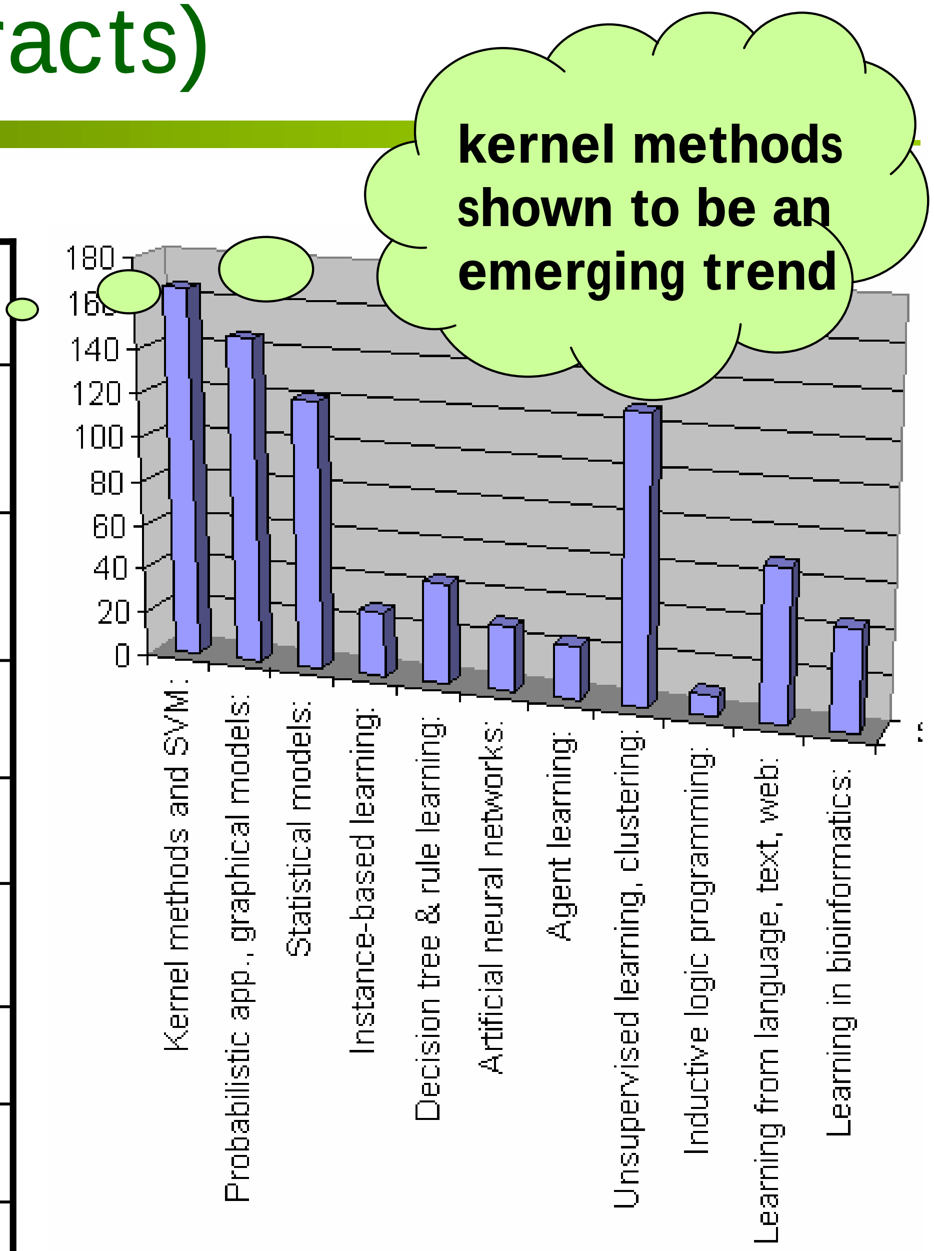
A Primer on Kernel Methods

- Paper of Jean Philippe Vert, Koji Tsuda, Bernhard Scholkopf, in *Kernel Methods in Computational Biology*, MIT 2004.
- *Learning with Kernels*, Bernhard Scholkopf and Alexander Smola, 2002, MIT.
- *Kernel Methods for Pattern Recognition*, John Shawe-Taylor, Nello Cristianini, Cambridge 2004.
- http://www.jaist.ac.jp/~bao/kernels/

Content

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



10 challenging problems in data mining
(IEEE ICDM'05)

1. Developing a unifying theory of data mining
2. Scaling up for high dimensional data/high speed streams
3. Mining sequence data and time series data
4. Mining complex knowledge from complex data
5. Data mining in a network setting
6. Distributed data mining and mining multi-agent data
7. Data mining for biological and environmental problems
8. Data-mining-process related problems
9. Security, privacy and data integrity
10. Dealing with non-static, unbalanced and cost-sensitive data

Kernel methods: 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.
- Schoolkopf 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.

Content

1. Introduction
2. **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)
3. Some Kernel Methods
4. Support Vector Machines
5. More Kernels
6. Designing Kernels

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 $\mathcal{X}$, which may be images, molecules, texts, etc.
- Majority of data analysis methods represent $\mathbb{S}$ by:
  - defining a representation $\phi(\mathbf{x}) \in \mathcal{F}$ for each object $\mathbf{x} \in \mathcal{X}$, where $\phi(\mathbf{x})$ can be a real-valued vector ($\mathcal{F} = \Re^p$) or a finite-length string, or more complex representation.
  - representing $\mathbb{S}$ by a set of representations of the objects

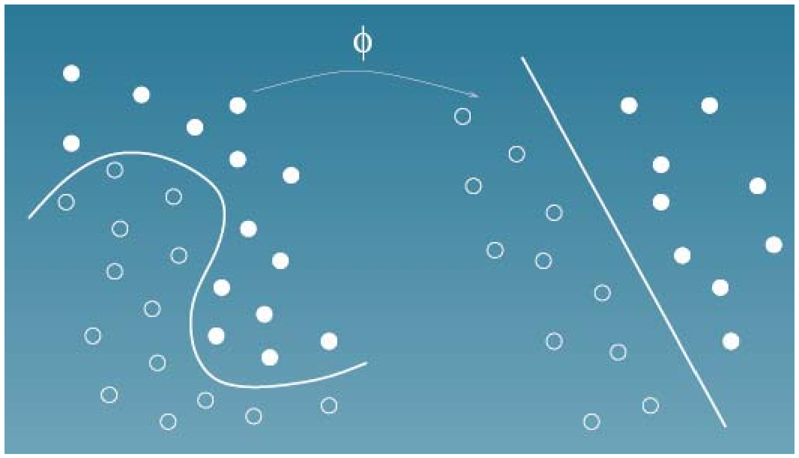
$$\phi(\mathbb{S}) = (\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n))$$

Kernel representation: idea

- Data are not represented individually anymore, but only through a set of pairwise comparisons.

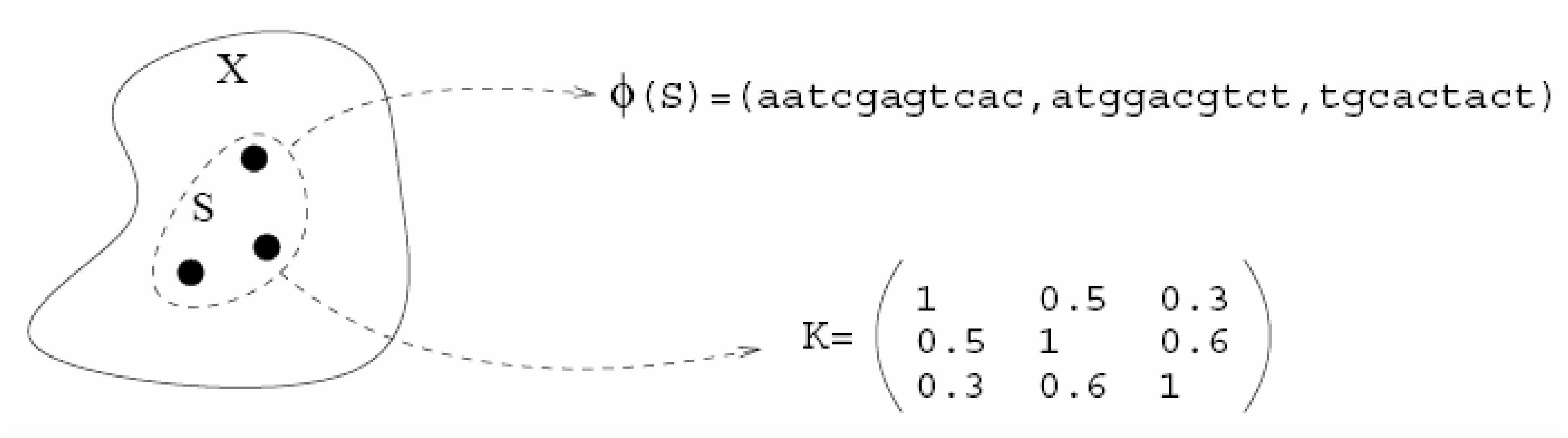
- Instead of using a mapping $\phi: \mathfrak{X} \rightarrow \mathfrak{F}$ to represent each object $\mathbf{x} \in \mathfrak{X}$ by $\phi(\mathbf{x}) \in \mathfrak{F}$, a real-valued "comparison function" (called kernel) $k: \mathfrak{X} \times \mathfrak{X} \rightarrow \mathfrak{R}$ is used, and the data set $\mathbb{S}$ is represented by the nxn matrix (Gram matrix) of pairwise comparisons $k_{i,j} = k(x_i, x_j)$.

- A main question is how to find a kernel k such that, in the new space, problem solving is easier (e.g. linear).



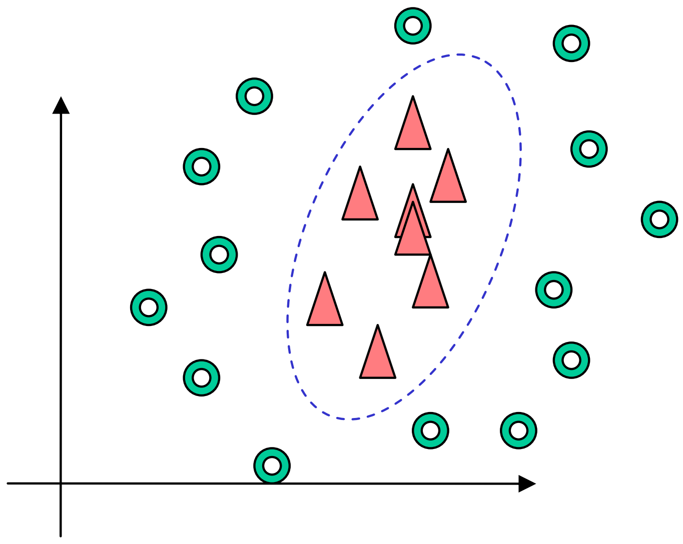
- All kernel methods have two parts:
  (1) find such a kernel k, and
  (2) process such Gram matrices.

Kernel representation: idea


X
S
$\phi(S) = (\text{aatcgagtcac}, \text{atggacgtct}, \text{tgcactact})$
$K = \begin{pmatrix} 1 & 0.5 & 0.3 \\ 0.5 & 1 & 0.6 \\ 0.3 & 0.6 & 1 \end{pmatrix}$

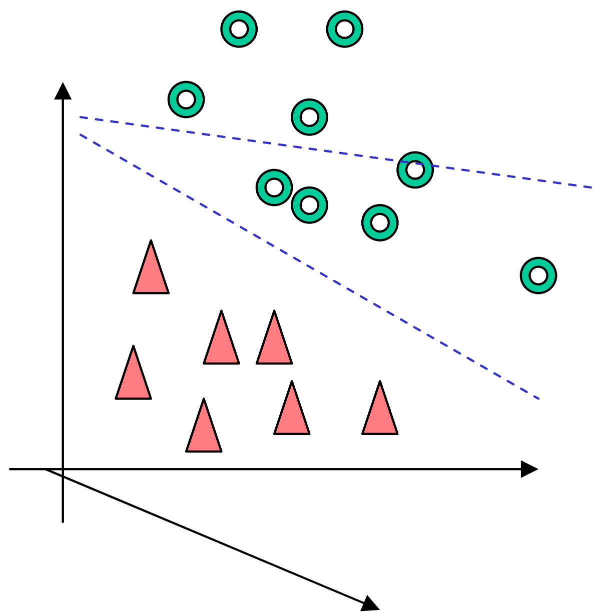
- $\mathcal{X}$ is the set of all oligonucleotides, $\mathbb{S}$ consists of three oligonucleoides.
- Traditionally, each oligonucleotide is represented by a sequence of letters.
- In kernel methods, $\mathbb{S}$ is represented as a matrix of pairwise similarity between its elements.

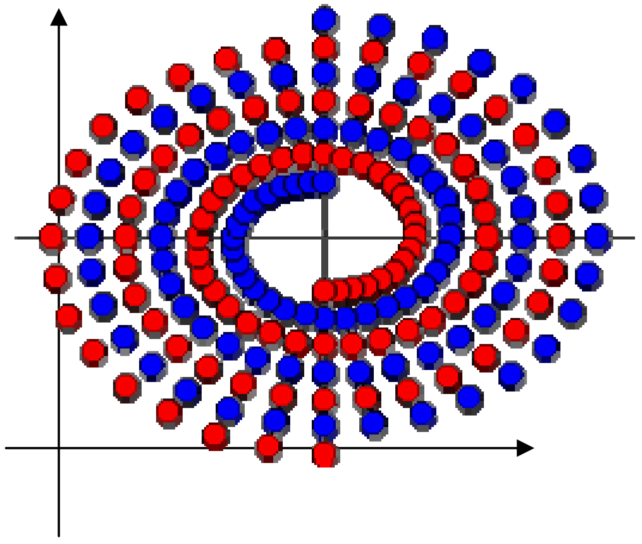
Examples of kernels



by polynomial kernel (n=2)

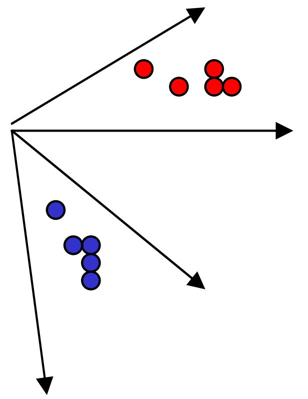
$$\phi:(x_1,x_2)\mapsto$$
$$(x_1^2,\sqrt{2}x_1x_2,x_2^2)$$





by RBF kernel (n=2)

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



Prerequisite 1. Dot product and Hilbert space

- A vector space (linear space) $\mathcal{H}$ of "vectors" over the reals $\Re$ if two operations,
  - *vector addition*: $\mathcal{H} \times \mathcal{H} \to \mathcal{H}$ denoted **v** + **w**, where **v**, **w** $\in \mathcal{H}$, and
  - *scalar multiplication*: $\Re \times \mathcal{H} \to \mathcal{H}$ denoted λ**v**, where $\lambda \in \Re$ 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 $\langle .,. \rangle$: $\mathcal{H} \times \mathcal{H} \to \Re$, $(\mathbf{x}, \mathbf{x}') \mapsto \langle \mathbf{x}, \mathbf{x}' \rangle$ that is strictly positive definite, i.e., $\forall \mathbf{x} \in \mathcal{H}$, $\langle \mathbf{x}, \mathbf{x} \rangle \geq 0$ with equality only for $\mathbf{x} = 0$.

- Norm $\|\mathbf{x}\| := \langle \mathbf{x}, \mathbf{x} \rangle^{1/2}$ ($\|\,.\,\|: \mathcal{H} \to \Re^+$), 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:

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

Prerequisite 1. Dot product and 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

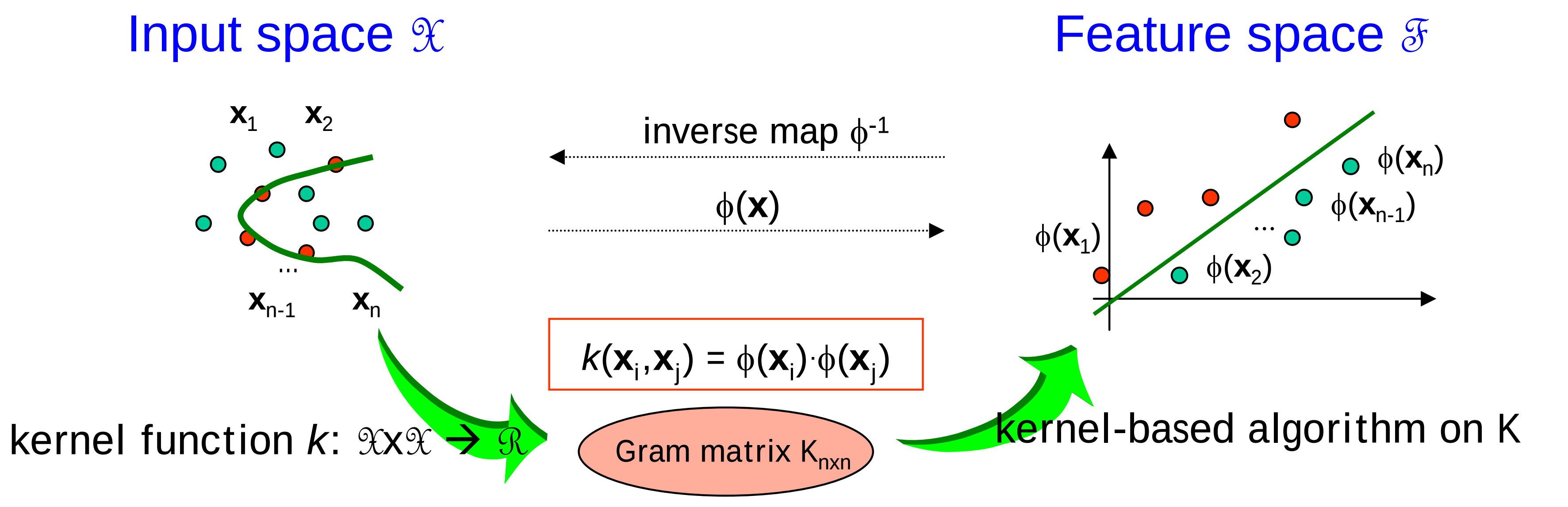
  $$\sup_{m>n} \|h_n - h_m\| \to 0, \text{ as } n \to \infty$$

  $\mathcal{H}$ 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.

Kernel methods: the scheme


Input space $\mathscr{X}$
Feature space $\mathscr{F}$
$\mathbf{x}_1$
$\mathbf{x}_2$
…
$\mathbf{x}_{n-1}$
$\mathbf{x}_n$
inverse map ϕ^{-1}
$\phi(\mathbf{x})$
$\phi(\mathbf{x}_1)$
$\phi(\mathbf{x}_2)$
…
$\phi(\mathbf{x}_{n-1})$
$\phi(\mathbf{x}_n)$
$k(\mathbf{x}_i,\mathbf{x}_j) = \phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j)$
kernel function k: $\mathscr{X} \times \mathscr{X} \rightarrow \mathfrak{R}$
Gram matrix K_{nxn}
kernel-based algorithm on K

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

Kernel representation: comments

- The representation as a square matrix does not depend on the nature of the objects to be analyzed.
    - an algorithm developed to process such a matrix can analyze images as well as molecules, if valid functions *k* can be defined.
    - a complete modularity exists between the design of a function *k* to represent data and the design of an algorithm to process the data representations $\rightarrow$ utmost importance in field where data of different nature need to be integrated and analyzed in a unified framework.
- The size of the matrix used to represent a dataset of *n* objects is always *nxn*, whatever the nature or the complexity of the objects.
- Comparing objects is an easier task in many cases than finding an explicit representation for objects that a given algorithm can process.

Valid kernels

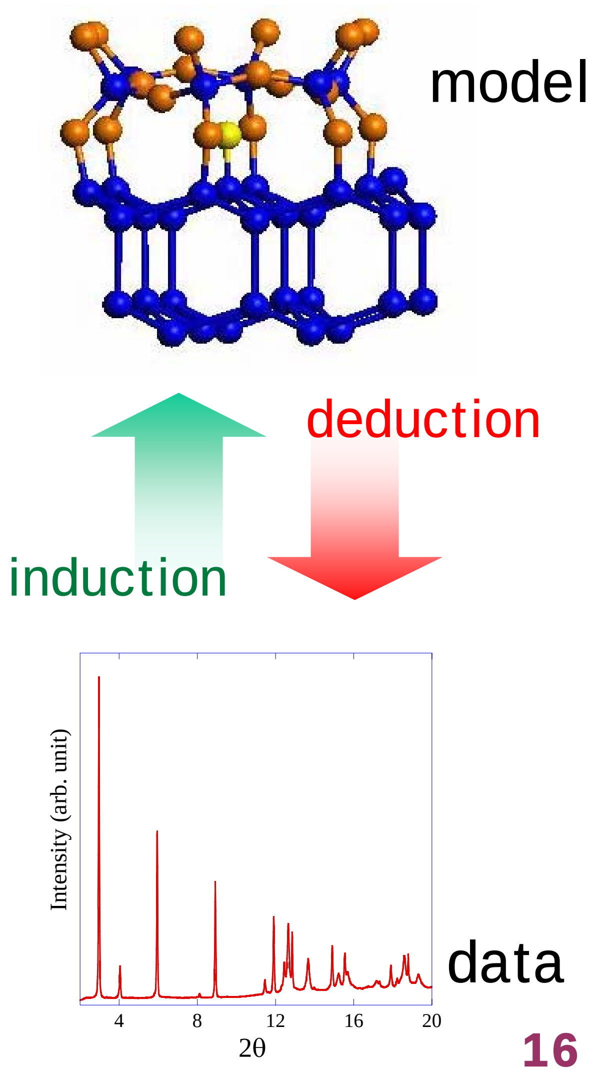
- Most kernel methods can only process symmetric positive definite matrix: $k_{i,j} = k_{j,i}$ for all i, j and $c^T k c \geq 0$, for any $c \in \Re^p$.
- **Definition 1**: *A function $k: \mathfrak{X} \times \mathfrak{X} \rightarrow \Re$ is called a positive definite kernel if it is symmetric (that is, $k(x, x') = k(x', x)$ $\forall x, x' \in \mathfrak{X}$), and positive definite, that is for any $n > 0$, any choice of n objects $x_1, \ldots, x_n \in \mathfrak{X}$, 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.

Prerequisite 2: Regularization (1/6)

- Classification is one inverse problem (induction): Data → Model parameters
- Inverse problems are typically ill posed, as opposed to the well-posed problems typically when modeling physical situations where the model parameters or material properties are known (a unique solution exists that depends continuously on the data).
- To solve these problems numerically one must introduce some additional information about the solution, such as an assumption on the smoothness or a bound on the norm.



Prerequisite 2: Regularization (2/6)

- **Input of the classification problem**: m pairs of training data (x_i, y_i) generated from some distribution $P(x,y)$, $x_i \in \mathcal{X}$, $y_i \in \mathcal{C} = \{C_1, C_2, \dots, C_k\}$ (training data).
- **Task**: Predict *y* given *x* at a new location, i.e., to find a function *f* (model) to do the task, $f: \mathcal{X} \rightarrow \mathcal{C}$.
- **Training error** (empirical risk): Average of a loss function on the training data, for example

$$R_{emp}[f] = \frac{1}{m}\sum_{i=1}^{m} c(x_i, y_i, f(x_i)) \qquad \text{for example, } c(x_i, y_i, f(x_i)) = \begin{cases} 1, & y_i = f(x_i) \\ 0, & y_i \neq f(x_i) \end{cases}$$

- **Target**: (risk minimization) to find a function *f* that minimizes the test error (expected risk)

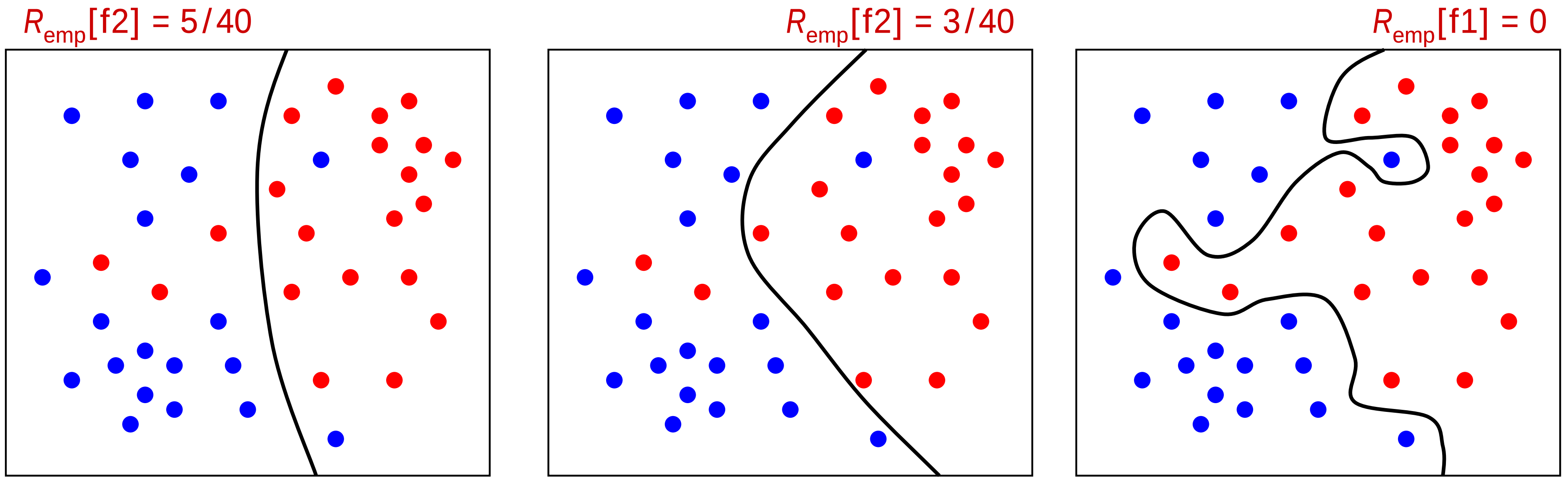
$$R[f] := E[R_{test}[f]] = E[c(x, y, f(x))] = \int_{\mathcal{X} \times \mathcal{Y}} c(x, y, f(x))\, dP(x, y)$$

Prerequisite 2: Regularization (3/6)

- **Problem:** Small $R_{emp}[f]$ does not always ensure small $R[f]$ (overfitting), i.e., we may get small

$$\text{Prob}\{\sup_{f\in\mathfrak{F}} \left|R_{emp}[f]-R[f]\right| < \varepsilon\}$$

- **Fact 1**: Statistical learning theory says that the difference is small if $\mathfrak{F}$ is small.
- **Fact 2**: Practical work says the difference is small if f is smooth.



Prerequisite 2: Regularization (4/6)

- **Regularization** is the restriction of a class $\mathfrak{F}$ of possible minimizers (with $f \in \mathfrak{F}$) of the empirical risk functional $R_{emp}[f]$ such that $\mathfrak{F}$ becomes a compact set.
- **Key idea**: Add a regularization (stabilization) term $\Omega[f]$ such that small $\Omega[f]$ corresponds to smooth f (or otherwise simple f) and minimize

$$R_{reg}[f] := R_{emp}[f] + \lambda\Omega[f]$$

  - $R_{reg}[f]$: regularized risk functionals;
  - $R_{emp}[f]$: empirical risk;
  - $\Omega[f]$: regularization term; and
  - λ: regularization parameter that specifies the trade-off between minimization of $R_{emp}[f]$ and the smoothness or simplicity enforced by small $\Omega[f]$ (i.e., complexity penalty).
- We need someway to measure if the set $\mathfrak{F}_C = \{f \mid \Omega[f] < C\}$ is a "small" class of functions.

Prerequisite 2: Regularization (5/6)

- **Representer theorem**: In many cases, the expansion of *f* in terms of $k(\mathbf{x}_i, \mathbf{x})$, where the $\mathbf{x}_i$ are training data, contains the minimizer of $R_{emp}[f] + \lambda\Omega[f]$.
- Maximization of the margin of classification in feature space by using the regularizing term $\frac{1}{2}\|w\|^2$, and thus minimizing

$$\Omega[f] := \frac{1}{2}\|w\|^2, \text{ and therefore } R_{reg}[f] + \frac{\lambda}{2}\|w\|^2$$

- Rewriting the above risk functional in terms of the reproducing kernel Hilbert space (RKHS) $\mathcal{H}$ associated with *k*, and we equivalently minimize

$$R_{reg}[f] := R_{emp}[f] + \frac{\lambda}{2}\|f\|_{\mathcal{H}}^2$$

Prerequisite 2: Regularization (6/6)

- **Reproducibility** is one of the main principles of the scientific method.
- An experimental description (thought experiment) produced by a particular researcher or group of researchers is generally evaluated by other independent researchers by attempting to reproduce it; they repeat the same experiment themselves and see if it gives equal results as reported by the original group.

Fundamental properties of kernels:

Kernels as inner product (1/3)

- 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*.

- Limitation: can only defined when data represented as vectors.

- 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 $k(\mathbf{x},\mathbf{x}') = \phi(\mathbf{x})^T\phi(\mathbf{x}')$ (2)

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

Fundamental properties of kernels:
Kernels as inner product (2/3)

- Any $\phi: \mathfrak{X} \to \Re^p$ for some $p \geq 0$ results in a valid kernel through (2).
- Whether there exist more general kernels than these? At least, it is possible if replacing $\Re^p$ by an infinite-dimensional Hilbert space.

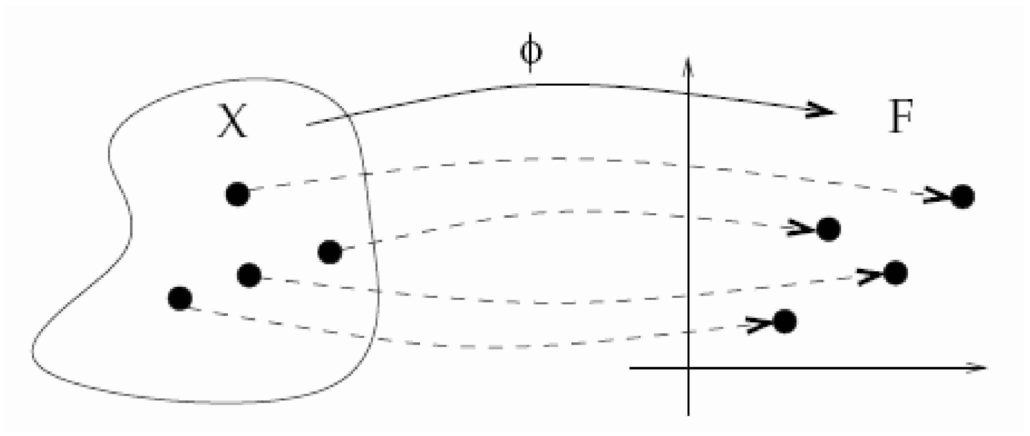
- **Theorem 2**: *For any kernel k on a space $\mathfrak{X}$, there exists a Hilbert space $\mathfrak{F}$ and a mapping $\phi: \mathfrak{X} \to \mathfrak{F}$ such that*

$$k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle, \quad \text{for any } \mathbf{x}, \mathbf{x}' \in \mathfrak{X},$$

where $\langle u, v \rangle$ represents the dot product in the Hilbert space between any two points $u, v \in \mathfrak{F}$.

Fundamental properties of kernels:

Kernels as inner product (3/3)



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 $\mathbb{S}$, 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.

Fundamental properties of kernels:

Kernels as measures of similarity (1/2)

- Kernels are often presented as measures of similarity, in the sense that $k(\mathbf{x}, \mathbf{x}')$ is "large" when $\mathbf{x}$ and $\mathbf{x}'$ are "similar".

- This motivates the design of kernels for particular types of data or applications, because particular prior knowledge might suggest a relevant measure of similarity in a given context.

- The dot product does not always fit one's intuition of similarity. There are cases where these notions coincide, e.g., Gaussian RBK kernel.

- For a general kernel k, where Hilbert distance $d(u,v)^2 = \langle (u-v), (u,v) \rangle$, $\| . \|$ is Hilbert norm (i.e., $\|u\|^2 = \langle u, u \rangle$), the following holds for any x, $x' \in \mathcal{F}$

$$k(x,x') = \frac{\|\phi(\mathbf{x})\|^2 + \|\phi(\mathbf{x}')\|^2 - d(\phi(\mathbf{x}),\phi(\mathbf{x}'))^2}{2}$$

$k(\mathbf{x}, \mathbf{x}')$ measures the similarity between $\mathbf{x}$ and $\mathbf{x}'$ as the opposite of the square distance $d(\phi(\mathbf{x}),\phi(\mathbf{x}'))^2$ between their images, up to square of their norms.

Fundamental properties of kernels:
Kernels as measures of similarity (2/2)

- For more general kernels, one should keep in mind the slight gap between the notion of dot product and similarity.
- It is generally relevant to think of a kernel as a measure of similarity, in particular when it is constant on the diagonal.
- This intuition is useful in designing kernels and in understanding kernel methods.

Fundamental properties of kernels:

Kernels as measures of function regularity (1/4)

- Let k be a kernel on a space $\mathcal{X}$, we show that k is associated with a set $\mathcal{H}_k = \{f: \mathcal{X} \to \Re\}$ of real-valued functions on $\mathcal{X}$, and $\mathcal{H}_k$ is endowed with a structure of Hilbert space (a dot product and a norm).

- Understand the functional space $\mathcal{H}_k$ and the norm associated with a kernel often helps in understanding kernel methods and in designing new kernels.

- **Example**: linear kernel on $\mathcal{X} = \Re^p$, i.e., $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{x}' = \sum x_i x_i'$. $\mathcal{H}_k$ is the space of linear functions $f: \Re^p \to \Re$ and associated norm

$$\mathcal{H}_k = \{f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}: \mathbf{w} \in \Re^p\} \text{ and } \| f \|_{\mathcal{H}_k} = \| \mathbf{w} \| \text{ for } f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}.$$

The norm $\| f \|_{\mathcal{H}_k}$ decreases if the "smoothness" of f increases, where the definition of smoothness depends on the kernel (for linear kernel it relates to function slope: smooth $\equiv$ flat). The notion of smoothness is dual to that of similarity: a function is "smooth" when it varies slowly between "similar" points.

Fundamental properties of kernels:

Kernels as measures of function regularity (2/4)

- The question is how to systematic construct $\mathcal{H}_k$ from k? We define $\mathcal{H}_k$ as the set of functions f: $\mathcal{X} \to \Re$ in $\mathcal{H}_k$ for n > 0, a finite number of points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{X}$ and weights $\alpha_1, \ldots, \alpha_n \in \Re$ as

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x}) \quad \text{and the norm} \quad \|f\|_{\mathcal{H}_k}^2 := \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) \qquad (*)$$

- One can prove that $\mathcal{H}_k$ is a Hilbert space, and the dot product for two functions

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x}), \text{ and } g(\mathbf{x}) = \sum_{j=1}^{m} \beta_j k(\mathbf{x}_j, \mathbf{x})$$

$$\langle f, g \rangle = \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \beta_j k(\mathbf{x}_i, \mathbf{x}_j) = \sum_{i=1}^{n} \alpha_i g(\mathbf{x}_i) = \sum_{j=1}^{m} \beta_j f(\mathbf{x}_j)$$

- Interesting property: the value $f(\mathbf{x})$ of $f \in \mathcal{H}_k$ at a point $\mathbf{x} \in \mathcal{X}$ can be expressed as a dot product in $\mathcal{H}_k$, i.e., if we take $g = k(\mathbf{x}, .)$

$$\langle f, k(\mathbf{x},.) \rangle = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x}) = f(\mathbf{x})$$

Fundamental properties of kernels:

Kernels as measures of function regularity (3/4)

- Taking $f(.) = k(\mathbf{x}',.)$, we derive the reproducing property valid for any $\mathbf{x}$, $\mathbf{x}' \in \mathcal{X}$:

$$k(\mathbf{x}, \mathbf{x}') = \langle k(\mathbf{x},.), k(\mathbf{x}',.)\rangle \qquad (3)$$

- $\mathcal{H}_k$ is usually called the reproducing kernel Hilbert space (RKHS) associated with k. $\mathcal{H}_k$ is one possible feature space associated with k when considering $\phi: \mathcal{X} \to \mathcal{H}_k$ defined by $\phi(\mathbf{x}) := k(\mathbf{x}, .)$

- A general Hilbert space usually contains many non-smooth functions. RKHS contains smooth functions, and is “smaller” than a general Hilbert space.

Fundamental properties of kernels:

Kernels as measures of function regularity (4/4)

- We should keep in mind the connection between kernels and norms on functional spaces. Most kernel methods have an interpretation in terms of functional analysis.
- Many kernel methods, including SVMs, can be defined as algorithm that, given a set $\mathbb{S}$ of objects, return a function that solves

$$\min_{f \in \mathcal{H}_k} R(f, \mathbb{S}) + c\|f\|_{\mathcal{H}_k} \qquad (4)$$

- Where $R(f,\mathbb{S})$ is small when f "fits" the data well, and the term $\|f\|_{\mathcal{H}_k}$ ensures the solution of the above equation is "smooth". In fact, besides fitting the data well and being smooth, the solution of the above equation turns out to have special properties that are useful for computational reasons (representer theorem).

Content

The kernel trick (1/5)

- Two concepts that underlie most kernel methods: the kernel trick and the representer theorem.
- The kernel trick is a simple and general principle based on the property that kernels can be thought of as inner product: *Any positive definite kernel $k(x, y)$ can be expressed as a dot product in a high-dimensional space.*
- The kernel trick is obvious but has huge practical consequences that were only recently exploited.
- **Proposition 3.** *Any algorithm for vectorial data that can be expressed only in terms of dot products between vectors can be performed implicitly in the feature space associated with any kernel, by replacing each dot product by a kernel evaluation.*

The kernel trick (2/5)

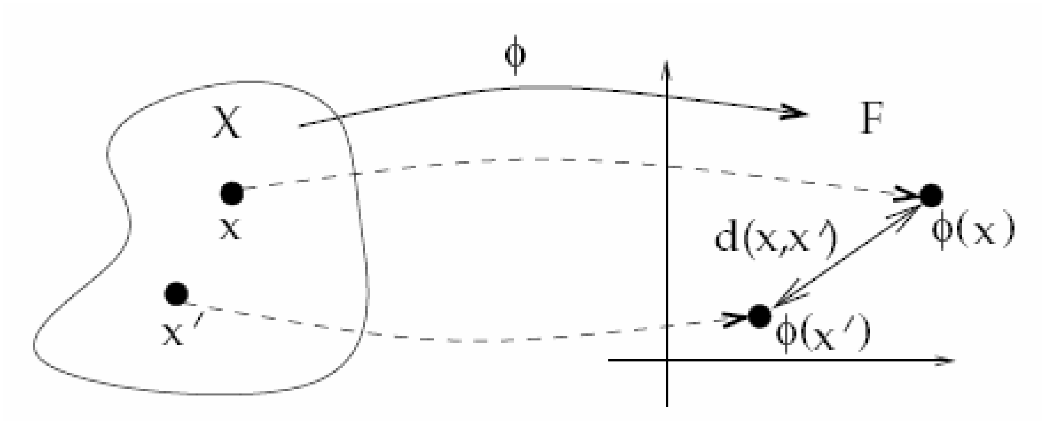
- **Kernelization**: Transforming linear methods → nonlinear methods
  - Kernel trick can be first used to transform linear methods (e.g., linear discriminant analysis or PCA) into non-linear methods by simply replacing the classic dot product by a more general kernel, such as the Gaussian RBF kernel.
  - Nonlinearity is obtained at no computational cost, as the algorithm remains exactly the same.
- **Non-vectorial data**:
  - The combination of the kernel trick with kernels defined on non-vectorial data permits the application of many classic algorithms on vectors to virtually any type of data as long as kernel can be defined.
  - Examples: to perform PCA on a set of sequences or graphs thanks to the availability of kernels for sequences and for graphs.

The kernel trick (3/5)

Example: Computing distances between objects

- Recall that the kernel can be expressed as $k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$ in a dot product space $\mathcal{F}$ for some mapping $\phi: \mathcal{X} \rightarrow \mathcal{F}$.
- Given $\mathbf{x}_1$, $\mathbf{x}_2 \in \mathcal{X}$, mapped to $\phi(\mathbf{x}_1)$, $\phi(\mathbf{x}_2) \in \mathcal{F}$. We define distance $d(\mathbf{x}_1, \mathbf{x}_2)$ as Hilbert distance between their images

$$d(\mathbf{x}_1, \mathbf{x}_2) := \left\| \phi(\mathbf{x}_1) - \phi(\mathbf{x}_2) \right\|$$



Given a space X endowed with a kernel, a distance can be defined between points of $\mathcal{X}$ mapped to the feature space $\mathcal{F}$ associated with the kernel. This distance can be computed without knowing the mapping ϕ thank to the kernel trick.

The kernel trick (4/5)

Example: Computing distances between objects

- The following equality shows the distance can be expressed in terms of dot products in $\mathcal{F}$:

$$\|\phi(\mathbf{x}_1)-\phi(\mathbf{x}_2)\|^2 = \langle\phi(\mathbf{x}_1),\phi(\mathbf{x}_1)\rangle + \langle\phi(\mathbf{x}_2),\phi(\mathbf{x}_2)\rangle - 2\langle\phi(\mathbf{x}_1),\phi(\mathbf{x}_2)\rangle$$

From here the distance can be computed in terms of the kernel

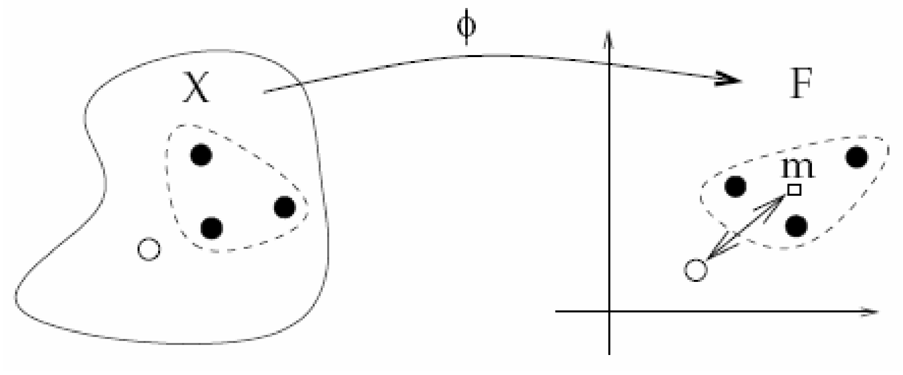
$$d(\mathbf{x}_1,\mathbf{x}_2) = \sqrt{k(\mathbf{x}_1,\mathbf{x}_1)+k(\mathbf{x}_2,\mathbf{x}_2)-2k(\mathbf{x}_1,\mathbf{x}_2)}$$

- The example shows the effect of the kernel trick: it is possible to perform operations implicitly in the feature space.

- A more general question: Let $\mathbb{S} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ be a fixed finite set of objects, and $\mathbf{x} \in \mathcal{X}$ a generic object. Is it possible to assess how “close” the object **x** is to the set of objects $\mathbb{S}$?

The kernel trick (5/5)

Example: Computing distances between objects



The distance between the white and the set of three black in $\mathcal{X}$ endowed with a kernel is defined as the distance between image of the white and the centroid m of the images of the black. The centroid m might have no preimage in $\mathcal{X}$. This distance can nevertheless be computed implicitly with the kernel trick.

$$\mathrm{dist}(\mathbf{x},\mathbb{S}) = \|\phi(\mathbf{x}) - m\| = \left\| \phi(\mathbf{x}) - \frac{1}{n}\sum_{i=1}^{n}\phi(\mathbf{x}_i) \right\|$$

Using kernel trick we obtain $\mathrm{dist}(\mathbf{x},\mathbb{S}) = \sqrt{k(\mathbf{x},\mathbf{x}) - \frac{2}{n}\sum_{i=1}^{n} k(\mathbf{x},\mathbf{x}_i) + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} k(\mathbf{x},\mathbf{x}_i)}$

$$dist(\mathbb{S}_1,\mathbb{S}_2) = \sqrt{\frac{1}{|\mathbb{S}_1|^2}\sum_{x,x'\in\mathbb{S}_1} k(\mathbf{x},\mathbf{x}') + \frac{1}{|\mathbb{S}_2|^2}\sum_{x,x'\in\mathbb{S}_2} k(\mathbf{x},\mathbf{x}') - \frac{1}{|\mathbb{S}_1||\mathbb{S}_2|}\sum_{x\in\mathbb{S}_1, x'\in\mathbb{S}2} k(\mathbf{x},\mathbf{x}')}$$

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) $\Omega(.)$ 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 $(\alpha_1, \ldots, \alpha_n) \in \Re^p$.

Kernel principal component analysis (kernel PCA)

- **PCA**: Given $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \Re^p$. Find: $\mathbf{x}'_1, \ldots, \mathbf{x}'_m \in \Re^d$, $d \ll p$ 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

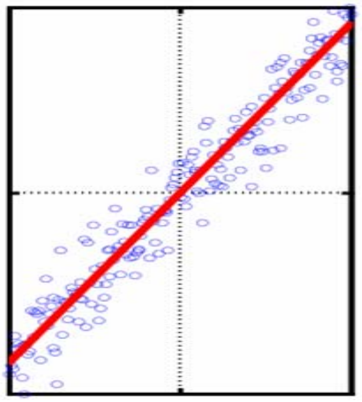
$$\lambda\upsilon = C\upsilon \qquad C = \frac{1}{m}\sum_{j=1}^{m} x_j x_j^T$$

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

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

- All solutions v with $\lambda \neq 0$ lie in the span of $x_1, \ldots, x_m$, ,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 principal component analysis (kernel PCA)

- 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

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

$$= \sum_j\sum_i \alpha i\phi(x_i)\left\langle\phi(x_i),\phi(x_j)\right\rangle$$

Kernel principal component analysis (kernel PCA)

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

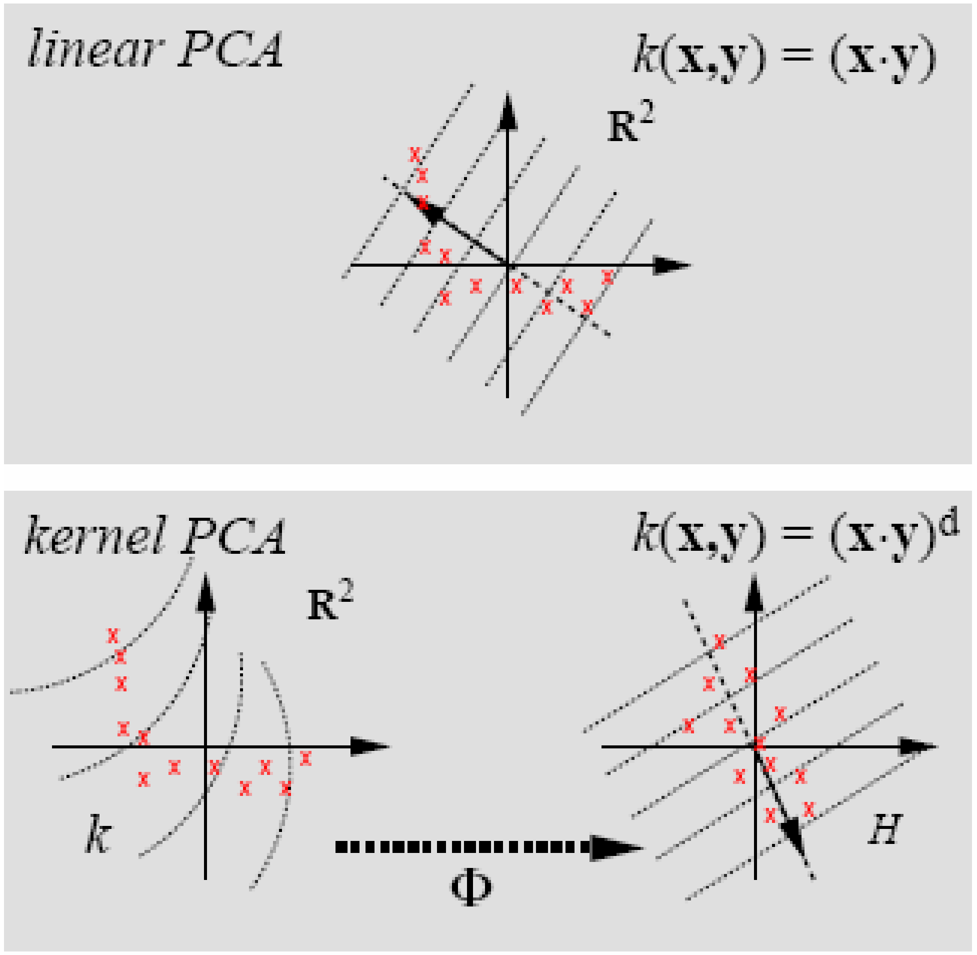
$$m\lambda \sum \alpha_i \phi(x_i) = \sum_i \sum_j \alpha_i \phi(x_i) K(x_i, x_j)$$

- 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

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

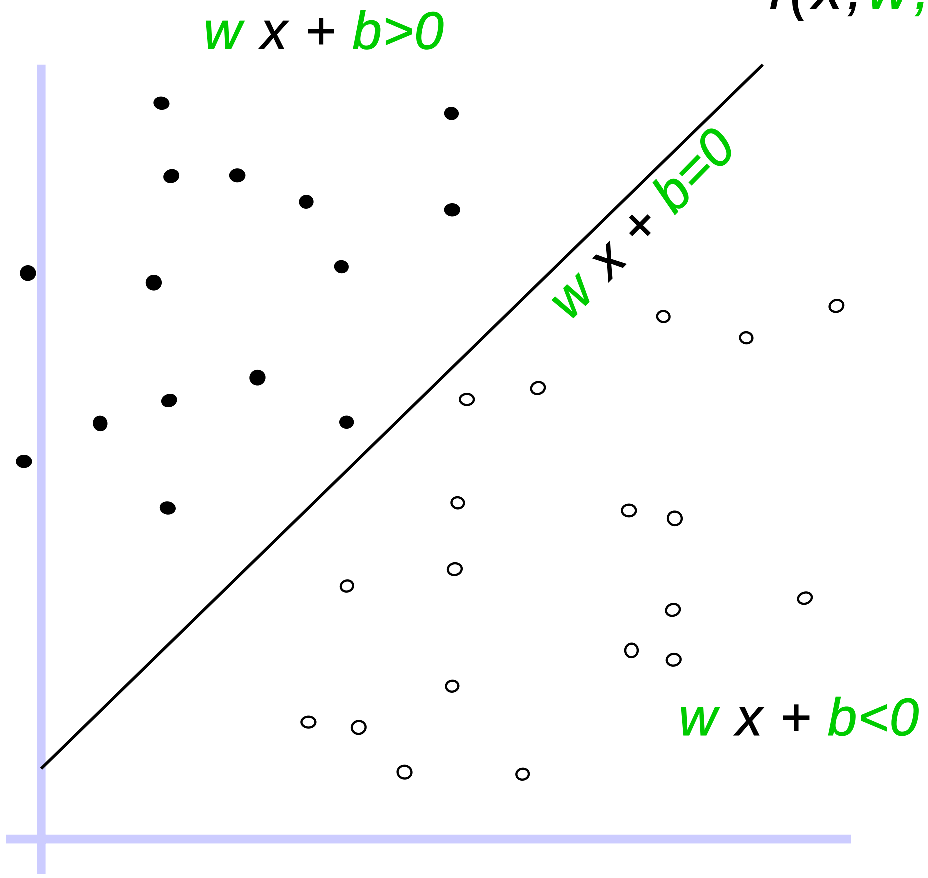

linear PCA
$k(\mathbf{x},\mathbf{y}) = (\mathbf{x}\cdot\mathbf{y})$
$\mathbf{R}^2$
kernel PCA
$k(\mathbf{x},\mathbf{y}) = (\mathbf{x}\cdot\mathbf{y})^d$
$\mathbf{R}^2$
k
Φ
H

Content

Linear classifiers

- ● denotes +1
- ○ denotes -1

$f(x,w,b) = sign(w\ x + b)$



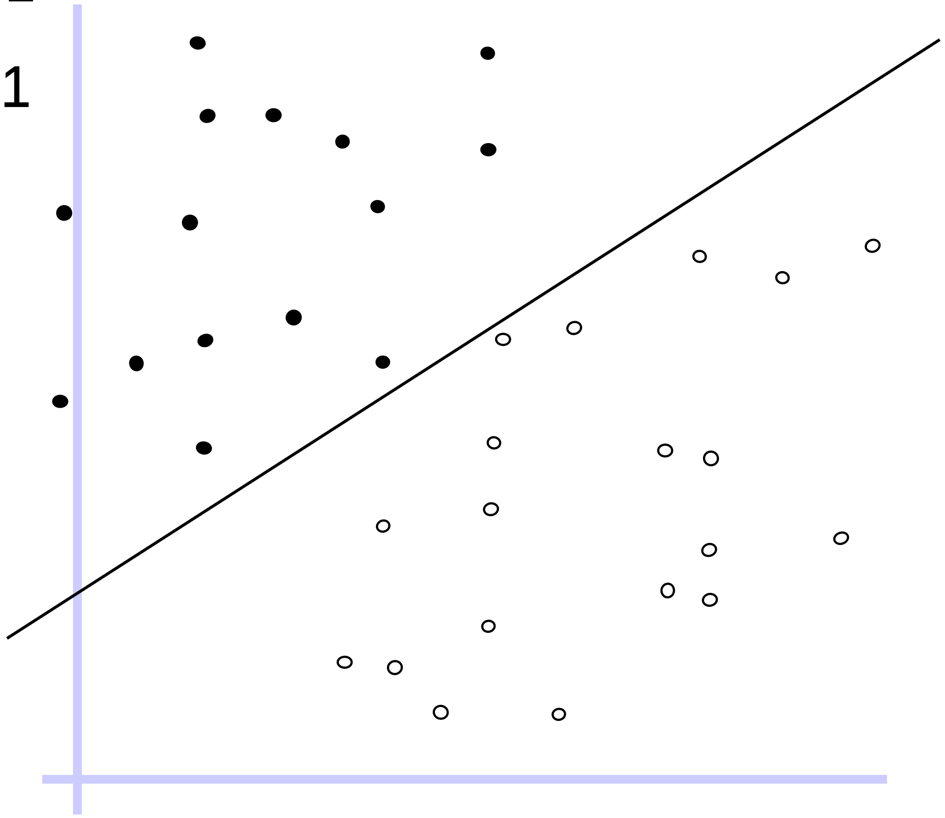
How would you classify this data?

The illustrated slides (p.42-48) are taken from Prof. Andrew Moore's SVM tutorial

Linear classifiers

- ● denotes +1
- ○ denotes -1

$f(x, w, b) = sign(w\ x + b)$



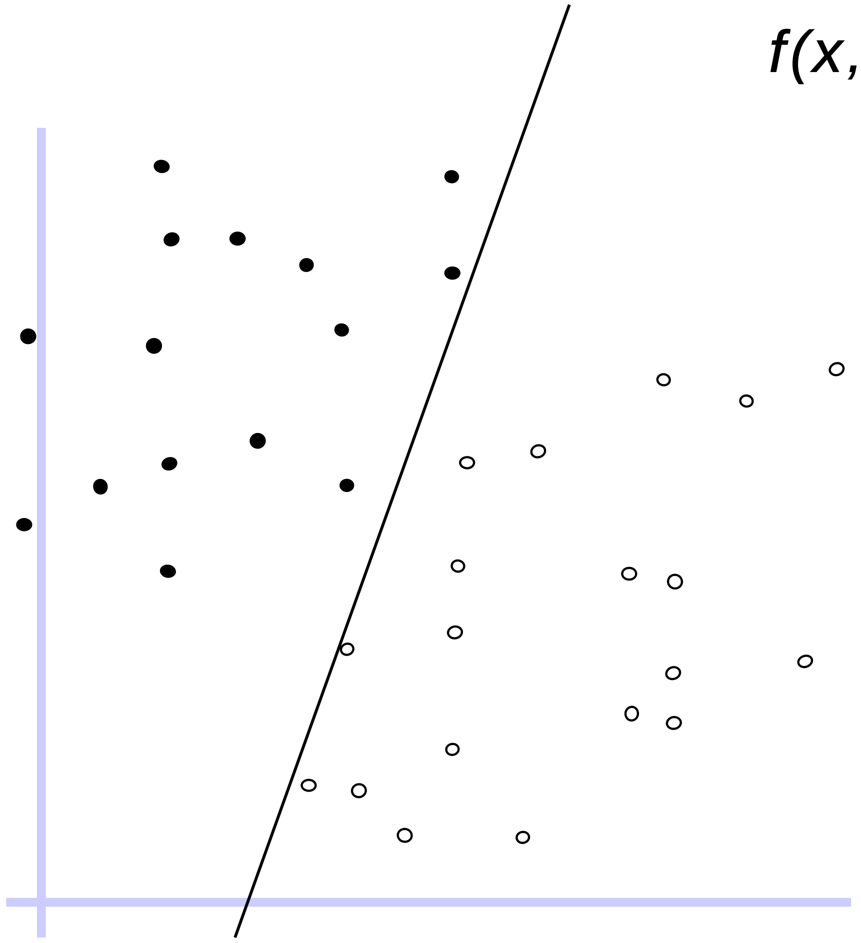
How would you classify this data?

Linear classifiers

● denotes +1

○ denotes -1

$f(x, w, b) = sign(w\, x + b)$



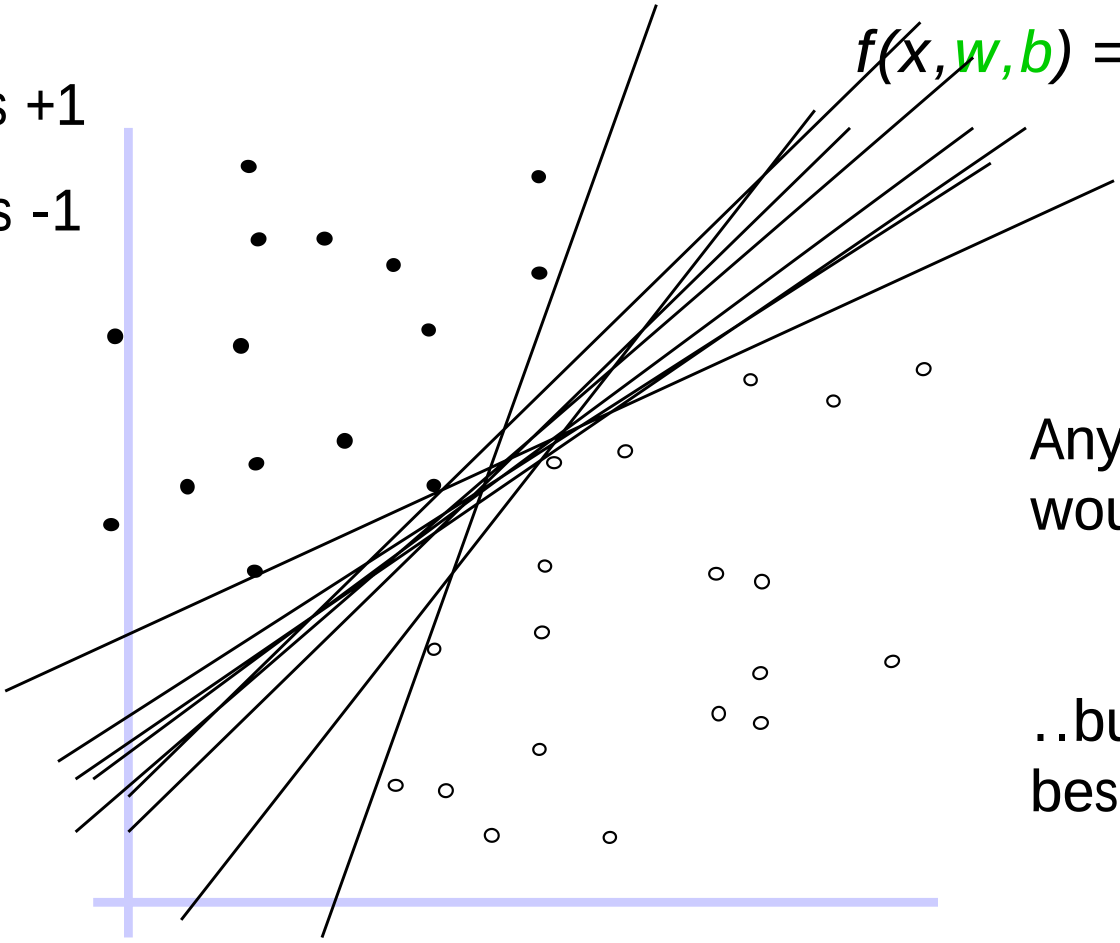
How would you classify this data?

Linear classifiers

● denotes +1

○ denotes -1

$f(x, w, b) = sign(w\ x + b)$



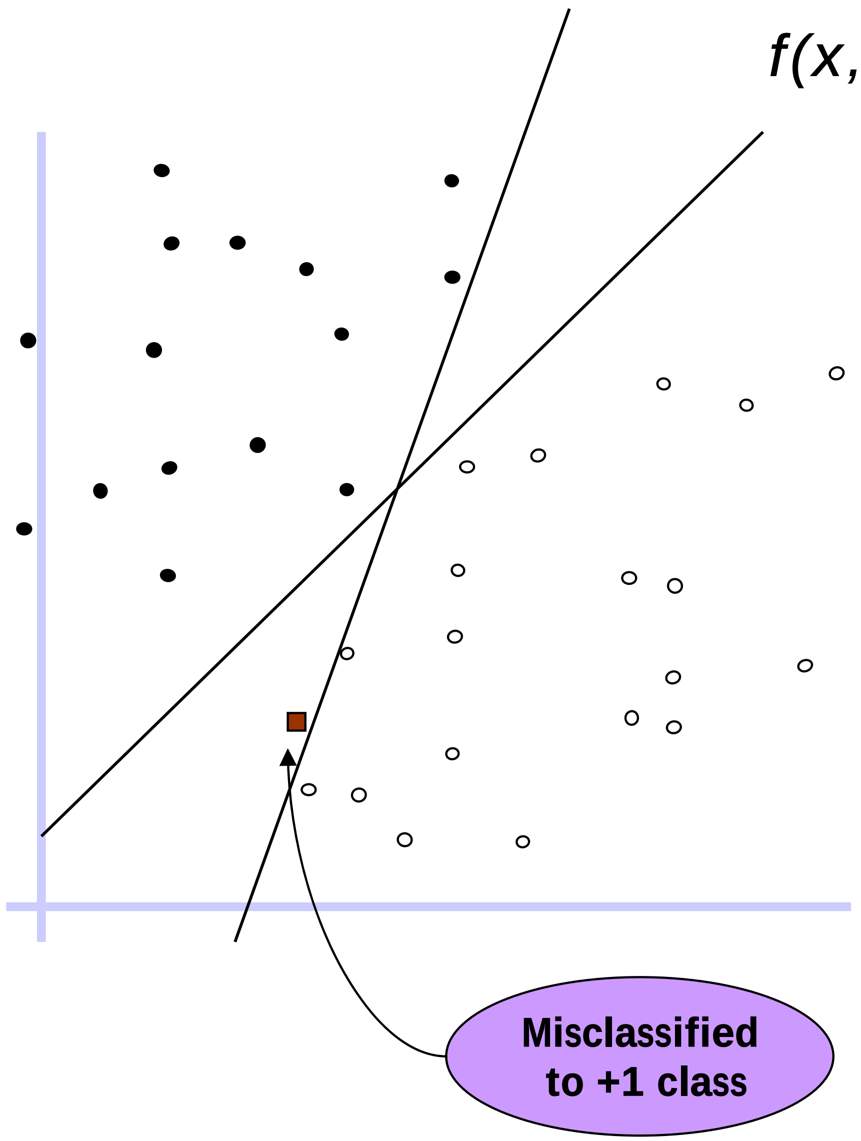
Any of these would be fine..

..but which is best?

Linear classifiers

- ● denotes +1
- ○ denotes -1

$f(x, w, b) = \text{sign}(w\, x + b)$

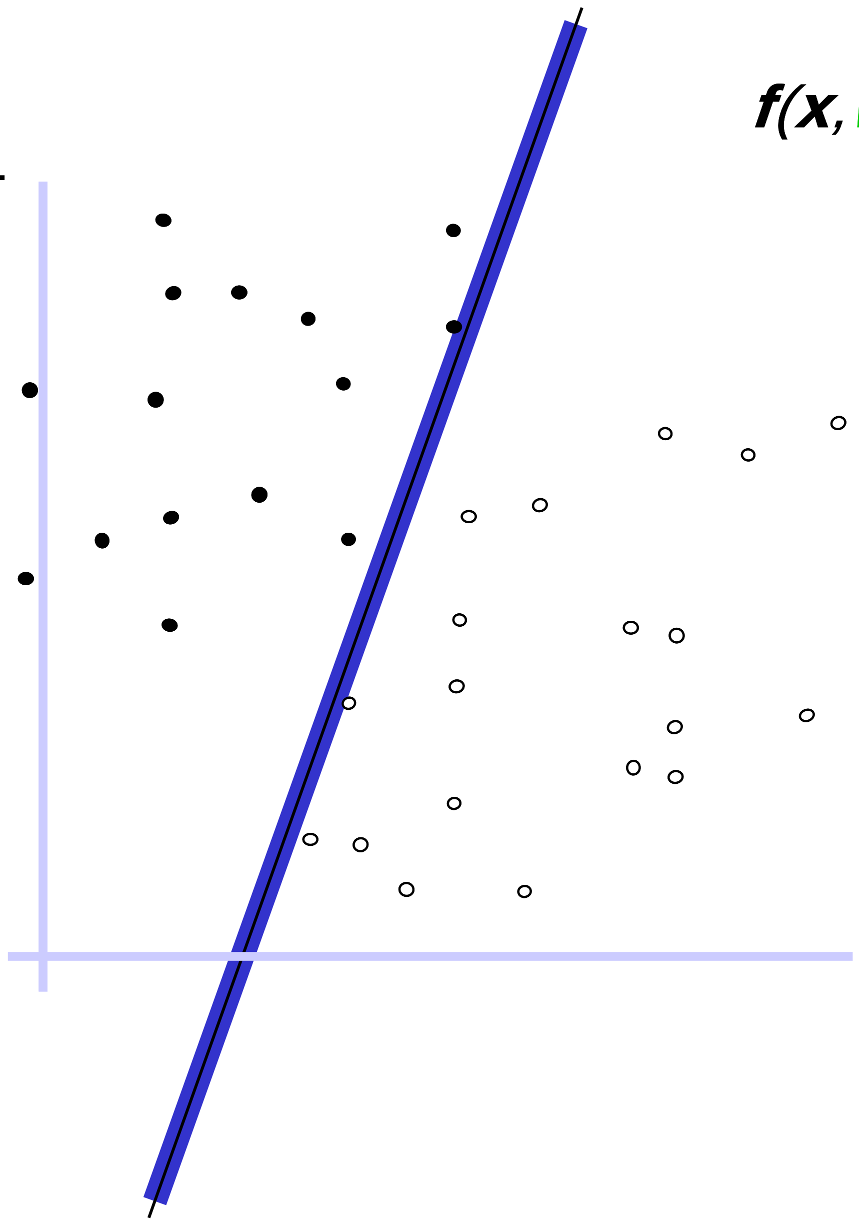


How would you classify this data?

Classifier margin

- ● denotes +1
- ○ denotes -1

$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\ \mathbf{x} + b)$



Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

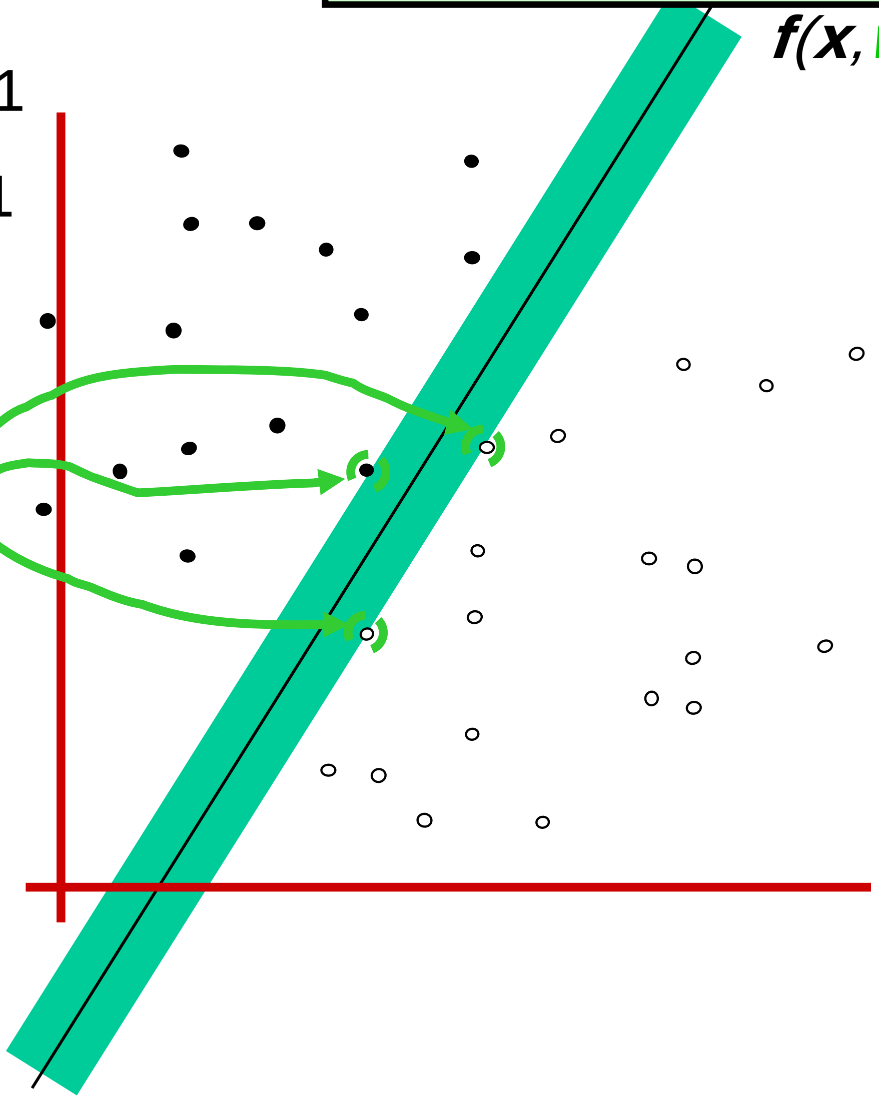
Maximum margin

1. Maximizing the margin is good according to intuition and PAC theory
2. Implies that only support vectors are important; other training examples are ignorable.
3. Empirically it works very very well.

$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\,\mathbf{x} + b)$

● denotes +1

○ denotes -1



Support vectors are those datapoints that the margin pushes up against

The maximum margin linear classifier is the linear classifier with the maximum margin.

This is the simplest kind of SVM (called an LSVM)

Summary

Soft margin problem

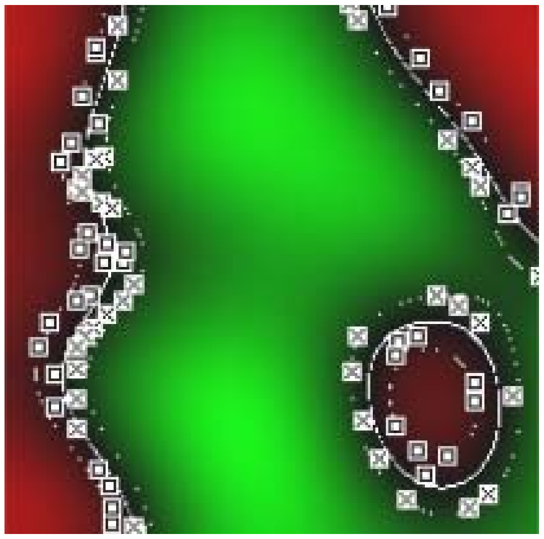
$$\min_{\mathbf{w},b,\xi_1,\dots,\xi_n} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$

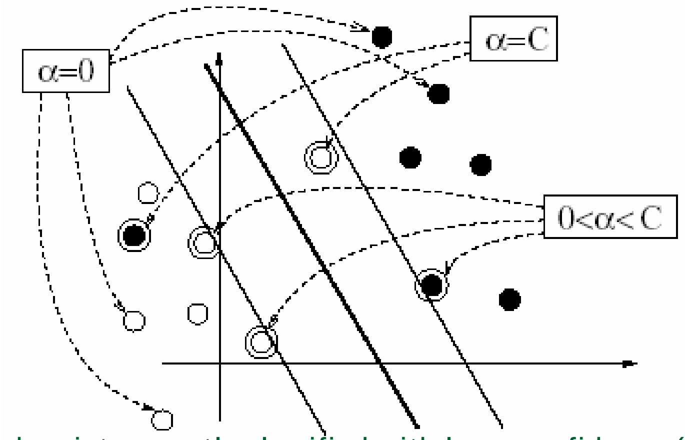
$$\forall i, \begin{cases} \xi_i \geq 0 \\ \xi_i - 1 + y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \geq 0 \end{cases}$$

Equivalent to dual problem

$$W(\alpha) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j + \sum_{i=1}^{n}\alpha_i$$

$$\begin{cases} \sum_{i=1}^{n} y_i\alpha_i = 0 \\ 0 \leq \alpha_i \leq C \quad \text{for } i = 1,\dots,n \end{cases}$$





Each point correctly classified with large confidence ($yf(x) > 1$) has a null multiplier. Other points are called support vector. They can be on the boundary., in which case the multiplier satisfies $0 \leq \alpha \leq C$, or on the wrong side of this boundary, in which case a = C.

Content

SVMs in practice

- Many free and commercial **software**: SVMlight, LIBSVM, mySVM, etc.
- **Multiclass problems**
  - SVM implemented for multiclass problem (just for few classes)
  - Combination of binary SVMs by one-against-all scheme
- **Kernel normalization**
  - Vectors: linearly scale each attribute to zero mean and unit variance and use Gaussian RBF is usually effective.
  - Strings and graphs: $k'_{ij} = k_{ij} / (k_{ii} k_{jj})^{1/2}$
- **Parameter setting**
  - The regularization parameter C of the SVM
  - The kernel (γ) and its parameters

$\Big\}$ Use cross-validation to choose C and γ

Content

Kernels for vectors

- The linear kernels $k_L = (\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{x}'$
- k_L is a particular case of polynomial kernels defined for $d \geq 0$

  $k_{Poly1} = (\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}')^d$ or $k_{Poly1} = (\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}')^d$

  they yields with $d = 2$ $\{x_1^2, x_1 x_2, x_2^2\}$, and $\{1, x_1, x_2, x_1^2, x_1 x_2, x_2^2\}$
- The Gaussian RBF kernel

$$k_G(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$

- The sigmoid kernel $k(\mathbf{x}, \mathbf{x}') = \tanh(k\mathbf{x}^T\mathbf{x}' + \theta)$

Kernels for strings

- Lodhi et al. (2002) in the NLP context. Basic idea:
  - Count the number of subsequences up to length *n* in a sequence.
  - Compose a high-dimensional feature vector by these counts.
  - Kernel = dot product between such feature vectors.
- Σ = set of symbols; a string $s = s_1, \ldots, s_{|s|} \in \Sigma^{|s|}$. $\mathcal{X} = \cup_{i=0,\ldots,\infty} \Sigma^i$
- An index set *i* of length *l* is an l-tuple of positions in *s*

$$i = (i_1, \ldots, i_l), \quad 1 \leq i_1 < \ldots < i_l < |s|$$

- Denoted by $s[i] = s_{i1}, \ldots, s_{il}$ the subsequence of s corresponding to *i*.
- Define weight of *i* by $\lambda^{l(i)}$, where $l(i) = i_l - i_1 + 1$, $\lambda < 1$.

Kernels for strings

- For $u \in \Sigma^k$, k is fixed, we define $\Phi_u\colon \mathcal{X} \rightarrow \Re$.

$$\forall s \in \mathcal{X}, \Phi_u(s) = \sum_{\boldsymbol{i}:s[i]=u} \lambda^{l(i)}$$

- Considering all sequences u of length n, we map each $s \in \mathcal{X}$ to a $|\Sigma|^n$–dimensional feature space by $s \rightarrow (\Phi_u(s))_{u\in\Sigma^n}$.

- We can then define a kernel for strings as the dot product between these representations

$$\forall s,t \in X, k_n(s,t) = \sum_{u\in\Sigma^n} \sum_{i:u=s[i]} \lambda^{l(i)} \sum_{j:u=t[]} \lambda^{l(j)}$$

$$= \sum_{u\in\Sigma^n} \sum_{i:u=s[i]} \sum_{j:u=t[j]} \lambda^{l(i)+l(j)}$$

The Fisher kernel

- Probabilistic models are convenient to represent families of complex objects that arise in computational biology.
- Hidden Markov models (HMMs) are a central tool for modeling protein families or finding genes from DNA sequences (Durbin et al., 1998). More complicated models called stochastic context-free grammars (SCFGs) are useful for modeling RNA sequences (Sakakibara et al., 1994).
- The Fisher kernel (Jaakkola and Haussler, 1999) provides a general principle to design a kernel for objects well modeled by a probabilistic distribution, or more precisely a parametric statistical model.

Content

How to choose 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.

Designing kernels:

Operations on kernels

- The class of kernel functions on $\mathfrak{X}$ 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.

Designing kernels:

Translation-invariant kernels and kernels on semi-groups

- When $\mathfrak{X} = \Re^p$, the class of translation-invariant kernels is defined as the class of kernels of the form, for some function $\psi : \Re^p \rightarrow \Re$

$$\forall \mathbf{x}, \mathbf{x}' \in \mathfrak{X},\ k(\mathbf{x}, \mathbf{x}') = \psi(\mathbf{x} - \mathbf{x}')$$

- More general, if (X, .) is a group, then the group kernels are defined, with $\psi : \mathfrak{X} \rightarrow \Re$, as functions of the form

$$k(\mathbf{x}, \mathbf{x}') = \psi(\mathbf{x}^{-1}\mathbf{x}')$$

Designing kernels:

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.

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

Designing kernels:

From similarity scores to kernels

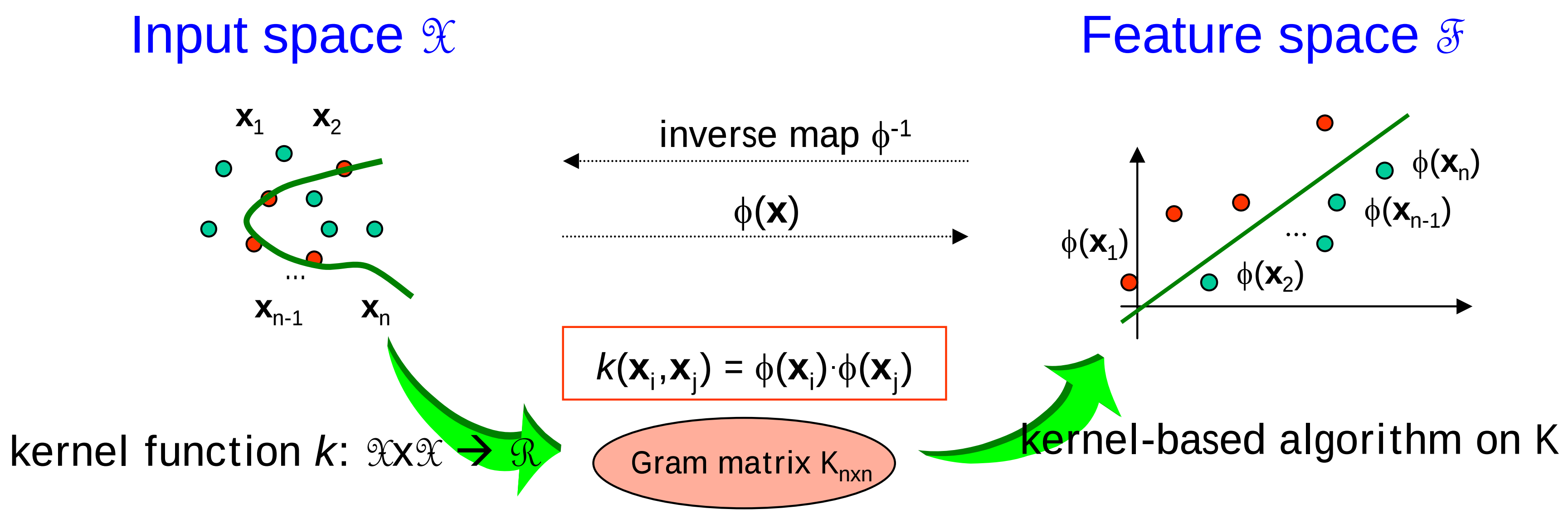
- $\mathfrak{X}$ be a set and s: $\mathfrak{X} \times \mathfrak{X} \rightarrow \Re$ a measure of similarity.
- Empirical kernel map (Tsuda, 1999): (1) Choosing a finite set of objects $t_1, \ldots, t_r \in \mathfrak{X}$ called templates. (2) $\mathbf{x} \in \mathfrak{X}$ is represented by a vector of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

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

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

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

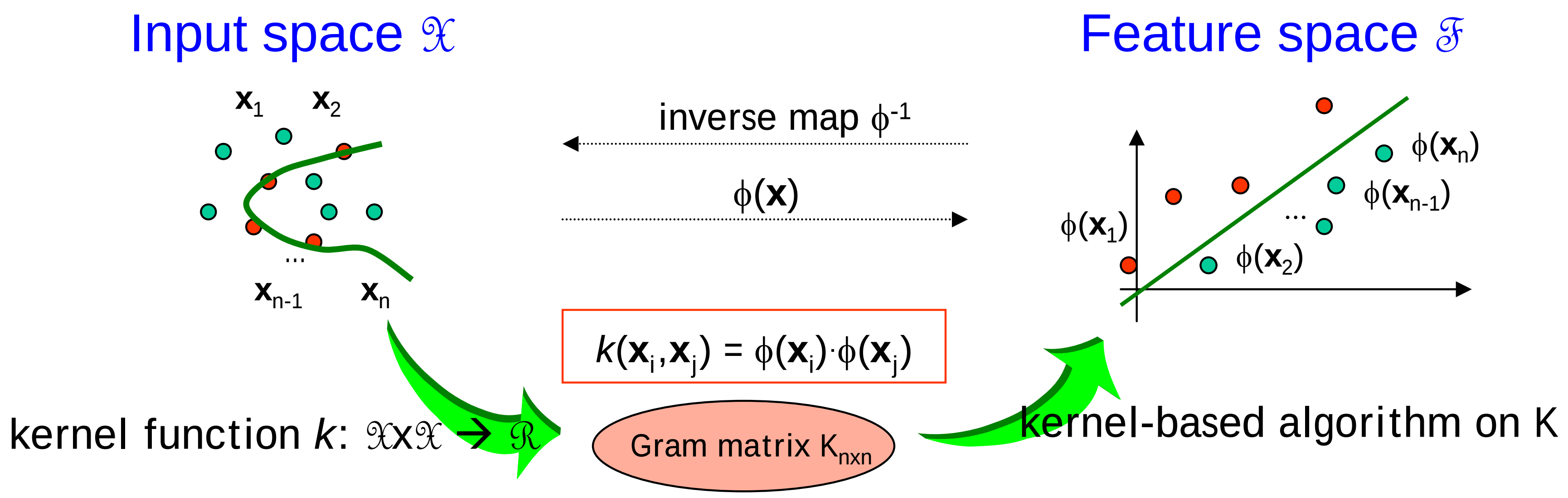
Summary



Input space $\mathfrak{X}$
Feature space $\mathfrak{F}$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_{n-1}$
$\mathbf{x}_n$
inverse map ϕ^{-1}
$\phi(\mathbf{x})$
$\phi(\mathbf{x}_1)$
$\phi(\mathbf{x}_2)$
$\phi(\mathbf{x}_{n-1})$
$\phi(\mathbf{x}_n)$
$k(\mathbf{x}_i,\mathbf{x}_j) = \phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j)$
kernel function k: $\mathfrak{X} \times \mathfrak{X} \to \mathfrak{R}$
Gram matrix K_{nxn}
kernel-based algorithm on K

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

Summary



Input space $\mathfrak{X}$
Feature space $\mathfrak{F}$
$\mathbf{x}_1$ $\mathbf{x}_2$ … $\mathbf{x}_{n-1}$ $\mathbf{x}_n$
inverse map ϕ^{-1}
$\phi(\mathbf{x})$
$\phi(\mathbf{x}_1)$ $\phi(\mathbf{x}_2)$ … $\phi(\mathbf{x}_{n-1})$ $\phi(\mathbf{x}_n)$
$k(\mathbf{x}_i,\mathbf{x}_j) = \phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j)$
kernel function k: $\mathfrak{X} \times \mathfrak{X} \rightarrow \mathfrak{R}$
Gram matrix K_{nxn}
kernel-based algorithm on K

- Linear algebra, probability/statistics, functional analysis, optimization
- **Mercer theorem**: Any positive definite function can be written as an inner product in some feature space.
- **Kernel trick**: Using kernel matrix instead of inner product in the feature space.
- **Representer theorem**:

  Every minimizer of $\min_{f \in \mathcal{H}} \{C(f,\{x_i, y_i\}) + \Omega(\|f\|_H)\}$ admits

  a representation of the form $f(.) = \sum_{i=1}^{m} \alpha_i K(., x_i)$