

Kernels Methods in Machine Learning

- Perceptron.
- Geometric Margins.
- Kernel Methods.

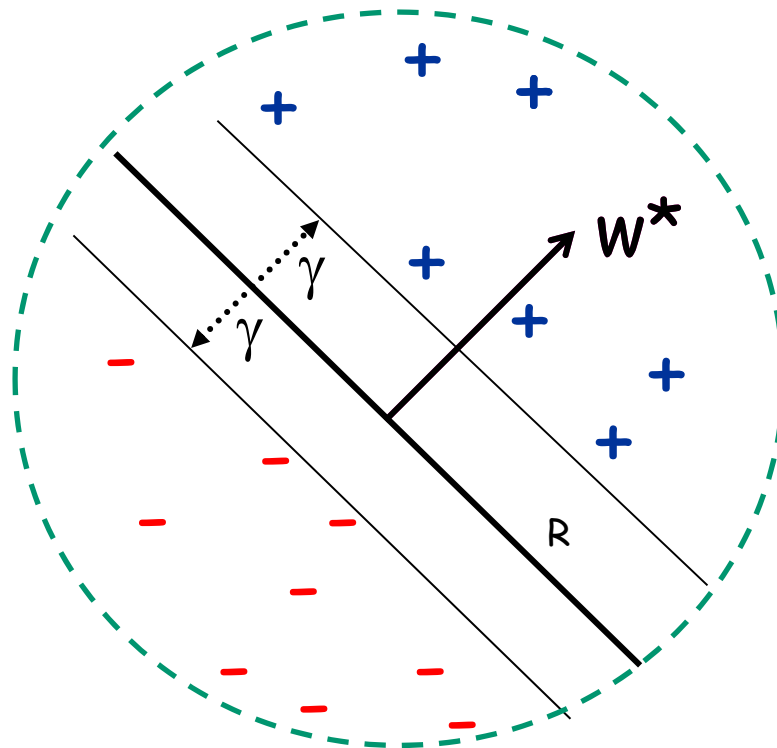
Maria-Florina Balcan

03/23/2015

Perceptron: Mistake Bound

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

(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 Algorithm: Analysis

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

Update rule:

- Mistake on positive: $w_{t+1} \leftarrow w_t + x$
- Mistake on negative: $w_{t+1} \leftarrow w_t - x$

Proof:

Idea: analyze $w_t \cdot w^*$ and $\|w_t\|$, where w^* is the max-margin sep, $\|w^*\| = 1$.

Claim 1: $w_{t+1} \cdot w^* \geq w_t \cdot w^* + \gamma$. (because $l(x)x \cdot w^* \geq \gamma$)

Claim 2: $\|w_{t+1}\|^2 \leq \|w_t\|^2 + R^2$. (by Pythagorean Theorem)

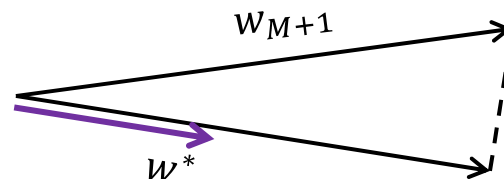
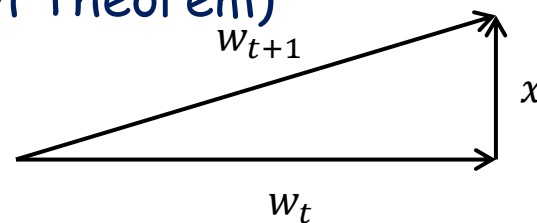
After M mistakes:

$$w_{M+1} \cdot w^* \geq \gamma M \text{ (by Claim 1)}$$

$$\|w_{M+1}\| \leq R\sqrt{M} \text{ (by Claim 2)}$$

$$w_{M+1} \cdot w^* \leq \|w_{M+1}\| \text{ (since } w^* \text{ is unit length)}$$

$$\text{So, } \gamma M \leq R\sqrt{M}, \text{ so } M \leq \left(\frac{R}{\gamma}\right)^2.$$



Perceptron Extensions

- Can use it to find a consistent separator (by cycling through the data).
- One 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!

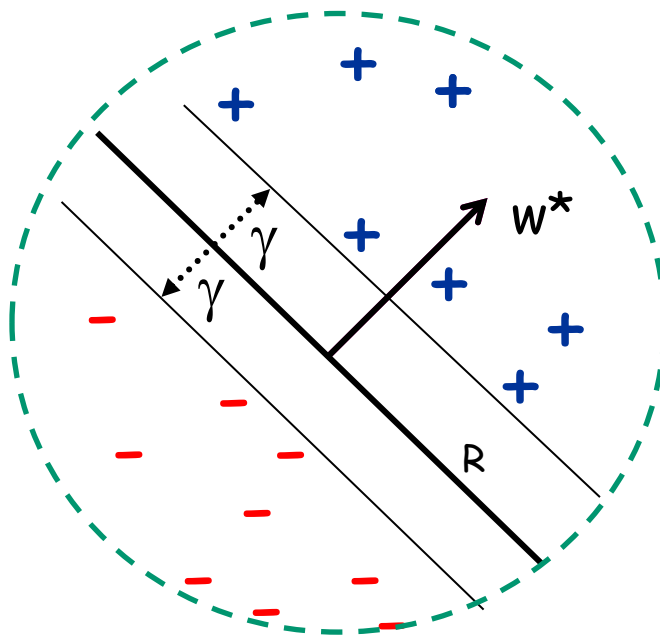
Perceptron Discussion

- Simple online algorithm for learning linear separators with a nice guarantee that depends only on the geometric (aka L_2, L_2) margin.
- It can be kernelized to handle non-linear decision boundaries --- see next class!
- Simple, but very useful in applications like Branch prediction; it also has interesting extensions to structured prediction.

Perceptron: Mistake Bound

Theorem: If data linearly separable by margin γ 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.)

Perceptron Extensions

- Can use it to find a consistent separator with a given set S linearly separable by margin γ (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 γ and points inside a ball of radius R , then Perceptron makes $\leq (R/\gamma)^2$ mistakes.

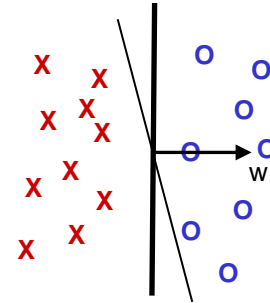
Implies that large margin classifiers have smaller complexity!

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 γ ?



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

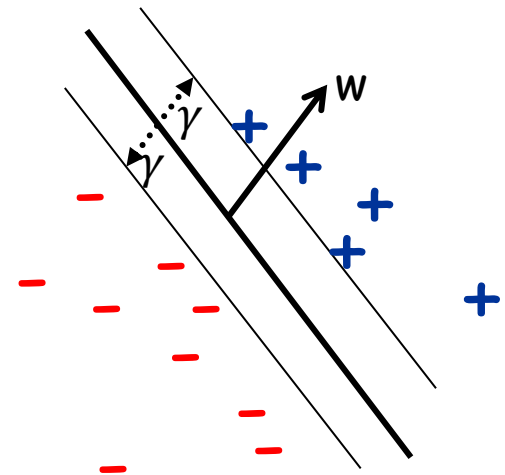
- So, large margin classifiers have smaller complexity!
 - Nice implications for usual distributional learning setting.
 - Less classifiers to worry about that will look good over the sample, but bad over all....
- Less prone to overfitting!!!!

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 γ and if alg. produces a large margin classifier, then amount of data needed depends only on R/γ [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"

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 Φ s.t. $K(\text{img1}, \text{img2}) = \Phi(\text{img1}) \cdot \Phi(\text{img2})$

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

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

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 Φ -space.
- If data is linearly separable by large margin in the Φ -space, then good sample complexity.

[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:

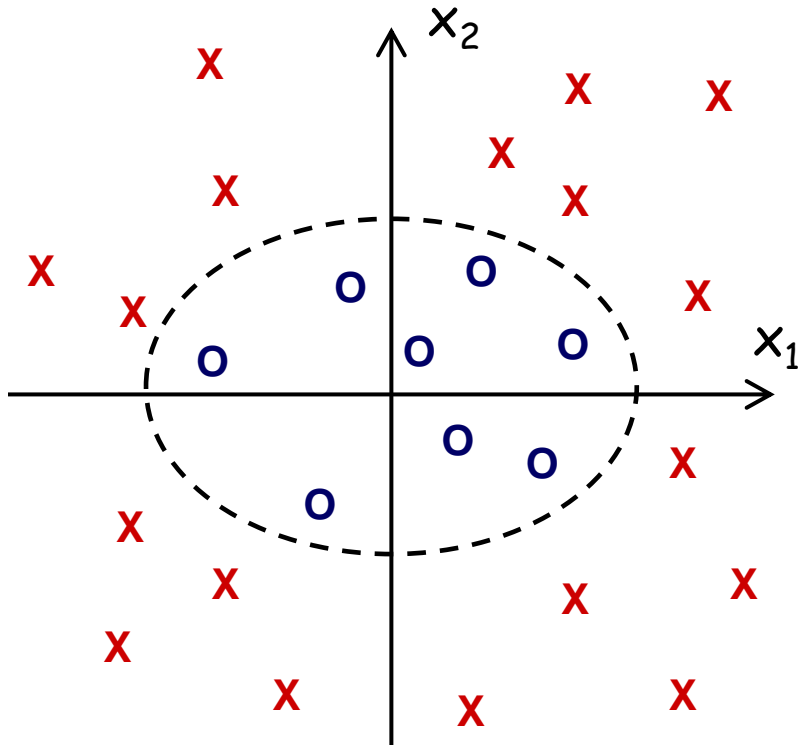
- $\exists \phi: X \rightarrow \mathbb{R}^N$ s.t. $K(x, z) = \phi(x) \cdot \phi(z)$
 - Range of ϕ is called the Φ -space.
 - N can be very large.
- But think of ϕ as **implicit**, not explicit!!!!

Example

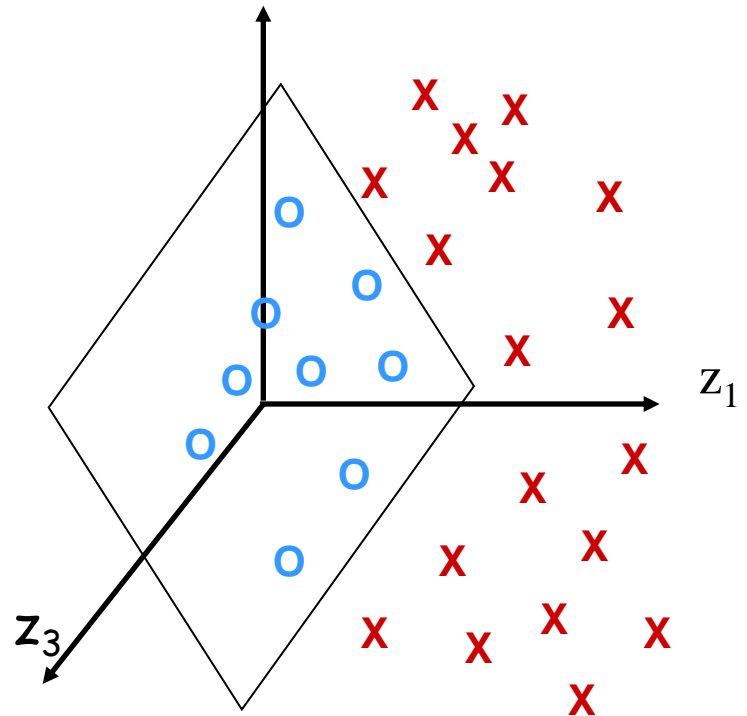
For $n=2$, $d=2$, the kernel $K(x, z) = (x \cdot z)^d$ corresponds to

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

Original space



Φ -space

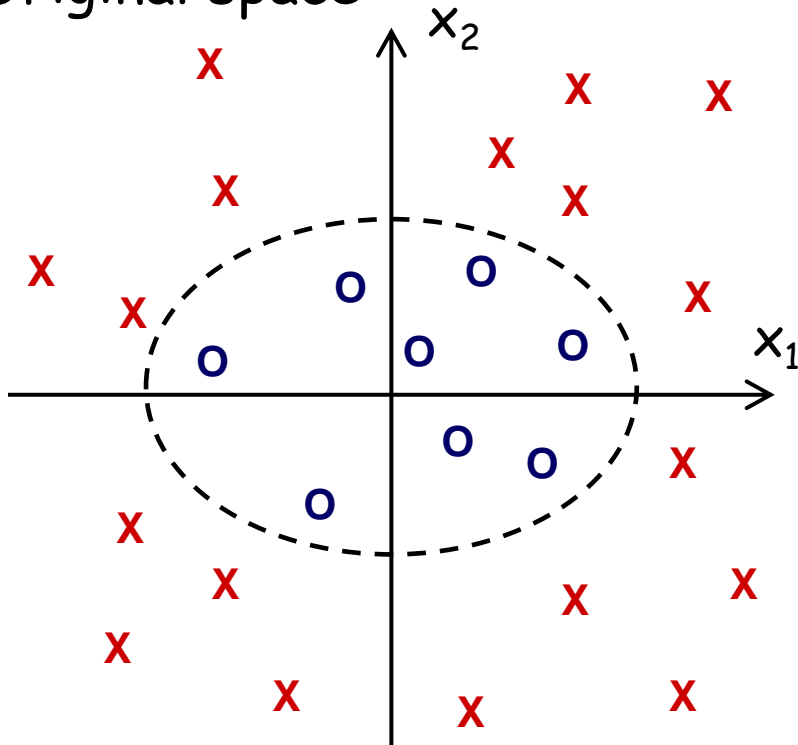


Example

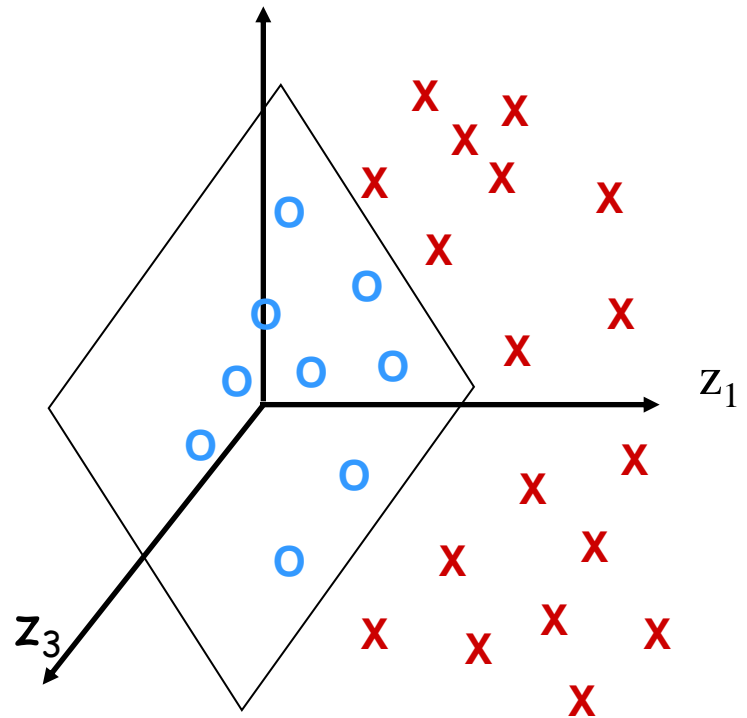
$$\phi: \mathbb{R}^2 \rightarrow \mathbb{R}^3, (x_1, x_2) \rightarrow \Phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

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

Original space



Φ -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 ϕ is called the Φ -space.
 - N can be very large.
- But think of ϕ 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 ϕ as **implicit**, not explicit!!!!

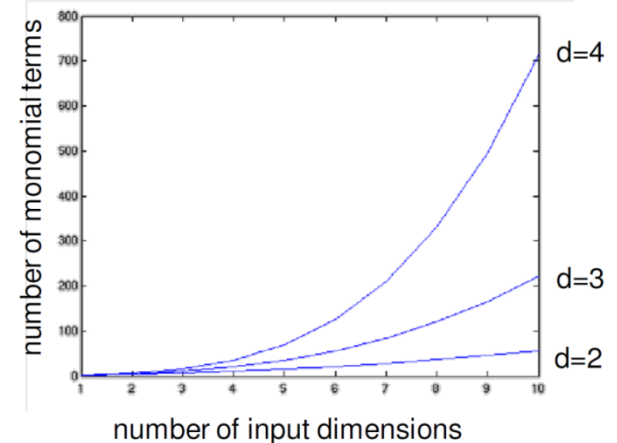
- Polynomial kernel degree d , $k(x, z) = (x^\top z)^d = \phi(x) \cdot \phi(z)$

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

- 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!

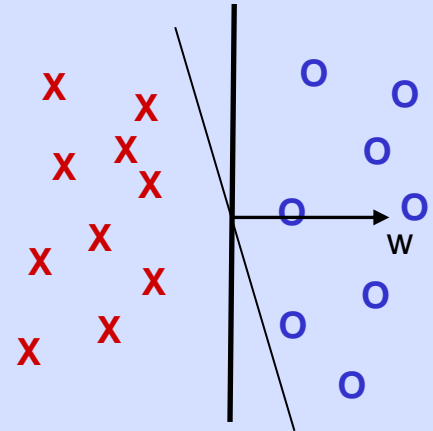
$$k(x, z) = (x^\top z)^d = \phi(x) \cdot \phi(z)$$

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

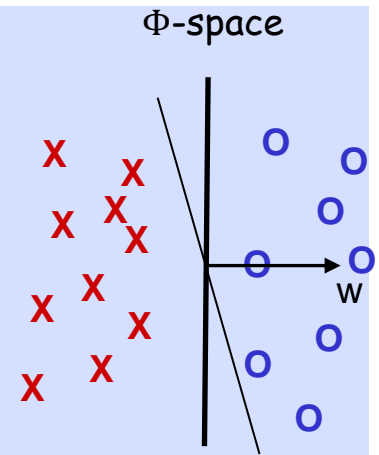
Note: need to store all the mistakes so far.

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

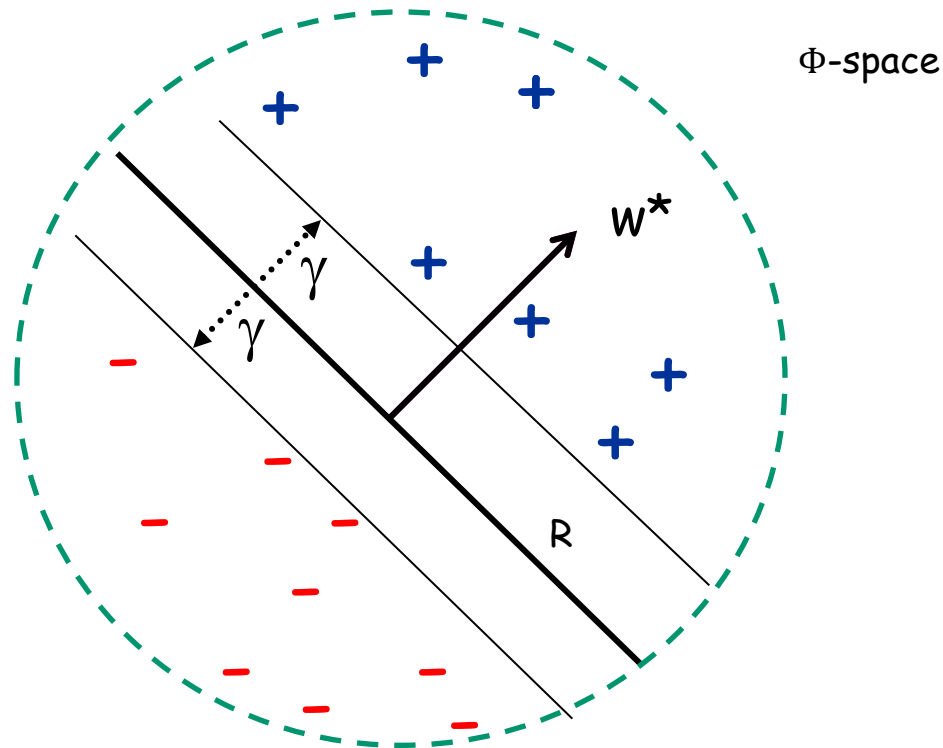
$$w_t \cdot x = a_{i_1} x_{i_1} \cdot x + \dots + a_{i_k} x_{i_k} \cdot x \quad \rightarrow \quad a_{i_1} K(x_{i_1}, x) + \dots + a_{i_k} K(x_{i_k}, x)$$

Exact same behavior/prediction rule as if mapped data in the ϕ -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 ϕ -space, then small mistake bound.
- If margin γ in ϕ -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 ϕ 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: $\phi(x) = (\phi_{1,i}(x) \phi_{2,j}(x))_{i \in \{1, \dots, n\}, j \in \{1, \dots, m\}}$

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

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., σ 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]