Kernels

Volker Tresp
Winter 2024-2025

Infinite Dimensions

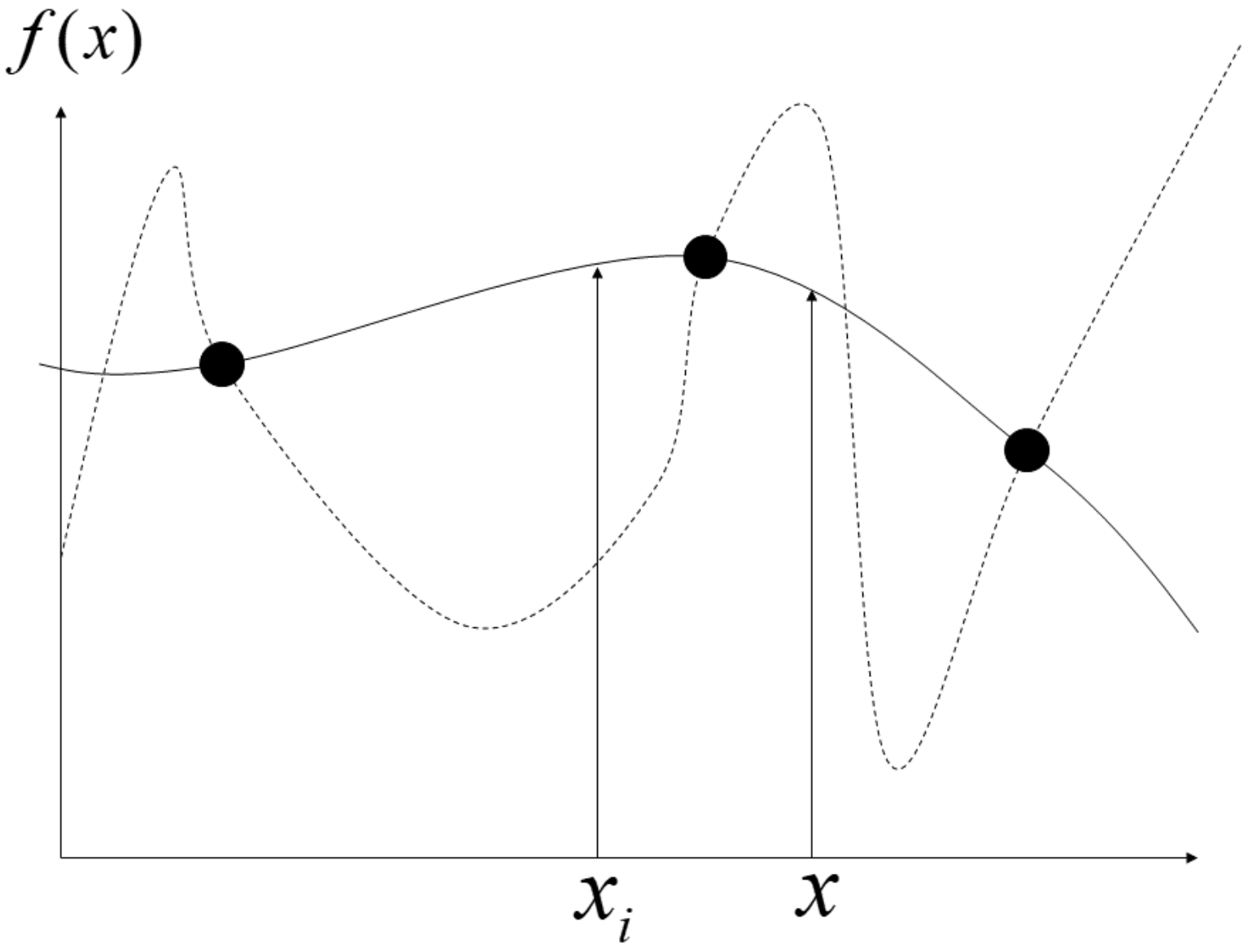
- In the lecture on complexity, we derived the equation: $M_\phi = \mathcal{O}(A^M)$ and working in high dimensions with the brute force approach can never work
- Today, we will see that this is not quite correct
- With kernels, we can work even with $M_\phi \to \infty$
- The idea is to represent the solution in a form that does not require the calculation of the weights

Dataset Size

- So far, we implied: the more data the better
- (Unfortunately), this is also not quite true with kernels

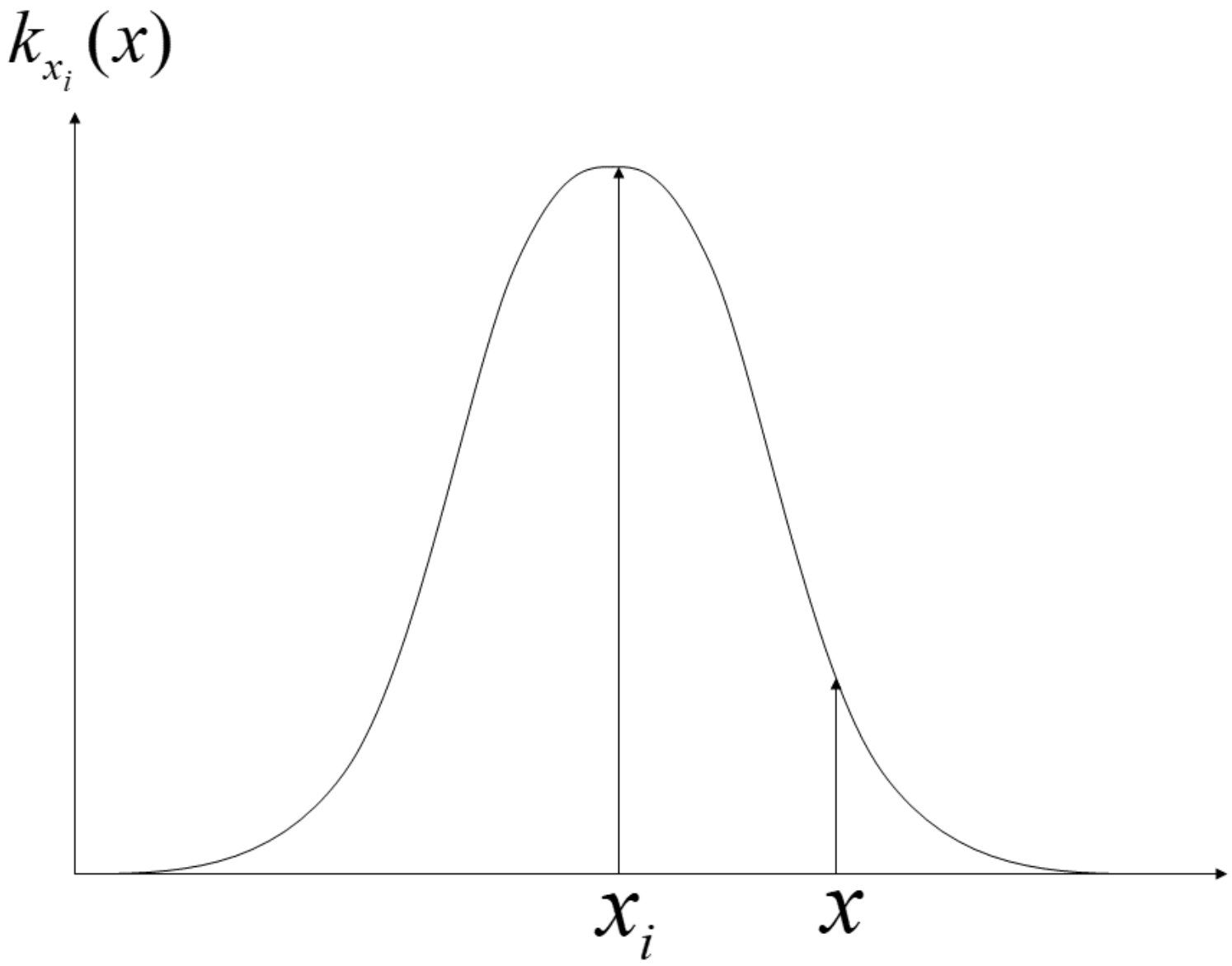
Smoothness Assumption

- In the lecture on basis functions, the assumption was that $f(\mathbf{x}_{i'})$ can be approximated by a weighted sum of basis functions; in the lecture on neural networks by a weighted sum of learned basis functions; in the lecture on deep neural networks by a weighted sum of highly complex learned basis functions
- Alternatively, it might make sense to have a preference for smooth functions: functional values close in input space should have similar functional values
- In the next figure it might make sense that the functional values at $\mathbf{x}_i$ and $\mathbf{x}_{i'}$ are similar (smoothness assumption)
- Thus, one might prefer the smooth (continuous) function in favor of the dashed function


$f(x)$
x_i
x

Introduction Kernels

- One can implement smoothness assumptions over kernel functions
- A kernel function $k(\mathbf{x}_i, \mathbf{x}_{i'}) = k_{\mathbf{x}_i}(\mathbf{x}_{i'})$ determines, how neighboring functional values are influenced when $f(\mathbf{x}_i)$ is given
- Example: Gaussian kernel


$k_{x_i}(x)$
x_i
x

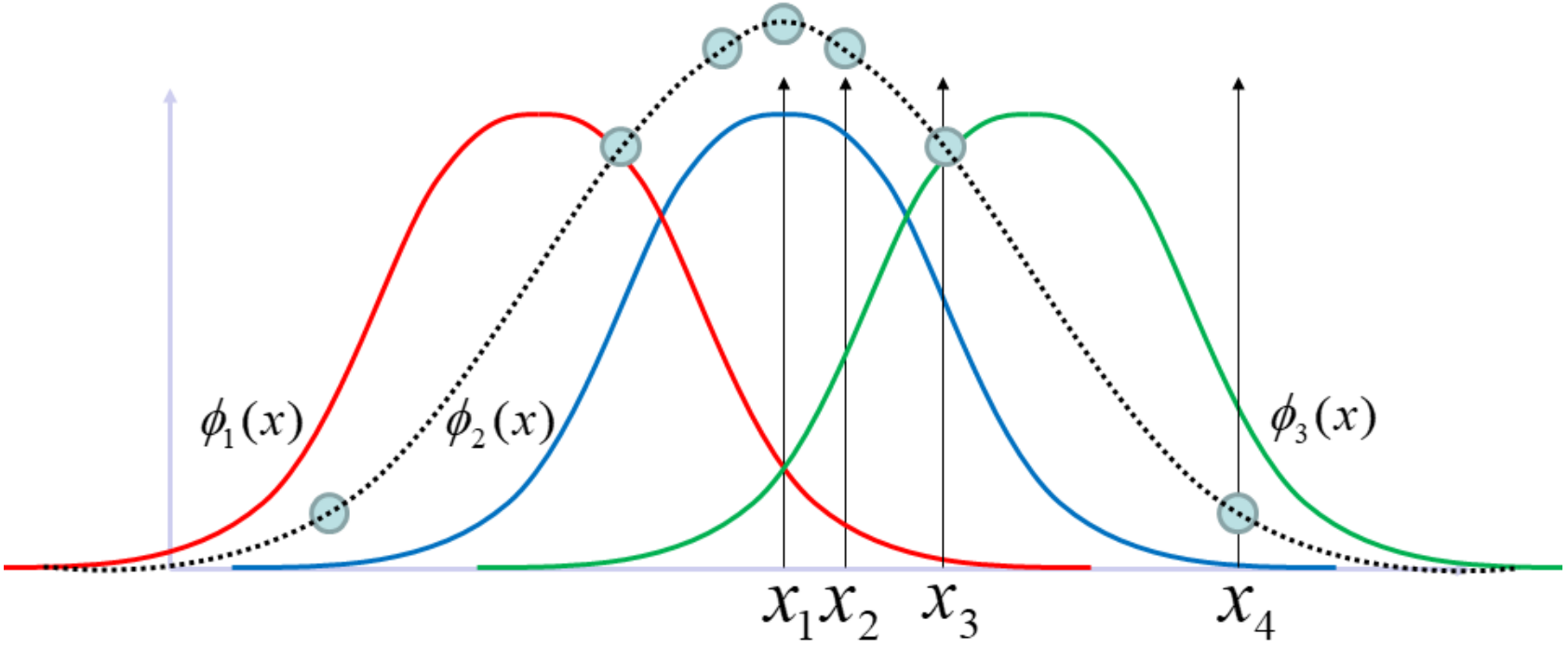
Kernels and Basis Functions

- It turns out that there is a close relationship between kernels and basis functions:

$$k(\mathbf{x}_i, \mathbf{x}_{i'}) = k_{\mathbf{x}_i}(\mathbf{x}_{i'}) = \sum_{j=1}^{M_\phi} \phi_j(\mathbf{x}_i)\phi_j(\mathbf{x}_{i'})$$

- It follows the symmetry: $k(\mathbf{x}_i, \mathbf{x}_{i'}) = k_{\mathbf{x}_i}(\mathbf{x}_{i'}) = k(\mathbf{x}_{i'}, \mathbf{x}_i) = k_{\mathbf{x}_{i'}}(\mathbf{x}_i)$
- Thus: given the M_ϕ basis functions, this equation gives you the corresponding kernel
- (Note the kernel is a function of weighted basis functions. The weight $\phi_j(\mathbf{x}_i)$ are the amplitudes of the basis functions at $\mathbf{x}_i$)
- As we see later: For positive definite kernels, we can also go the other way: given the kernels I can give you a corresponding set of basis functions (not unique)

Gaussian basis functions (continuous)
Kernel: dotted



$$\vec{\phi}(x_1) = (0.25, 1.00, 0.25)^T \qquad k(x_1, x_1) = \vec{\phi}^T(x_1)\vec{\phi}(x_1) = 1.12$$
$$\vec{\phi}(x_2) = (0.10, 0.90, 0.50)^T \qquad k(x_1, x_2) = \vec{\phi}^T(x_1)\vec{\phi}(x_2) = 1.05$$
$$\vec{\phi}(x_3) = (0.02, 0.60, 0.90)^T \qquad k(x_1, x_3) = \vec{\phi}^T(x_1)\vec{\phi}(x_3) = 0.83$$
$$\vec{\phi}(x_4) = (0.00, 0.01, 0.30)^T \qquad k(x_1, x_4) = \vec{\phi}^T(x_1)\vec{\phi}(x_4) = 0.08$$

Kernel Prediction

- Regression

$$\widehat{y}(\mathbf{x}_{i'}) = \sum_{i=1}^{N} v_i k(\mathbf{x}_i, \mathbf{x}_{i'})$$

- Classification

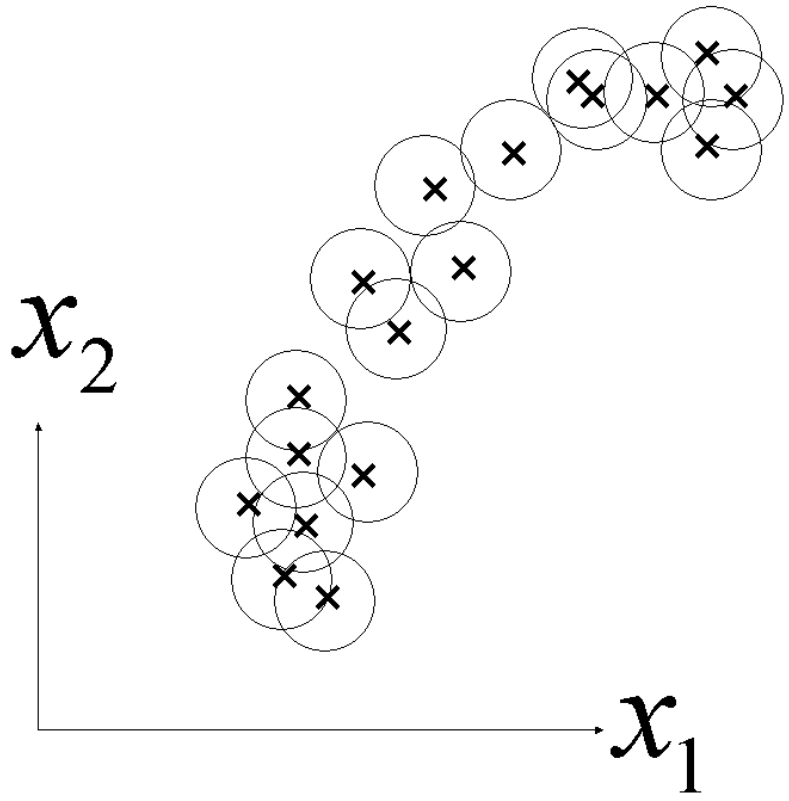
$$\widehat{y}(\mathbf{x}_{i'}) = \text{sign}\left(\sum_{i=1}^{N} v_i k(\mathbf{x}_i, \mathbf{x}_{i'})\right)$$

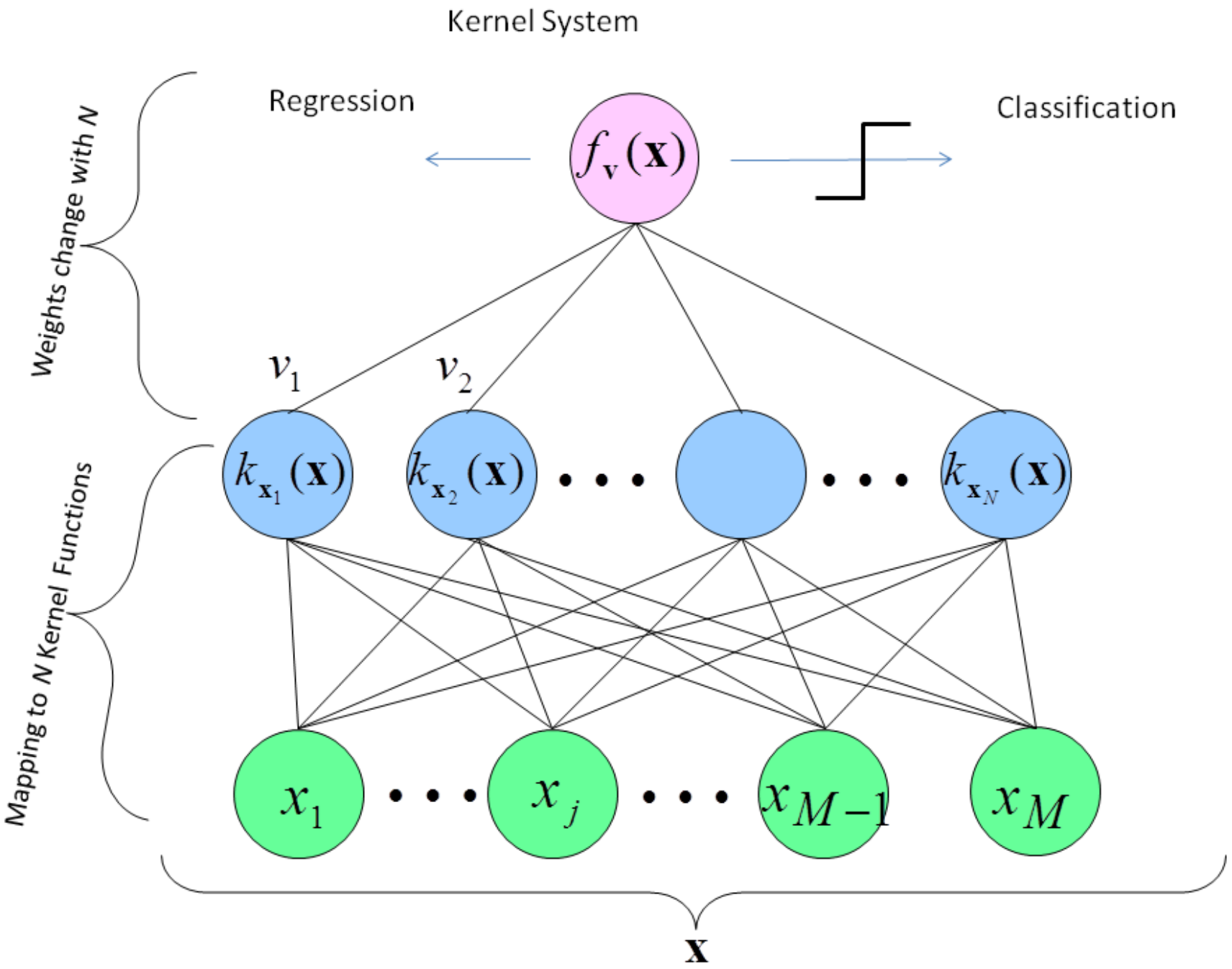
- The solution contains as many kernels as there are data points N (independent on the number of underlying basis functions M_ϕ)

Kernels: Infinite Number of Basis Functions

- Thus with $M_\phi \to \infty$: I can work with a finite number N of kernels, instead of an infinite number of basis functions
- Thus no matter how many training data points: a perfect fit can be possible (with $\lambda/\sigma^2 \to 0$)
- So in neural networks, one makes a model of basis functions more flexible by introducing hidden parameters for tuning the basis functions, with kernels one makes the model more flexible by working with an infinite number of fixed basis functions

One Kernel for Each Data Point




Kernel System
Regression
Classification
$f_{\mathbf{v}}(\mathbf{x})$
Weights change with N
v_1
v_2
$k_{\mathbf{x}_1}(\mathbf{x})$
$k_{\mathbf{x}_2}(\mathbf{x})$
$k_{\mathbf{x}_N}(\mathbf{x})$
Mapping to N Kernel Functions
x_1
x_j
x_{M-1}
x_M
$\mathbf{x}$

Starting with the Cost Function

- We start with the penalized least squares cost function for models with basis functions
- Regularized cost function

$$\text{cost}^{pen}(\mathbf{w}) = \sum_{i=1}^{N}(y_i - \sum_{j=1}^{M_\phi} w_j \phi_j(\mathbf{x}_i))^2 + \lambda \sum_{j=1}^{M_\phi} w_j^2$$

$$= (\mathbf{y} - \mathbf{\Phi w})^T(\mathbf{y} - \mathbf{\Phi w}) + \lambda \mathbf{w}^T\mathbf{w}$$

where $\mathbf{\Phi}$ is the design matrix design with $(\mathbf{\Phi})_{i,j} = \phi_j(\mathbf{x}_i)$.

Implicit Solution

- We calculate the first derivatives and set them to zero,

$$\frac{\partial \text{cost}^{pen}(\mathbf{w})}{\partial \mathbf{w}} = -2\mathbf{\Phi}^T(\mathbf{y} - \mathbf{\Phi}\mathbf{w}) + 2\lambda\mathbf{w} = 0$$

It follows that one can write,

$$\mathbf{w}_{pen} = \frac{1}{\lambda}\mathbf{\Phi}^T(\mathbf{y} - \mathbf{\Phi}\mathbf{w}_{pen})$$

Approach

- This is not an explicit solution ($\mathbf{w}_{pen}$ appears on both sides of the equation). But we know now, that we can write the solution as a linear combination of the input vectors

$$\mathbf{w}_{pen} = \mathbf{\Phi}^T \mathbf{v} = \sum_{i=1}^{N} v_i \vec{\phi}(\mathbf{x}_i)$$

- Note that we have a sum over N data points (and not M_ϕ basis functions)

Kernel Model

- We immediately get,

$$\hat{f}(\mathbf{x}_{i'}) = \sum_{j=1}^{M_\phi} w_{j,pen}\phi_j(\mathbf{x}_{i'}) = \vec{\phi}(\mathbf{x}_{i'})^T \mathbf{w}_{pen}$$

$$= \vec{\phi}(\mathbf{x}_{i'})^T \mathbf{\Phi}^T \mathbf{v} = \sum_{i=1}^{N} v_i k(\mathbf{x}_i, \mathbf{x}_{i'})$$

with $\mathbf{v} = (v_1, \ldots, v_N)^T$ and

$$k(\mathbf{x}_i, \mathbf{x}_{i'}) = \vec{\phi}(\mathbf{x}_i)^T \vec{\phi}(\mathbf{x}_{i'}) = \sum_{j=1}^{M_\phi} \phi_j(\mathbf{x}_i)\phi_j(\mathbf{x}_{i'})$$

- But note that not all functions that can be represented by the basis functions can be written in this form, only the functions that minimize the cost function!

A New Form of the Cost Function

- We can substitute the constraints, and obtain as a cost function with kernel weights as free parameters

$$\text{cost}^{pen}(\mathbf{v}) = (\mathbf{y} - \mathbf{\Phi}\mathbf{\Phi}^T\mathbf{v})^T(\mathbf{y} - \mathbf{\Phi}\mathbf{\Phi}^T\mathbf{v}) + \lambda\mathbf{v}^T\mathbf{\Phi}\mathbf{\Phi}^T\mathbf{v}$$

$$= (\mathbf{y} - \mathbf{K}\mathbf{v})^T(\mathbf{y} - \mathbf{K}\mathbf{v}) + \lambda\mathbf{v}^T\mathbf{K}\mathbf{v}$$

Explicitly

$$\text{cost}^{pen}(\mathbf{v}) = \sum_{i=1}^{N}\left(y_i - \sum_{i'=1}^{N} v_j k(\mathbf{x}_i, \mathbf{x}_{i'})\right)^2 + \lambda\sum_{i=1}^{N}\sum_{i'=1}^{N} v_i v_{i'} k(\mathbf{x}_i, \mathbf{x}_{i'})$$

Here K is an $N \times N$ matrix with elements

$$k_{i,i'} = \vec{\phi}(\mathbf{x}_i)^T \vec{\phi}(\mathbf{x}_{i'}) = \sum_{j=1}^{M_\phi} \phi_j(\mathbf{x}_i)\phi_j(\mathbf{x}_{i'})$$

- An important result: **We can write the cost function, such that only dot products of the basis functions appear (i.e., the kernels), but not the basis functions themselves!**

Kernel Parameters

- Now we can take the derivative of the cost function with respect to $\mathbf{v}$ (note, that $\mathbf{K} = \mathbf{K}^T$)

$$\frac{\partial \mathbf{cost}^{pen}(\mathbf{v})}{\partial \mathbf{v}} = 2\mathbf{K}(\mathbf{K}\mathbf{v} - \mathbf{y}) + 2\lambda\mathbf{K}\mathbf{v}$$

  such that (if $\mathbf{K}$ is full rank)

$$\mathbf{v}_{pen} = (\mathbf{K} + \lambda I)^{-1}\mathbf{y}$$

Kernel Prediction

- A prediction can be written as ($\mathbf{w}, \mathbf{v}$ are the penalized least squares solutions)

$$\widehat{f}(\mathbf{x}_{i'}) = \vec{\phi}(\mathbf{x}_{i'})^T \mathbf{w} = \vec{\phi}(\mathbf{x}_{i'})^T \mathbf{\Phi}^T \mathbf{v} = \sum_{i=1}^{N} v_i k(\mathbf{x}_i, \mathbf{x}_{i'})$$

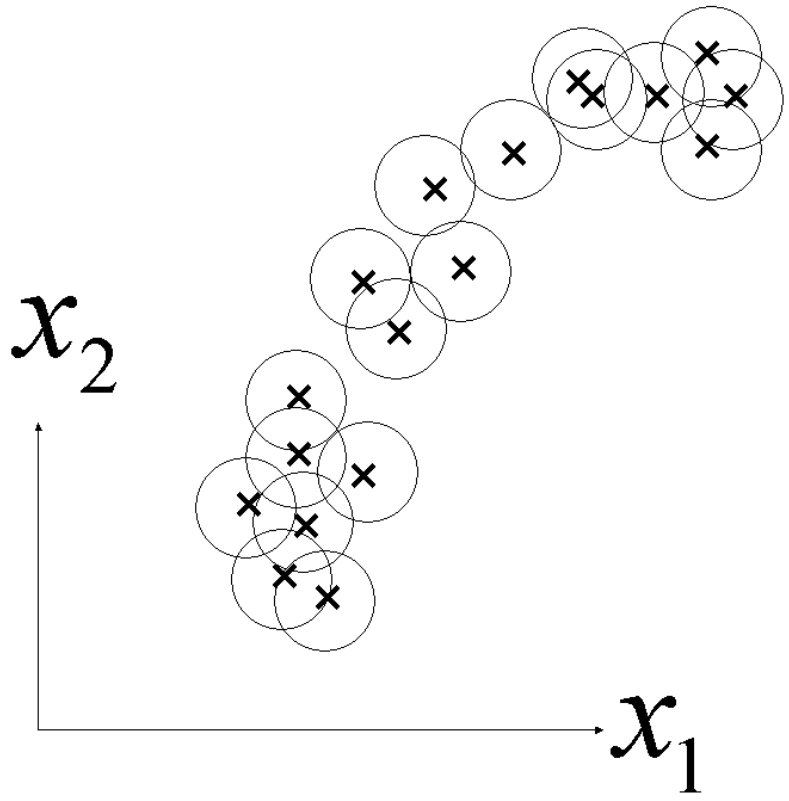
  with

$$k(\mathbf{x}_i, \mathbf{x}_{i'}) = \vec{\phi}(\mathbf{x}_i)^T \vec{\phi}(\mathbf{x}_{i'})^T$$

- Another important result: we can write the solution such that only dot products are used; **the solution can be written as a weighted sum of N kernels**.

- We want to point out again, that *not* each function that can be written as $\sum_j w_j \phi_j(\mathbf{x}_{i'})$ can be expressed in this way, only a subset of the functions and in particular that one which minimizes the cost function based on the specific N training data points!

One Kernel for Each Data Point

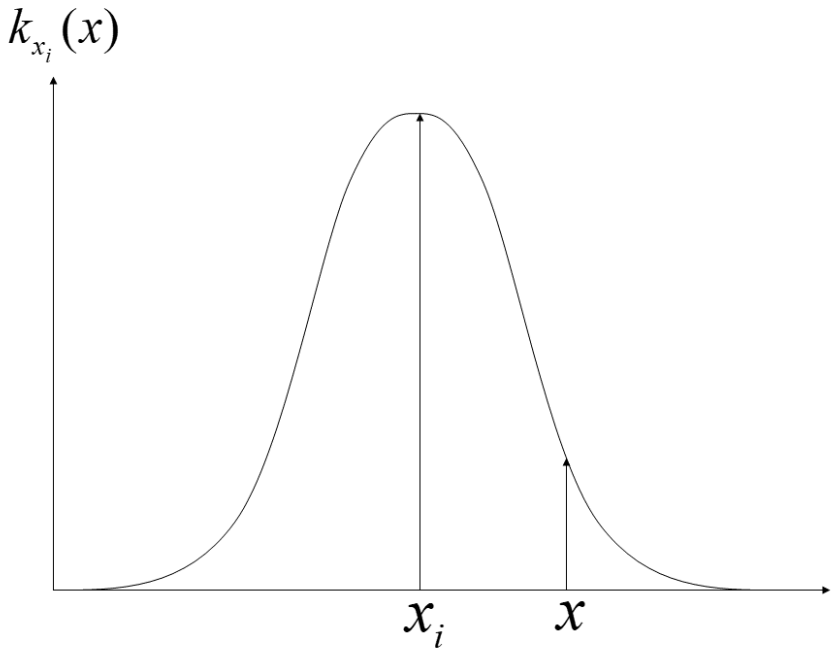


With only One Training Data Point

- With only one training data point we get

$$f(\mathbf{x}_{i'}) = v_1 k(\mathbf{x}_1, \mathbf{x}_{i'})$$

- As discussed previously:



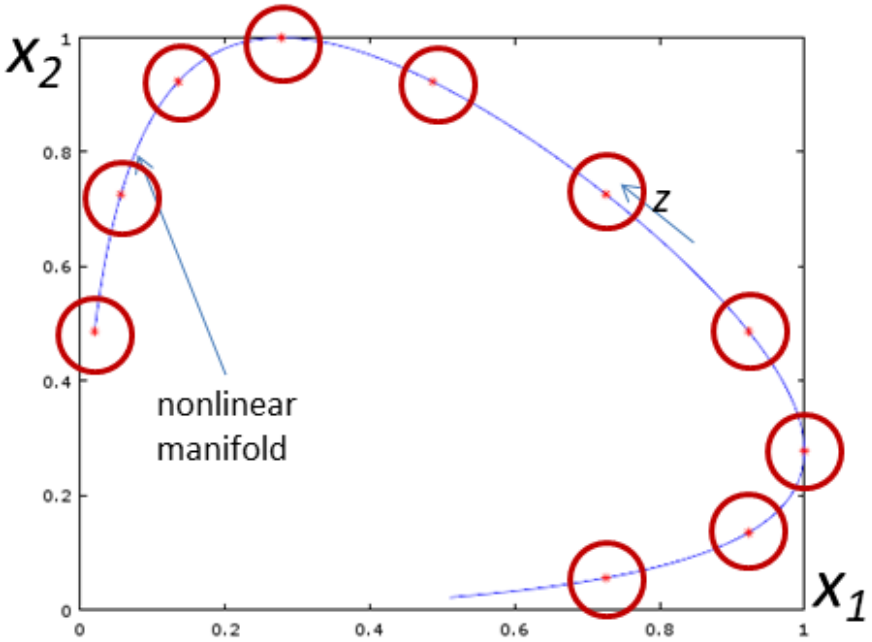
Manifold View

- Computationally, kernel approaches can deal with functions in high dimensions and with a high bandwidth! Computation scales as N^3, independent of M_ϕ and only weakly dependent on M
- If data are on a low-dimensional manifold in high dimensions, Gaussian kernels would be placed on the manifold and the kernel width can be adapted to fit the complexity of the function on the manifold (Case Ib (manifold))
- Thus ϵ can also be quite small **for test data points on the manifold**
- Outside the manifold, $f(\mathbf{x}_{i'}) \to 0$ and ϵ can be quite large; kernel solutions might not perform well when the test data is not on the manifold of the training data

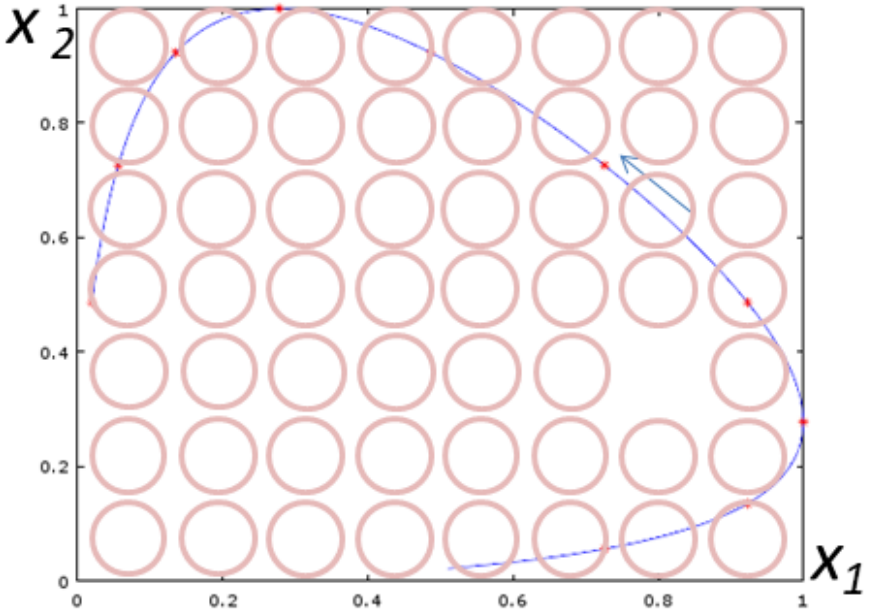
Case Ib

By design, Gaussian kernels are placed on the manifold.
They implicitly perform a dimensionality reduction (by placing themselves on the manifold) , followed by a mapping into infinite basis function space (by the kernels).

Placing Gaussian basis functions on a grid, ignoring the manifold



Gaussian kernels with centers on data points



Gaussian basis functions, uniformly placed

One-class classification (OCC)

- Using the same argument, kernel systems might perform well in one-class classification (one only has data points for class 1 and not data points for class 0)
- The kernel prediction in regions of input space on or closer to the manifold will be larger that in regions of input space far away from the manifold
- Thus if the prediction of a test data point gives an output $>> 0$, the data point is "normal", if the output ≈ 0, the data point might be an outlier (abnormal, novel)
- This can be employed in condition monitoring

Comments and Interpretation of a Kernel

- This is interesting, since there can be more basis functions than data points; in particular this result is valid, even if we work with an **infinite number of basis functions**!
- It is even possible to start with the kernels, without knowing exactly, what the underlying basis functions are

Input Space

- A dot product between to data points in input space is

$$\mathbf{x}_i^T \mathbf{x}_{i'} = \sum_{j=1}^{M} x_{i,j} x_{i',j}$$

  which is the linear kernel

Basis Function Space

- A dot product between two data points in basis function space is

$$\vec{\phi}(\mathbf{x}_i)^T \vec{\phi}(\mathbf{x}_{i'}) = \sum_{j=1}^{M_\phi} \phi_j(\mathbf{x}_i)\phi_j(\mathbf{x}_{i'}) = k(\mathbf{x}_i, \mathbf{x}_{i'})$$

  and this is exactly the kernel which belongs to this basis function space!

Gaussian Process: Prior Mean Function

- Kernel solutions can be derived from different perspectives; we consider Gaussian processes
- Assume that the **prior distribution** of of the **basis function weights** has a zero mean and a unit covariance

$$\mathbf{w} \sim \mathcal{N}(0, I)$$

- Then, a priori, the functions have zero mean $\vec{\phi}(\mathbf{x}) \times 0 = 0$

Gaussian Process: Prior Covariance

- Then, a priori, the covariance matrix of the functional values at two inputs is

$$\Sigma_{(f(\mathbf{x}_i);f(\mathbf{x}_{i'}))} = \begin{pmatrix} \vec{\phi}^T(\mathbf{x}_i)\Sigma_{\mathbf{w}}\vec{\phi}(\mathbf{x}_i) & \vec{\phi}^T(\mathbf{x}_i)\Sigma_{\mathbf{w}}\vec{\phi}(\mathbf{x}_{i'}) \\ \vec{\phi}^T(\mathbf{x}_{i'})\Sigma_{\mathbf{w}}\vec{\phi}(\mathbf{x}_i) & \vec{\phi}^T(\mathbf{x}_{i'})\Sigma_{\mathbf{w}}\vec{\phi}(\mathbf{x}_{i'}) \end{pmatrix}$$

$$= \begin{pmatrix} k(\mathbf{x}_i, \mathbf{x}_i) & k(\mathbf{x}_i, \mathbf{x}_{i'}) \\ k(\mathbf{x}_{i'}, \mathbf{x}_i) & k(\mathbf{x}_{i'}, \mathbf{x}_{i'}) \end{pmatrix}$$

- Special case: $\text{var}_{prior}(f(\mathbf{x}_i)) = k(\mathbf{x}_i, \mathbf{x}_i)$
- This interpretation is used in Gaussian processes: the kernel represents the covariance between the function values, evaluated at different inputs

Gaussian Process: Prediction

- Note, how cool this is: we can deal with probabilities over functions, not just functions themselves
- $f(\cdot)$ has an infinite number of functional values; $P(f(\cdot))$ is describes as an infinite Gaussian with zero mean (the most likely function a priori is $f(\mathbf{x}) = 0$) and a nontrivial covariance, described by the above equations (a priori: before training data points are available)
- A posteriori, the mean becomes the prediction as described

$$\mu_p(\mathbf{x}_{i'}) = \sum_{i=1}^{N} v_i^{pen} k(\mathbf{x}_i, \mathbf{x}_{i'})$$

with (as before) $\mathbf{v}^{pen} = \mathbf{K}^{-1}\mathbf{y}$ (we assume zero noise for the measurements)

Gaussian Process: Uncertainty in the Prediction

- The a posteriori covariance kernel becomes

$$k_p(\mathbf{x}_i, \mathbf{x}_{i'}) = k(\mathbf{x}_i, \mathbf{x}_{i'}) - \mathbf{k}^T(\mathbf{x}_i)\mathbf{K}^{-1}\mathbf{k}(\mathbf{x}_{i'})$$

  where $\mathbf{k}(\mathbf{x}_i)$ is a vector containing the kernel values between $\mathbf{x}_i$ and all training data points

- Special case:

$$\text{var}_{post}((f(\mathbf{x}_i')) = \text{var}_{prior}((f(\mathbf{x}_i')) - \mathbf{k}^T(\mathbf{x}_i')\mathbf{K}^{-1}\mathbf{k}(\mathbf{x}_i')$$

- Gaussian processes are popular in safety critical applications, like robotics: they provide an estimate of the uncertainty of the prediction and they "know when they don't know"

- With noisy measurements, set $\mathbf{K} \leftarrow \mathbf{K} + \sigma^2\mathbf{I}$ in the previous equations

Computational Complexity

- When $N >> M_\phi$ it is computationally more efficient to work with basis functions (requiring $M_\phi^3 + M_\phi^2 N$ operations). When $M_\phi >> N$, the kernel version is more efficient, requiring $N^3 + N^2 M_\phi$ operations. If the kernels are known a priori (i.e., if they do not need to be calculates via dot product), the kernel solution requires N^3 operations.

Mercer's Theorem

- Still, not all functions are valid kernel functions. Mercer's theorem addresses that issue
- (From Vapnik: The nature of statistical learning theory. Springer, 2000)
- *Mercer's Theorem:* To guarantee, that the symmetric functions $k(\mathbf{x}_i, \mathbf{x}_{i'}) = k(\mathbf{x}_{i'}, \mathbf{x}_i)$ from L_2 permits an expansion as

$$k(\mathbf{x}_i, \mathbf{x}_{i'}) = \sum_{h=1}^{\infty} \lambda_h \phi_h^T(\mathbf{x}_i)\phi_h(\mathbf{x}_{i'})$$

with positive coefficients $\lambda_h > 0$, it is necessary and sufficient, that

$$\int \int k(\mathbf{x}_i, \mathbf{x}_{i'}) g(\mathbf{x}_i) g(\mathbf{x}_{i'})\, d\mathbf{x}_i d\mathbf{x}_{i'} > 0$$

for all $g \neq 0$, for which

$$\int g^2(\mathbf{x}_{i'}) d\mathbf{x}_{i'} < \infty$$

- The theorem says, that for so-called positive-definite kernels ("Mercer kernels"), a decomposition in basis functions is possible!
- Each kernel-matrix $\mathbf{K}$ is then also positive definite, $\mathbf{a}^T\mathbf{K}\mathbf{a} > 0$, for all vectors $\mathbf{a} \neq 0$. A symmetric matrix is positive definite iff all its eigenvalues are positive
- The results also generalize to the non-negative (positive-semidefinite) case

Kernel Design

- Linear Kernel

$$k(\mathbf{x}_i, \mathbf{x}_{i'}) = \mathbf{x}_i^T \mathbf{x}_{i'}$$

The kernel matrix is then $\mathbf{K} = \mathbf{X}\mathbf{X}^T$. $\mathbf{X}$ is the design matrix $(N \times M)$. (Recall that the empirical correlation between the input dimensions is $\mathbf{X}^T\mathbf{X}$)

- Polynomial kernel (1)

$$k(\mathbf{x}_i, \mathbf{x}_{i'}) = (\mathbf{x}_i^T \mathbf{x}_{i'})^d$$

The basis functions are all ordered polynomials of order d

- Polynomial kernel (2)

$$k(\mathbf{x}_i, \mathbf{x}_{i'}) = (\mathbf{x}_i^T \mathbf{x}_{i'} + R)^d$$

The corresponding basis functions are all polynomials of order d **or smaller**. R is a tuning parameter

- Gauss-kernels (RBF-kernels)

$$k(\mathbf{x}_i, \mathbf{x}_{i'}) = \exp\left(-\frac{1}{2s^2}\|\mathbf{x}_i - \mathbf{x}_{i'}\|^2\right)$$

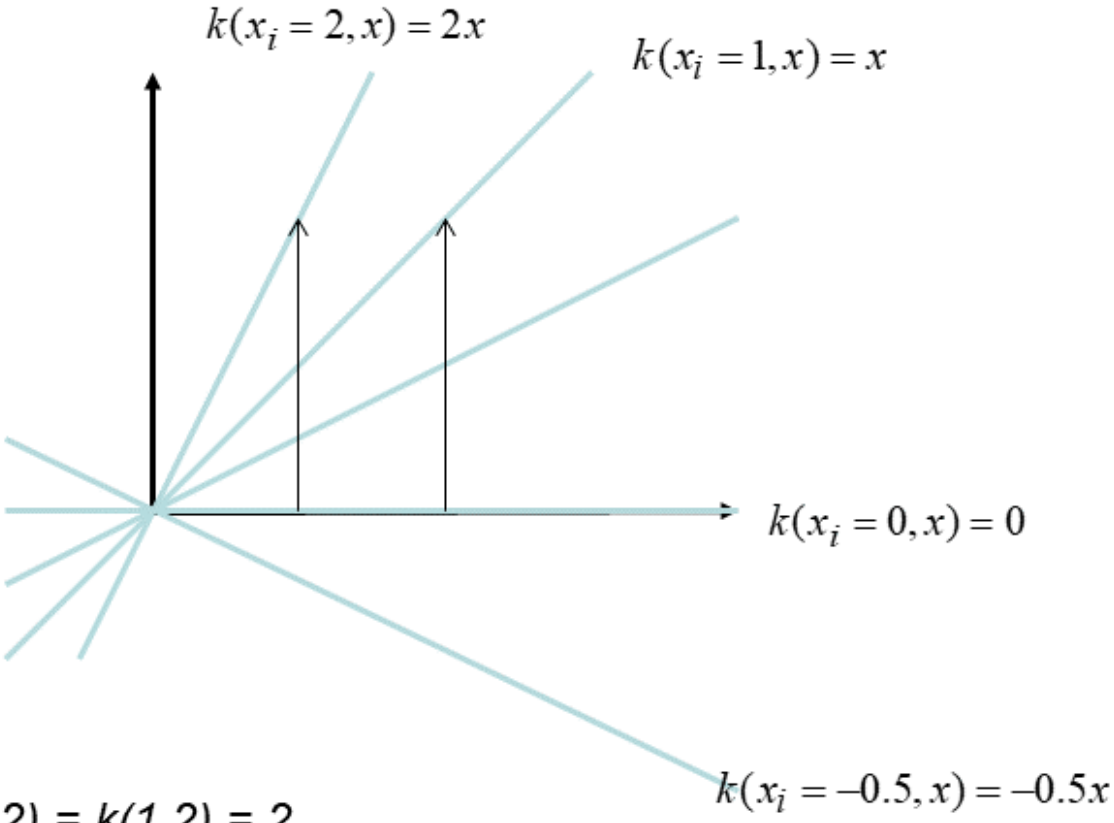
  These kernels correspond to infinitely many Gaussian basis functions

- Sigmoid ("neural network") kernels

$$k(\mathbf{x}_i, \mathbf{x}_{i'}) = \text{sig}\left(\mathbf{x}_i^T \mathbf{x}_{i'}\right)$$

Kernels do not need to look symmetrical: linear kernel in 1-D

$$k(x_i, x) = x_i x$$



Symmetry: *k(1,2) = k(1,2) = 2*

Generalization

- Recall that

$$f(\mathbf{x}_i) = \vec{\phi}^T(\mathbf{x}_i)\mathbf{w}$$

$$f(\mathbf{x}_{i'}) = \vec{\phi}^T(\mathbf{x}_{i'})\mathbf{w}$$

- Then one can write

$$k(\mathbf{x}_i, \mathbf{x}_{i'}) = \vec{\phi}(\mathbf{x}_i)^T\vec{\phi}(\mathbf{x}_{i'}) = \left(\frac{\partial f(\mathbf{x}_i)}{\partial \mathbf{w}}\right)^T \left(\frac{\partial f(\mathbf{x}_{i'})}{\partial \mathbf{w}}\right)$$

  which generalizes, to nonlinear models, e.g., to neural networks

- This is then called a neural tangent kernel

Comment on Valid Kernels

- A necessary condition is that $k(\mathbf{x}_i, \mathbf{x}_{i'}) = k(\mathbf{x}_{i'}, \mathbf{x}_i)$!
- So any function of $\|\mathbf{x}_i - \mathbf{x}_{i'}\|$ would be a good candidate. These kernels also appear symmetrical, like a Gaussian kernel
- But note that also any function of $\mathbf{x}_i^T \mathbf{x}_{i'}$ would be a good candidate. They don't necessarily look symmetrical, like the linear kernel or the polynomial kernel
- Here is an example of a kernel that violates the necessary condition

$$k(\mathbf{x}_i, \mathbf{x}_{i'}) = \mathbf{x}_i^T \mathbf{x}_{i'} + \alpha \|\mathbf{x}_i\|^2$$

- The kernels discussed here are called dot-product kernels, Mercer kernels, or kernels in a reproducing kernel Hilbert space
- Kernels are widely used in mathematics. The kernels used here should, for example, not to be confused with the kernels used in kernel smoothing!

Sometimes it is Easier to Define Sensible Kernels than it is to Define Sensible Basis Functions

- Example: Classification of chemical graphs
  - Molecules can be described as graphs (structural formula, chemical graph theories)
  - Task: I know from N molecules, if these have a particular medical effect (training data). Can I predict the medical effect of a new molecule?
  - Features which describe a chemical structure formula are difficult to describe; it is easier to define graph kernels
- Example: Classification of a person in a social network
  - Kernels reflect similarity with respect to a network topology. For example, one can define a kernel based on the number of overlapping substructures of two persons in their mutual neighborhoods
  - A simple example: one forms a feature vector, where $\phi_{i,i'} = 1$ if node i is a neighbor of node i', and is zero otherwise; if a linear kernel is used, it simply counts the number of common neighbors

Representer Theorem

- *Representer Theorem:* Let Ω be a strictly monotonously increasing function and let loss() be an arbitrary loss function, then the minimizer of the loss function

$$\sum_{i=1}^{N} \text{loss}(y_i, f(\mathbf{x}_i)) + \Omega(\|\mathbf{f}\|_\phi)$$

  can be represented as

$$f(\mathbf{x}_{i'}) = \sum_{i=1}^{N} v_i k(\mathbf{x}_i, \mathbf{x}_{i'})$$

- $\|\mathbf{f}\|_\phi = \sqrt{\langle \mathbf{f}, \mathbf{f} \rangle_\phi}$ is a norm in a *reproducing kernel Hilbert space* (RKHS) and includes $\|\mathbf{f}\|_\phi = \sqrt{\mathbf{w}^T \mathbf{w}}$
- So kernel solutions are possible for **all cost functions we are considering**!

Dilemma

- Kernel approaches work well with data on low-dimensional manifolds (as discussed); this might be their real strength! (Case Ib (manifold))
- Now consider that data is not on a manifold and a Gaussian kernel:
- If I assume the function is complex (high bandwidth) then I need many fixed basis functions and a Gaussian kernel should have small width
- If I have a small kernel width, the kernel model will mostly predict zero, when the test data point is a bit different from the training data point
- I would need a huge number of training data points to model the function well, but then kernel models do not handle large data sets well
- Essentially I have to "assume" that the function has low bandwith and I use a kernel with a large width; but then I am back to Cases II/III, which are not really challenging