Kernel Methods

Machine Learning

Daniele Loiacono

POLITECNICO MILANO 1863

Outline and References

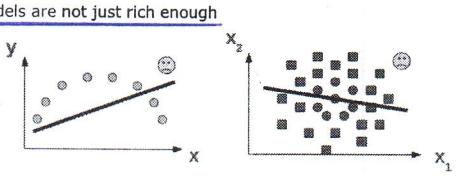
- Outline
  - Introduction [PRML 6]
  - Kernel Ridge Regression [PRML 6.1]
  - Kernel Design [PRML 6.2]
  - Kernel Regression [PRML 6.3, 2.5.1]
  - Gaussian Processes [PRML 6.4]

- References
  - This slides are based on material of prof. Marcello Restelli
  - Pattern Recognition and Machine Learning, Bishop [PRML]

Introduction to Kernel Methods

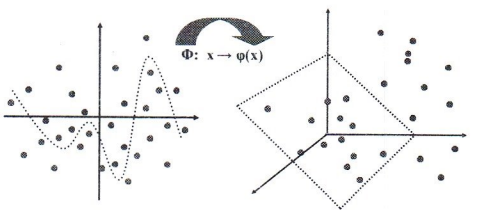
Feature Mapping

- 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**



- **Kernel methods** allow to make linear models work in nonlinear settings by **mapping data to higher dimensions** where it exhibits linear patterns

Feature Mapping (2)



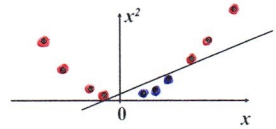
INPUT SPACE → FEATURES SPACE

almost always this space's dimension is higher than the input space's one.

Example: 1D Binary Classification

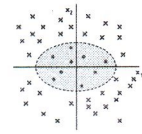
- Let consider this binary classification problem for which **no linear separator** exists:

- Now let map the input space (single variable x) to a feature space with **two features**: $x \rightarrow \{x, x^2\}$
- Data is now linear separable:

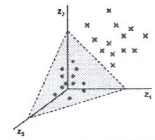


Example: 2D Binary Classification

- Let consider the following binary classification problem for which **no linear separator** exists in the input space $\mathbf{x} = \{x_1, x_2\}$



- We can make data linearly separable by mapping the input space to a suitable feature space: $\mathbf{x} = \{x_1, x_2\} \rightarrow \mathbf{z} = \{x_1^2, \sqrt{2}x_1x_2, x_2^2\}$



Why kernel methods?

- Let consider a **quadratic mapping** for a problem with M input variables:

$$\mathbf{x} = \{x_1, \ldots, x_M\} \rightarrow \phi(\mathbf{x}) = \{x_1^2, x_2^2, \ldots, x_M^2, x_1x_2, x_2x_3, \ldots, x_1x_M, \ldots, x_{M-1}x_M\}$$

- **Curse of dimensionality!** The number of features grows significantly with the number of input variable and the mapping becompe quickly **computationally unfeasible**
- **Kernels methods** deal with this issue:
  - they **don't require to explicitly compute** the feature mapping
  - they are **expensive** but computationally **feasible**

Kernel Functions

- The **kernel function** is defined as the **scalar product** between the feature vectors of two data samples:

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

  - Kernel function is **symmetric**: $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$
  - Kernel function can be interpreted as a **similarity measure** between $\mathbf{x}$ and $\mathbf{x}'$
  - Very large feature vectors (even non finite ones) might result in an easy to compute kernel function
- Special class of kernels
  - **Stationary kernels:** $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x} - \mathbf{x}')$
  - **Homogeneous kernels** (or **radial basis functions**): $k(\mathbf{x}, \mathbf{x}') = k(||\mathbf{x} - \mathbf{x}'||)$

we may think that we solved nothing since we still have to compute $\phi(\cdot)^T \phi(\cdot)$ (and so we have to det. $\phi(\cdot)$). However, it's possible to obtain $k(\cdot,\cdot)$ even without computing the scalar product !! in fact, we'll try to construct a kernel function s.t. it respects some properties. At that point it'll be still possible to represent the kernel function as $\phi(\cdot)^T \phi(\cdot)$ but we won't be interested.

Kernel Trick

- What kernel function is used for?
  - It is possible to rework the representation of linear models to replace all the terms that involve $\phi(\mathbf{x})$ with other terms that involve only $k(\mathbf{x}, \cdot)$
  - In other words, the output of linear model can be computed only on the basis of the similaraties between data samples (computed with the kernel function)
- This approach, called **kernel trick**, is used in several learning algorithm
  - Ridge Regression
  - K-NN Regression
  - Perceptron
  - (Nonlinear) PCA
  - Support Vector Machines
  - ...

!!

Kernel Trick

- What kernel function is used for?
  - It is possible to rework the representation of linear models to replace all the terms that involve $\phi(\mathbf{x})$ with other terms that involve only $k(\mathbf{x}, \cdot)$
  - In other words, the output of linear model can be computed only on the basis of the similarities between data samples (computed with the kernel function)
- This approach, called **kernel trick**, is used in several learning algorithm
  - **Ridge Regression**
  - **K-NN Regression**
  - Perceptron
  - (Nonlinear) PCA
  - **Gaussian Processes**
  - **Support Vector Machines**
  - ...

Kernel Ridge Regression

Dual Representation

- Let's go back to the loss function used for the **ridge regression:** (residual sum of squares + regulariz. term)

$$L(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_n) - t_n\right)^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} = \frac{1}{2}(\mathbf{t}-\boldsymbol{\Phi}\mathbf{w})^T(\mathbf{t}-\boldsymbol{\Phi}\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

- To solve it, we set to zero the gradient of L with respect to $\mathbf{w}$:

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = \lambda\mathbf{w} - \boldsymbol{\Phi}^T(\mathbf{t}-\boldsymbol{\Phi}\mathbf{w}) = 0$$

$\underline{\underline{\Phi}} = \begin{bmatrix} \underline{\Phi}(x_1) \\ \vdots \\ \underline{\Phi}(x_N) \end{bmatrix}$: we've **not** able to compute it

$(\underline{\Phi}(x_1) = [\Phi_1(x_1), \dots, \Phi_M(x_1)])$

- Now, instead of solving it for $\mathbf{w}$, let do a variable change:

$$\mathbf{a} = \lambda^{-1}(\mathbf{t}-\boldsymbol{\Phi}\mathbf{w})$$

$$\mathbf{w} = \boldsymbol{\Phi}^T\lambda^{-1}(\mathbf{t}-\boldsymbol{\Phi}\mathbf{w}) = \boldsymbol{\Phi}^T\mathbf{a}$$

Dual Representation (2)

- Now we can replace $\mathbf{w}$ in the gradient:

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = \lambda\mathbf{w} - \boldsymbol{\Phi}^T(\mathbf{t}-\boldsymbol{\Phi}\mathbf{w}) = 0 \qquad \mathbf{w} = \boldsymbol{\Phi}^T\mathbf{a}$$

$$\boldsymbol{\Phi}^T\left(\lambda\mathbf{a} - \left(\mathbf{t} - \boldsymbol{\Phi}\boldsymbol{\Phi}^T\mathbf{a}\right)\right) = 0$$

$$\boldsymbol{\Phi}\boldsymbol{\Phi}^T\mathbf{a} + \lambda\mathbf{a} = \mathbf{t}$$

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

$$\mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^T$$

Gram Matrix

$\underline{\underline{\Phi}} \in \mathbb{R}^{N \times M}$; $\underline{\underline{K}} \in \mathbb{R}^{N \times N}$

Gram Matrix and Kernel Function

- The **Gram matrix** is a $N \times N$ matrix, where each element is the inner product between the feature vectors:

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

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

  - K matrix represents the **similaraties** between each pair of samples in the training data

Prediction Function

- How can we compute the prediction using the dual representation?

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

$\mathbf{w} = \Phi^T\mathbf{a}$

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

$\underline{\underline{\Phi}} = \begin{bmatrix} \underline{\phi}(\underline{x}_1) \\ \vdots \\ \underline{\phi}(\underline{x}_N) \end{bmatrix} \Rightarrow \underline{\underline{\Phi}}\cdot\underline{\phi}(\underline{x}) = \begin{bmatrix} \underline{\phi}(\underline{x}_1) \\ \vdots \\ \underline{\phi}(\underline{x}_N) \end{bmatrix}\cdot\underline{\phi}(\underline{x}) = \begin{bmatrix} \underline{\phi}(\underline{x}_1)^T\cdot\underline{\phi}(\underline{x}) \\ \vdots \\ \underline{\phi}(\underline{x}_N)^T\cdot\underline{\phi}(\underline{x}) \end{bmatrix} = \begin{bmatrix} k_1(\underline{x}_1,\underline{x}) \\ \vdots \\ k_N(\underline{x}_N,\underline{x}) \end{bmatrix} = \underline{\mathbf{k}}(\underline{x})$

Prediction Function

- How can we compute the prediction using the dual representation?

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

  - Where $\mathbf{k}(\mathbf{x})$ is such that $k_n(\mathbf{x}) = k(\mathbf{x}_n,\mathbf{x})\ \forall\mathbf{x}_n \in \mathcal{D}$

- Accordingly the prediction is computed as the linear combination of the **target values** of the samples in the **training set**

Original vs Dual Representation?

- Original representation
  - Requires to compute the inverse of $(\boldsymbol{\Phi}^T\boldsymbol{\Phi}+\lambda\boldsymbol{I}_M)$ which is a MxM matrix
  - Is computationally convenient when M is rather small
- Dual representation
  - Requires to compute the inverse of $(\mathbf{K}+\lambda\boldsymbol{I}_N)$ which is a NxN matrix
  - Is computationally convenient when M **is very large or even infinite**
  - Does not require to explicitly compute $\boldsymbol{\Phi}$, making it possible to apply this approach also to complex type of data (e.g., graphs, sets, strings, text, etc.)
  - The **similarity between data samples** (i.e., the **kernel function**) is generally both less expensive to compute and easy to design than computing $\boldsymbol{\Phi}$

Kernel Function Design

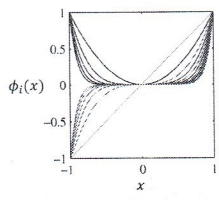
Design of Kernels

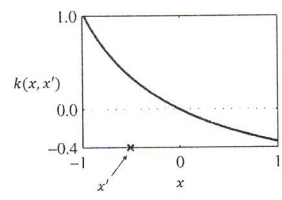
- We defined the kernel function as the dot product in the feature space:

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

If we restrict to this then it's all useless (because we're passing through $\phi(\cdot)$ to calculate $k(\cdot,\cdot)$)

- Examples (1D input space)



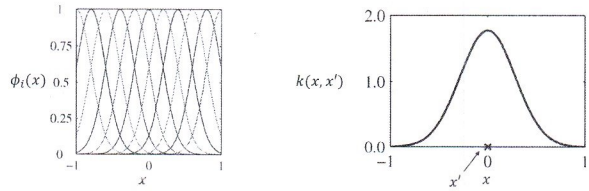


Design of Kernels

- We defined the kernel function as the dot product in the feature space:

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

- Examples (1D input space)

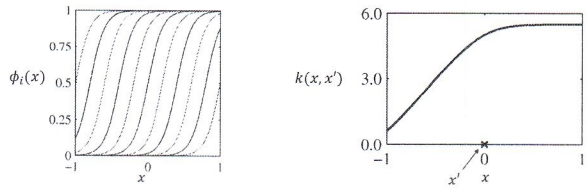


Design of Kernels

- We defined the kernel function as the dot product in the feature space:

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

- Examples (1D input space)



Design of Kernels: alternative methods

- We do not necessarily have to compute the kernel function starting from the feature space, as we do not want to explicitly compute the feature vectors
- There are two major alternatives to design a kernel function:
  - **Directly** design the kernel functions from scratch
  - Design **from existing** kernel functions by applying a **set of rules**
- In general, we must make sure that the designed kernel functions is **valid**, that is it correspond to a scalar product into any feature space
- **Mercer Theorem:** Any **continuous, symmetric, positive semi-definite** kernel function $k(x, x')$ can be expressed as a dot product in a high-dimensional space
  - **Necessary and sufficient** condition for a function $k(\mathbf{x}, \mathbf{x}')$ to be a valid kernel is that Gram matrix K is positive semi-definite for all possible choice of $\mathcal{D} = \{\mathbf{x}_i\}$
  - It means $\mathbf{x}^T K \mathbf{x} > 0$ for any non zero real vector $\mathbf{x}$, i.e., $\sum_i \sum_j K_{ij} x_i x_j$ for any real numbers x_i and x_j

Kernel direct design: an example

- Let consider the following kernel function: $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}')^2$
- Is it a valid kernel function?
- Let check it in a **two dimensional space:**

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}')^2 = (x_1x_1' + x_2x_2')^2 = x_1^2x_1'^2 + 2x_1x_1'x_2x_2' + x_2^2x_2'^2$$
$$= (x_1^2, \sqrt{2}x_1x_2, x_2^2)(x_1'^2, \sqrt{2}x_1'x_2', x_2'^2)^T = \boldsymbol{\phi}(\mathbf{x})^T\boldsymbol{\phi}(\mathbf{x}')$$

  - It does correspond to the scalar product in a feature space with only second order terms (also notice that is **less computationally expensive to compute**)
- To get also constant and linear terms, we can define $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}' + c)^2$
- To get all the terms up to degree z, we can define $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}' + c)^z$

Rules to design valid kernels

- Given valid kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$ the following rules can be applied to design a new valid kernel:

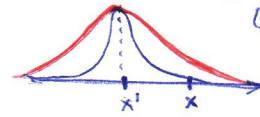
1. $k(\mathbf{x}, \mathbf{x}') = ck_1(\mathbf{x}, \mathbf{x}')$, where $c > 0$ is a constant
2. $k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})k_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}')$, where $f(\cdot)$ is any function
3. $k(\mathbf{x}, \mathbf{x}') = q(k_1(\mathbf{x}, \mathbf{x}'))$, where $q(\cdot)$ is a polynomial with non-negative coefficients
4. $k(\mathbf{x}, \mathbf{x}') = exp(k_1(\mathbf{x}, \mathbf{x}'))$
5. $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')$
6. $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}')$
7. $k(\mathbf{x}, \mathbf{x}') = k_3(\phi(\mathbf{x}), \phi(\mathbf{x}'))$, where $\phi(\mathbf{x})$ maps $\mathbf{x}$ to $\mathbb{R}^M$ and $k_3(\cdot,\cdot)$ is a valid kernel in $\mathbb{R}^M$
8. $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{A}\mathbf{x}'$, where $\mathbf{A}$ is a symmetric semidefinite matrix
9. $k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}_a') + k_b(\mathbf{x}_b, \mathbf{x}_b')$
10. $k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}_a')k_b(\mathbf{x}_b, \mathbf{x}_b')$

Where $\mathbf{x} = \{\mathbf{x}_a\} \cup \{\mathbf{x}_b\}$ are two subsets no necessarily disjoints of variables and k_a, k_b are valid kernels

Gaussian Kernel

- This is a commonly used kernel:

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



$\sigma^2 > \sigma^2$
(Note: σ^2 is also called BANDWIDTH)

  - We can check it is a valid kernel, expanding the square:

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

  - ➡ $k(\mathbf{x}, \mathbf{x}') = \exp(-\mathbf{x}^T\mathbf{x}/2\sigma^2)\exp(-\mathbf{x}^T\mathbf{x}'/\sigma^2)\exp(-\mathbf{x}'^T\mathbf{x}'/2\sigma^2)$ Rule 2 and 4

- The feature space corresponding to Gaussian kernel has infinite dimension → this kernel corresponds to the dot product of two infinite-dim vectors (dim(features space) = ∞)
- We can extend Gaussian Kernel by replacing $\mathbf{x}^T\mathbf{x}'$ with a nonlinear kernel $\kappa(\mathbf{x}, \mathbf{x}')$:

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

Basically kernels allow us to avoid to deal with feature spaces.

The original problem of choosing the right feature space now becomes the problem of choosing the right kernel

Kernels for Symbolic Data

- Kernel methods can be extended also to inputs different from real vectors, such as graphs, sets, strings, texts, etc.
- In fact, the kernel function represents a measure of the similarity between two samples
- A common kernel used for set is:

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

number of elements that appear in both sets

Kernels Based on Generative Models

- It is also possible to define a kernel function based on probability distribution
- Given a generative model $p(\mathbf{x})$ we can define a kernel as:

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

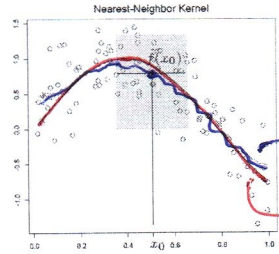
- It is a valid kernel, since it corresponds to the inner product in the one dimensional feature space defined mapping $\mathbf{x}$ to $p(\mathbf{x})$

Kernel Regression

k-NN Regression

- The k Nearest Neighbour (k-NN) can be applied to solve a regression problem by averaging the K nearest samples in the training data:

$$\hat{f}(\mathbf{x}) = \frac{1}{k} \sum_{\mathbf{x}_i \in N_k(\mathbf{x})} t_i$$



obtained regression

true function

What is a problem? There is a lot of noise: moving a little bit makes the function change a lot

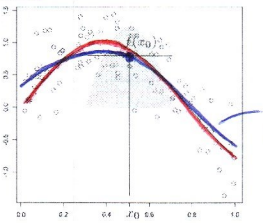
→ To mitigate it we can introduce a kernel function

Nadaraya-Watson Model

- In k-NN regression the model output is generally very noisy due to the discontinuity of neighborhood averages
- **Nadaraya-Watson** model (or **kernel regression**) deal with this issue by using kernel function to compute a weighted average of samples:

$$\hat{f}(\mathbf{x}) = \frac{\sum_{i=1}^{N} k(\mathbf{x}, \mathbf{x}_i) t_i}{\sum_{i=1}^{N} k(\mathbf{x}, \mathbf{x}_i)}$$

- Typical choices for kernels are:
  - Epanechnikov Kernel (bounded support)
  - Gaussian Kernel (infinite support)



this time the obtained regression function is more smooth

Gaussian Processes

application of kernel methods to BAYESIAN REGRESSION

Linear Regression Revisited

- Let's start from the same assumptions used in Bayesian Linear Regression

$$y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x})$$
$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|0, \tau^2 \mathbf{I})$$

linear model with a prior distribution over the weights

- Now compute the prior distribution of the outputs of the regression function:

$$\mathbf{y} = \boldsymbol{\Phi}\mathbf{w} \Rightarrow p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\boldsymbol{\mu}, \mathbf{S})$$

$\mathbf{y}$ is not a random variable

$$\boldsymbol{\mu} = \mathbb{E}[\mathbf{y}] = \boldsymbol{\Phi}\mathbb{E}[\mathbf{w}] = \mathbf{0}$$

$$\mathbf{S} = \mathrm{cov}[\mathbf{y}] = \mathbb{E}[\mathbf{y}\mathbf{y}^T] = \boldsymbol{\Phi}\mathbb{E}[\mathbf{w}\mathbf{w}^T]\boldsymbol{\Phi}^T = \tau^2 \boldsymbol{\Phi}\boldsymbol{\Phi}^T = \mathbf{K}$$

Gram matrix

Now, instead of modelling the distribution of the weights, we try to model the outputs of the regression function (the "y's"). (To do it we start from the assumpt. that we did on the weights: $\underline{w} \sim N(\underline{0}, \tau^2 \mathbf{I})$)

Note that we're including τ^2

Gaussian Processes and Gram Matrix

- In general, a **Gaussian Process** is defined as a distribution probability over a function y(x) such that **the set of values y(x_i)** – for an arbitrary $\{x_i\}$ – **jointly have a Gaussian Distribution.**
- In our specific case,

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K})$$

  - where **K** is the Gram matrix defined as:

$$K_{nm} = k(\mathbf{x}_n, \mathbf{x}_m) = \tau^2 \boldsymbol{\phi}(\mathbf{x}_n)^T \boldsymbol{\phi}(\mathbf{x}_m)$$

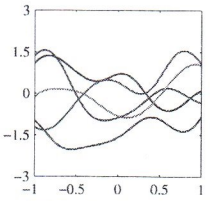
- This gives a probabilistic interpretation of the Kernel function as:

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

the kernel function computed between two samples represents how much the output of the model will be correlated in the two samples. This is why the kernel function can be interpreted as similarity. The kernel shapes how correlated will be the output of the model for each pair of points.

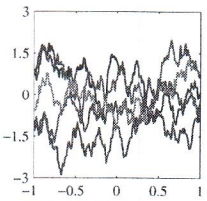
Kernel Design

- We can apply the usual approaches to design the kernels
- Two families of kernels typically used with GP are:



Gaussian Kernel

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



Exponential Kernel

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

larger σ^2 smoother the function

Gaussian Processes for Regression

- To apply GP to regression we must take into account also noise on the target:

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

- Assuming the noise Gaussian and independent on each point, the joint distribution of the targets will be:

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

- We can compute the marginal distribution p(**t**) from p(**t**|**y**) and p(**y**) as:

$$p(\mathbf{t}) = \int p(\mathbf{t}|\mathbf{y})p(\mathbf{y})\mathrm{d}\mathbf{y}$$

$\mathcal{N}(\mathbf{t}|\mathbf{y}, \sigma^2\mathbf{I})$ $\quad$ $\mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K})$

easy since both are gaussian. The result will be a gaussian with the mean which is the product of the two means and the variance which is the sum of the two variances

Gaussian Processes for Regression

- To apply GP to regression we must take into account also noise on the target:

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

- Assuming the noise Gaussian and independent on each point, the joint distribution of the targets will be:

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

- We can compute the marginal distribution p(**t**) from p(**t**|**y**) and p(**y**) as:

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

we now want to use this distribution to make prediction

  - Being p(**t**|**y**) and p(**y**) their covariances simply add:

$$\delta_{nm} = \begin{cases} 1 \text{ if } n = m \\ 0 \text{ otherwise} \end{cases}$$

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

$C = K + \sigma^2 I$

Making Prediction with GP

- We observe {**x**,**t**} and want to make a prediction on a new point x_{N+1}
- First, we need to compute the joint distribution of $\mathbf{t}_{N+1}$ (that will be still Gaussian):

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

  - $\mathbf{t}_{N+1} = (t_1, t_2, \cdots, t_{N+1})^T$ — target that we want to predict (t_{N+1})
  - $\mathbf{C}_{N+1} = \begin{pmatrix} \mathbf{C}_N & \mathbf{k} \\ \mathbf{k}^T & c \end{pmatrix}$
  - **k** is a vector $k(\mathbf{x}_i, \mathbf{x}_{N+1}) \quad \forall i = 1, \ldots, N$
  - $c = k(\mathbf{x}_{N+1}, \mathbf{x}_{N+1}) + \sigma^2$

Making Prediction with GP (2)

- ... now we can use the joint distribution to derive the conditioned distribution that will be used in practice to make the prediction:

$$p(t_{N+1}|\mathbf{t}) = \mathcal{N}(m, \sigma^2)$$

  - where:

$$m(\mathbf{x}_{N+1}) = \mathbf{k}^T\mathbf{C}_N^{-1}\mathbf{t}$$

$$\sigma^2(\mathbf{x}_{N+1}) = c - \mathbf{k}^T\mathbf{C}_N^{-1}\mathbf{k}$$ → uncertainty of our prediction

- Computational cost is:
  - $O(N^3)$ for inversion of **C** (required only once for training)
  - O(N) for computing m
  - $O(N^2)$ for computing σ^2

Kernel design for regression

- Popular choice is $k(\mathbf{x}_n, \mathbf{x}_m) = \theta_0 \exp\left(-\frac{\theta_1}{2}\|\mathbf{x}_n - \mathbf{x}_m\|_2^2\right) + \theta_2 + \theta_3 \mathbf{x}_n{}^T \mathbf{x}_m$
- How to choose the kernel hyperparameters?
  - Grid Search (very expensive)
  - Maximization of marginal likelihood over the training set using gradient optimization