Kernel methods

Shikui Tu

Department of Computer Science and Engineering, Shanghai Jiao Tong University

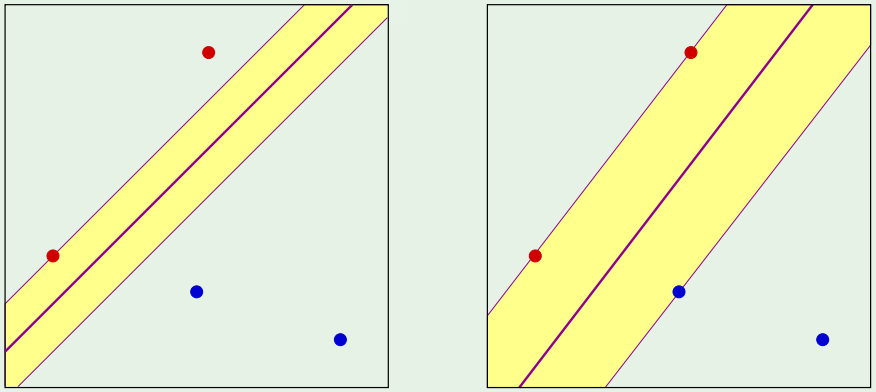
2019-04-23

Outline

- **Review of SVM in the previous lecture**
- A bit of history of kernel methods and SVM
- Kernel methods
- Kernel PCA
- Designing kernels

Maximize margin

- The margin

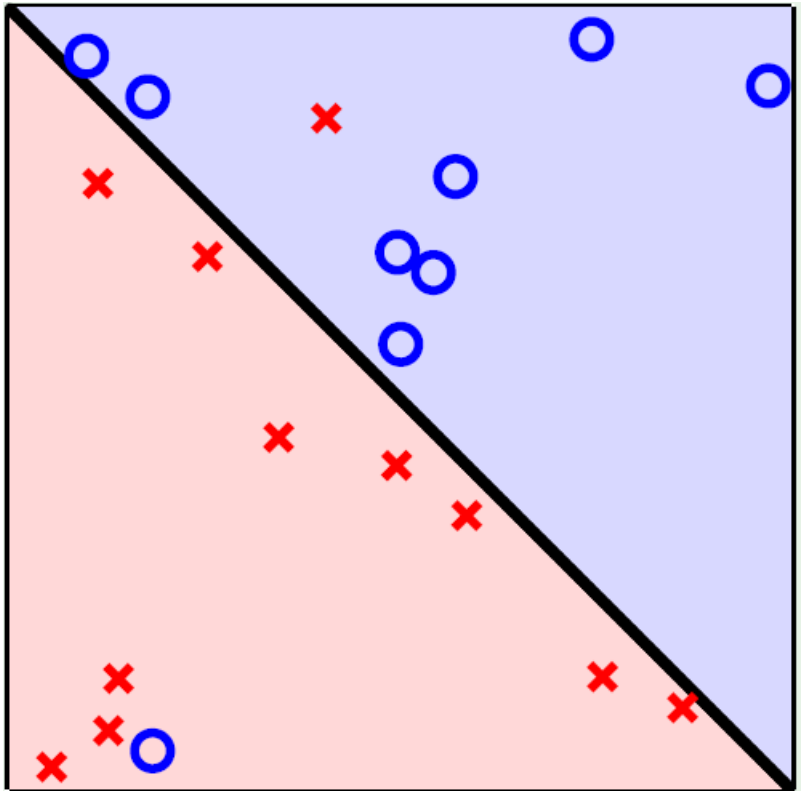


Maximizing the margin $\Longrightarrow$ dual problem:

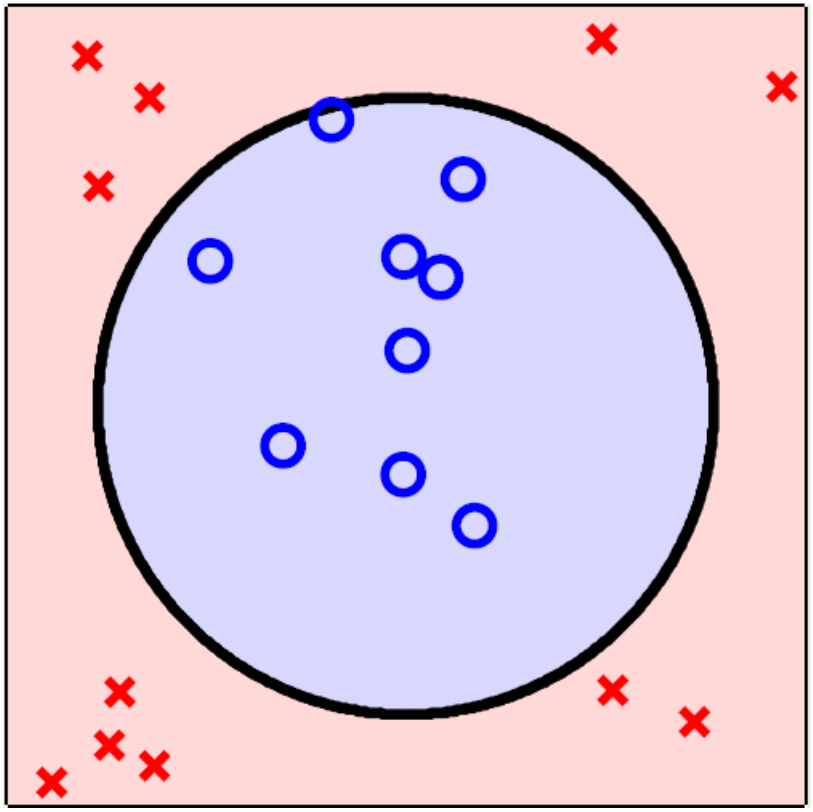
$$\mathcal{L}(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \, \alpha_n \alpha_m \, \mathbf{x}_n^{\mathsf{T}} \mathbf{x}_m$$

quadratic programming

Non-separable Data



Slightly
non-separable

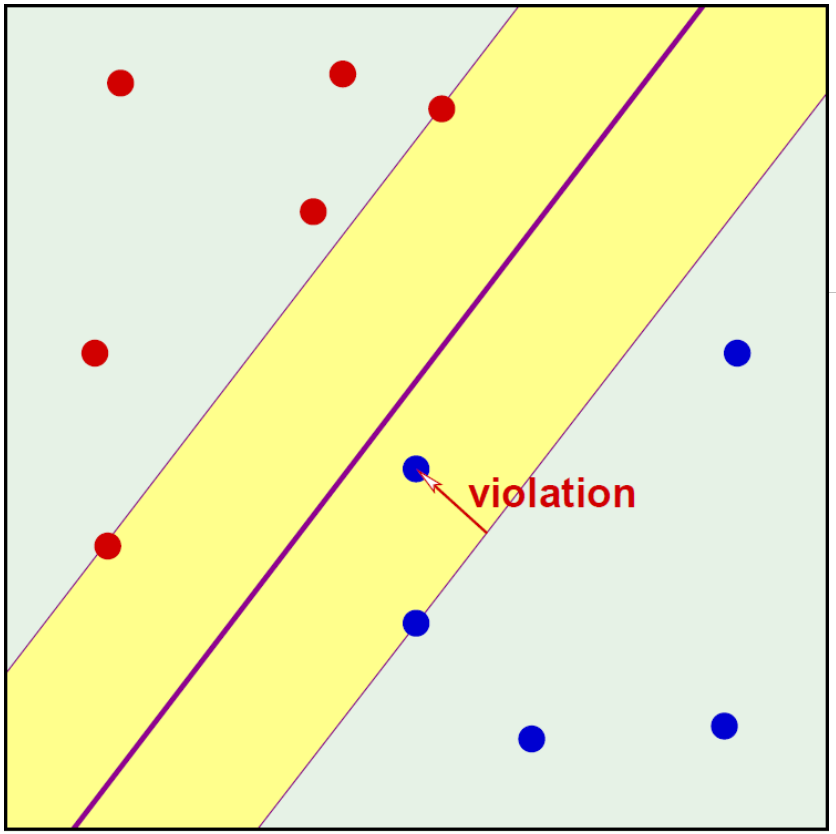


Seriously
non-separable

Soft-margin SVM

Margin violation: $y_n \left(\mathbf{w}^\intercal \mathbf{x}_n + b\right) \geq 1$ fails

Quantify: $y_n \left(\mathbf{w}^\intercal \mathbf{x}_n + b\right) \geq 1 - \xi_n$



$$\text{Total violation} = \sum_{n=1}^{N} \xi_n$$

ξ_n = ksi = "amount" of margin violation

$$\xi_n \geq 0$$

Soft-margin SVM optimization

- Similar to hard-margin optimization, but get compromise between maximizing the margin and allowing for violations

Still get large margin after term minimization

Allow for small violations by minimizing

$$\text{Minimize} \quad \frac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w} + C\sum_{n=1}^{N}\xi_n$$

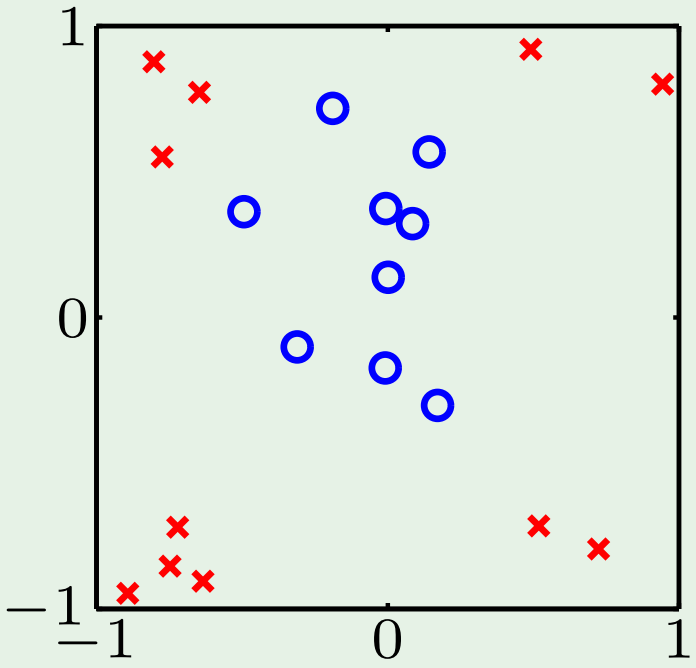
$$\text{subject to} \quad y_n\left(\mathbf{w}^{\mathsf{T}}\mathbf{x}_n + b\right) \geq 1 - \xi_n$$

$$\text{for} \quad n = 1, \ldots, N$$

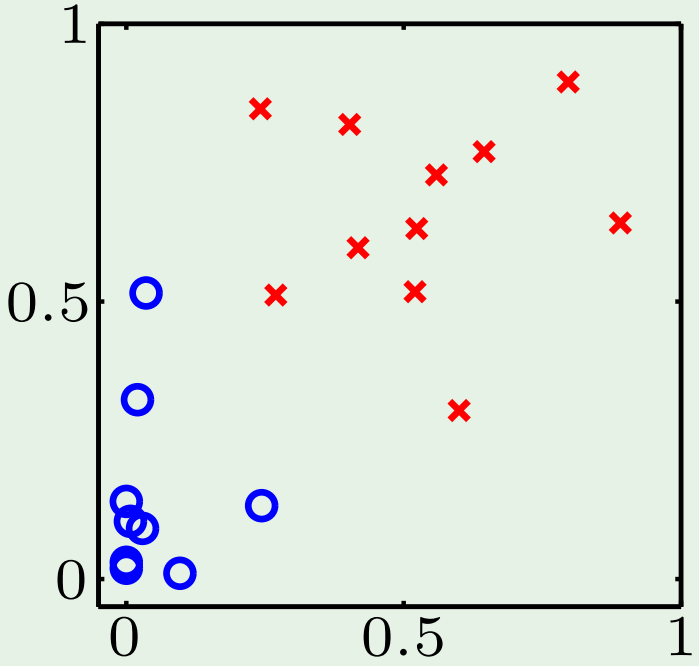
$$\text{and} \quad \xi_n \geq 0 \quad \text{for} \quad n = 1, \ldots, N$$

Kernel transform

$$\mathcal{L}(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \, \alpha_n \alpha_m \, \mathbf{z}_n^{\mathsf{T}} \mathbf{z}_m$$



$\mathcal{X} \longrightarrow \mathcal{Z}$



Outline

- Review of SVM in the previous lecture

- **A bit of history of kernel methods and SVM**

- Kernel methods

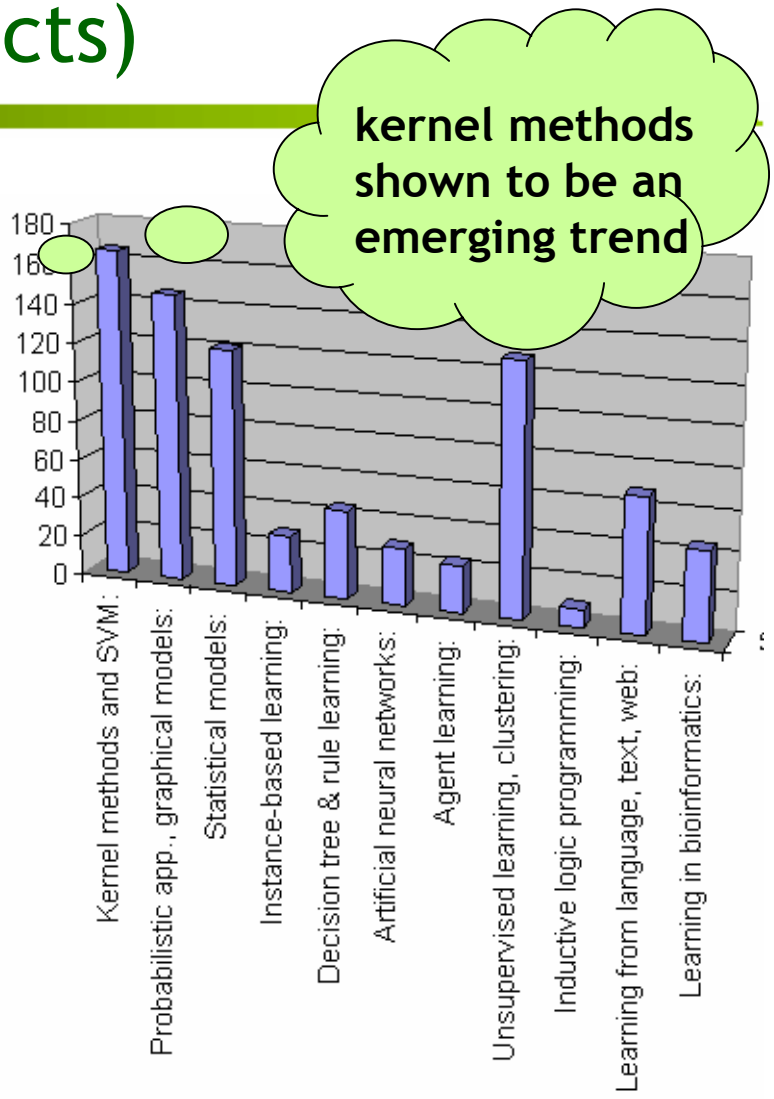
- Kernel PCA

- Designing kernels

Very popular around 2006

ICML 2006 (720 abstracts)

Kernel methods & SVM	**166**
Probabilistic, graphical models	146
Unsupervised learning, clustering	128
Statistical models	121
Language, Text & web	68
Learning in bioinformatics	45
ANN	29
ILP	9
CRF	13



A bit of history

- Aronszajn (1950) and Parzen (1962) were some of the first to employ positive definite kernels in statistics.
- Aizerman et al. (1964) used positive definite kernels in a way closer to the kernel trick.
- Boser et al. (1992) constructed the SVMs, a generalization of optimal hyperplane algorithm worked for vectorial data.
- Scholkopf (1997): kernels can work with nonvectorial data by providing a vectorial representation of the data in the feature space.
- Scholkopf et al. (1998): kernels can be used to build generalizations of any algorithm that can be carried out in terms of dot products.
- A large number of "kernelizations" of various algorithms since last 5 years.

Computation from the Z space

$$\mathcal{L}(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \, \alpha_n \alpha_m \, \mathbf{z}_n^{\mathsf{T}} \mathbf{z}_m$$

Constraints: $\alpha_n \geq 0$ for $n = 1, \cdots, N$ and $\sum_{n=1}^{N} \alpha_n y_n = 0$

$$\boxed{g(\mathbf{x}) = \text{sign}\left(\mathbf{w}^{\mathsf{T}} \mathbf{z} + b\right)} \qquad \text{need } \mathbf{z}_n^{\mathsf{T}} \mathbf{z}$$

where $\mathbf{w} = \sum_{\mathbf{z}_n \text{ is SV}} \alpha_n y_n \mathbf{z}_n$

and b: $y_m \left(\mathbf{w}^{\mathsf{T}} \mathbf{z}_m + b\right) = 1$ need $\mathbf{z}_n^{\mathsf{T}} \mathbf{z}_m$

Generalized inner product

Given two points $\mathbf{x}$ and $\mathbf{x}' \in \mathcal{X}$, we need $\mathbf{z}^\mathsf{T}\mathbf{z}'$

Let $\mathbf{z}^\mathsf{T}\mathbf{z}' = K(\mathbf{x}, \mathbf{x}')$ (the kernel) "inner product" of $\mathbf{x}$ and $\mathbf{x}'$

Second order transform:

Example: $\mathbf{x} = (x_1, x_2) \longrightarrow$ 2nd-order Φ

$$\mathbf{z} = \Phi(\mathbf{x}) = (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)$$

$$K(\mathbf{x}, \mathbf{x}') = \mathbf{z}^\mathsf{T}\mathbf{z}' = 1 + x_1 x'_1 + x_2 x'_2 + x_1^2 x'^2_1 + x_2^2 x'^2_2 + x_1 x'_1 x_2 x'_2$$

Without transforming explicitly

- Can we compute $K(\boldsymbol{x}, \boldsymbol{x}')$ without transforming $\boldsymbol{x}$ and $\boldsymbol{x}'$?

Example:

$$K(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^\top \mathbf{x}')^2 = (1 + x_1 x'_1 + x_2 x'_2)^2$$

$$= 1 + x_1^2 x'^2_1 + x_2^2 x'^2_2 + 2x_1 x'_1 + 2x_2 x'_2 + 2x_1 x'_1 x_2 x'_2$$

Inner product

$$(\ 1\ ,\ x_1^2\ ,\ x_2^2\ ,\ \sqrt{2}\,x_1\ ,\ \sqrt{2}\,x_2\ ,\ \sqrt{2}\,x_1 x_2\ )$$

$$(\ 1\ ,\ x'^2_1\ ,\ x'^2_2\ ,\ \sqrt{2}x'_1\ ,\ \sqrt{2}x'_2\ ,\ \sqrt{2}x'_1 x'_2\ )$$

The polynomial kernel

$\mathcal{X} = \mathbb{R}^d$ and $\Phi : \mathcal{X} \to \mathcal{Z}$ is polynomial of order Q

The "equivalent" kernel $K(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^\intercal \mathbf{x}')^Q$

$$= (1 + x_1 x'_1 + x_2 x'_2 + \cdots + x_d x'_d)^Q$$

Adjust the scales:

$$K(\mathbf{x}, \mathbf{x}') = (a\mathbf{x}^\intercal \mathbf{x}' + b)^Q$$

We only need Z to exist

- If $K(\boldsymbol{x}, \boldsymbol{x}')$ is an inner product in some space Z, that is ok.
- For example,

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2\right)$$

Infinite-dimensional $\mathcal{Z}$: take simple case

$$K(x, x') = \exp\left(-(x - x')^2\right)$$

$$= \exp\left(-x^2\right) \exp\left(-x'^2\right) \underbrace{\sum_{k=0}^{\infty} \frac{2^k (x)^k (x')^k}{k!}}_{\exp(2xx')}$$

- How do we know whether Z exists or not?
  1. By construction
  2. Math properties (Mercer's condition)
  3. Who cares?

Kernel SVM

$$\min_{\boldsymbol{\alpha}} \quad \frac{1}{2}\boldsymbol{\alpha}^{\mathsf{T}} \underbrace{\begin{bmatrix} y_1 y_1\, \mathbf{x}_1^{\mathsf{T}}\mathbf{x}_1 & y_1 y_2\, \mathbf{x}_1^{\mathsf{T}}\mathbf{x}_2 & \dots & y_1 y_N\, \mathbf{x}_1^{\mathsf{T}}\mathbf{x}_N \\ y_2 y_1\, \mathbf{x}_2^{\mathsf{T}}\mathbf{x}_1 & y_2 y_2\, \mathbf{x}_2^{\mathsf{T}}\mathbf{x}_2 & \dots & y_2 y_N\, \mathbf{x}_2^{\mathsf{T}}\mathbf{x}_N \\ \dots & \dots & \dots & \dots \\ y_N y_1\, \mathbf{x}_N^{\mathsf{T}}\mathbf{x}_1 & y_N y_2\, \mathbf{x}_N^{\mathsf{T}}\mathbf{x}_2 & \dots & y_N y_N\, \mathbf{x}_N^{\mathsf{T}}\mathbf{x}_N \end{bmatrix}}_{\text{quadratic coefficients}} \boldsymbol{\alpha} + \underbrace{(-\mathbf{1}^{\mathsf{T}})}_{\text{linear}}\boldsymbol{\alpha}$$

subject to $\underbrace{\mathbf{y}^{\mathsf{T}}\boldsymbol{\alpha} = 0}_{\text{linear constraint}}$

$$\underbrace{\mathbf{0}}_{\text{lower bounds}} \leq \boldsymbol{\alpha} \leq \underbrace{\infty}_{\text{upper bounds}}$$

Kernel formulation

$$\underbrace{\begin{bmatrix} y_1 y_1\, K(\mathbf{x}_1, \mathbf{x}_1) & y_1 y_2\, K(\mathbf{x}_1, \mathbf{x}_2) & \dots & y_1 y_N K(\mathbf{x}_1, \mathbf{x}_N) \\ y_2 y_1\, K(\mathbf{x}_2, \mathbf{x}_1) & y_2 y_2\, K(\mathbf{x}_2, \mathbf{x}_2) & \dots & y_2 y_N K(\mathbf{x}_2, \mathbf{x}_N) \\ \dots & \dots & \dots & \dots \\ y_N y_1 K(\mathbf{x}_N, \mathbf{x}_1) & y_N y_2 K(\mathbf{x}_N, \mathbf{x}_2) & \dots & y_N y_N K(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix}}_{\text{quadratic coefficients}}$$

Solution to kernel SVM

Express $g(\mathbf{x}) = \text{sign}\left(\mathbf{w}^{\mathsf{T}}\mathbf{z} + b\right)$ in terms of $K(-,-)$

$$\mathbf{w} = \sum_{\mathbf{z}_n \text{ is SV}} \alpha_n y_n \mathbf{z}_n \quad \Longrightarrow \quad g(\mathbf{x}) = \text{sign}\left(\sum_{\alpha_n > 0} \alpha_n y_n K(\mathbf{x}_n, \mathbf{x}) + b\right)$$

where $b = y_m - \sum_{\alpha_n > 0} \alpha_n y_n K(\mathbf{x}_n, \mathbf{x}_m)$

for any support vector $(\alpha_m > 0)$

Mercer's condition

$K(\mathbf{x}, \mathbf{x}')$ is a valid kernel iff

1. It is symmetric and 2. The matrix:

$$\begin{bmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & K(\mathbf{x}_1, \mathbf{x}_2) & \dots & K(\mathbf{x}_1, \mathbf{x}_N) \\ K(\mathbf{x}_2, \mathbf{x}_1) & K(\mathbf{x}_2, \mathbf{x}_2) & \dots & K(\mathbf{x}_2, \mathbf{x}_N) \\ \dots & \dots & \dots & \dots \\ K(\mathbf{x}_N, \mathbf{x}_1) & K(\mathbf{x}_N, \mathbf{x}_2) & \dots & K(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix}$$

is **positive semi-definite**

for any $\mathbf{x}_1, \cdots, \mathbf{x}_N$ (Mercer's condition)

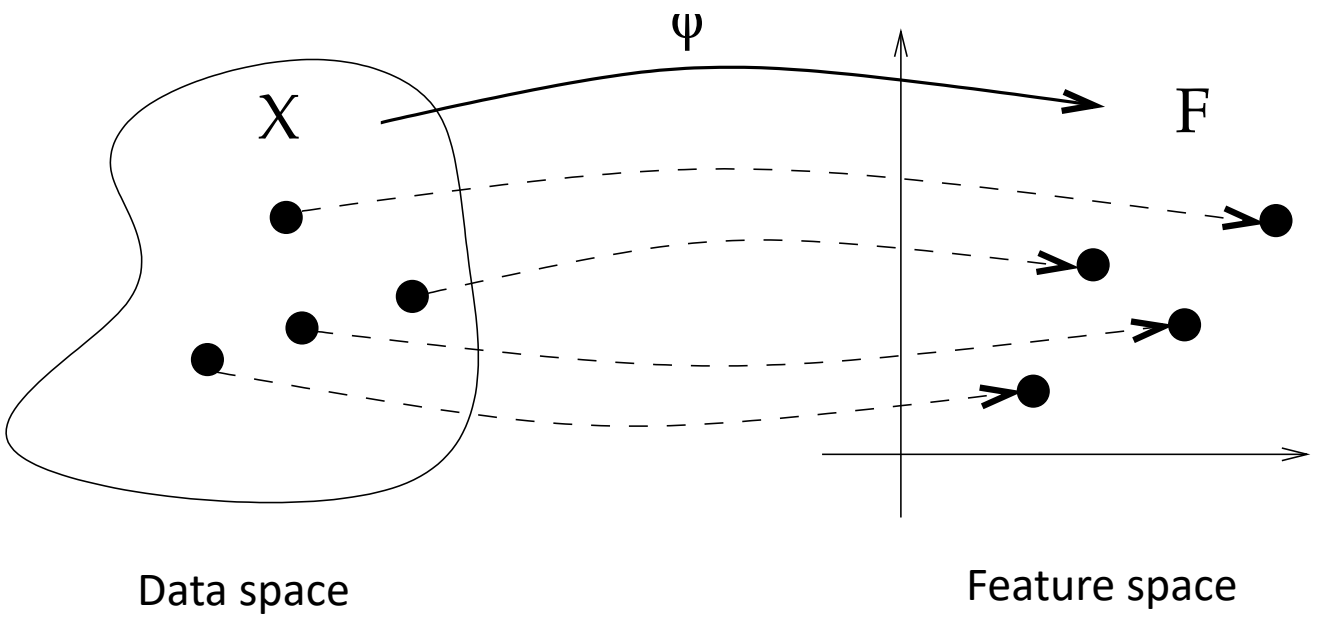
Then, you can design your own kernel as above.

Outline

- Review of SVM in the previous lecture
- A bit of history of kernel methods and SVM
- **Kernel methods**
- Kernel PCA
- Designing kernels

Kernels

- a general framework to represent data and must satisfy some math conditions that give them properties useful to understand kernel methods and kernel design



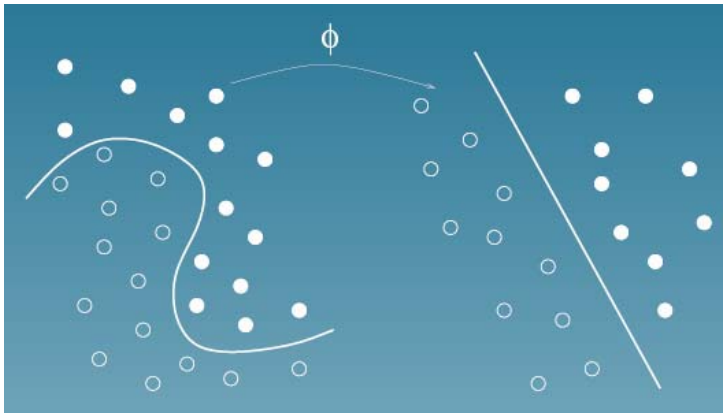
The issue of data representation

- Let $\mathbb{S} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ a set of *n* objects to be analyzed.
- Suppose that each object $\mathbf{x}_i$ is an element of a set $\mathbb{X}$, which may be images, molecules, texts, etc.
- Majority of data analysis methods represent $\mathbb{S}$ by:
  - defining a representation $\phi(\mathbf{x}) \in \mathbb{F}$ for each object $\mathbf{x} \in \mathbb{X}$, where $\phi(\mathbf{x})$ can be a real-valued vector ($\mathbb{F} = \Re^p$) or a finite-length string, or more complex representation.
  - representing $\mathbb{S}$ by a set of representations of the objects

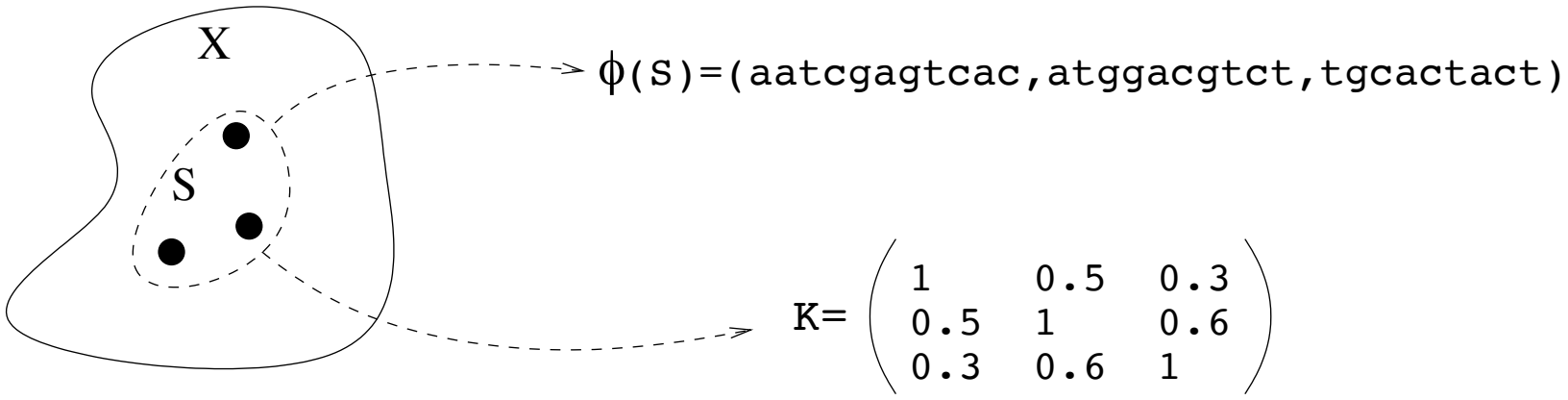
$$\phi(\mathbb{S}) = (\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n))$$

The idea of kernel representation

- Data are not represented individually anymore, but only through a set of pairwise comparisons.
- Instead of using a mapping $\phi: \mathcal{X} \to \mathcal{F}$ to represent each object $\mathbf{x} \in \mathcal{X}$ by $\phi(\mathbf{x}) \in \mathcal{F}$, a real-valued "comparison function" (called kernel) $k: \mathcal{X} \times \mathcal{X} \to \Re$ is used, and the data set $\mathcal{S}$ is represented by the nxn matrix (Gram matrix) of pairwise comparisons $k_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$.
- A main question is how to find a kernel k such that, in the new space, problem solving is easier (e.g. linear).
- All kernel methods have two parts:
  (1) find such a kernel k, and
  (2) process such Gram matrices.

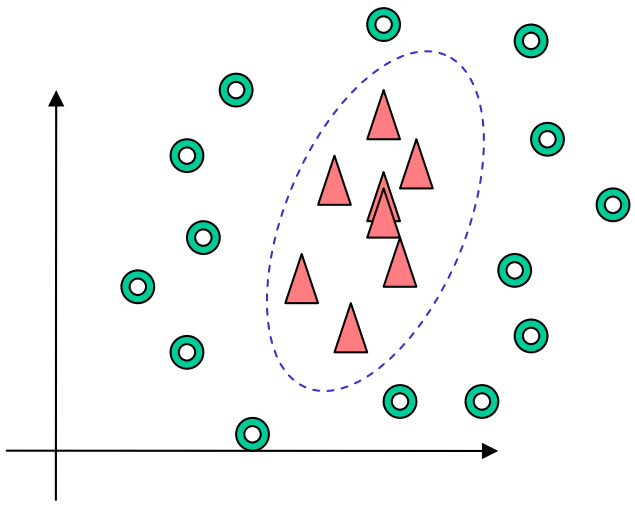


Kernel representation: example



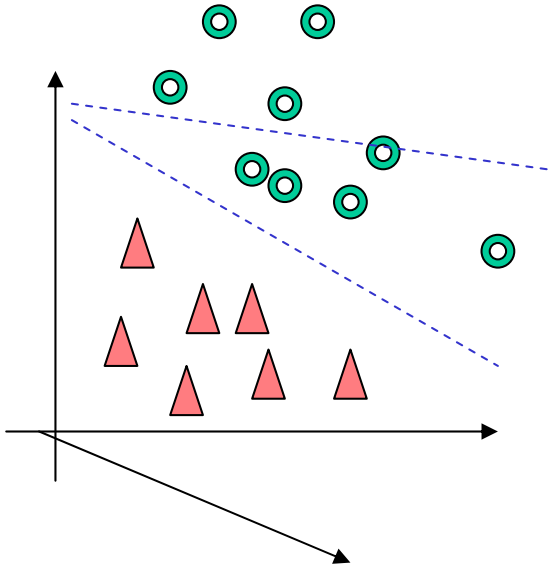
- X is the set of all oligonucleotides, S consists of three oligonucleotides.
- Traditionally, each oligonucleotide is represented by a sequence of letters.
- In kernel methods, S is represented as a matrix of pairwise similarity between its elements.

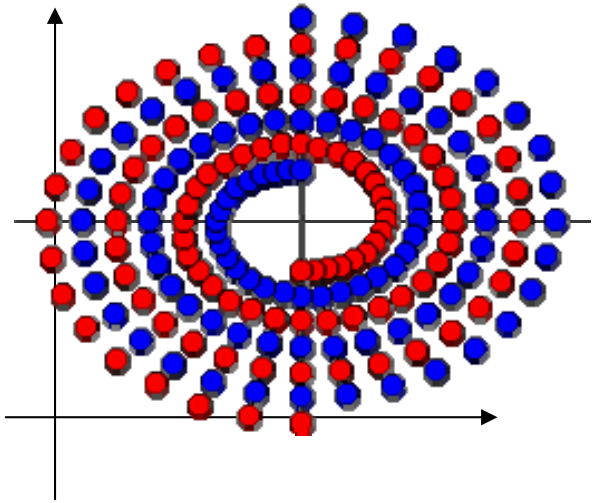
Examples of kernels



by polynomial kernel (n=2)

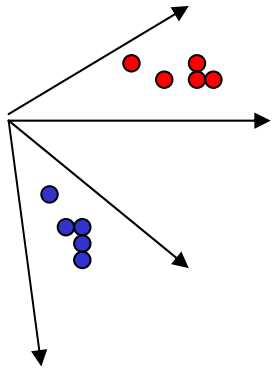
$$\phi : (x_1, x_2) \mapsto (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$





by RBF kernel (n=2)

$$\phi : (x_1, x_2) \mapsto \exp\left(-\frac{d(x_1, x_2)^2}{2\sigma^2}\right)$$



(Jean-Michel Renders, ICFT'04)

Dot product

- A vector space (linear space) $\mathcal{H}$ of "vectors" over the reals $\mathfrak{R}$ if two operations,
  - *vector addition*: $\mathcal{H} \times \mathcal{H} \rightarrow \mathcal{H}$ denoted **v** + **w**, where **v**, **w** $\in \mathcal{H}$, and
  - *scalar multiplication*: $\mathfrak{R} \times \mathcal{H} \rightarrow \mathcal{H}$ denoted λ**v**, where $\lambda \in \mathfrak{R}$ and **v** $\in \mathcal{H}$

  such that some axioms are satisfied (http://www.answers.com/topic/vector-space).

- Dot product (inner, scalar product) on $\mathcal{H}$ is a symmetric bilinear form $\langle .,. \rangle$: $\mathcal{H}$ x $\mathcal{H} \rightarrow \mathfrak{R}$, **(x, x')** $\mapsto \langle$**x, x'**$\rangle$ that is strictly positive definite, i.e., $\forall$**x** $\in \mathcal{H}$, $\langle$**x, x**$\rangle \geq 0$ with equality only for **x** = 0.

- Norm $\| \mathbf{x} \| := \langle \mathbf{x}, \mathbf{x} \rangle^{1/2}$ ($\| . \| : \mathcal{H} \rightarrow \mathfrak{R}^{+}$), p-norm: $\|x\|_p := \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}}$

- An dot product space (inner product space) $\mathcal{H}$ is a vector space endowed with a dot product. The dot product space allows us to introduce geometrical notions such as angles and lengths of vectors:

$$\cos\theta = \frac{\langle x, x' \rangle}{\|x\| \ \|x'\|}$$

Hilbert space

- A Hilbert space $\mathcal{H}$ is a dot product space with the additional properties that it is *separable* and *complete*.

  Completeness means every Cauchy sequence $\{h_n\}_{n\geq 1}$ of elements of $\mathcal{H}$ converges to an element $h \in \mathcal{H}$, where a Cauchy sequence is one satisfying

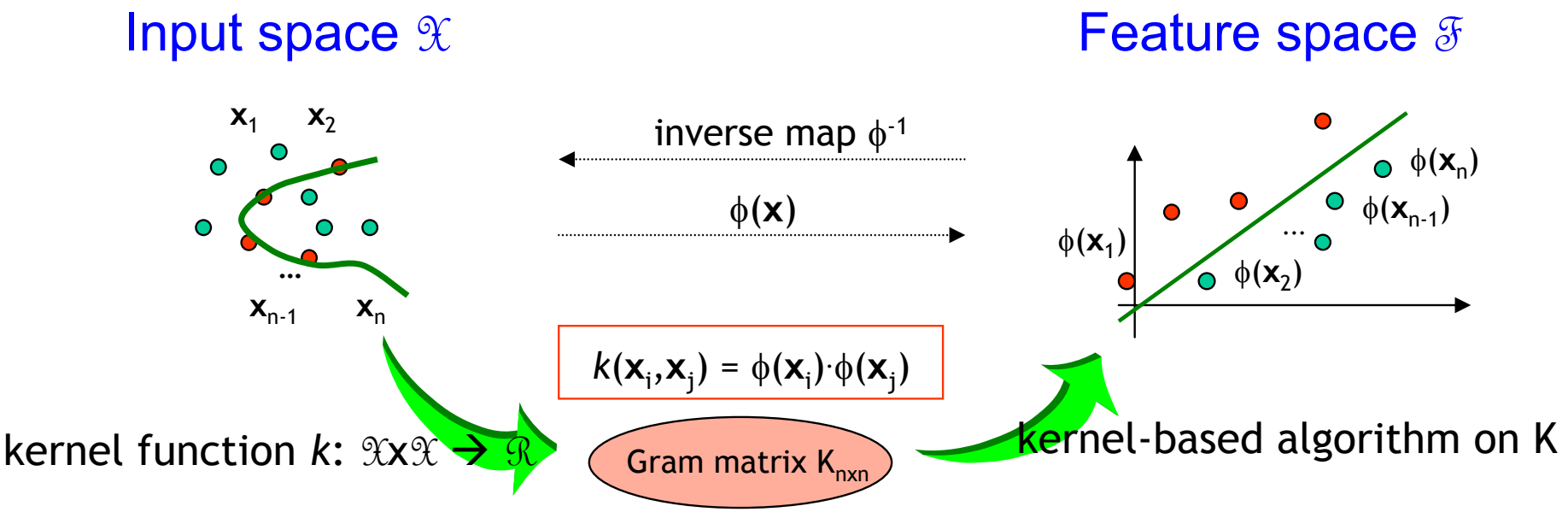
$$\sup_{m>n} \|h_n - h_m\| \to 0, \text{ as } n \to \infty$$

  $\mathcal{H}$ is separable if for any $\varepsilon > 0$ there is a finite set of elements $h_1, \ldots, h_N$ of $\mathcal{H}$ such that for all $h \in \mathcal{H}$,

$$\min_i \|h_i - h\| < e$$

- **Importance**: As all elements of a Hilbert space $\mathcal{H}$ are linear functions in $\mathcal{H}$ via the dot product: for a point $\mathbf{z}$ the corresponding function is $f_{\mathbf{z}}(\mathbf{x}) = \langle \mathbf{z}, \mathbf{x} \rangle$. Our target is to learn linear functions represented by a weight vector in the feature space. Finding the weight vector is therefore equivalent to identifying an appropriate point of the feature space.

General scheme of kernel methods



- Map the data from $\mathcal{X}$ into a (high-dimensional) vector space, the feature space $\mathcal{F}$, by applying the feature map ϕ on the data points **x**.
- Find a linear (or other easy) pattern in $\mathcal{F}$ using a well-known algorithm (that works on the Gram matrix).
- By applying the inverse map, the linear pattern in $\mathcal{F}$ can be found to correspond to a complex pattern in $\mathcal{X}$.
- This implicitly by only making use of inner products in $\mathcal{F}$ (kernel trick)

Valid kernels

- Most kernel methods can only process symmetric positive definite matrix: $k_{i,j} = k_{j,i}$ for all i, j and $\boldsymbol{c}^T k \boldsymbol{c} \geq 0$, for any $\boldsymbol{c} \in \Re^p$.

- **Definition 1**: *A function k:* $\mathcal{X} \times \mathcal{X} \rightarrow \Re$ *is called a positive definite kernel if it is symmetric (that is,* $k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}', \boldsymbol{x}) \ \forall \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$*), and positive definite, that is for any* $n > 0$*, any choice of n objects* $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathcal{X}$*, and any choice of real number* $c_1, \ldots, c_n \in \Re$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0 \quad (1)$$

- A positive definite kernel *k* is called a valid kernel (or simply kernel).
- **Mercer's theorem**: Any positive definite function can be written as an inner product in some feature space.

Kernels as inner product (1/2)

- Suppose $\mathcal{X} = \Re^p$ and $\mathbf{x} = (x_1, \ldots, x_p)^T$. Comparing such vectors by inner product, for any $\mathbf{x}, \mathbf{x}' \in \Re^p$,

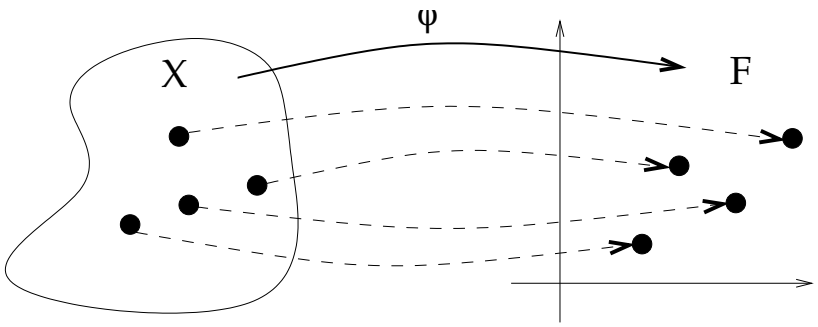
$$k_L(\mathbf{x},\mathbf{x}') := x^T x' = \sum_{i=1}^{p} x_i x_i' = \langle x, x' \rangle$$

- This function is a kernel as it is symmetric and positive definite, usually called *linear kernel*.

- One systematic way to define kernels for general objects:
  1. representing each object $\mathbf{x} \in \mathcal{X}$ as a vector $\phi(\mathbf{x}) \in \Re^p$
  2. defining a kernel for any $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ as inner product $k(\mathbf{x},\mathbf{x}') = \phi(\mathbf{x})^T\phi(\mathbf{x}')$

Can define kernel without requiring $\mathcal{X}$ to be a vector space.

Kernels as inner product (2/2)



Any kernel on a space $\mathcal{X}$ can be represented as an inner product after the space $\mathcal{X}$ is mapped to a Hilbert space $\mathcal{F}$, called feature space.

- Kernels can all be thought of as dot products in some space $\mathcal{F}$, called the *feature space*. Using a kernel boils down to representing each object $\mathbf{x} \in \mathcal{X}$ as a vector $\phi(\mathbf{x}) \in \mathcal{F}$, and computing dot products.
- We don't need to compute explicitly $\phi(\mathbf{x})$ for each point in $\mathcal{S}$, but only the pairwise dot products. $\mathcal{F}$ usually is infinite-dimensional, and $\phi(\mathbf{x})$ is tricky to represent though the kernel is simple to compute.
- Most kernel methods possess such an interpretation when the points $\mathbf{x} \in \mathcal{X}$ are viewed as points $\phi(\mathbf{x})$ in the feature space.

The representer theorem

- **Theorem** (Kimeldoft, Wahba, 1971): *If the problem*

$$\min_{f \in \mathcal{H}} \{C(f, \{x_i, y_i\}) + \Omega(\|f\|_{\mathcal{H}}) \qquad (5)$$

satisfies (1) C is pointwise; i.e., C(f,{x_i, y_i}) = C({x_i, y_i, f(x_i)}) which only depends on {f(x_i)}) and (2) Ω(.) is monotonically increasing, then every minimizer of the problem admits a representation of the form

$$f(.) = \sum_{i=1}^{m} \alpha_i k(x_i, .) \qquad (6)$$

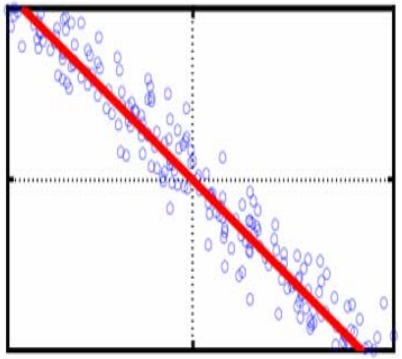
- The quantity $\|.\|_{\mathcal{H}_k}$ included in the function to optimize is a penalization that forces the solution to be smooth and usually a powerful protection against over-fitting of the data.
- Meaning: Instead of finding optimal solution in infinite-dimensional space, (5) can be reformulated as a *n*-dimensional optimization problem, by plugging (6) into (5) and optimizing over $(\alpha_1, \ldots, \alpha_n) \in \mathfrak{R}^p$.

Outline

- Review of SVM in the previous lecture
- A bit of history of kernel methods and SVM
- Kernel methods
- **Kernel PCA**
- Designing kernels

Principal component analysis (PCA)

- **PCA**: Given $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^p$. Find: $\mathbf{x'}_1, \ldots, \mathbf{x'}_m \in \mathbb{R}^d$, d << p so that $\mathbf{x'}_1, \ldots, \mathbf{x'}_m$ preserve most information.



- PCA finds the principal axes by diagonalizing the covariance matrix C with singular value decomposition

$$\lambda \upsilon = C\upsilon \qquad C = \frac{1}{m}\sum_{j=1}^{m} x_j x_j^T$$

$$\lambda\upsilon = C\upsilon = \frac{1}{m}\sum x_j x_j^T \upsilon$$

$$\upsilon = \frac{1}{m\lambda}\sum x_j x_j^T \upsilon = \frac{1}{m\lambda}\sum (x_j . \upsilon) x_j$$

- All solutions v with $\lambda \neq 0$ lie in the span of $x_1,..,x_m,$,i.e.

$$\upsilon = \sum_i \alpha_i x_i$$

1. Find eigenvectors, order them in decreasing.
2. Projects points onto eigenvectors.
3. Use new coordinates to do things

Kernel trick

- Do PCA in feature space? The covariance matrix

$$\overline{C} = \frac{1}{m}\sum_{j=1}^{m}\phi(x_j)\phi(x_j)^T$$

- can be diagonalized with nonnegative eigenvalues satisfying

$$\lambda V = \overline{C}V$$

- As *V* lie in the span of $\phi(x_i)$, so we have

$$\lambda\sum\alpha_i\phi(x_i) = \frac{1}{m}\sum\phi(x_j)\phi(x_j)^T.\sum\alpha_i\phi(x_i) = \frac{1}{m}\sum\sum\alpha_i\phi(x_j)\phi(x_j)^T\phi(x_i)$$

$$= \frac{1}{m}\sum_j\sum_i \alpha_i\phi(x_i)\left\langle\phi(x_i),\phi(x_j)\right\rangle$$

Kernel PCA

- Apply kernel trick, we have $K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$

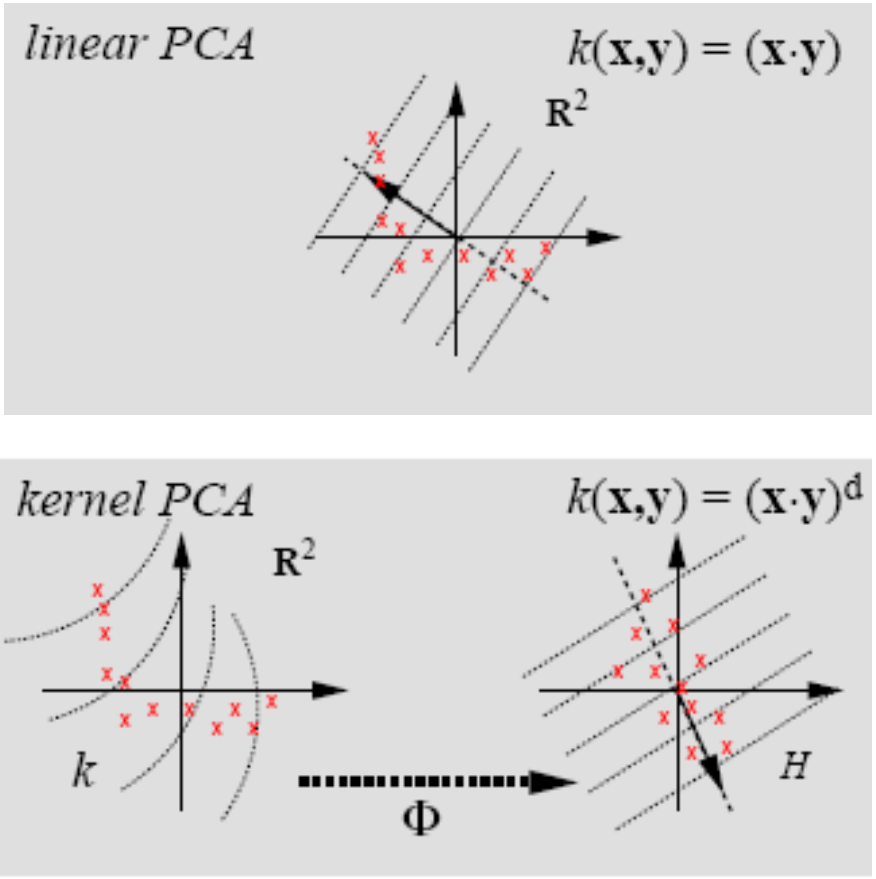
$$m\lambda \sum \alpha_i \phi(x_i) = \sum_i \sum_j \alpha_i \phi(x_i) K(x_i, x_j)$$

- And we can finally write the expression as the eigenvalue problem

$$\mathbf{K}\alpha = \lambda\alpha$$

- For a test pattern **x**, we extract a nonlinear component via

$$\langle V_k, x \rangle = \sum_{i=1}^{m} \alpha_i^k K(x_i, x)$$



Outline

- Review of SVM in the previous lecture
- A bit of history of kernel methods and SVM
- Kernel methods
- Kernel PCA
- **Designing kernels**

How to chose kernels

- There is no absolute rules for choosing the right kernel, adapted to a particular problem
- Kernel design can start from the desired feature space, from combination or from data
- Some considerations are important:
  - Use kernel to introduce a priori (domain) knowledge
  - Be sure to keep some information structure in the feature space
  - Experimentally, there is some “robustness” in the choice, if the chosen kernels provide an acceptable trade-off between
    - simpler and more efficient structure (e.g. linear separability), which requires some “explosion”
    - Information structure preserving, which requires that the “explosion” is not too strong.

Kernels built from data

- In general, this mode of kernel design can use both labeled and unlabeled data of the training set! Very useful for semi-supervised learning
- Basic ideas:
  - Intuitively, kernels define clusters in the feature space, and we want to find interesting clusters, i.e. cluster components that can be associated with labels.
  - Convex linear combination of kernels in a given family: find the best coefficient of eigen-components of the (complete) kernel matrix by maximizing the alignment on the labeled training data.
  - Build a generative model of the data, then use the Fischer Kernel.

Some operations on kernels

- The class of kernel functions on $\mathcal{X}$ has several useful closure properties. It is a convex cone, which means that if k_1 and k_2 are two kernels, then any linear combination, $\lambda_1 k_1 + \lambda_2 k_2$, with $\lambda_1, \lambda_2 \geq 0$ is a kernel.
- If k_1 and k_2 are two kernels, then $k(\mathbf{x}, \mathbf{x}') := k_1(\mathbf{x}, \mathbf{x}')\, k_2(\mathbf{x}, \mathbf{x}')$ is also a kernel. Several other operations are possible.
- Some other operations are forbidden: if k is a kernel, then log(k) is not positive definite in general, and neither is k^β for $0 < \beta < 1$. However, these two operations can be linked.

Combining kernels

- Multiple kernel matrices k_1, k_2, …, k_c for the same set of objects are available.
- We may design a single kernel k from several basic kernels k_1, k_2, …, k_c. A simple way to achieve this is to take the sum of the kernels:

$$k = \sum_{i=1}^{c} k_i$$

- Multiple kernel matrices k_1, k_2, …, k_c for the same set of objects are available.

$$k = \sum_{i=1}^{c} \mu_i k_i$$

From similarity scores to kernels

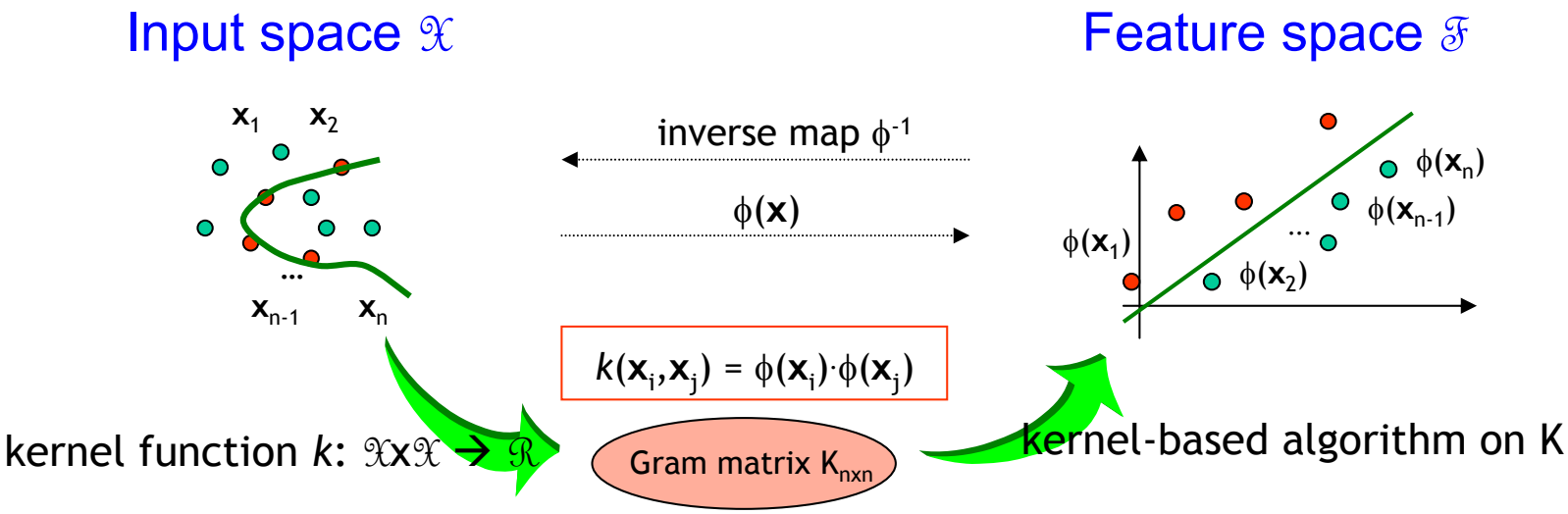
- $\mathcal{X}$ be a set and s: $\mathcal{X}$ **x** $\mathcal{X} \rightarrow \Re$ a measure of similarity.
- Empirical kernel map (Tsuda, 1999): (1) Choosing a finite set of objects $t_1, \ldots, t_r \in \mathcal{X}$ called templates. (2) $\mathbf{x} \in \mathcal{X}$ is represented by a vector of similarity

$$\mathbf{x} \in \mathcal{X} \rightarrow \phi(\mathbf{x}) = (s(\mathbf{x}, t_1), \ldots, s(\mathbf{x}, t_r))^{\mathrm{T}} \in \Re^p$$

- The kernel is defined as the dot product between two similarity vectors

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X},\ k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^{\mathrm{T}} \phi(\mathbf{x}) = \sum_{i=1}^{r} s(\mathbf{x}, t_i) s(\mathbf{x}', t_i)$$

Summary



- Map the data from $\mathcal{X}$ into a (high-dimensional) vector space, the feature space $\mathcal{F}$, by applying the feature map ϕ on the data points **x**.
- Find a linear (or other easy) pattern in $\mathcal{F}$ using a well-known algorithm (that works on the Gram matrix).
- By applying the inverse map, the linear pattern in $\mathcal{F}$ can be found to correspond to a complex pattern in $\mathcal{X}$.
- This implicitly by only making use of inner products in $\mathcal{F}$ (kernel trick)

Thank you!