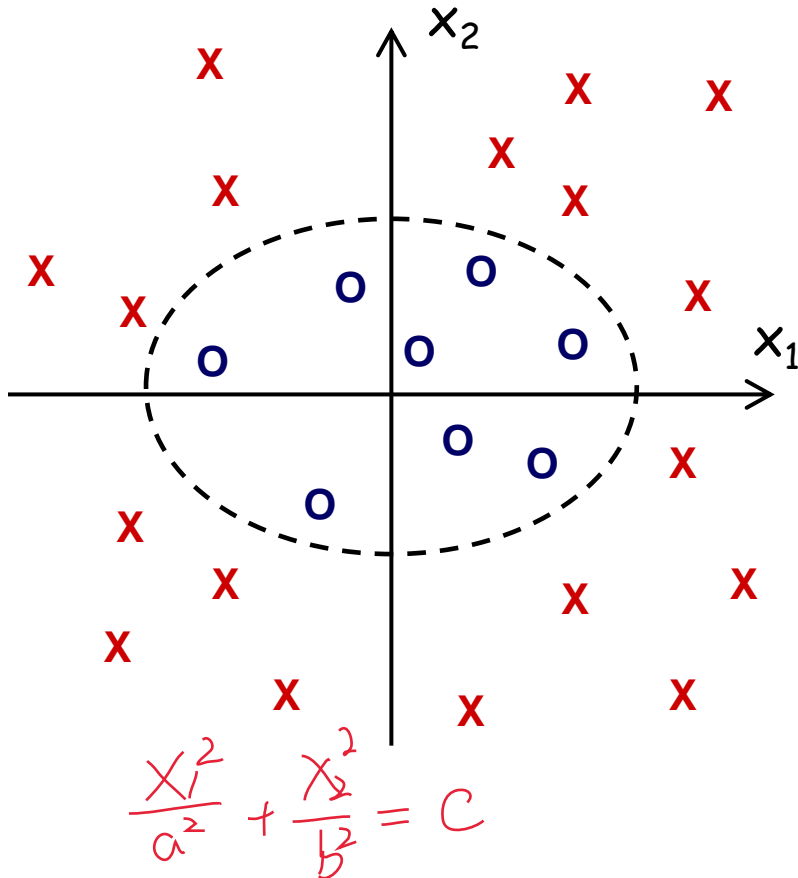
# Example

$$= \langle \phi(x), \phi(z) \rangle$$

For  $n=2$ ,  $d=2$ , the kernel  $K(x, z) = (x \cdot z)^d$  corresponds to

$$(x_1, x_2) \rightarrow \Phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

Original space  $x \in \mathbb{R}^2$   $z \in \mathbb{R}^3$   $\Phi$ -space



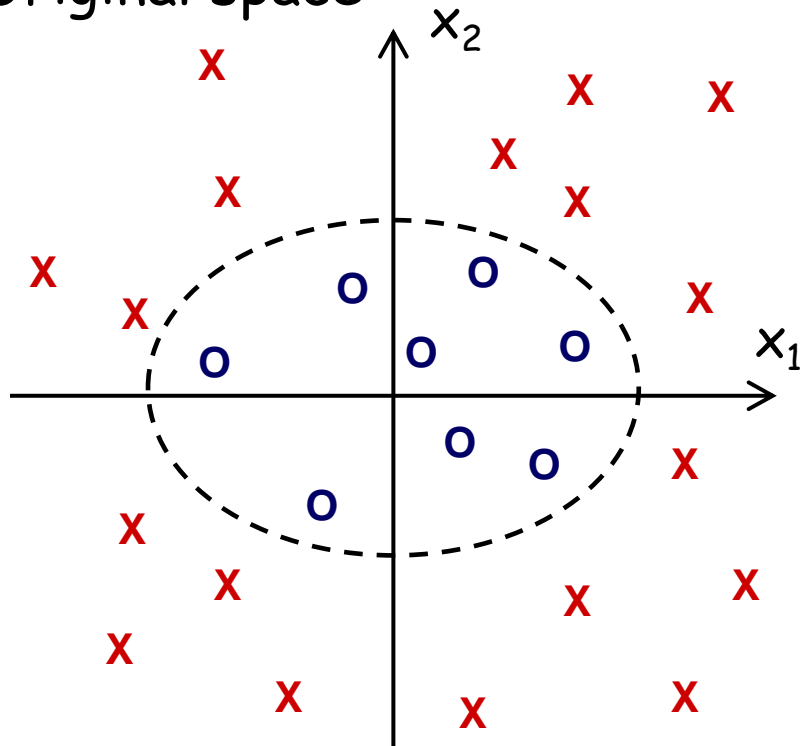
# Example

$$\phi: \mathbb{R}^2 \rightarrow \mathbb{R}^3, (x_1, x_2) \rightarrow \Phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

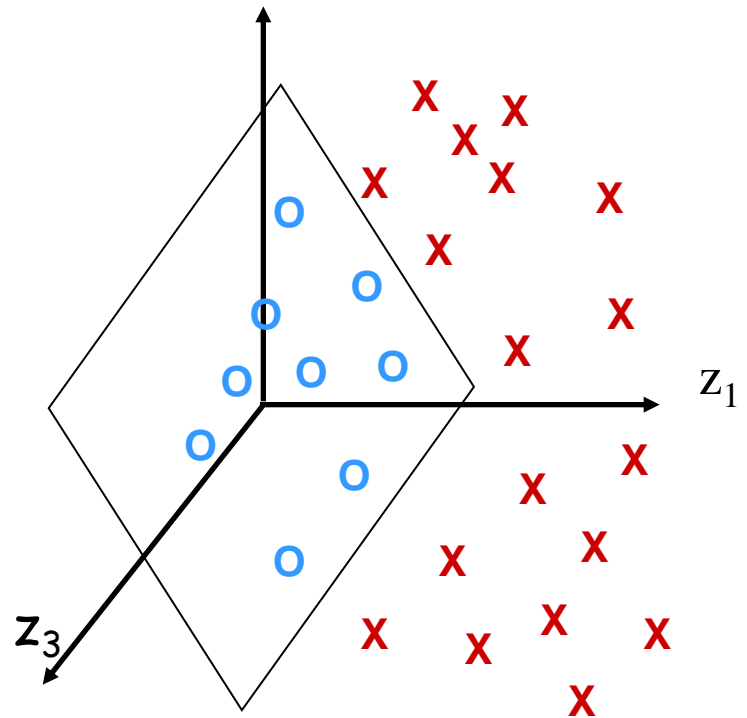
$$\phi(x) \cdot \phi(z) = (x_1^2, x_2^2, \sqrt{2}x_1x_2) \cdot (z_1^2, z_2^2, \sqrt{2}z_1z_2)$$

$$\begin{aligned} K(x, z) &= (\underbrace{x_1 z_1 + x_2 z_2}_x \cdot \underbrace{z_1 z_2}_z)^2 = (x \cdot z)^2 = K(x, z) = (x \cdot z)^{d=2} \\ &= \left( \underbrace{(x_1, x_2)}_x \cdot \underbrace{(z_1, z_2)}_z \right)^2 \end{aligned}$$

Original space



$\Phi$ -space



# Kernels

## Definition

$K(\cdot, \cdot)$  is a kernel if it can be viewed as a legal definition of inner product:

- $\exists \phi: X \rightarrow \mathbb{R}^N$  s.t.  $K(x, z) = \phi(x) \cdot \phi(z)$ 
  - Range of  $\phi$  is called the  $\Phi$ -space.
  - $N$  can be very large.
- But think of  $\phi$  as **implicit**, not explicit!!!!

# Example

**Note:** feature space might not be unique.

$$\phi: \mathbb{R}^2 \rightarrow \mathbb{R}^3, (x_1, x_2) \rightarrow \Phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

$$\begin{aligned}\phi(x) \cdot \phi(z) &= (x_1^2, x_2^2, \sqrt{2}x_1x_2) \cdot (z_1^2, z_2^2, \sqrt{2}z_1z_2) \\ &= (x_1z_1 + x_2z_2)^2 = (x \cdot z)^2 = K(x, z)\end{aligned}$$

$$\phi: \mathbb{R}^2 \rightarrow \mathbb{R}^4, (x_1, x_2) \rightarrow \Phi(x) = (x_1^2, x_2^2, x_1x_2, x_2x_1)$$

$$\begin{aligned}\phi(x) \cdot \phi(z) &= (x_1^2, x_2^2, x_1x_2, x_2x_1) \cdot (z_1^2, z_2^2, z_1z_2, z_2z_1) \\ &= (x \cdot z)^2 = K(x, z)\end{aligned}$$

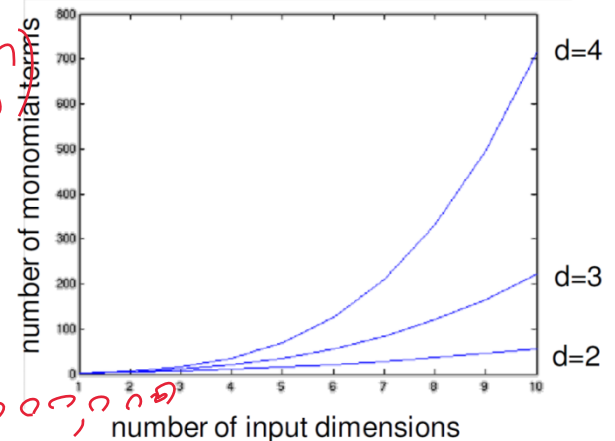
# Avoid explicitly expanding the features

Feature space can grow really large and really quickly....

Crucial to think of  $\phi$  as **implicit**, not explicit!!!!  $(a_1 + a_2 + \dots + a_n)^d$

- Polynomial kernel degree  $d$ ,  $k(x, z) = \underbrace{(x^\top z)^d}_{(x^\top z + 1)^d} = \phi(x) \cdot \phi(z)$

- $x_1^d, x_1 x_2 \dots x_d, x_1^2 x_2 \dots x_{d-1}$   $(x^\top z + 1)^d$
- Total number of such feature is  $\binom{d+n-1}{d} = \frac{(d+n-1)!}{d!(n-1)!}$   $d(n^d)$
- $d = 6, n = 100$ , there are 1.6 billion terms  $\phi: \mathbb{R}^{100} \rightarrow \mathbb{R}^{1,600,000,000}$



$O(n)$  computation!

$$k(x, z) = (x^\top z)^d = \phi(x) \cdot \phi(z)$$

$\begin{pmatrix} 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix} (d=3)$

$\begin{pmatrix} d+n-1 \\ n-1 \end{pmatrix} \begin{array}{|c|c|c|c|} \hline & & & \\ \hline \end{array} (n=4)$

$$= \langle \phi(x), \phi(z) \rangle$$

$$K(x, z) = (\underline{x \cdot z})^d$$

$$\phi(x) = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{bmatrix}$$

$$\phi(z) \in \mathbb{R}^p$$

$$= \underline{(a_1 + a_2 + \dots + a_n)}^d$$

$$= \underline{a_1^d + a_2^d + \dots + a_n^d}$$

$$\underline{a_1^d + a_2^d + \dots + a_n^d}$$

$p = \# \text{ terms}$

$$\underline{a_1, a_2, \dots, a_d}$$

$$\underline{x \cdot z = a_1 + a_2 + \dots + a_n}$$

$\downarrow \quad \downarrow \quad \downarrow$   
 $x_1 z_1 \quad x_2 z_2 \quad x_n z_n$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n$$

$$K \in \mathbb{R}^{m \times m}$$

$j$ -th

$i$ -th

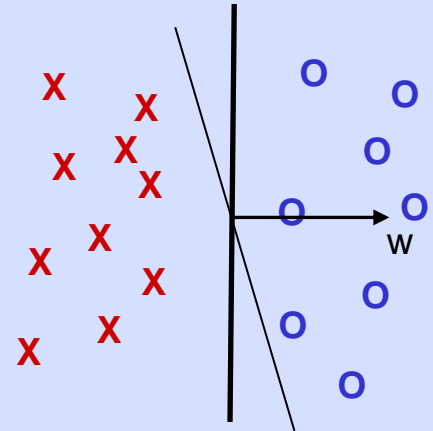
$K(x_i, x_j)$

# Kernelizing a learning algorithm

- If all computations involving instances are in terms of inner products then:
  - Conceptually, work in a very high diml space and the alg's performance depends only on linear separability in that extended space.
  - Computationally, only need to modify the algo by replacing each  $\mathbf{x} \cdot \mathbf{z}$  with a  $K(\mathbf{x}, \mathbf{z})$ .
- Examples of kernalizable algos:
  - classification: Perceptron, SVM.
  - regression: linear, ridge regression.
  - clustering: k-means.

# Kernelizing the Perceptron Algorithm

- Set  $t=1$ , start with the all zero vector  $w_1$ .
- Given example  $x$ , predict + iff  $w_t \cdot x \geq 0$
- On a mistake, update as follows:
  - Mistake on positive,  $w_{t+1} \leftarrow w_t + x$
  - Mistake on negative,  $w_{t+1} \leftarrow w_t - x$



Easy to kernelize since  $w_t$  is weighted sum of incorrectly classified examples  $w_t = a_{i_1}x_{i_1} + \dots + a_{i_k}x_{i_k}$

Replace  $w_t \cdot x = a_{i_1}x_{i_1} \cdot x + \dots + a_{i_k}x_{i_k} \cdot x$  with  $a_{i_1} K(x_{i_1}, x) + \dots + a_{i_k} K(x_{i_k}, x)$

Note: need to store all the mistakes so far.



# Kernelizing the Perceptron Algorithm

- Given  $x$ , predict + iff

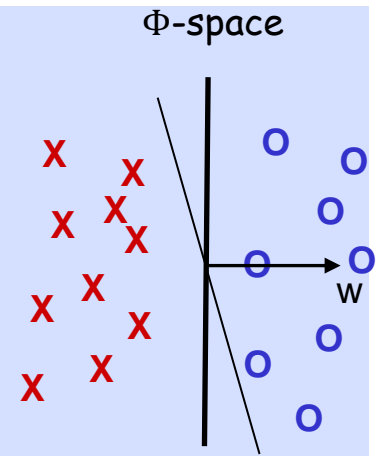
$$a_{i_1} K(x_{i_1}, x) + \dots + a_{i_{t-1}} \underbrace{K(x_{i_{t-1}}, x)}_{\phi(x_{i_{t-1}}) \cdot \phi(x)} \geq 0$$

- On the  $t$ th mistake, update as follows:

- Mistake on positive, set  $a_{i_t} \leftarrow 1$ ; store  $x_{i_t}$
- Mistake on negative,  $a_{i_t} \leftarrow -1$ ; store  $x_{i_t}$

$$w = \sum x^T a$$

$$O(N)$$



Perceptron  $w_t = a_{i_1} x_{i_1} + \dots + a_{i_k} x_{i_k}$   $w_t \cdot \phi(x) = \sum_i a_i \phi(x_{i_1}) \cdot \phi(x)$

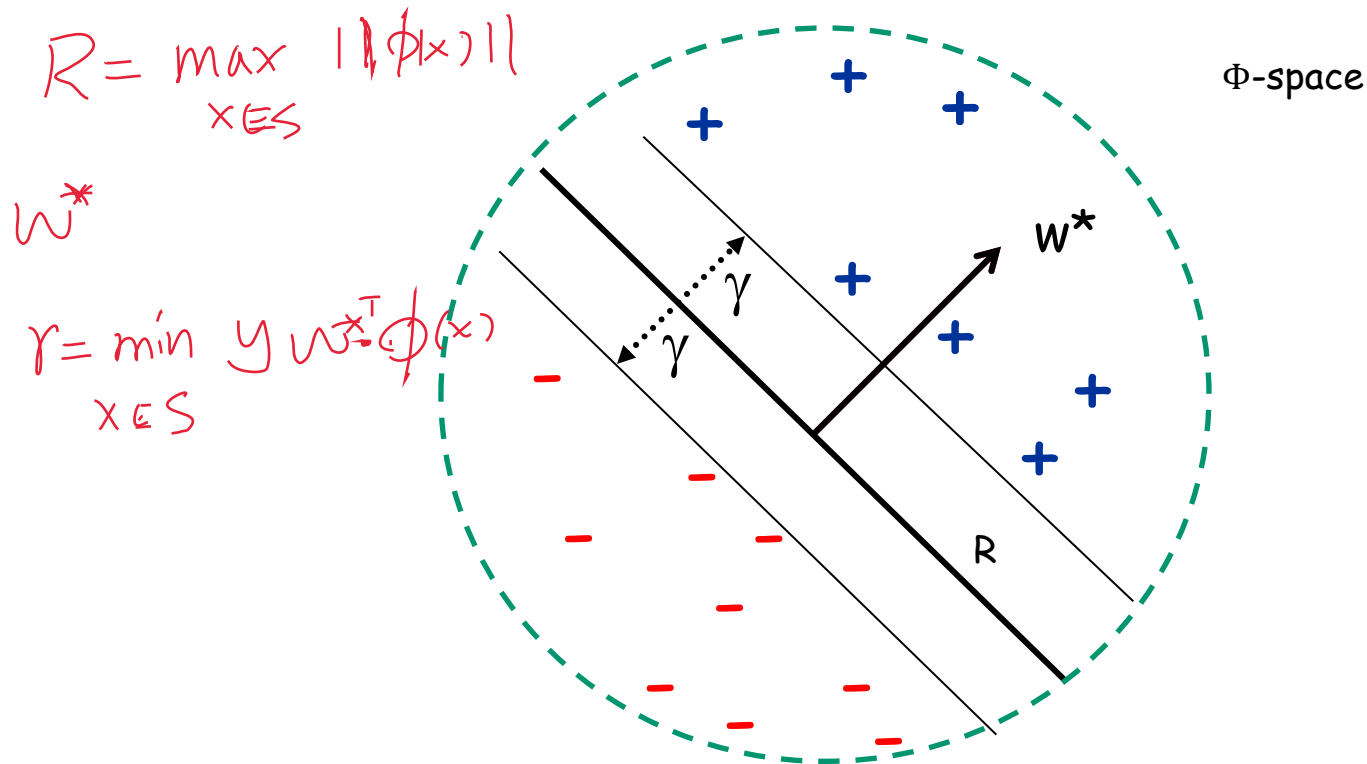
$$w_t \cdot x = a_{i_1} x_{i_1} \cdot x + \dots + a_{i_k} x_{i_k} \cdot x \rightarrow a_{i_1} K(x_{i_1}, x) + \dots + a_{i_k} K(x_{i_k}, x)$$

Exact same behavior/prediction rule as if mapped data in the  $\phi$ -space and ran Perceptron there!

Do this implicitly, so computational savings!!!!

# Generalize Well if Good Margin

- If data is linearly separable by margin in the  $\phi$ -space, then small mistake bound.
- If margin  $\gamma$  in  $\phi$ -space, then Perceptron makes  $\left(\frac{R}{\gamma}\right)^2$  mistakes.



# Kernels: More Examples

$$K(x, y) = \phi(x) \cdot \phi(y)$$

- Linear:  $K(x, z) = x \cdot z$   $\phi(x) = x$
- Polynomial:  $K(x, z) = (x \cdot z)^d$  or  $K(x, z) = (1 + x \cdot z)^d$   $O(n^d)$
- Gaussian:  $K(x, z) = \exp \left[ -\frac{\|x-z\|^2}{2\sigma^2} \right]$   $\frac{x \cdot z}{\text{infinite}}$   $\frac{e^{-(x \cdot z)}}{\phi(x) \rightarrow \infty}$
- Laplace Kernel:  $K(x, z) = \exp \left[ -\frac{\|x-z\|}{2\sigma^2} \right]$
- Kernel for non-vectorial data, e.g., measuring similarity between sequences.  
 $x \cdot z \rightarrow K(x, z) \leftarrow \text{similarity}$

$$e^x = \sum_{j=0}^{\infty} \frac{x^j}{j!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

# Properties of Kernels

## Theorem (Mercer)

$K$  is a kernel if and only if:

- $K$  is symmetric
- For any set of training points  $x_1, x_2, \dots, x_m$  and for any  $a_1, a_2, \dots, a_m \in \mathbb{R}$ , we have:

$$\sum_{i,j} a_i a_j K(x_i, x_j) \geq 0$$

$$a^T K a \geq 0$$

I.e.,  $K = (K(x_i, x_j))_{i,j=1,\dots,n}$  is positive semi-definite.


$$\begin{aligned} K(x, z) &= K(z, x) \\ &= \langle \phi(z), \phi(x) \rangle \end{aligned}$$

Gram matrix

$$K \in \mathbb{R}^{m \times m}$$

( $m$ : #examples)

# Kernel Methods

- Offer great **modularity**. 
- No need to change the underlying learning algorithm to accommodate a particular choice of kernel function.
- Also, we can substitute a different algorithm while maintaining the same kernel.

# Kernel, Closure Properties

$$\phi(x) : \mathbb{R}^n \rightarrow \mathbb{R}^N$$

Easily create new kernels using basic ones!



$$\phi_1(x) : \mathbb{R}^n \rightarrow \mathbb{R}^{N_1}$$

$$\phi_2 : \mathbb{R}^n \rightarrow \mathbb{R}^{N_2}$$

**Fact:** If  $K_1(\cdot, \cdot)$  and  $K_2(\cdot, \cdot)$  are kernels  $c_1 \geq 0, c_2 \geq 0$ ,

then  $K(x, z) = c_1 K_1(x, z) + c_2 K_2(x, z)$  is a kernel.

$$= \phi(x) \cdot \phi(z)$$

**Key idea:** concatenate the  $\phi$  spaces.

$$\phi(x) = \begin{bmatrix} \sqrt{c_1} \phi_1(x) \\ \sqrt{c_2} \phi_2(x) \end{bmatrix} \in \mathbb{R}^{N_1 + N_2}$$

$$\phi(x) = (\sqrt{c_1} \phi_1(x), \sqrt{c_2} \phi_2(x))$$

$$\phi(x) \cdot \phi(z) = c_1 \phi_1(x) \cdot \phi_1(z) + c_2 \phi_2(x) \cdot \phi_2(z)$$

$$K_1(x, z)$$

$$K_2(x, z)$$

# Kernel, Closure Properties

$$\langle A, B \rangle = \text{Tr}(A^T B)$$

Easily create new kernels using basic ones!



**Fact:** If  $K_1(\cdot, \cdot)$  and  $K_2(\cdot, \cdot)$  are kernels,  $\phi: \mathbb{R}^n \rightarrow \mathbb{R}^{N_1 \times N_2}$  then  $K(x, z) = \underbrace{K_1(x, z)}_{\mathbb{R}^{N_1}} \underbrace{K_2(x, z)}_{\mathbb{R}^{N_2}}$  is a kernel.

**Key idea:**  $\phi(x) = (\phi_{1,i}(x) \phi_{2,j}(x))_{i \in \{1, \dots, n\}, j \in \{1, \dots, m\}}$

$$\begin{aligned} \phi(x) \cdot \phi(z) &= \sum_{i,j} \phi_{1,i}(x) \phi_{2,j}(x) \phi_{1,i}(z) \phi_{2,j}(z) \\ &= \sum_i \phi_{1,i}(x) \phi_{1,i}(z) \left( \sum_j \phi_{2,j}(x) \phi_{2,j}(z) \right) \\ &= \sum_i \phi_{1,i}(x) \phi_{1,i}(z) K_2(x, z) = K_1(x, z) K_2(x, z) \end{aligned}$$

$K_1, K_2$   
 $\stackrel{K_1(x,z) + K_2(x,z)}{=} \left( K_1(x,z) + K_2(x,z) \right)^2$   
 $K(x,z)$

# Kernels, Discussion

- If all computations involving instances are in terms of inner products then:
  - Conceptually, work in a very high diml space and the alg's performance depends only on linear separability in that extended space.
  - Computationally, only need to modify the algo by replacing each  $\mathbf{x} \cdot \mathbf{z}$  with a  $K(\mathbf{x}, \mathbf{z})$ .
- Lots of Machine Learning algorithms are kernalizable:
  - classification: Perceptron, SVM.
  - regression: linear regression.
  - clustering: k-means.



$$y \sim w^T x$$

$$\min_w \frac{1}{2} \|y - Xw\|_2^2 + \frac{\lambda}{2} \|w\|_2^2 \quad Q(w)$$

$$\frac{\partial Q(w)}{\partial w} = (-X^T)(y - Xw) + \lambda w = 0$$

$$w = \frac{1}{\lambda} X^T (y - Xw) = \underline{X^T a}$$

$$a = \frac{1}{\lambda} (y - Xw)$$

$$= \frac{1}{\lambda} (y - X X^T a)$$

$$\Rightarrow a = (\underline{X X^T} + \lambda I)^{-1} y$$

$$K \left\{ \begin{array}{l} a = (K + \lambda I)^{-1} y \end{array} \right.$$

$$\begin{matrix} n \\ \left\{ \begin{array}{c} \boxed{X} \end{array} \right\}^T x \\ Xx \end{matrix}$$

$$K(x, z) = x \cdot z \quad \phi(x) = x$$

Gram matrix  $X \in \mathbb{R}^{m \times n}$

$$K = \underline{X X^T} \quad \begin{matrix} m \\ \left\{ \begin{array}{c} \boxed{i \quad j} \end{array} \right\} \end{matrix} \rightarrow \langle x_i, x_j \rangle$$

$$\downarrow \quad \begin{matrix} m \\ \left\{ \begin{array}{c} \boxed{i \quad j} \end{array} \right\} \end{matrix} \rightarrow K(x_i, x_j) = \underline{\exp(-\dots)}$$

$x$ : testing point

$$\hat{y} = \underline{X^T} \cdot w$$

$$= \underline{X^T X^T} \cdot a$$

$$= (\underline{X X^T})^T \cdot (X X^T + \lambda I)^{-1} y$$

$$= \underline{K^T} \cdot \underline{(K + \lambda I)^{-1} y}$$

$$\downarrow \text{GR}^m \quad \downarrow K(x_i, x_j)$$

$$\downarrow K(x_i, x)$$

# Kernels, Discussion

- If all computations involving instances are in terms of inner products then:
  - Conceptually, work in a very high diml space and the alg's performance depends only on linear separability in that extended space.
  - Computationally, only need to modify the algo by replacing each  $x \cdot z$  with a  $K(x, z)$ .

## How to choose a kernel:

- Kernels often encode domain knowledge (e.g., string kernels)
- Use Cross-Validation to choose the parameters, e.g.,  $\sigma$  for

Gaussian Kernel  $K(x, z) = \exp \left[ -\frac{\|x-z\|^2}{2\sigma^2} \right]$

*Multi-kernel learning*

- **Learn** a good kernel; e.g., [Lanckriet-Cristianini-Bartlett-El Ghaoui-Jordan'04]

$$K = G_1 K_1 + G_2 K_2 + G_3 K_3$$