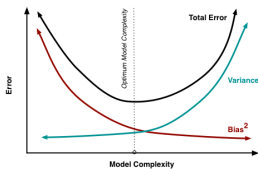


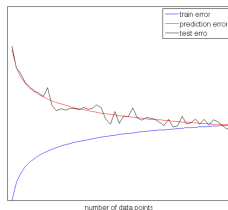
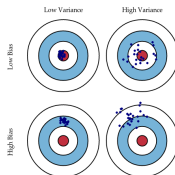
# Machine Learning

## Kernel Methods



Marcello Restelli

April 11, 2024



# Outline

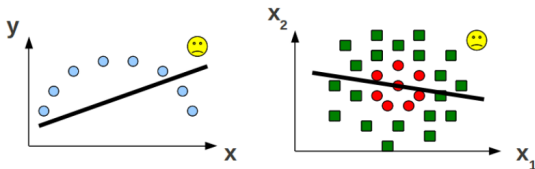
- 1 Kernel Methods
  - Dual Representation
  - Constructing Kernels
- 2 Radial Basis Functions Networks
- 3 Gaussian Processes

# Kernel Methods

- Kernel methods are memory-**based** (like nearest neighbor)
  - **Training data** are used in **prediction phase**
  - **Fast** to train, **slow** to predict
  - Require a **metric** to be defined

# Motivations

- Often we want to **capture nonlinear patterns** in the data
  - Nonlinear Regression: input-output relationship may not be linear
  - Nonlinear Classification: Classes may not be separable by a linear boundary
- Linear models are **not just rich enough**



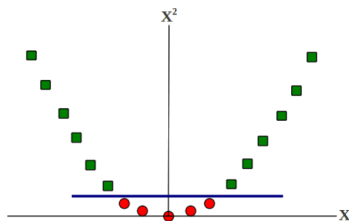
- **Kernels**: make linear models work in nonlinear settings
  - By **mapping data to higher dimensions** where it exhibits linear patterns
  - Apply the **linear model** in the **new input space**
  - Mapping  $\equiv$  changing the **feature representation**
- Mappings can be **expensive** to compute
- Kernels give such mappings **for (almost) free: Kernel trick!**

# Example: 1D

- Consider this binary classification problem

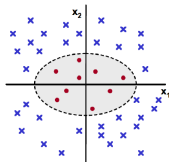


- Each example represented by a **single feature**  $x$
- No linear separator** exists for this data
- Now map each example as  $x \rightarrow \{x, x^2\}$ 
  - Each example now has **two features** (derived from the old representation)
- Data now becomes **linear separable** in the **new representation**

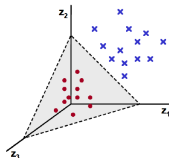


# Example: 2D

- Let's look at another example:



- Each example defined by a two features  $\mathbf{x} = \{x_1, x_2\}$
- No linear separator** exists for this data
- Now map each example as  $\mathbf{x} = \{x_1, x_2\} \rightarrow \mathbf{z} = \{x_1^2, \sqrt{2}x_1x_2, x_2^2\}$ 
  - Each example now has **three features**
- Data now becomes **linearly separable** in the **new representation**



# Feature Mapping

- Consider the following mapping  $\phi$  for an example  $\mathbf{x} = \{x_1, \dots, x_M\}$

$$\phi : \mathbf{x} \rightarrow \{x_1^2, x_2^2, \dots, x_M^2, x_1x_2, x_2x_3, \dots, x_1x_M, \dots, x_{M-1}x_M\}$$

- It's an example of a **quadratic** mapping
  - Each new feature uses a **pair** of the original features
- Problem:** Mapping usually leads to the number of features **blow up!**
  - Computing the mapping itself can be **inefficient**
  - Moreover, **using** the mapped representation could be inefficient too
- Thankfully, **kernels** help avoid both these issues!
  - The mapping doesn't have to be **explicitly** computed
  - Computations with the mapped features remain **efficient**

# Kernel Functions

- Many linear **parametric** models can be re-cast into equivalent **dual representations** where predictions are based on a **kernel function** evaluated at **training points**
- Kernel function is given by

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

- where  $\phi(\mathbf{x})$  is a fixed nonlinear feature space mapping (**basis function**)
- Kernel is a **symmetric** function of its arguments

$$k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$$

- Kernel function can be interpreted as **similarity** of  $\mathbf{x}$  and  $\mathbf{x}'$
- Simplest is **identity mapping** in feature space:  $\phi(\mathbf{x}) = \mathbf{x}$ 
  - In which case  $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$
  - Called **linear kernel**



# Kernel Trick

- Formulated as **inner product** allows extending well-known algorithms
  - by using the **kernel trick**
- **Basic idea** of kernel trick
  - If an input vector  $\mathbf{x}$  appears only in the form of **scalar products** then we can replace scalar products with some other choice of kernel
- **Used widely**
  - Ridge Regression
  - Perceptron
  - Nonlinear variant of PCA
  - Support Vector Machines
  - Lots more...

# Other Forms of Kernel Functions

- Function of difference between arguments

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

- Called **stationary kernel** since invariant to translation in space
- **Homogeneous kernels**, also known as **radial basis functions**

$$k(\mathbf{x}, \mathbf{x}') = k(\|\mathbf{x} - \mathbf{x}'\|)$$

- Depend only on the magnitude of the **distance** between arguments
- Note that the kernel function is a **scalar** value while  $\mathbf{x}$  is an  $M$ -dimensional **vector**
- The kernel functions are valid if can be expressed as
$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$$

# Dual Representation

- Many linear models for regression and classification can be reformulated in terms of dual representation in which the **kernel function arises naturally**
- Plays important role in SVMs (we see this later)
- Consider linear regression model
  - The parameters are obtained by minimizing **regularized sum-of-squares** error function

$$L_{\mathbf{w}} = \frac{1}{2} \sum_{n=1}^N (\mathbf{w}^T \phi(\mathbf{x}_n) - t_n)^2 + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}$$

- Setting the gradient of  $L_{\mathbf{w}}$  with respect to  $\mathbf{w}$  equal to zero:

$$\mathbf{w} = -\frac{1}{\lambda} \sum_{n=1}^N (\mathbf{w}^T \phi(\mathbf{x}_n) - t_n) \phi(\mathbf{x}_n) = \sum_{n=1}^N a_n \phi(\mathbf{x}_n) = \Phi^T \mathbf{a}$$

- $\Phi$  is the design matrix whose  $n^{th}$  row is  $\phi(\mathbf{x}_n)^T$
- The coefficients  $a_n$  are functions of  $\mathbf{w}$ :  $a_n = -\frac{1}{\lambda} (\mathbf{w}^T \phi(\mathbf{x}_n) - t_n)$

# Gram Matrix and Kernel Function

- Define the **Gram matrix**  $K = \Phi \times \Phi^T$  an  $N \times N$  matrix, with elements

$$K_{nm} = \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m)$$

- Given  $N$  vectors, the Gram Matrix is the matrix of all **inner products**

$$K = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & \dots & k(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix}$$

- Notes:
  - $\Phi$  is  $N \times M$  and  $K$  is  $N \times N$
  - $K$  is a matrix of **similarities** of pairs of samples (is **symmetric**)

# Error Function in Terms of Gram Matrix of Kernel

- Substituting  $\mathbf{w} = \Phi^T \mathbf{a}$  into  $L_{\mathbf{w}}$  gives

$$L_{\mathbf{w}} = \frac{1}{2} \mathbf{a}^T \Phi \Phi^T \Phi \Phi^T \mathbf{a} - \mathbf{a}^T \Phi \Phi^T \mathbf{t} + \frac{1}{2} \mathbf{t}^T \mathbf{t} + \frac{\lambda}{2} \mathbf{a}^T \Phi \Phi^T \mathbf{a}$$

where  $\mathbf{t} = (t_1, \dots, t_N)^T$

- Sum of squares error function is written in terms of Gram matrix as

$$L_{\mathbf{a}} = \frac{1}{2} \mathbf{a}^T K K \mathbf{a} - \mathbf{a}^T K \mathbf{t} + \frac{1}{2} \mathbf{t}^T \mathbf{t} + \frac{\lambda}{2} \mathbf{a}^T K \mathbf{a}$$

- Solving for  $\mathbf{a}$  by combining  $\mathbf{w} = \Phi^T \mathbf{a}$  and  $a_n = -1/\lambda(\mathbf{w}^T \phi(\mathbf{x}_n) - t_n)$

$$\mathbf{a} = (K + \lambda \mathbf{I}_N)^{-1} \mathbf{t}$$

- Solution for  $\mathbf{a}$  can be expressed as a linear combination of elements of  $\phi(\mathbf{x})$  whose coefficients are **entirely in terms of kernel**  $k(\mathbf{x}, \mathbf{x}')$  from which we can recover original formulation in terms of parameters  $\mathbf{w}$

# Prediction Function

- Prediction for new input  $\mathbf{x}$ 
  - We can write  $\mathbf{a} = (K + \lambda \mathbf{I}_N)^{-1} \mathbf{t}$
  - Substituting back into linear regression model

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) = \mathbf{a}^T \Phi \phi(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T (K + \lambda \mathbf{I}_N)^{-1} \mathbf{t}$$

where  $\mathbf{k}(\mathbf{x})$  has elements  $k_n(\mathbf{x}) = k(\mathbf{x}_n, \mathbf{x})$

- Prediction is a linear combination of the **target values** from the **training set**

# Advantage of Dual Representation

- Solution for  $\mathbf{a}$  is expressed entirely in terms of kernel function  $k(\mathbf{x}, \mathbf{x}')$
- Once we get  $\mathbf{a}$  we can recover  $\mathbf{w}$  as linear combination of elements of  $\phi(\mathbf{x})$  using  $\mathbf{w} = \Phi^T \mathbf{a}$
- In parametric formulation, solution is  $\mathbf{w}_{ML} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{t}$ 
  - Instead of inverting an  $M \times M$  matrix we are inverting an  $N \times N$  matrix (an **apparent disadvantage**)
- But, advantage of dual formulation is that we can work with kernel function  $k(\mathbf{x}, \mathbf{x}')$  and therefore
  - **avoid** working with a feature vector  $\phi(\mathbf{x})$  and
  - problems associated with **very high or infinite dimensionality** of  $\mathbf{x}$
  - kernel functions can be defined not only over simply vectors of real numbers, but also **over objects** as diverse as graphs, sets, string, and text documents

# Constructing Kernels

- To exploit kernel substitution need **valid** kernel functions
- First method
  - **Choose a feature space** mapping  $\phi(\mathbf{x})$  and use it to find corresponding kernel
  - One-dimensional input space

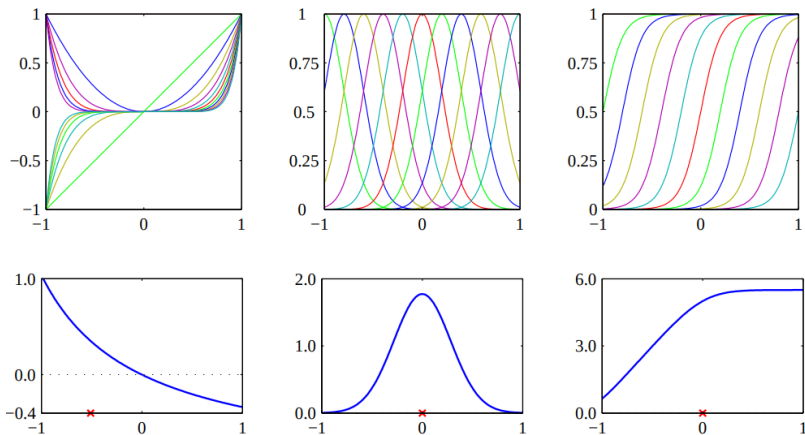
$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}') = \sum_{i=1}^M \phi_i(\mathbf{x}) \phi_i(\mathbf{x}')$$

- where  $\phi(\mathbf{x})$  are basis functions such as polynomial
- for each  $i$  we choose  $\phi_i(\mathbf{x}) = \mathbf{x}^i$



# Construction of Kernel Functions from Basis Functions

## One-dimensional input space



## Second Method: Direct Construction of Kernels

- Function we choose has to correspond to a **scalar product** in some (perhaps infinite dimensional) space
- Consider kernel function  $k(x, z) = (x^T z)^2$

- In **two dimensional space**

$$\begin{aligned} k(x, z) &= (\mathbf{x}^T \mathbf{z})^2 = (x_1 z_1 + x_2 z_2)^2 = x_1^2 z_1^2 + 2x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\ &= (x_1^2, \sqrt{2}x_1 x_2, x_2^2)(z_1^2, \sqrt{2}z_1 z_2, z_2^2)^T = \phi(\mathbf{x})^T \phi(\mathbf{z}) \end{aligned}$$

- Feature mapping takes the form  $\phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1 x_2, x_2^2)$
- Comprises of **all second order terms** with a specific weighting
  - Inner product needs computing six feature values and  $3 \times 3 = 9$  multiplications
  - Kernel function  $k(\mathbf{x}, \mathbf{z})$  has 2 multiplications and a squaring
- By considering  $(\mathbf{x}^T \mathbf{z} + c)^2$  we get constant, linear, second order terms
- By considering  $(\mathbf{x}^T \mathbf{z} + c)^p$  we get all terms up to degree  $p$

# Testing whether a function is a valid kernel

- Without having to construct the function  $\phi(\mathbf{x})$  explicitly
- **Necessary and sufficient** condition for a function  $k(\mathbf{x}, \mathbf{x}')$  to be a kernel is
  - Gram matrix  $K$ , whose elements are given by  $k(\mathbf{x}_n, \mathbf{x}_m)$  is positive semi-definite for all possible choices of the set  $\{\mathbf{x}_n\}$ 
    - Positive semi-definite is **not** the same thing as a matrix whose elements are non-negative
    - It means  $\mathbf{x}^T K \mathbf{x} \geq 0$  for non-zero vectors  $\mathbf{x}$  with real entries, i.e.,  

$$\sum_n \sum_m K_{n,m} \mathbf{x}_n \mathbf{x}_m \geq 0$$
 for any real numbers  $\mathbf{x}_n, \mathbf{x}_m$

## Theorem

*Mercer's theorem* **Any** continuous, symmetric, positive semi-definite kernel function  $k(x, y)$  can be expressed as a **dot product** in a high-dimensional space

- **New kernels** can be constructed from simpler kernels as **building blocks**

# Techniques for Constructing Kernels

Given valid kernels  $k_1(\mathbf{x}, \mathbf{x}')$  and  $k_2(\mathbf{x}, \mathbf{x}')$  the following new kernels will be valid

- ❶  $k(\mathbf{x}, \mathbf{x}') = ck_1(\mathbf{x}, \mathbf{x}')$
- ❷  $k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})k_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}')$ , where  $f(\cdot)$  is any function
- ❸  $k(\mathbf{x}, \mathbf{x}') = q(k_1(\mathbf{x}, \mathbf{x}'))$ , where  $q(\cdot)$  is a polynomial with non-negative coefficients
- ❹  $k(\mathbf{x}, \mathbf{x}') = \exp(k_1(\mathbf{x}, \mathbf{x}'))$
- ❺  $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')$
- ❻  $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}')$
- ❼  $k(\mathbf{x}, \mathbf{x}') = k_3(\phi(\mathbf{x}), \phi(\mathbf{x}'))$ , where  $\phi(\mathbf{x})$  is a function from  $\mathbf{x}$  to  $\mathbb{R}^M$
- ❽  $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T A \mathbf{x}'$ , where  $A$  is a symmetric positive semidefinite matrix
- ❾  $k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b)$ , where  $x_a$  and  $x_b$  are variables with  $\mathbf{x} = (x_a, x_b)$
- ❿  $k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}'_a)k_b(\mathbf{x}_b, \mathbf{x}'_b)$

# Gaussian Kernel

- Commonly used kernel is

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2 / 2\sigma^2)$$

- It is seen as a valid kernel by expanding the square

$$\|\mathbf{x} - \mathbf{x}'\|^2 = \mathbf{x}^T \mathbf{x} + \mathbf{x}'^T \mathbf{x}' - 2\mathbf{x}^T \mathbf{x}'$$

- To give

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

- From kernel construction rules 2 and 4, together with validity of linear kernel  $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$
- Can be extended to non-Euclidean distances

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2\sigma^2}(\kappa(\mathbf{x}, \mathbf{x}) + \kappa(\mathbf{x}', \mathbf{x}') - 2\kappa(\mathbf{x}, \mathbf{x}'))\right)$$

# Kernels for Symbolic Data

- Kernels can be extended to inputs that are **symbolic**, rather than simply vectors of real numbers
- Kernel functions can be defined over objects as diverse as graphs, sets, strings, and text documents
- Consider a simple kernel over sets:

$$k(A_1, A_2) = 2^{|A_1 \cap A_2|}$$

# Kernels based on Generative Models

- Given a generative model  $p(\mathbf{x})$  we define a kernel by

$$k(\mathbf{x}, \mathbf{x}') = p(\mathbf{x})p(\mathbf{x}')$$

- A valid kernel since it is an inner product in the one-dimensional feature space defined by the mapping  $p(\mathbf{x})$
- Two inputs  $\mathbf{x}$  and  $\mathbf{x}'$  are **similar** if they have high probabilities

# Radial Basis Function Networks

- **Radial basis functions:** each basis function depends only on the radial distance (typically Euclidean) from a center

$$\phi_j(\mathbf{x}) = h(\|\mathbf{x} - \boldsymbol{\mu}_j\|_2)$$

- Used for **exact interpolation**:

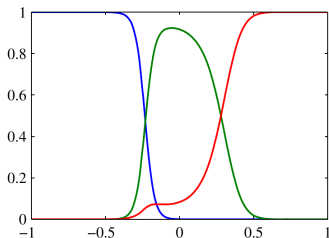
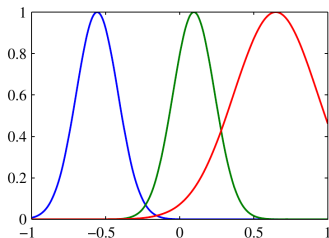
$$f(\mathbf{x}) = \sum_{n=1}^N w_n h(\|\mathbf{x} - \mathbf{x}_n\|_2)$$

- Because the data in ML are generally **noisy**, exact interpolation is not very useful



# Normalized Basis Functions

- Uses **normalized** radial functions as basis
- Normalization is sometimes used in practice as it avoids having regions of input space where all basis functions take **small values**, which would necessarily lead to **predictions** in such regions that are either **small** or controlled purely by the **bias parameter**



# Nadaraya-Watson Model

- Given a training set  $\{\mathbf{x}_n, t_n\}$  the **joint distribution** of the two variables can be estimated with **Parzen window**:

$$p(\mathbf{x}, t) = \frac{1}{N} \sum_{n=1}^N f(\mathbf{x} - \mathbf{x}_n, t - t_n)$$

- We want to find the regression function  $y(\mathbf{x})$

$$\begin{aligned} y(\mathbf{x}) &= \mathbb{E}[t|\mathbf{x}] = \int_{-\infty}^{\infty} t p(t|\mathbf{x}) dt = \frac{\int t p(\mathbf{x}, t) dt}{\int p(\mathbf{x}, t) dt} \\ &= \frac{\sum_{n=1}^N g(\mathbf{x} - \mathbf{x}_n) t_n}{\sum_{m=1}^N g(\mathbf{x} - \mathbf{x}_m)} = \sum_{n=1}^N k(\mathbf{x}, \mathbf{x}_n) t_n \end{aligned}$$

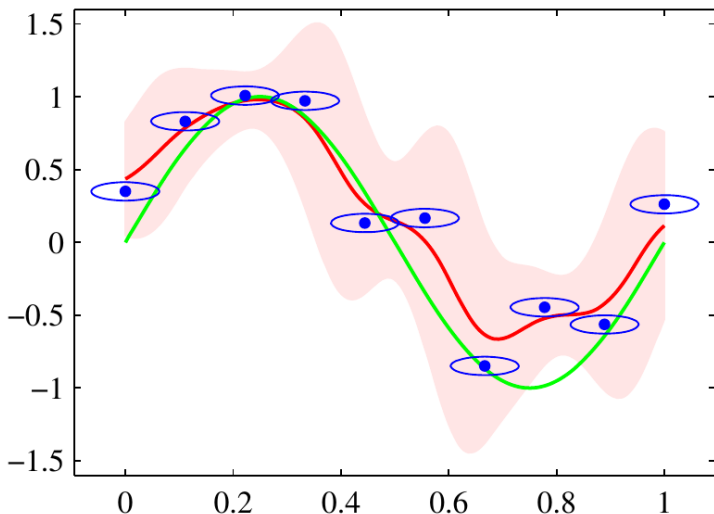
- where  $k(\mathbf{x}, \mathbf{x}_n) = \frac{g(\mathbf{x} - \mathbf{x}_n)}{\sum_m g(\mathbf{x} - \mathbf{x}_m)}$  and  $g(\mathbf{x}) = \int_{-\infty}^{\infty} f(\mathbf{x}, t) dt$

# Nadaraya-Watson Model

- Also called **kernel regression**
- For a localized kernel function, it has the property of giving **more weight** to the data points  $\mathbf{x}_n$  that are **close** to  $\mathbf{x}$
- The model defines not only a conditional expectation, but also a **full conditional distribution**

# Example

Isotropic Gaussian kernels centered around the data points  $z_n = (x_n, t_n)$



# Gaussian Processes

- We have seen kernels as a **dual model** for a **non-probabilistic** model for regression
- Extend kernels to **probabilistic discriminative models**
- In linear model for regression, we have introduced a prior distribution over  $\mathbf{w}$
- Given the training data set, we evaluated the posterior distribution over  $\mathbf{w} \Rightarrow$  posterior distribution over the regression functions  $\Rightarrow$  predictive distribution  $p(t|\mathbf{x})$  for new input  $\mathbf{x}$

# Gaussian Processes

- Now we define a **prior** probability distribution **over functions directly**
- Might seem difficult to work with a distribution over the uncountable infinite space of functions
- However, for a **finite** training set, we only need to consider the values of the function at the discrete set of input values  $\mathbf{x}_n$  of the training set
- So, in practice, we can work in **finite space**

# Revisiting Linear Regression

- Let's apply kernels to probabilistic discriminative models

$$y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \phi(\mathbf{x})$$

- Instead of prior over  $\mathbf{w}$ , let's define a prior over functions directly

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w} | \mathbf{0}, \tau \mathbf{I})$$

$$\mathbf{y} = \Phi \mathbf{w}$$

$$p(\mathbf{y}) = ?$$

- $\mathbf{y}$  is a **linear combination of Gaussian** distributed variables given by the elements of  $\mathbf{w}$  and hence is itself **Gaussian**

# Revisiting Linear Regression

- Thus:

$$\begin{aligned}\mathbb{E}[\mathbf{y}] &= \mathbf{\Phi} \mathbb{E}[\mathbf{w}] = \mathbf{0} \\ \text{cov}[\mathbf{y}] &= \mathbb{E}[\mathbf{y}\mathbf{y}^T] = \mathbf{\Phi} \mathbb{E}[\mathbf{w}\mathbf{w}^T] \mathbf{\Phi}^T = \tau \mathbf{\Phi} \mathbf{\Phi}^T = \mathbf{K}\end{aligned}$$

- where  $\mathbf{K}$  is the Gram matrix with elements

$$K_{nm} = k(\mathbf{x}_n, \mathbf{x}_m) = \tau \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m)$$

- So the marginal distribution  $p(\mathbf{y})$  is defined by a Gram matrix so that  $p(\mathbf{y}) = \mathcal{N}(\mathbf{y} | \mathbf{0}, \mathbf{K})$



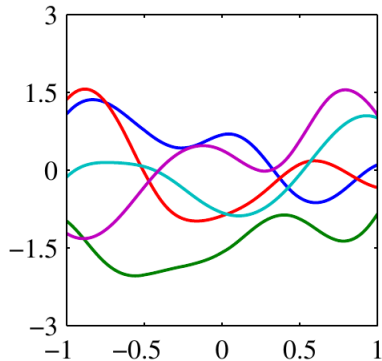
# Gaussian Processes: Definition

- A Gaussian process is defined as a **probability distribution over functions**  $y(\mathbf{x})$  such that the set of values of  $y(\mathbf{x})$  evaluated at an arbitrary set of points  $\mathbf{x}_1, \dots, \mathbf{x}_N$  **jointly have a Gaussian distribution**
- This distribution is completely specified by the second-order statistics, the mean and the covariance
  - Usually, we do not have any prior information about the **mean** of  $y(\mathbf{x})$ , so we'll take it to be **zero**
  - The **covariance** is given by the **kernel function**

$$\mathbb{E}[y(\mathbf{x}_n)y(\mathbf{x}_m)] = k(\mathbf{x}_n, \mathbf{x}_m)$$

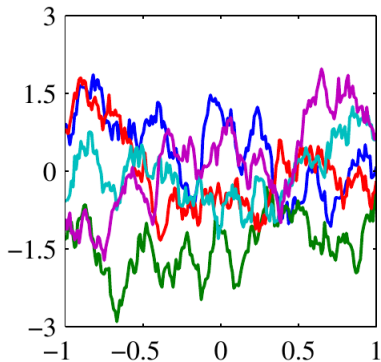
# Gaussian Processes: Example

We can also define the kernel function **directly**, rather than indirectly through a choice of basis function



Gaussian Kernel:

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|_2^2 / 2\sigma^2)$$



Exponential Kernel:

$$k(x, x') = \exp(-\theta|x - x'|)$$

# Gaussian Processes for Regression

- Take into account the noise on the target

$$t_n = y(\mathbf{x}_n) + \epsilon_n$$

- Random noise under a **Gaussian distribution**

$$p(t_n|y(\mathbf{x}_n)) = \mathcal{N}(t_n|y(\mathbf{x}_n), \sigma^2)$$

- Because the noise is **independent** on each data point, the joint distribution is still **Gaussian**:

$$p(\mathbf{t}|\mathbf{y}) = \mathcal{N}(\mathbf{t}|\mathbf{y}, \sigma^2 \mathbf{I}_N)$$

- Since  $p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K})$ , we can compute the marginal distribution:

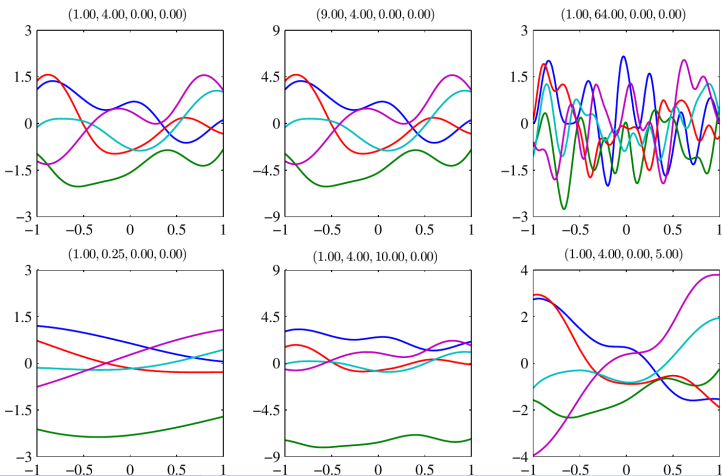
$$p(\mathbf{t}) = \int p(\mathbf{t}|\mathbf{y})p(\mathbf{y})d\mathbf{y} = \mathcal{N}(\mathbf{t}|\mathbf{0}, \mathbf{C})$$

where  $C(\mathbf{x}_n, \mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m) + \sigma^2 \delta_{nm}$

- Since the two Gaussians are **independent** their covariances simply **add**

# Gaussian Processes Examples

$$k(\mathbf{x}_n, \mathbf{x}_m) = \theta_0 \exp \left( -\frac{\theta_1}{2} \|\mathbf{x}_n - \mathbf{x}_m\|_2^2 \right) + \theta_2 + \theta_3 \mathbf{x}_n^T \mathbf{x}_m$$



# Making Predictions

- Make prediction for a **new data input**, given the training data
- **Goal:** predict  $t_{N+1}$  given  $\mathbf{x}_{N+1}$
- Need to evaluate the predictive distribution  $p(t_{N+1} | \mathbf{t}_N, \mathbf{x}_1, \dots, \mathbf{x}_{N+1})$

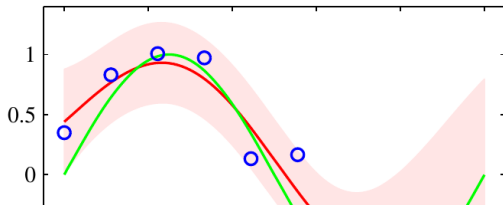
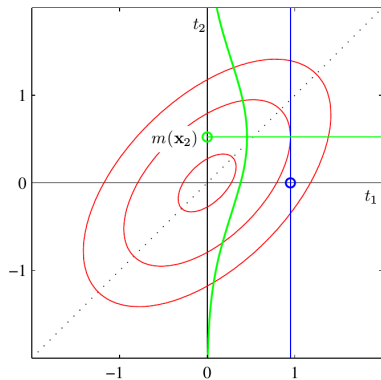
$$p(\mathbf{t}_{N+1}) = \mathcal{N}(\mathbf{t}_{N+1} | \mathbf{0}, \mathbf{C}_{N+1})$$

- where  $\mathbf{C}_{N+1} = \begin{pmatrix} \mathbf{C}_N & \mathbf{k} \\ \mathbf{k}^T & c \end{pmatrix}$ 
  - $\mathbf{k}$  is a vector  $k(\mathbf{x}_i, \mathbf{x}_{N+1})$  for  $i = 1, \dots, N$
  - $c$  is a scalar:  $c = k(\mathbf{x}_{N+1}, \mathbf{x}_{N+1}) + \sigma^2$
- $p(t_{N+1} | \mathbf{t}_N, \mathbf{x}_1, \dots, \mathbf{x}_{N+1})$  is a Gaussian:
  - $m(\mathbf{x}_{N+1}) = \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{t}$
  - $\sigma^2(\mathbf{x}_{N+1}) = c - \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{k}$
- The mean and variance both depend on  $\mathbf{x}_{N+1}$
- $\mathbf{C}$  has to be positive definite  $\Leftrightarrow$  the kernel function is positive semi-definite

# Computational cost

- The prediction requires to compute the **inversion** of  $\mathbf{C}_N$ : cost  $\mathcal{O}(N^3)$ 
  - Need to be done **only once** for the given training set
- The computation of the **mean** costs  $\mathcal{O}(N)$
- The computation of the **variance** costs  $\mathcal{O}(N^2)$
- For **large** training sets **approximated** methods are used
  - random sampling
  - clustering

# Prediction Example



# Estimating the Kernel Parameters

- The performance of a GP is strongly affected by the choice of **parameters** for the kernels
- The choice of the kernel parameters is a **model selection** problem
  - We can consider a discrete grid of values and use **cross validation**
    - Robust, but **slow**
  - Maximization of the **marginal likelihood** using gradient optimization
    - **Faster**, but multiple local optima