

# Kernels Methods in Machine Learning

## Kernelized Perceptron

Maria-Florina Balcan

02/18/2019

# Admin

HWK 3: out today, due on Feb 25

Midterm Exam: March 4th

# Quick Recap about Perceptron and Margins

# The Online Learning Model

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

Afterwards observe the outcome.

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



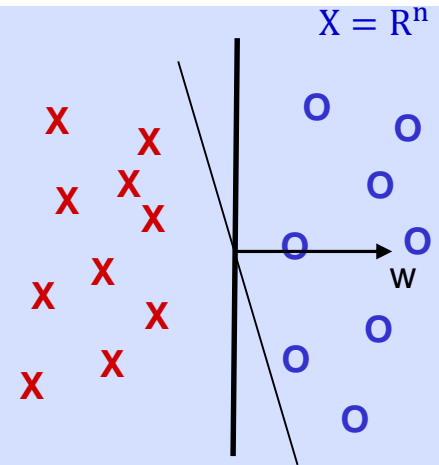
## Mistake bound model

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

# Perceptron Algorithm in Online Model

WLOG homogeneous linear separators

- 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

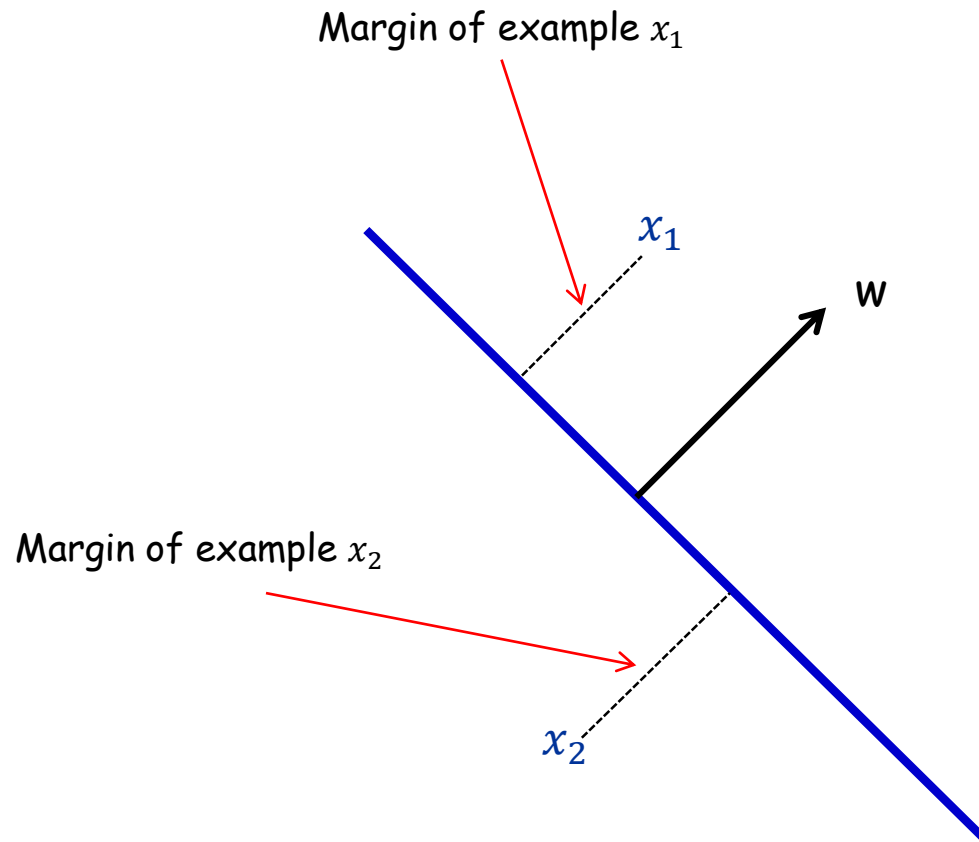
$$w_t = a_{i_1}x_{i_1} + \dots + a_{i_k}x_{i_k} \quad \text{So, } w_t \cdot x = a_{i_1}x_{i_1} \cdot x + \dots + a_{i_k}x_{i_k} \cdot x$$

**Note 2:** Number of mistakes ever made depends only on the geometric margin (amount of wiggle room) of examples seen.

- 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  $w \cdot x = 0$ .



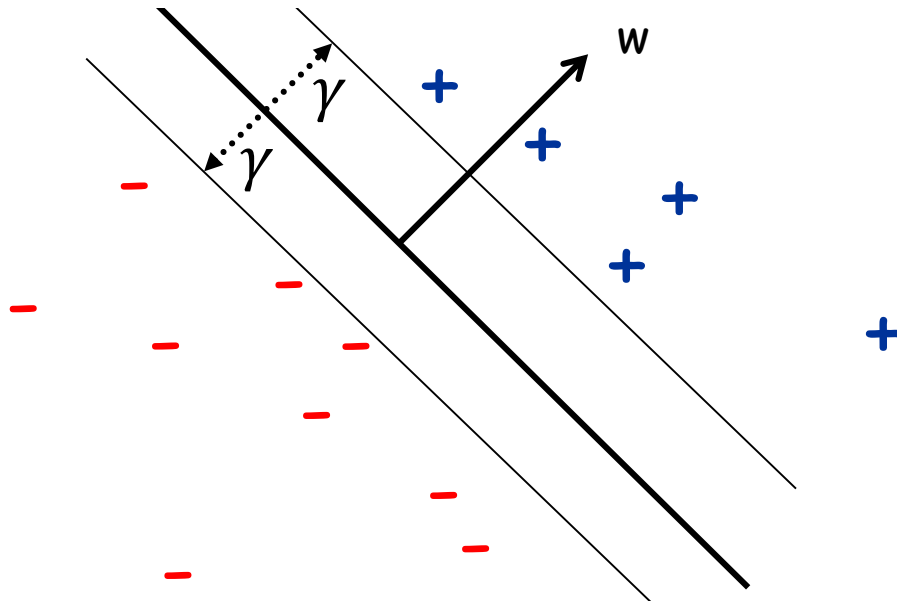
If  $\|w\| = 1$ , margin of  $x$  w.r.t.  $w$  is  $|x \cdot w|$ .

# Geometric Margin

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

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

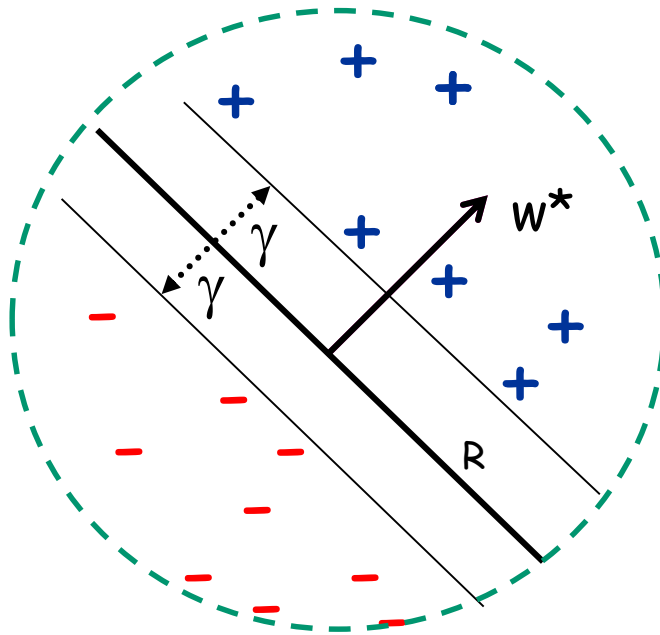
**Definition:** The margin  $\gamma$  of a set of examples  $S$  is the **maximum**  $\gamma_w$  over all linear separators  $w$ .



# 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.

(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.)



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"

# Overview of Kernel Methods

## What is a Kernel?

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

$$\text{E.g., } K(x, z) = (x \cdot z + 1)^d$$

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

## Why Kernels matter?

- 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.

# Kernels

## Definition

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

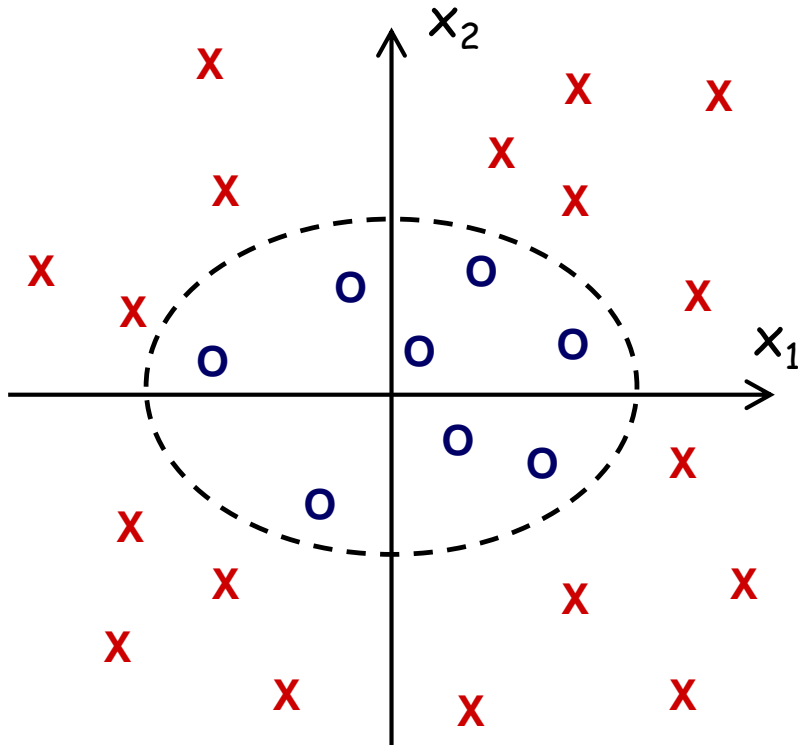
- $\exists \phi: X \rightarrow \mathbb{R}^N$  s.t.  $K(x, z) = \phi(x) \cdot \phi(z)$ 
  - Range of  $\phi$  is called the  $\Phi$ -space.
  - $N$  can be very large.
- But think of  $\phi$  as **implicit**, not explicit!!!!

# Example

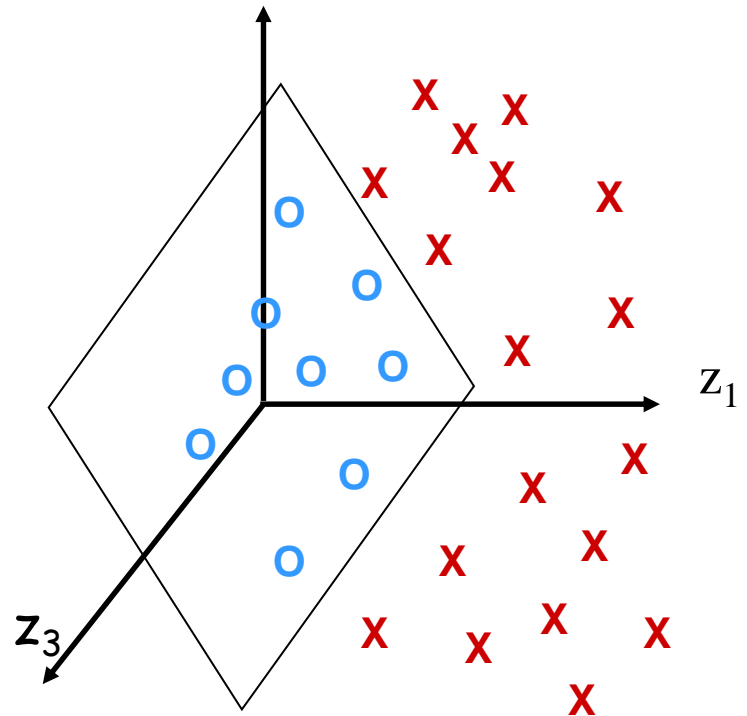
For  $n=2$ ,  $d=2$ , the kernel  $K(x, z) = (x \cdot z)^d$  corresponds to

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

Original space



$\Phi$ -space

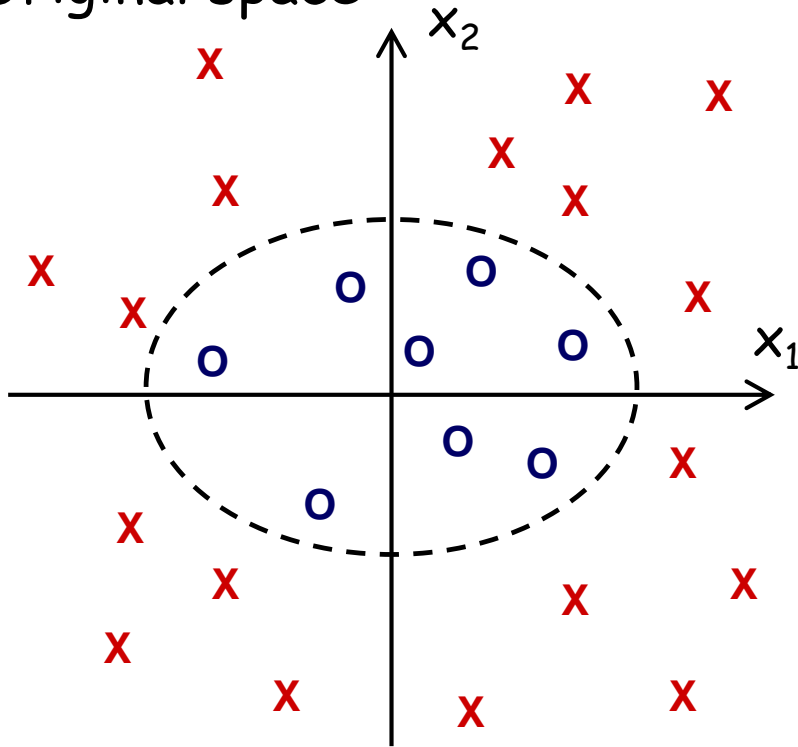


# Example

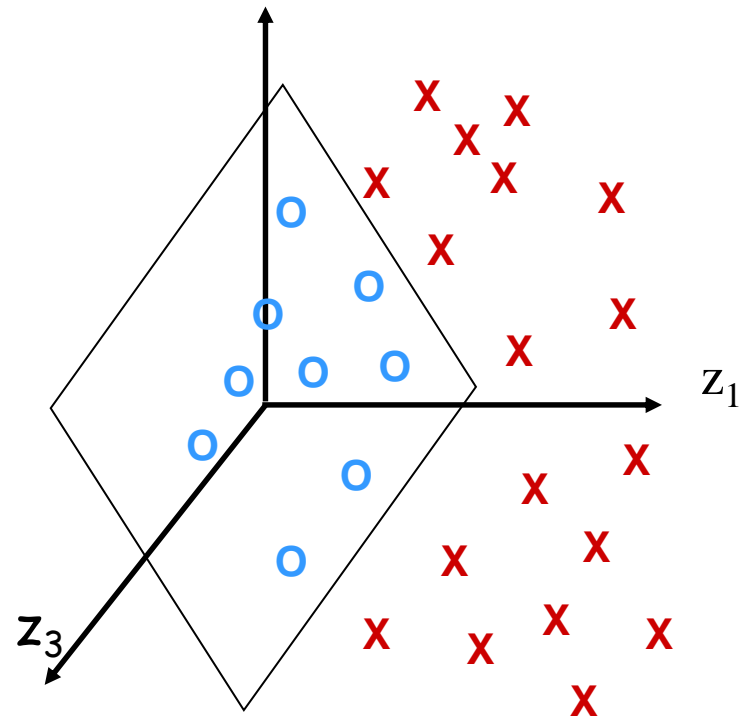
$$\phi: \mathbb{R}^2 \rightarrow \mathbb{R}^3, (x_1, x_2) \rightarrow \Phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

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

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!!!!

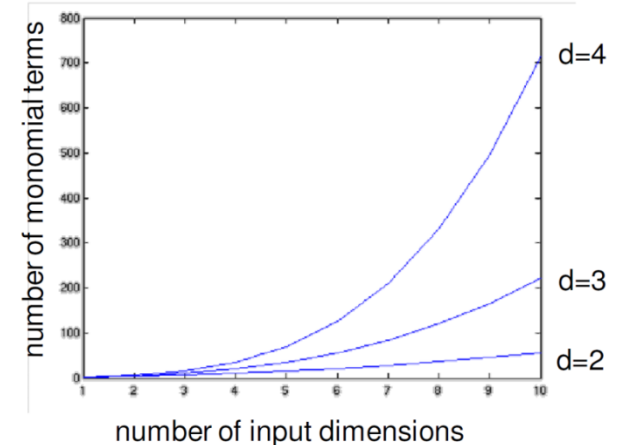
- Polynomial kernel degree  $d$ ,  $k(x, z) = (x^\top z)^d = \phi(x) \cdot \phi(z)$

- $x_1^d, x_1 x_2 \dots x_d, x_1^2 x_2 \dots x_{d-1}$

- Total number of such feature is

$$\binom{d+n-1}{d} = \frac{(d+n-1)!}{d! (n-1)!}$$

- $d = 6, n = 100$ , there are 1.6 billion terms



$O(n)$  computation!

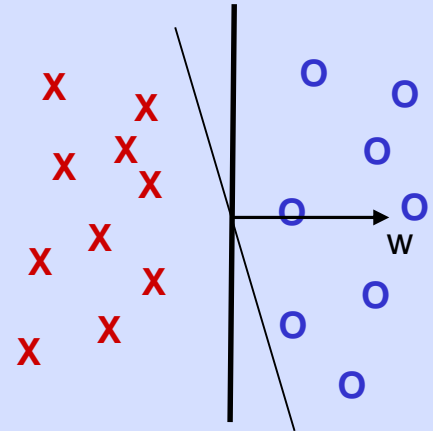
$$k(x, z) = (x^\top z)^d = \phi(x) \cdot \phi(z)$$

# 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  $a_{i_1} K(x_{i_1}, x) + \dots + a_{i_k} K(x_{i_k}, x)$

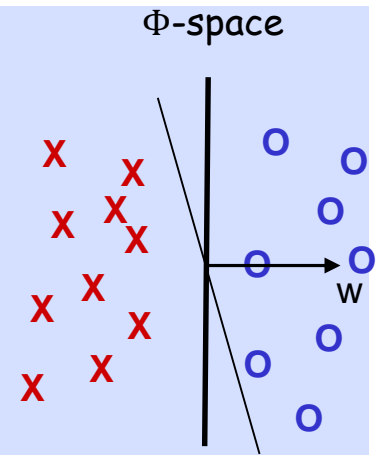
Note: need to store all the mistakes so far.

# 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  $x_{i_t}$



Perceptron  $w_t = a_{i_1}x_{i_1} + \dots + a_{i_k}x_{i_k}$

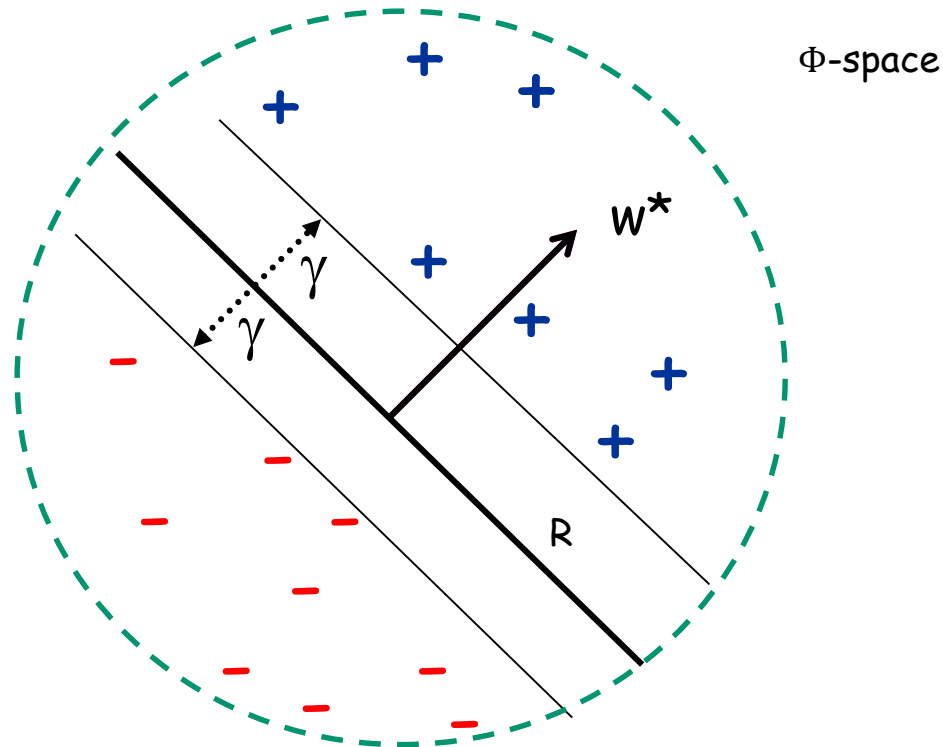
$$w_t \cdot x = a_{i_1}x_{i_1} \cdot x + \dots + a_{i_k}x_{i_k} \cdot x \quad \rightarrow \quad a_{i_1} K(x_{i_1}, x) + \dots + a_{i_k} K(x_{i_k}, x)$$

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

- Linear:  $K(x, z) = x \cdot z$
- Polynomial:  $K(x, z) = (x \cdot z)^d$  or  $K(x, z) = (1 + x \cdot z)^d$
- Gaussian:  $K(x, z) = \exp \left[ -\frac{\|x - z\|^2}{2 \sigma^2} \right]$
- 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.

# 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.

# 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

Easily create new kernels using basic ones!



**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.

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

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

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

$$K_1(x, z)$$

$$K_2(x, z)$$

# Kernel, Closure Properties

Easily create new kernels using basic ones!



**Fact:** If  $K_1(\cdot, \cdot)$  and  $K_2(\cdot, \cdot)$  are kernels,

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

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

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

# 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.

# 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  $K(\mathbf{x}, \mathbf{z})$ .

## 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  $K(\mathbf{x}, \mathbf{z}) = \exp \left[ -\frac{\|\mathbf{x} - \mathbf{z}\|^2}{2 \sigma^2} \right]$
- **Learn** a good kernel; e.g., [Lanckriet-Cristianini-Bartlett-El Ghaoui-Jordan'04]