Kernels Methods in Machine Learning

- Perceptron.
- Geometric Margins.
- Kernel Methods.

Maria-Florina Balcan

03/23/2015

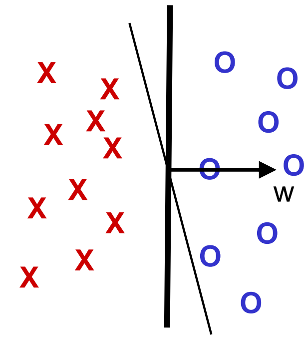
Theorem: If data linearly separable by margin γ and points inside a ball of radius R, then Perceptron makes $\leq (R/\gamma)^2$ mistakes.

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).

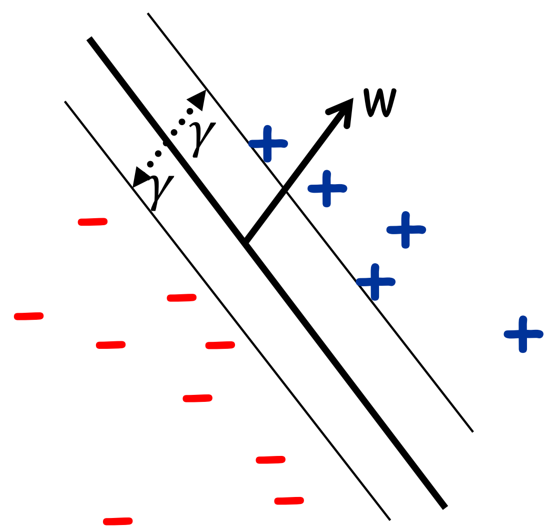
- So, large margin classifiers have smaller complexity!
  - Nice implications for usual distributional learning setting.
  - Less classifiers to worry about that will look good over the sample, but bad over all….
- Less prone to overfitting!!!!

Margin Important Theme in ML.

Both sample complexity and algorithmic implications.

Sample/Mistake Bound complexity:

- If large margin, # mistakes Peceptron 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"
  - 决策树 神经网络
  - (e.g., decision trees, neural networks, boosting).
- "Use a Kernel" (a neat solution that attracted a lot of attention)
- "Use a Deep Network" 更多层的网络
- "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(, ) =Φ()$\cdot$ Φ()

可看作K(<x1,x2>), 对内积的函数

E.g., $K(x,y) = (x \cdot y + 1)^d$ 超参数.

$\Phi: \mathbb{R}^n \to \mathbb{R}^N, N \gg n$

$K(x_1, x_2) = \langle \Phi(x_1), \Phi(x_2) \rangle$

$= \Phi(x_1)^T \Phi(x_2)$

$= \Phi(x_1) \cdot \Phi(x_2)$

$\Phi(x) \in \mathbb{R}^N$

$O(N)$

ϕ: (n-dimensional space) $\to$ n^d-dimensional space $O(n)$

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

Why Kernels matter?

kernel trick

可以隐式转化为高维进行内积 但是 $N \gg n$, 复杂度极高

- 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.]

RKHS : Reproducing Kernel Hilbert Space

kernel trick : $\langle K(x_1,\cdot), K(x_2,\cdot) \rangle = K(x_1, x_2)$ $K(x_1,\cdot)$是把x_1经过Φ后映射的结果

令 $f(x_1) = K(x_1, \cdot)$ $\Rightarrow$ $\langle f(x), K(x_2, \cdot) \rangle = f(x_1, x_2)$ $\Rightarrow$ 再生核.

Kernels

Definition

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

- $\exists\ \phi: X \to R^N$ 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

原始空间维度

kernel的度

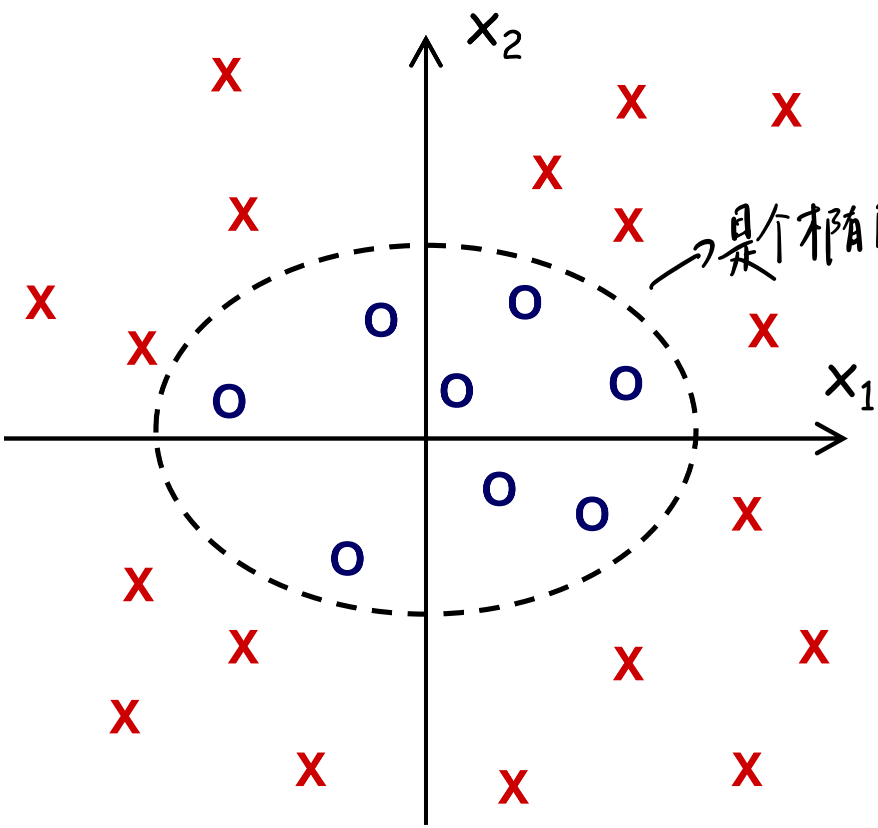
比较特殊：只有d次项

一般若要保留所有次项，则需 $(x\cdot z+1)^d$ 表达更强.

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

$$(x_1,x_2)\rightarrow\Phi(x)=\underbrace{(x_1^2,x_2^2,\sqrt{2}x_1x_2)}_{\text{一般不需要知道这个}}$$

Original space



是个椭圆 => 二次项

$\beta_1 x_1^2+\beta_2 x_2^2+\beta_3 x_1 x_2+\beta_0=0$

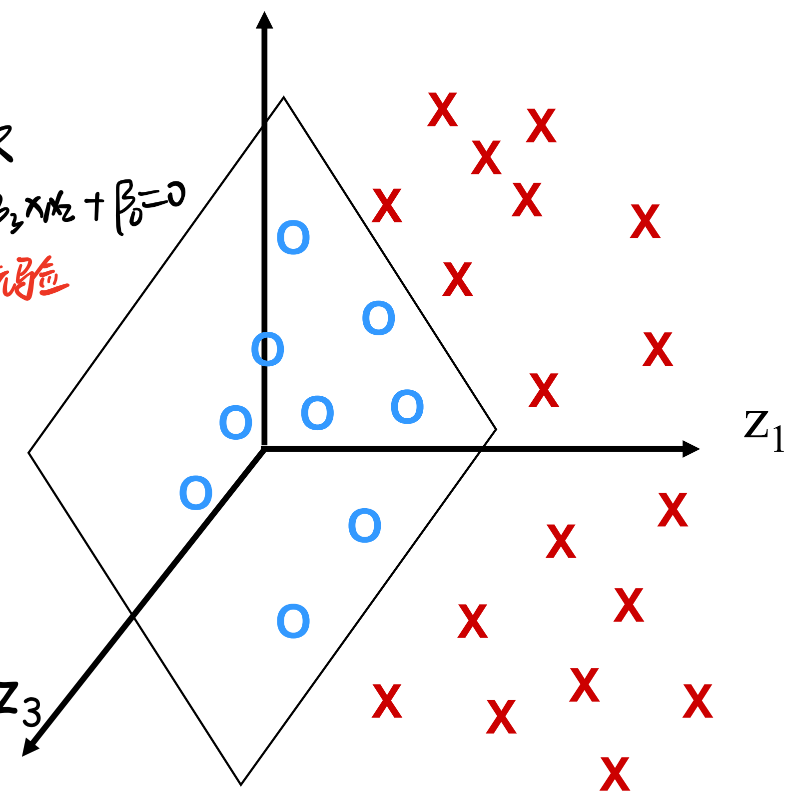
先验

$\Downarrow$

取 $d=2$

超参.

Φ-space



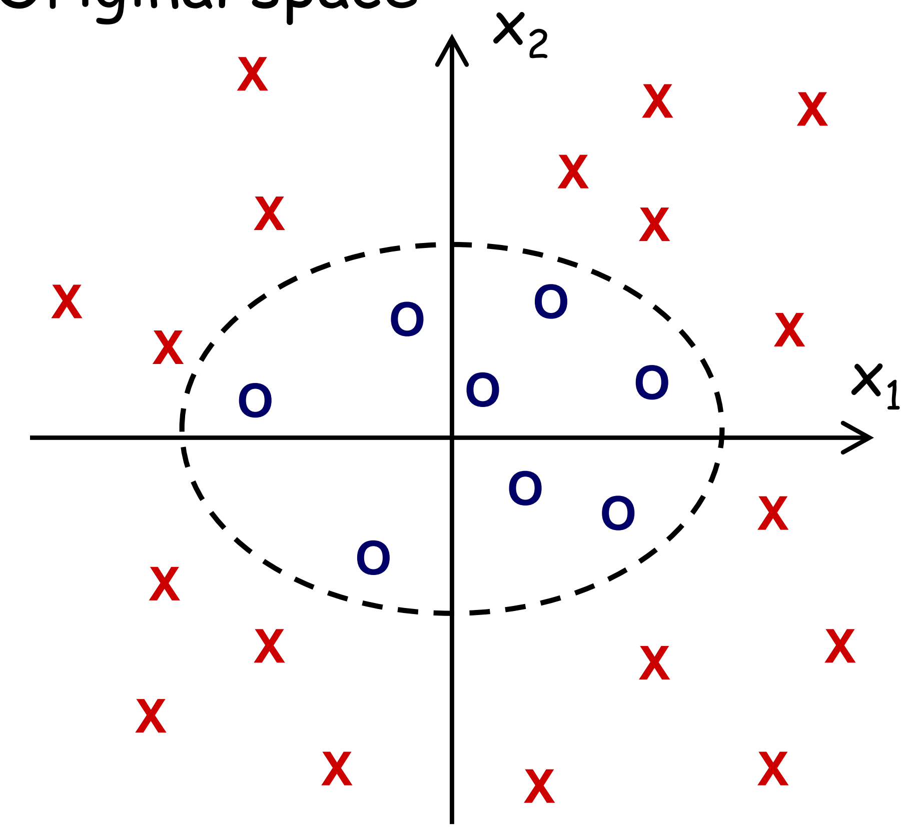
Example

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

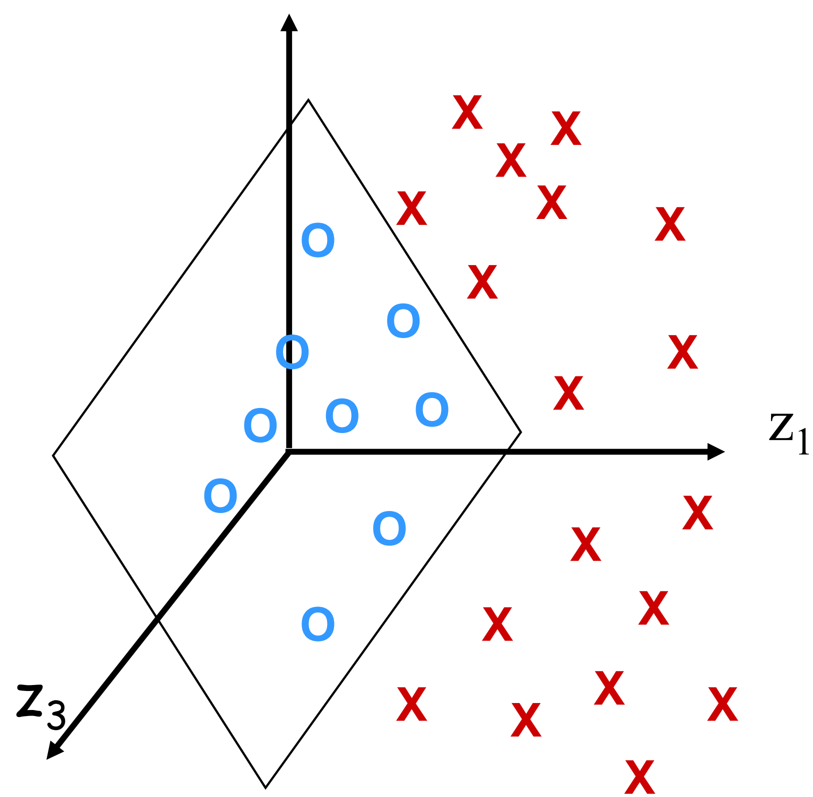
$\phi(x) \cdot \phi(z) = \left(x_1^2, x_2^2, \sqrt{2}x_1x_2\right) \cdot (z_1^2, z_2^2, \sqrt{2}z_1z_2)$

$= (x_1^2 z_1^2 + x_2^2 z_2^2 + 2x_1x_2z_1z_2) = (x_1z_1 + x_2z_2)^2 = (x \cdot z)^2 = K(x, z)$

Original space



Φ-space



Kernels

Definition

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

- $\exists\ \phi: X \rightarrow R^N$ s.t. $K(x, z) = \phi(x) \cdot \phi(z)$
    - Range of ϕ is called the Φ-space.
    - N can be very large. $(N = n^d)$

- But think of ϕ as implicit, not explicit!!!!

Example

Note: feature space might not be unique.

三维.

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

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

$$= (x_1z_1 + x_2z_2)^2 = (x \cdot z)^2 = K(x, z)$$

对同一个 kernel function
可能对应多个Φ.
但不用关心Φ. kernel 映射是
隐式的

四维

$$\phi: R^2 \to R^4, (x_1, x_2) \to \Phi(x) = (x_1^2, x_2^2, x_1x_2, x_2x_1)$$

$$\phi(x) \cdot \phi(z) = (x_1^2, x_2^2, x_1x_2, x_2x_1) \cdot (z_1^2, z_2^2, z_1z_2, z_2z_1)$$

$$= (x \cdot z)^2 = K(x, z)$$

Avoid explicitly expanding the features

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

Crucial to think of ϕ as implicit, not explicit!!!!

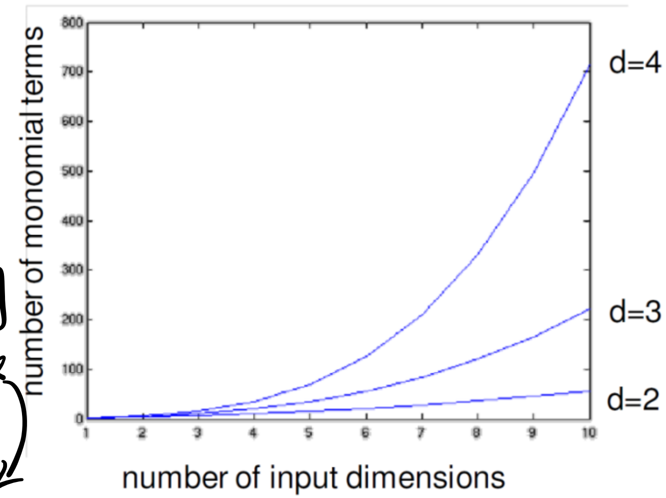
- Polynomial kernel degreee d, $k(x, z) = (x^\top z)^d = \phi(x) \cdot \phi(z)$
  - $x_1^d, x_1 x_2 \dots x_d, x_1^2 x_2 \dots x_{d-1}$
  - Total number of such feature is

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

→映射到空间的维度

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

所以要用kernel trick来计算　　或者说维度，即有多少项



$O(n)$ computation!

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

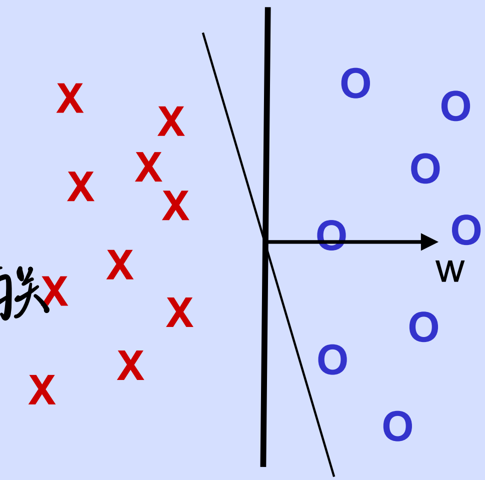
Kernelizing a learning algorithm

- **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 $x \cdot z$ with a $K(x, z)$.

- **Examples of kernalizable algos:**
  - classification: Perceptron, SVM.
  - regression: linear, ridge regression. Logistic Regression 也可以用
  - clustering: k-means.
    聚类.

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:
  → w_{final} 只和分类错误的点有关
  - 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 $w_t = a_{i_1} x_{i_1} + \cdots + a_{i_k} x_{i_k}$

→错误的点加权和

Replace $w_t \cdot x = \underline{a_{i_1} x_{i_1}} \cdot x + \cdots + \underline{a_{i_k} x_{i_k}} \cdot x$ with

$a_{i_1} K(x_{i_1}, x) + \cdots + a_{i_k} K(x_{i_k}, x)$ → 需要消耗更多的内存保存 $a_i x_i$
(原先只用存 w_t)

Note: need to store all the mistakes so far.

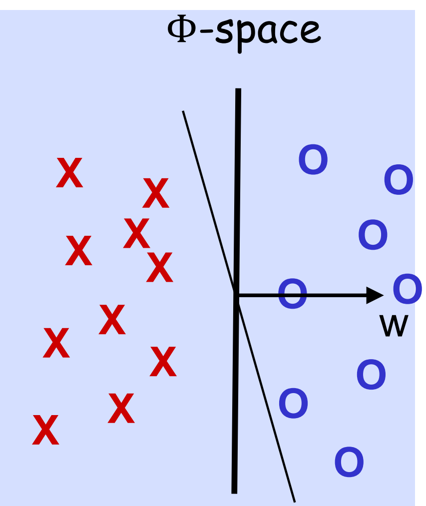
Kernelizing the Perceptron Algorithm

- Given x, predict + iff

$\phi(x_{i_{t-1}}) \cdot \phi(x)$

$$a_{i_1} K(x_{i_1}, x) + \cdots + a_{i_{t-1}} K(x_{i_{t-1}}, x) \geq 0$$

- 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 $w_t = a_{i_1} x_{i_1} + \cdots + a_{i_k} x_{i_k}$

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

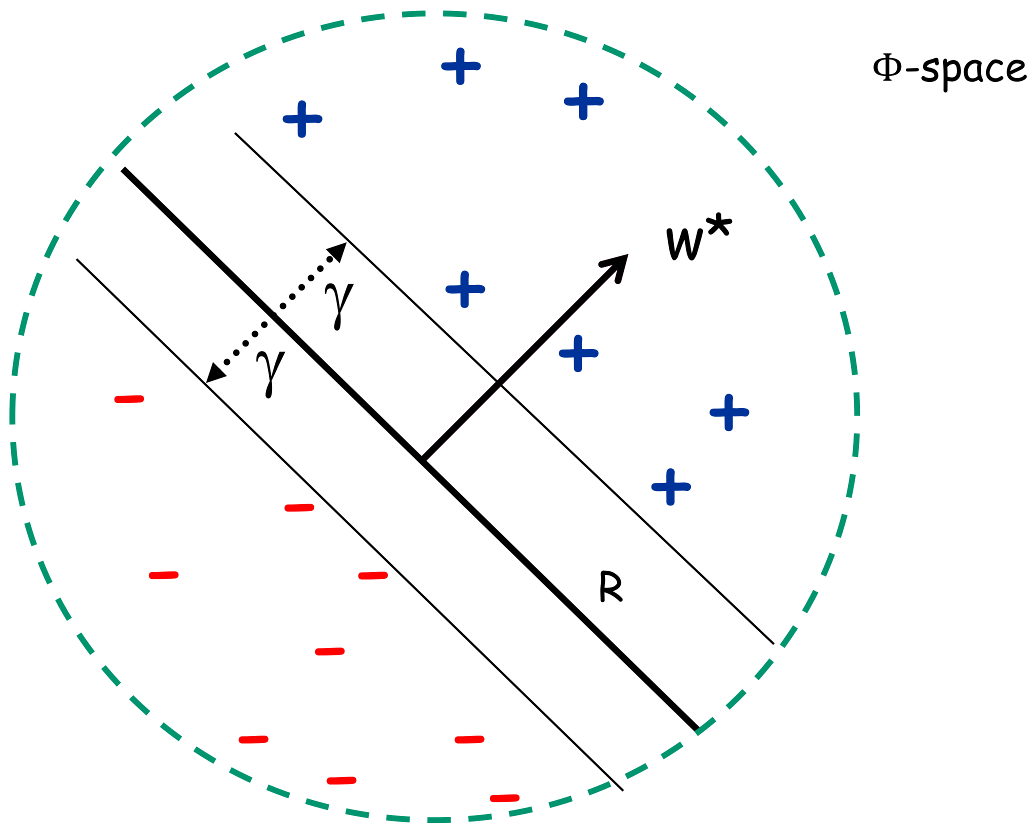
Exact same behavior/prediction rule as if mapped data in the ϕ-space and ran Perceptron there!

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. $R = \max \|\Phi(x_i)\|$
- If margin γ in ϕ-space then Perceptron makes $\left(\frac{R}{\gamma}\right)^2$ mistakes.

$$\gamma = \min_{x \in S} \frac{y_i w^{*T} \Phi(x_i)}{\|w^*\|}$$



Kernels: More Examples

- Linear: $K(x, z) = x \cdot z$ 线性核函数.

常用

- Polynomial: $K(x, z) = (x \cdot z)^d$ or $K(x, z) = (1 + x \cdot z)^d$
- Gaussian: $K(x, z) = \exp\left[-\frac{||x-z||^2}{2\,\sigma^2}\right]$ → 超参 (方差) $\sigma\uparrow$: bais↓ var↑ ; $\sigma\downarrow$: bais↑ var↓

  → $e^x = \sum_{j=1}^{\infty} \frac{x^j}{j!} = x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$ 是无穷维的.

- 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. 若两个输入不为vector时会有其它的kernel定义.

有m个样本: Gram matrix: $K: \mathbb{R}^{m\times m}$ 是对称矩阵 $K_{ij} = \overset{\text{kernel}}{k}(x_i, x_j)$

Properties of Kernels

Theorem (Mercer) → 判断 kernel function 是否合法.

$\mathbb{K}$ is a kernel if and only if:

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

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

kernel

$$a^T \mathbb{K} a \geq 0$$

kernel

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

是一个半正定矩阵

Kernel Methods

模块化：同个算法套不同kernel
对同个kernel套不同算法.

- 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!

$\Phi_1: \mathbb{R}^n \to \mathbb{R}^{N_1}$ $\Phi_2: \mathbb{R}^n \to \mathbb{R}^{N_2}$ $\Phi: \begin{bmatrix} \sqrt{c_1}\Phi_1 \\ \sqrt{c_2}\Phi_2 \end{bmatrix}: \mathbb{R}^n \to \mathbb{R}^{N_1+N_2}$

前提

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.

两个kernel的线性组合仍是一个kernel

Key idea: concatenate the ϕ spaces.

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

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

$K_1(x,z)$ $K_2(x,z)$

=> 可以设计新的kernel: $K = c_1 K_L + c_2 K_p + c_3 K_G$

Linear Polynomial Gaussian

多核学习 Multiple kernel Learning, MKL

Kernel, Closure Properties

Easily create new kernels using basic ones!

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

$\Phi: \mathbb{R}^n \to \mathbb{R}^{N_1 \times N_2}$ 成为一个矩阵. $\Phi_1 \times \Phi_2$ 外积.

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

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

$$\phi(x)\cdot\phi(z) = \sum_{i,j} \phi_{1,i}(x)\ \phi_{2,j}(x)\phi_{1,i}(z)\ \phi_{2,j}(z)$$

$$= \sum_i \phi_{1,i}(x)\ \phi_{1,i}(z)\left(\sum_j \phi_{2,j}(x)\ \phi_{2,j}(z)\right)$$

$$= \sum_i \phi_{1,i}(x)\ \phi_{1,i}(z)K_2(x,z) = K_1(x,z)\ K_2(x,z)$$

$\Rightarrow K = (c_1K_1 + c_2K_2)^d$ 也是一个 kernel

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 $K(x, z)$.

- 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 $\mathrm{x} \cdot \mathrm{z}$ with a $\mathrm{K}(\mathrm{x}, \mathrm{z})$.

How to choose a kernel:

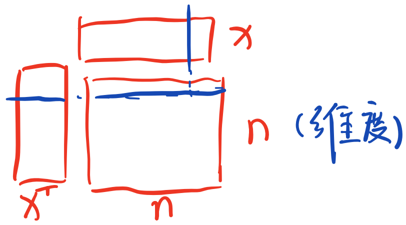
- Kernels often encode domain knowledge (e.g., string kernels)
- Use ~~Cross-Validation~~ to choose the parameters, e.g., σ for Gaussian Kernel $\mathrm{K}(\mathrm{x}, z) = \exp\left[-\frac{\left\|x-z\right\|^2}{2\,\sigma^2}\right]$
- Learn a good kernel; e.g., [Lanckriet-Cristianini-Bartlett-El Ghaoui-Jordan'04]

如何给线性回归加 kernel：

$$\min_{\beta}\left(\frac{1}{2}\|y-X\beta\|_2^2+\frac{\lambda}{2}\|\beta\|_2^2\right) \Rightarrow \beta=(X^TX+\lambda I)^{-1}X^Ty$$

$\rightarrow O(n^3)$

要使用kernel要找内积. 但 X^TX 是 x_i, x_j 的外积.



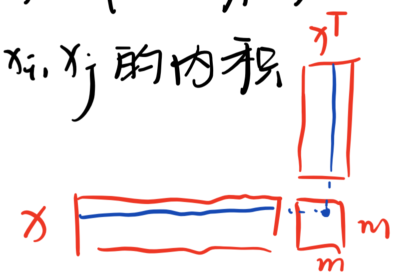
对偶变量

$$\frac{\partial l}{\partial \beta} = -X^T(y-X\beta)+\lambda\beta=0 \Rightarrow \beta=\frac{1}{\lambda}X^T(y-X\beta)=X^T\underline{a}$$

$$a=\frac{1}{\lambda}(y-X\beta)=\frac{1}{\lambda}(y-XX^Ta) \Rightarrow a=\left(\frac{1}{\lambda}I+XX^T\right)\frac{1}{\lambda}y$$

而 XX^T 是 x_i, x_j 的内积

出内积后即可应用核函数进行升维.



$\rightarrow$ 第i行第j列即为 $\langle x_i, x_j\rangle$

$$\Rightarrow \beta = X^Ta = \sum_{i=1}^{m} x_i a_i$$

数据量

考虑 perceptron 形式 $w_t=\sum_{i=1}^{m} a_i x_i$

可以为任意回归或分析最终表达为 $\sum x_i a_i$ Represent Theorem.