[Spring, 2017]

# Kernel Methods

## Pattern Recognition (BRI623)


**Heung-Il Suk**

hisuk@korea.ac.kr

http://www.ku-milab.org


Department of Brain and Cognitive Engineering,
Korea University

## Contents

1. Introduction
2. Dual Representations
3. Constructing Kernels
4. Radial Basis Function Networks
5. Gaussian Processes

# Introduction

- Linear parametric models for regression and classification: $y(\mathbf{x}, \mathbf{w})$
  - Training: finding either a maximum likelihood estimate of $\mathbf{w}$ or a posterior distribution over $\mathbf{w}$
  - Prediction is performed based only on $\mathbf{w}$ (discarding training data)
- Also true for neural networks and deep learning
- Another class of methods *uses the training samples or a subset of them*



# Memory-based Methods

- Training data points are used in the prediction phase
  - Parzen probability density model (refer to Chapter 2): Linear combination of kernel functions centered on each training data point
  - Nearest neighborhood classification
- Typically require a metric to be defined that measures the *similarity* of any two vectors in input space
- Generally fast to 'train' but slow at making predictions for test data points

# Kernel Functions

- Many linear parametric models can be re-cast into an equivalent 'dual representation.'
- Predictions are also based on linear combinations of a *kernel function* evaluated at the training data points

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

$\phi(\mathbf{x})$: fixed non-linear feature space mapping

- Kernel is a symmetric function of its arguments
  - Kernel function can be interpreted as the similarity of its arguments

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

- Linear kernel: simplest identity mapping in feature space $\phi(\mathbf{x}) = \mathbf{x}$

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



# Kernel Trick

- The concept of a kernel function formulated as inner product allows extending well-known algorithms by making use of the *kernel trick*
- Also called *kernel substitution*
- 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.*
- Applications
  - Support Vector Machines (SVMs) [Boser *et al.*, 1992]
  - Developing non-linear variant of Principal Component Analysis (PCA) [Schölkopf *et al.*, 1998]
  - Kernel Fisher discriminant [Mika *et al.*, 1999; Roth and Steinhage, 2000; Baudat and Anouar, 2000]
  - Kernel Independent Component Analysis (ICA) [Bach and Jordan, 2002]

# Examples of Kernel Functions

- Difference between arguments
  - Called *stationary* kernel since invariant to translation in space

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

- *Homogeneous* kernels, also known as *radial basis functions*
  - Depend only on the magnitude of the distance between arguments

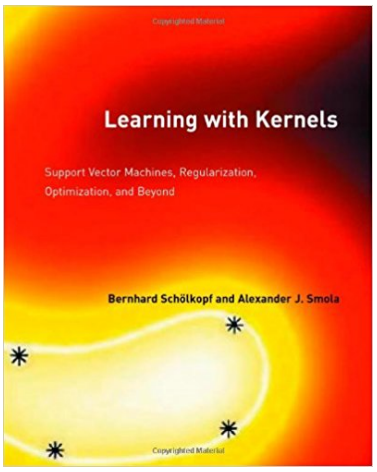
$$k\left(\mathbf{x}, \mathbf{x}'\right) = k\left(\|\mathbf{x} - \mathbf{x}'\|\right)$$

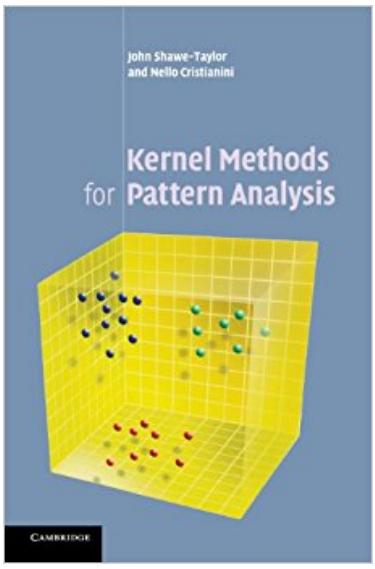
For functions to be a *valid* kernel, they should be shown to have the property

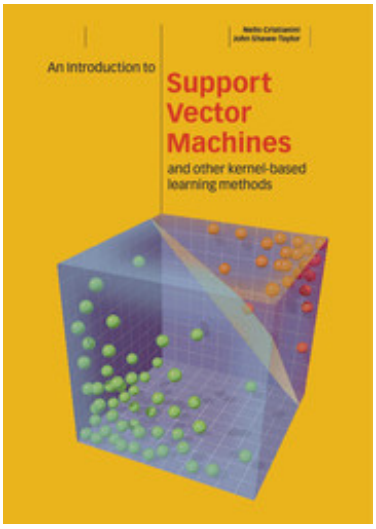
$$k\left(\mathbf{x}, \mathbf{x}'\right) = \phi\left(\mathbf{x}\right)^{\top} \phi\left(\mathbf{x}'\right)$$



## Textbooks on kernel methods



# Dual Representations



## Dual Representations

- Linear models for regression and classification can be reformulated in terms of a dual representation in which kernel function arises naturally
- Consider a regularized linear regression model with a sum-of-squares error function

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{\mathbf{w}^{\top}\phi(\mathbf{x}_n) - t_n\right\}^2 + \frac{\lambda}{2}\mathbf{w}^{\top}\mathbf{w}$$

$$(\text{where } \lambda \geq 0)$$

- By equating derivative $J(\mathbf{w})$ w.r.t. $\mathbf{w}$ to zero and solving for $\mathbf{w}$

$$\mathbf{w} = -\frac{1}{\lambda}\sum_{n=1}^{N}\left\{\mathbf{w}^{\top}\phi(\mathbf{x}_n) - t_n\right\}\phi(\mathbf{x}_n)$$

Let $\mathbf{a}$ and $\Phi$ defined as follows:

$$a_n = -\frac{1}{\lambda}\left\{\mathbf{w}^\top \phi\left(\mathbf{x}_n\right) - t_n\right\}$$

where $\Phi = \begin{bmatrix} \phi_0\left(\mathbf{x}_1\right) & \phi_1\left(\mathbf{x}_1\right) & \cdots & \phi_{M-1}\left(\mathbf{x}_1\right) \\ \phi_0\left(\mathbf{x}_2\right) & \phi_1\left(\mathbf{x}_2\right) & \cdots & \phi_{M-1}\left(\mathbf{x}_2\right) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0\left(\mathbf{x}_n\right) & \phi_1\left(\mathbf{x}_n\right) & \cdots & \phi_{M-1}\left(\mathbf{x}_n\right) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0\left(\mathbf{x}_N\right) & \phi_1\left(\mathbf{x}_N\right) & \cdots & \phi_{M-1}\left(\mathbf{x}_N\right) \end{bmatrix}$, called '*design matrix*'

$$\mathbf{w} = \sum_{n=1}^{N} a_n \phi\left(\mathbf{x}_n\right) = \Phi^\top \mathbf{a}$$

where $\mathbf{a} = \left[a_1, \ldots, a_n, \ldots, a_N\right]^\top$



$$J\left(\mathbf{w}\right) = \frac{1}{2}\sum_{n=1}^{N}\left\{\mathbf{w}^\top \phi\left(\mathbf{x}_n\right) - t_n\right\}^2 + \frac{\lambda}{2}\mathbf{w}^\top \mathbf{w}$$

- Instead of working with the parameter vector $\mathbf{w}$, we can now reformulate the least-squares algorithm in terms of the parameter vector $\mathbf{a}$, giving rise to a *dual representation*

$$J\left(\mathbf{a}\right) = \frac{1}{2}\mathbf{a}^\top \Phi\Phi^\top \Phi\Phi^\top \mathbf{a} - \mathbf{a}^\top \Phi\Phi^\top \mathbf{t} + \frac{1}{2}\mathbf{t}^\top \mathbf{t} + \frac{\lambda}{2}\mathbf{a}^\top \Phi\Phi^\top \mathbf{a}$$

- Define the *Gram matrix* $\mathbf{K} = \Phi\Phi^\top$
  - $N \times N$ matrix

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

  - $k\left(\mathbf{x}_n, \mathbf{x}_m\right)$: *kernel function*

- In terms of the Gram matrix, sum-of-squares error function can be written as

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^\top \mathbf{K}\mathbf{K}^\top \mathbf{a} - \mathbf{a}^\top \mathbf{K}\mathbf{t} + \frac{1}{2}\mathbf{t}^\top \mathbf{t} + \frac{\lambda}{2}\mathbf{a}^\top \mathbf{K}\mathbf{a}$$

- Setting the gradient of $J(\mathbf{a})$ w.r.t. $\mathbf{a}$ to zero

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



- By substituting back into the linear regression model

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

  - $\mathbf{k}(\mathbf{x})$: a vector with elements $k_n(\mathbf{x}) = k(\mathbf{x}_n, \mathbf{x})$

The dual formulation allows the solution to the least-squares problem to be expressed entirely in terms of the kernel function $k(\mathbf{x}, \mathbf{x}')$

In dual formulation,

- we determine the parameter vector $\mathbf{a}$ by inverting an $N \times N$ matrix,

$$\mathbf{w} = \Phi^{\top}\mathbf{a} = \Phi^{\top}\left(\mathbf{K} + \lambda \mathbf{I}_N\right)^{-1}\mathbf{t}$$

In the original parameter space formulation,

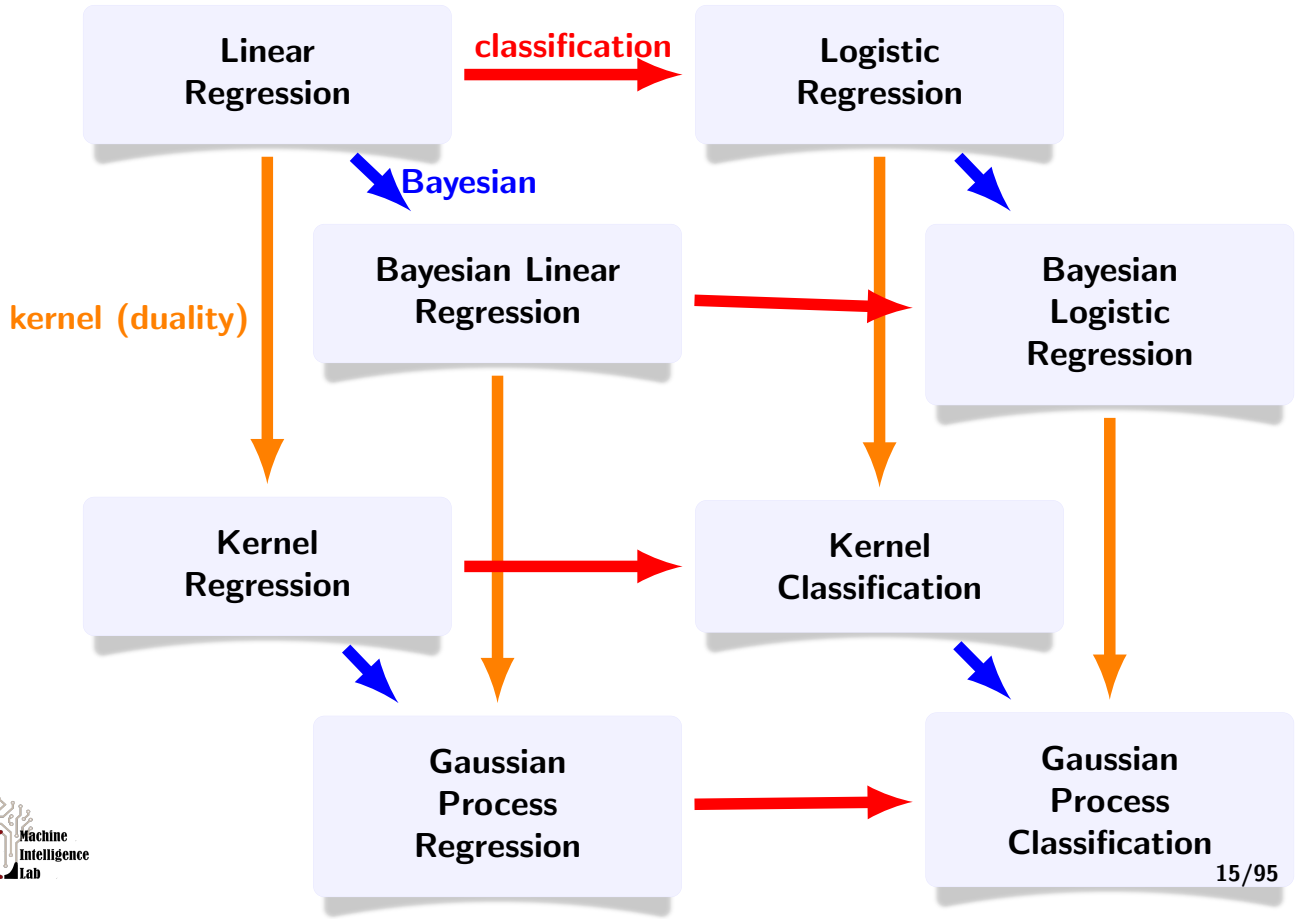
- we had to invert $M \times M$ matrix in order to determine $\mathbf{w}$

$$\mathbf{w}_{ML} = \left(\Phi^{\top}\Phi\right)^{-1}\Phi^{\top}\mathbf{t}$$

  - When $M > N$, obviously the dual formulation is useful
  - When $M < N$, the dual formulation does not seem to be particularly useful

Due to the kernel function $k(\mathbf{x}, \mathbf{x}')$ in dual formulation, we can work in terms of kernels and avoid the explicit introduction of the feature vector $\phi(\mathbf{x})$, allowing implicitly to use feature spaces of high, even infinite, dimensionality.





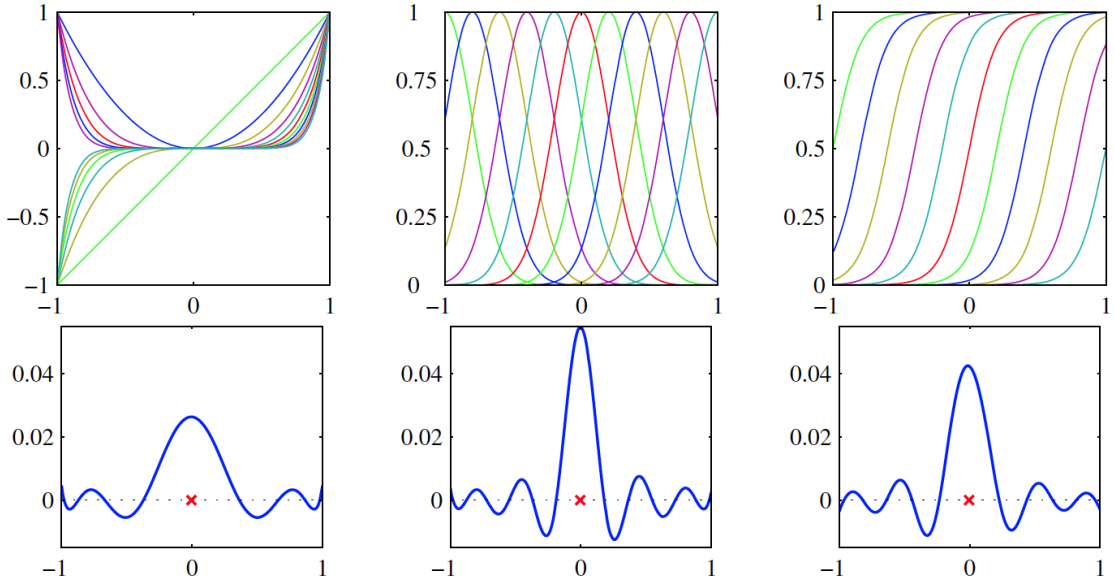
# Constructing Kernels



## Constructing Kernels

1. Choose a feature space mapping $\phi(\mathbf{x})$ and then use this to find the corresponding kernel

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



2. Construct kernel functions directly
   - Ensure that the function we choose is valid
   - Valid kernel: corresponding to a scalar product in some (perhaps infinite dimensional) feature space

$$\textit{e.g.)}\ k(\mathbf{x},\mathbf{z}) = \left(\mathbf{x}^\top \mathbf{z}\right)^2$$

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



Need a simple way to test whether a function constitutes a valid kernel without having to construct the function $\phi(\mathbf{x})$ explicitly

## Necessary and sufficient condition

- Gram matrix $\mathbf{K}$, whose elements are given by $k(\mathbf{x}_n, \mathbf{x}_m)$ is *positive semidefinite* for all possible choices of the set $\{\mathbf{x}_n\}$

$$\mathbf{z}^\top \mathbf{K} \mathbf{z} \geq 0 \text{ for nonzero vectors } \mathbf{z} \text{ with real entries}$$

- **Mercer's theorem** (Mercer, 1909)
  - Any continuous, symmetric, positive semidefinite kernel function $k(\mathbf{x},\mathbf{z})$ can be expressed as a dot product in a high-dimensional space.

**New kernels can be constructed with simpler kernels as building blocks!!!**

## Techniques for Constructing New Kernels.

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

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= c k_1(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= q\left(k_1(\mathbf{x}, \mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= \exp\left(k_1(\mathbf{x}, \mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= k_3\left(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^{\mathrm{T}} \mathbf{A} \mathbf{x}' \\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b) \\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) k_b(\mathbf{x}_b, \mathbf{x}'_b)
\end{aligned}
$$

where $c > 0$ is a constant, $f(\cdot)$ is any function, $q(\cdot)$ is a polynomial with nonnegative coefficients, $\boldsymbol{\phi}(\mathbf{x})$ is a function from $\mathbf{x}$ to $\mathbb{R}^M$, $k_3(\cdot, \cdot)$ is a valid kernel in $\mathbb{R}^M$, $\mathbf{A}$ is a symmetric positive semidefinite matrix, $\mathbf{x}_a$ and $\mathbf{x}_b$ are variables (not necessarily disjoint) with $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$, and $k_a$ and $k_b$ are valid kernel functions over their respective spaces.



Requirements for specific applications

- Symmetricity
- Gram matrix: positive semidefinite
- Appropriate form of similarity between $\mathbf{x}$ and $\mathbf{z}$

For more information on '*kernel engineering*', refer to [Shawe-Taylor and Cristianini, 2004].

- Polynomial (degree of $q$) functions

$$\begin{aligned} k(\mathbf{x},\mathbf{y}) &= \left(\mathbf{x}^\top\mathbf{y}+c\right)^2 && \text{(e.g., } q=2\text{)} \\ &= (x_1y_1+x_2y_2+c)^2 && \text{(where } d=2\text{)} \\ &= c^2+2cx_1y_1+2cx_2y_2+2cx_1x_2y_1y_2+x_1^2y_1^2+x_2^2y_2^2 \\ &= \Phi(\mathbf{x})^\top\Phi(\mathbf{y}) \end{aligned}$$

$$\text{where } \Phi(\mathbf{x}) = \left[c, \sqrt{2c}x_1, \sqrt{2c}x_2, \sqrt{2c}x_1x_2, x_1^2, x_2^2\right]^\top$$

$$k(\mathbf{x},\mathbf{y}) = \left(\mathbf{x}^\top\mathbf{y}+c\right)^q : \textbf{polynomial kernel function}$$



- **Gaussian kernel**

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

- Expanding the square

$$\|\mathbf{x}-\mathbf{x}'\|^2 = \mathbf{x}^\top\mathbf{x}+(\mathbf{x}')^\top\mathbf{x}'-2\mathbf{x}^\top\mathbf{x}'$$

$$k(\mathbf{x},\mathbf{x}') = \exp\left[-\mathbf{x}^\top\mathbf{x}/2\sigma^2\right]\exp\left[\mathbf{x}^\top\mathbf{x}'/\sigma^2\right]\exp\left[-(\mathbf{x}')^\top\mathbf{x}'/2\sigma^2\right]$$

- Valid kernel

$$\because \begin{cases} k(\mathbf{x},\mathbf{x}') = \mathbf{x}^\top\mathbf{x}' \text{ (valid)}, \\ k(\mathbf{x},\mathbf{x}') = ck_1(\mathbf{x},\mathbf{x}'), \\ k(\mathbf{x},\mathbf{x}') = f(\mathbf{x})k_1(\mathbf{x},\mathbf{x}')f(\mathbf{x}'), \\ k(\mathbf{x},\mathbf{x}') = \exp(k_1(\mathbf{x},\mathbf{x}')) \end{cases}$$

**The feature vector that corresponds to the Gaussian kernel has infinite dimensionality.**

- Variants of Gaussian kernel by using different distance measure

$$
\begin{aligned}
k\left(\mathbf{x}, \mathbf{x}'\right) &= \exp\left[-\frac{1}{2\sigma^2}\left(k\left(\mathbf{x}, \mathbf{x}\right) + k\left(\mathbf{x}', \mathbf{x}'\right) - 2k\left(\mathbf{x}, \mathbf{x}'\right)\right)\right] \\
k\left(\mathbf{x}, \mathbf{x}'\right) &= \exp\left[-\frac{1}{2}\|\mathbf{x} - \mathbf{x}'\|^{\top}\mathbf{S}^{-1}\|\mathbf{x} - \mathbf{x}'\|\right] \\
k\left(\mathbf{x}, \mathbf{x}'\right) &= \exp\left[-\frac{\mathcal{D}\left(\mathbf{x}, \mathbf{x}'\right)}{2\sigma^2}\right]
\end{aligned}
$$

  - $\mathbf{S}$: a covariance matrix
  - $\mathcal{D}\left(\mathbf{x}, \mathbf{x}'\right)$: some distance function



- Extensible to symbolic inputs and objects as diverse as graphs, sets, strings, and text documents

- *e.g.*, Similarity between two documents $D_1$ and $D_2$
  - $\Phi\left(D_i\right)$: $M$-dimensional binary vector whose dimension $i$ is 1 if word $i$ appears in $D_i$ (called '*Bag-of-Words*' representation)
  - $\Phi\left(D_1\right)^{\top}\Phi\left(D_2\right)$: counts the number of shared words

$$\Downarrow$$

  - Define a kernel $k\left(D_1, D_2\right)$ as counting the number of words appearing in two documents
    - Don't need to preselect $M$ words
    - Allows $M$ to be as large as we want

- Kernels based on generative models
  - If $p(\mathbf{x})$ is a probability density, the following is a valid kernel

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

    - 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 both have high probabilities

  - Allowing to apply generative models in a discriminative setting
    - Define a kernel using a generative model
    - Use this kernel in a discriminative approach



- Kernels based on generative models (cont.)
  - Extension by considering sums over products of different probability distributions

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

$$k(\mathbf{x}, \mathbf{x}') = \int p(\mathbf{x}|\mathbf{y})\, p(\mathbf{x}'|\mathbf{y})\, d\mathbf{y}$$

    - $i$, $\mathbf{y}$: discrete/continuous latent variables

  Two inputs $\mathbf{x}$ and $\mathbf{x}'$ will give a large value for the kernel function, and hence appear similar, if they have significant probability under a range of different components.

- Kernels based on generative models (cont.)
  - Use a hidden Markov model for a time series data

$$k(\mathbf{X}, \mathbf{X}') = \sum_{\mathbf{Z}} p(\mathbf{X}|\mathbf{Z})\, p(\mathbf{X}'|\mathbf{Z})\, p(\mathbf{Z})$$

    - $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_L\}$: ordered sequence of length $L$
    - $\mathbf{Z}$: a corresponding sequence of hidden states
    - Can easily be extended to allow sequences of differing length to be compared



- Fisher kernel [Jaakkola and Haussler, 1998]
  - Parameterize the generative model as $p(\mathbf{x}|\theta)$ and learn $\theta$ from data
  - Consider the gradient w.r.t. $\theta$, which defines a vector in a 'feature' space having the same dimensionality as $\theta$

$$\mathbf{g}(\theta, \mathbf{x}) = \nabla_\theta \ln p(\mathbf{x}|\theta) \qquad \textit{Fisher score}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{g}(\theta, \mathbf{x})^\top \mathbf{F}^{-1} \mathbf{g}(\theta, \mathbf{x}')$$

$$\mathbf{F} = \mathbb{E}_{\mathbf{x}}\left[\mathbf{g}(\theta, \mathbf{x})\, \mathbf{g}(\theta, \mathbf{x})^\top\right] \qquad \textit{Fisher information matrix}$$

- Fisher information matrix
  - Presence of Fisher information matrix causes kernel to be invariant under non-linear parametrization of the density model $\theta \rightarrow \psi(\theta)$
  - In practice, infeasible to evaluate Fisher information matrix
  - Instead use the approximation

$$\mathbf{F} \approx \frac{1}{N} \sum_{n=1}^{N} \mathbf{g}(\theta, \mathbf{x}_n) \mathbf{g}(\theta, \mathbf{x}_n)^{\top}$$

    - Covariance matrix of the Fisher scores
  - Fisher kernel corresponds to whitening of the Fisher scores
  - More simply, omit $\mathbf{F}$ and use non-invariant kernel

$$k(\mathbf{x}, \mathbf{z}) = \mathbf{g}(\theta, \mathbf{x})^{\top} \mathbf{g}(\theta, \mathbf{z})$$



- Sigmoidal kernel functions

$$k(\mathbf{x}, \mathbf{z}) = \tanh\left(a\mathbf{x}^{\top}\mathbf{z} + b\right)$$

  - Provides a link between SVMs and neural networks
  - Its Gram matrix is not positive semidefinite
  - But, used in practice because it gives SVMs a superficial resemblance to neural networks

- **Multi-kernel method:** useful to combine heterogeneous data

$$k(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{m} \eta_i k_i(\mathbf{x}, \mathbf{y})$$

  - $\eta_i \geq 0$: weight of the $i$-th kernel
    - should be learned from data
    - $\sum_i \eta_i = 1$: convex combination
  - Multiple kernel learning [Gönen and Alpaydin, JMLR 2011]

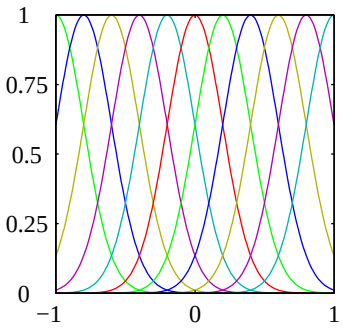


# Radial Basis Function Networks

# Radial Basis Function Networks

- Radial basis functions
  - Depends on the radial distance (typically Euclidean) from a center $\boldsymbol{\mu}_j$

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






- Historically, introduced for the purpose of exact function interpolation [Powell, 1987]
- Given a set of input vectors and target values $\{\mathbf{x}_n, t_n\}_{n=1,\ldots,N}$, the goal is to find a smooth function $f(\mathbf{x}) = t_n$ for $n = 1, \ldots, N$

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

  - $\{w_n\}$: found by least squares
  - Because there are the same number of coefficients as there are constraints, the result is a function that fits every value exactly.
  - In reality, the target values are generally noisy, and exact interpolation is undesirable due to an overfitted solution.

- Interpolation problem when the input (rather than the target) variables are noisy
- Describing the noise on the input variable $\mathbf{x}$ by $\boldsymbol{\xi}$, having a distribution $\nu(\boldsymbol{\xi})$
- Sum-of-squares error function

$$E = \frac{1}{2}\sum_{n=1}^{N}\int \{y(\mathbf{x}_n + \boldsymbol{\xi}) - t_n\}^2 \nu(\boldsymbol{\xi})\, d\boldsymbol{\xi}$$



- Using the calculus of variations (Appendix D), we can optimize w.r.t. the function $y(\mathbf{x})$ to give

$$y(\mathbf{x}) = \sum_{n=1}^{N} t_n h(\mathbf{x} - \mathbf{x}_n)$$

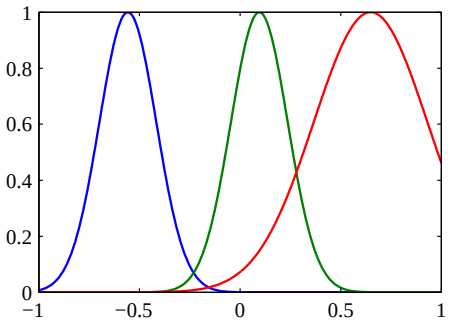
$$h(\mathbf{x} - \mathbf{x}_n) = \frac{\nu(\mathbf{x} - \mathbf{x}_n)}{\sum_{m=1}^{N} \nu(\mathbf{x} - \mathbf{x}_m)}$$

  - One basis function centered on every data point
  - Known as '*Nadaraya-Watson*' model

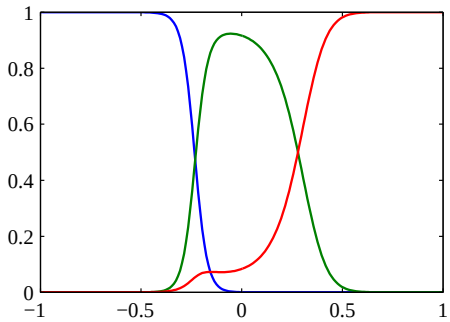
- If the noise distribution $\nu(\boldsymbol{\xi})$ is isotropic, so that it is a function only of $\|\boldsymbol{\xi}\|$, then the basis functions will be radial.
- Normalized basis functions

$$h(\mathbf{x}-\mathbf{x}_n) = \frac{\nu(\mathbf{x}-\mathbf{x}_n)}{\sum_{m=1}^{N}\nu(\mathbf{x}-\mathbf{x}_m)}; \quad \sum_{n}^{N} h(\mathbf{x}-\mathbf{x}_n) = 1$$

  - Normalization avoids having regions of input space where all of the basis functions take small values, which would necessarily lead to predictions in such regions that are either small or controlled purely by the basis parameter.



Gaussian basis functions



Normalized basis functions



- Since there is one basis function per data point, the corresponding computational model will be costly to evaluate for new data points.
- Methods for choosing a smaller set
  - Use a random subset of data points
  - Orthogonal least squares [Chen *et al.*, 1991]
    - At each step the next data point to be chosen as basis function center is the one that gives the greatest reduction in least-squared error.
  - Clustering algorithms (*e.g.*, $K$-means)
    - Basis function centers that no longer coincide with training data points

# Nadaraya-Watson Model

- Given a training set $\{\mathbf{x}_n, t_n\}_{n=1,\ldots,N}$
- Use a Parzen density estimator to model the joint distribution $p(\mathbf{x}, t)$

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

  - $f(\mathbf{x}, t)$: component density function; one component centered on each data point



- Find an expression for the regression function $y(\mathbf{x})$, corresponding to the conditional average of the target variable conditioned on the input variable

$$\begin{aligned} y(\mathbf{x}) &= \mathbb{E}[t|\mathbf{x}] = \int_{-\infty}^{\infty} t p(\mathbf{x}|t)\, dt \\ &= \frac{\int t p(\mathbf{x}, t)\, dt}{\int p(\mathbf{x}, t)\, dt} \\ &= \frac{\sum_n \int t f(\mathbf{x} - \mathbf{x}_n, t - t_n)\, dt}{\sum_m \int f(\mathbf{x} - \mathbf{x}_m, t - t_m)\, dt} \end{aligned}$$

- For simplicity, assume that the component density functions have zero mean for all values of $\mathbf{x}$

$$\int_{-\infty}^{\infty} f(\mathbf{x}, t)\, dt = 0$$

- Using a simple change of variable and by defining $g(\mathbf{x}) = \int_{-\infty}^{\infty} f(\mathbf{x}, t)\, dt$

$$\begin{aligned} y(\mathbf{x}) &= \frac{\sum_n g(\mathbf{x} - \mathbf{x}_n)\, t_n}{\sum_m g(\mathbf{x} - \mathbf{x}_m)} \\ &= \sum_n k(\mathbf{x}, \mathbf{x}_n)\, t_n \end{aligned}$$

$$k(\mathbf{x}, \mathbf{x}_n) = \frac{g(\mathbf{x} - \mathbf{x}_n)}{\sum_m g(\mathbf{x} - \mathbf{x}_m)}$$

*Nadaraya-Watson* model or *kernel regression*

[Nadaraya, 1964; Watson, 1964]



- This model defines not only a conditional expectation, but also a full conditional distribution

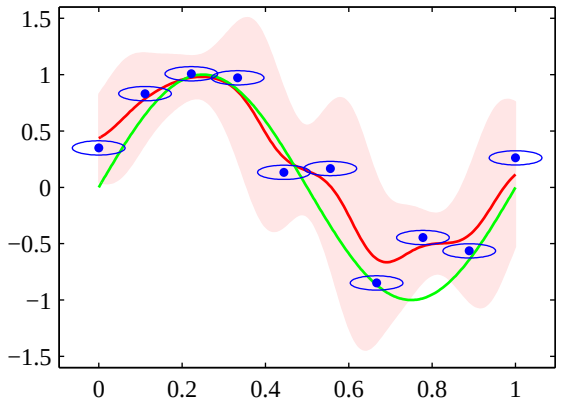
$$p(t|\mathbf{x}) = \frac{p(t, \mathbf{x})}{\int p(t, \mathbf{x})\, dt} = \frac{\sum_n f(\mathbf{x} - \mathbf{x}_n, t - t_n)}{\sum_m \int f(\mathbf{x} - \mathbf{x}_m, t - t_m)\, dt}$$

from which other expectations can be evaluated.

As an illustration, consider a single input $x$ in which $f(x,t)$ is given by a zero-mean isotropic Gaussian over the variable $\mathbf{z} = (x,t)$ with variance $\sigma^2$.

$$p(t|\mathbf{x}) = \frac{p(t,\mathbf{x})}{\int p(t,\mathbf{x})\,dt} = \frac{\sum_n f(\mathbf{x}-\mathbf{x}_n, t-t_n)}{\sum_m \int f(\mathbf{x}-\mathbf{x}_m, t-t_m)\,dt}$$ : given by a Gaussian mixture

- Green: original sine function
- Blue dot: data points; each is the center of an isotropic Gaussian kernel
- Blue ellipse: one standard deviation contour for the corresponding kernel (elongated due to different scales of the axes)
- Red: mean of the resulting regression function
- Red shading: two-standard-deviation region for $p(t|x)$






- Extension by allowing for more flexible forms of Gaussian components
  - *e.g.*, different variance parameters for the input and target variables
- More generally, could model the joint distribution $p(t,\mathbf{x})$ using a Gaussian mixture models; then find the corresponding conditional distribution $p(t|\mathbf{x})$
  - Gaussian mixture models: trained by EM algorithm
  - No longer have a representation in terms of kernel functions evaluated at the training set data points
  - The number of components in the mixture model can be smaller than the number of training set points.
    - Resulting in a model that is faster to evaluate for test data points
    - Increased computational cost during the training phase

# Gaussian Processes

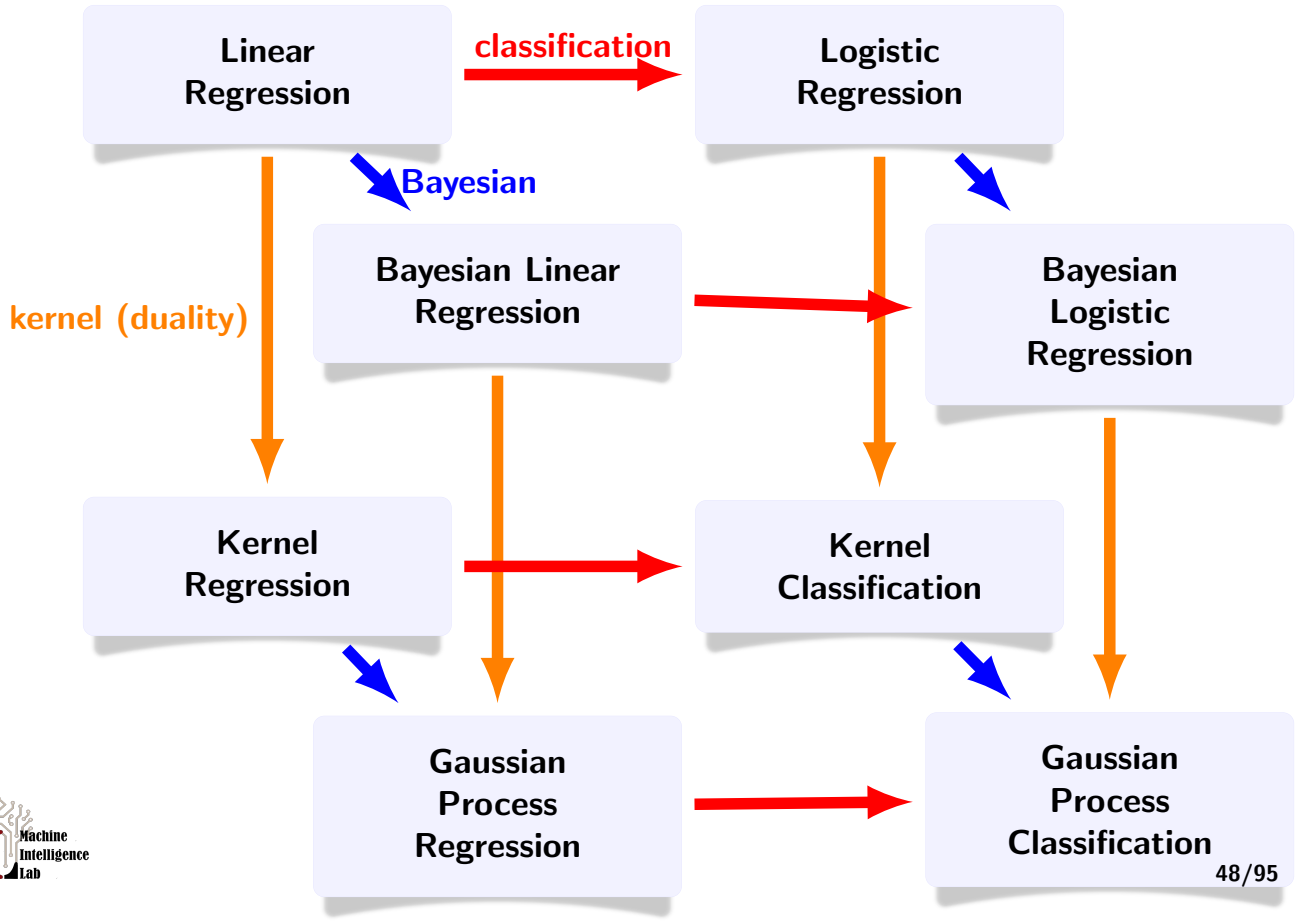


## Introduction

Introduced kernel by applying the concept of duality to a non-probabilistic model for linear regression

### GP as a kernel method

- Extending the role of kernels to *probabilistic discriminative models*
- Kernels arise naturally in a Bayesian setting
- Models that can be views as Gaussian process models
  - ARMA (AutoRegressive Moving Average) models
  - Kalman filters
  - Radial basis function networks




[Modified from Slides by Z. Ghahramani, NIPS 2016]

## Bayesian linear regression

- Model: $y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^\top \phi(\mathbf{x})$
- $p(\mathbf{w})$: prior over parameters $\mathbf{w}$
- Given a training dataset $\mathcal{D}$, $p(\mathbf{w}|\mathcal{D})$: posterior distribution over $\mathbf{w}$
- thereby, posterior distribution over regression functions
- Predictive distribution: $p(t|\mathbf{x})$ for a new input vector $\mathbf{x}$

## Gaussian process regression

- Dispense with the parametric model (parameter space)
- Define a prior probability distribution over functions directly
- **Non-parametric method**
- Infinite space: hard to work
- (in practice) working with training set and test set data points, thus working in a finite space

# Probabilistic Linear Regression Revisited

- Consider a model with $M$ fixed basis functions

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

- Prior distribution over $\mathbf{w}$ given by an isotropic Gaussian

$$p(\mathbf{w}) = \mathcal{N}\left(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I}\right)$$

- Induces a probability distribution over functions $y(\mathbf{x})$

- In practice, we wish to evaluate the function $y(\mathbf{x})$ at specific values of $\mathbf{x}$, *e.g.*, at the training data points $\mathbf{x}_1, \ldots, \mathbf{x}_N$.

**Joint distribution of the function values $y(\mathbf{x}_1), \ldots, y(\mathbf{x}_N)$**



$$\mathbf{y} = \begin{bmatrix} y(\mathbf{x}_1) & \cdots & y(\mathbf{x}_N) \end{bmatrix}^\top = \mathbf{\Phi}\mathbf{w}$$
$$\mathbf{\Phi} = [\Phi_{nk} = \phi_k(\mathbf{x})] \quad \text{(design matrix)}$$

- $\mathbf{y}$: a linear combination of Gaussian distributed variables given by the elements of $\mathbf{w} \Rightarrow p(\mathbf{y})$ **is Gaussian distributed.**
- For specification, need to find its mean and covariance

$$\mathbb{E}[\mathbf{y}] = \mathbf{\Phi}\mathbb{E}[\mathbf{w}] = \mathbf{0}$$
$$\text{Cov}[\mathbf{y}] = \mathbb{E}\left[\mathbf{y}\mathbf{y}^\top\right] = \mathbf{\Phi}\mathbb{E}\left[\mathbf{w}\mathbf{w}^\top\right]\mathbf{\Phi}^\top = \frac{1}{\alpha}\mathbf{\Phi}\mathbf{\Phi}^\top = \mathbf{K}$$

$$\mathbf{K} = \left[K_{nm} = \underbrace{k(\mathbf{x}_n, \mathbf{x}_m)}_{\text{kernel function}} = \frac{1}{\alpha}\phi(\mathbf{x}_n)^\top \phi(\mathbf{x}_m)\right] \quad \text{(Gram matrix)}$$

**A Gaussian distributed weight vector $\mathbf{w}$ induces a Gaussian joint distribution over functions.**

# General Definition of Gaussian Process

**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 $\mathbf{x}_1, \ldots, \mathbf{x}_N$ jointly have a multivariate Gaussian distribution.
  - For a single point $\mathbf{x}_1$, the output $y_1$ is a univariate Gaussian.
  - For two points $\mathbf{x}_1, \mathbf{x}_2$, the outputs $y_1, y_2$ are jointly a bivariate Gaussian.
  - ...
- Gaussian random field: cases where the input vector $\mathbf{x}$ is two dimensional
- Generally, a *stochastic process** $y(\mathbf{x})$ is specified by giving the joint probability distribution for any finite set of values $y_1, \ldots, y_N$ in a consistent manner.

* random process: a collection of random variables; this is often used to represent the evolution of some random value, or system, over time [from Wikipedia]



Probability distributions

- Describing the distribution of variables $x$ or $\mathbf{x}$

Stochastic processes

- Describing the distribution of functions $f(x)$ or $f(\mathbf{x})$
- Can think of a function as a very long vector: each entry in the vector is $f(x)$ or $f(\mathbf{x})$

- Gaussian process: a generalization of Gaussian distributions, where it is describing the distributions of functions
- Since the vector is infinite-dimensional, we constrain it to only points of training and test samples.

**A stochastic GP is specified completely by the second-order statistics, i.e., mean and covariance.**

- Zero-mean assumption
  - Mostly not have any prior knowledge about the mean of $y(\mathbf{x})$ and so by symmetry we take it to be zero
    (Equivalent to choosing the mean of the prior over weight values $p(\mathbf{w}|\alpha)$ to be zero in the basis function viewpoint)
- Covariance of $y(\mathbf{x})$ evaluated at any two values of $\mathbf{x}$, given by the *kernel function*

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

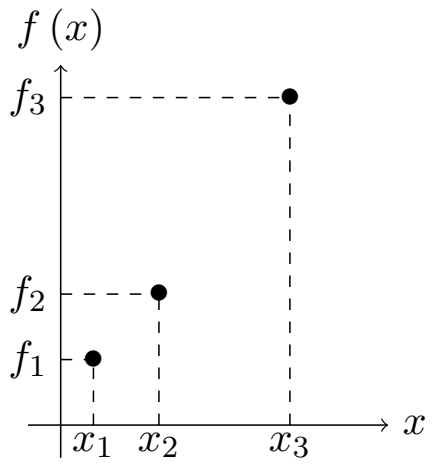
**e.g.)** Linear regression model $y(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x})$ with $p(\mathbf{w}) = \mathcal{N}\left(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I}\right)$

$$k(\mathbf{x}_n, \mathbf{x}_m) = \frac{1}{\alpha}\phi(\mathbf{x}_n)^\top \phi(\mathbf{x}_m)$$



# Defining Kernel Functions

- Indirectly through a choice of basis function
- Possible to define the kernel function directly

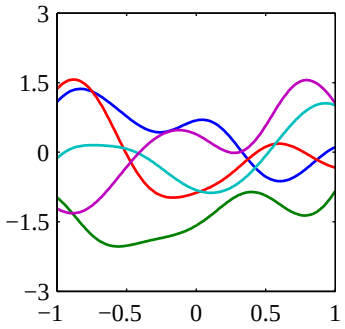



$$\begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} \sim \left( \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} \right) \sim \mathcal{N}(0, \mathbf{K})$$

$$K_{ij} = \exp\left[-\lambda \|x_i - x_j\|^2\right] = \begin{cases} 0 & \|x_i - x_j\|^2 \to \infty \\ 1 & x_i = x_j \end{cases}$$

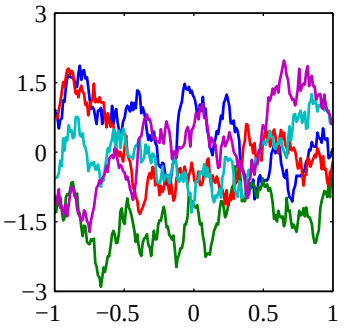
- Samples of functions from GP for two different choices of kernel function

▸ Multivariate Gaussian Sampling




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

: Gaussian kernel




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

: exponential kernel

(a.k.a., Ornstein-Uhlenbeck process: to describe Brownian motion)



# Gaussian Process for Regression

- Need to take account of the noise on the observed target values

$$t_n = y_n + \epsilon_n$$

  - $y_n = y(\mathbf{x}_n)$
  - $\epsilon_n$: random noise variable whose value is chosen independently for each observation $n$

- Gaussian distribution over $t_n$

$$p(t_n | y_n) = \mathcal{N}\left(t_n | y_n, \beta^{-1}\right)$$

  - $\beta$: hyperparameter representing the precision of the noise

- Joint distribution of the target values $\mathbf{t} = [t_1, \ldots, t_N]^\top$ conditioned on $\mathbf{y} = (y_1, \ldots, y_N)^\top$

$$p(\mathbf{t}|\mathbf{y}) = \mathcal{N}\left(\mathbf{t}|\mathbf{y}, \beta^{-1}\mathbf{I}_N\right)$$

  - $\mathbf{I}_N$: $N \times N$ unit matrix

- From the definition of a GP, the marginal distribution $p(\mathbf{y})$

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

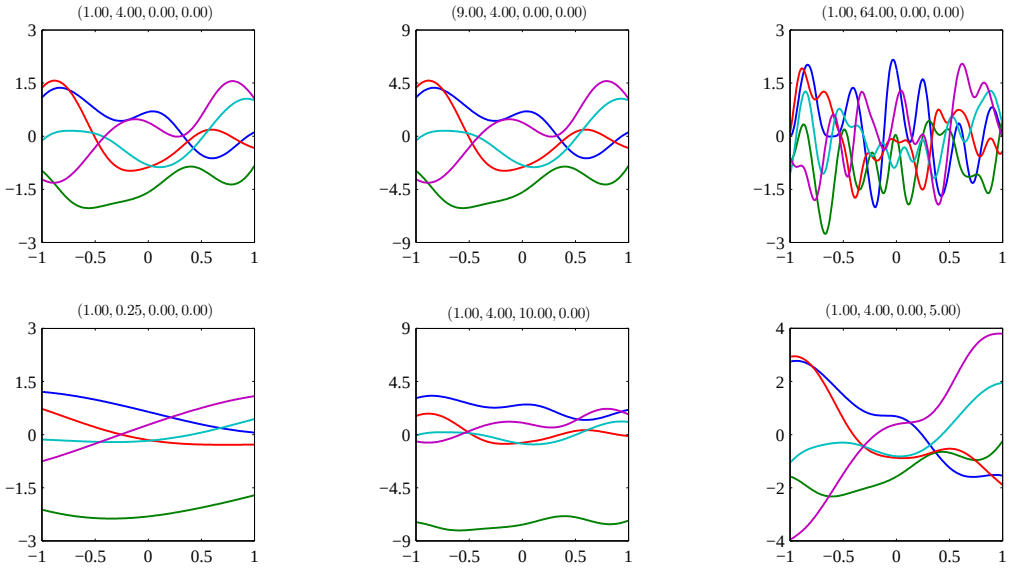
  - $\mathbf{K}$: determined by the kernel function
    - Typically chosen to express the property that, for points $\mathbf{x}_n$ and $\mathbf{x}_m$ that are "similar", the corresponding values $y(\mathbf{x}_n)$ and $y(\mathbf{x}_m)$ will be more strongly correlated than for dissimilar points
    - Notion of the similarity: application-dependent



- Exponential of a quadratic form kernel function (widely used for GP)

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

  - $\theta_3 \mathbf{x}_n^\top \mathbf{x}_m$: corresponding to a parametric model that is a linear function of the input variables



- Marginal distribution $p(\mathbf{t})$ conditioned on the input values $\mathbf{x}_1, \ldots, \mathbf{x}_N$ by integrating over $\mathbf{y}$

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

$$C(\mathbf{x}_n, \mathbf{x}_m) = \underbrace{k(\mathbf{x}_n, \mathbf{x}_m)}_{\text{due to } y(\mathbf{x})} + \underbrace{\beta^{-1}\delta_{nm}}_{\text{due to } \epsilon}$$

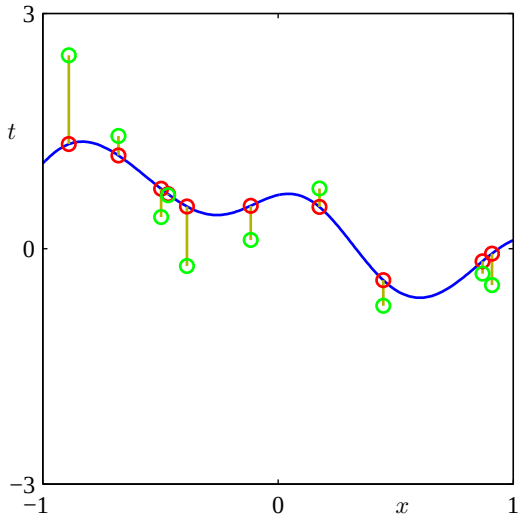
  - additive covariances because of independence of two Gaussian sources

▸ Marginal and Conditional Gaussians



Illustration of the sampling of data points $\{t_n\}$ from a Gaussian process

- Blue curve: a sample function from the Gaussian process prior over functions
- Red points: $y_n$ obtained by evaluating the function at a set of input values $\{x_n\}$
- Green: $\{t_n\}$ obtained by adding independent Gaussian noise to each of the $\{y_n\}$

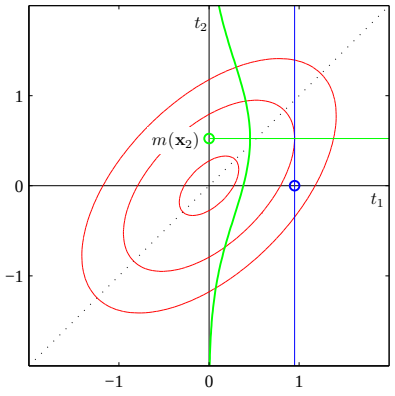


# Making Predictions using GP

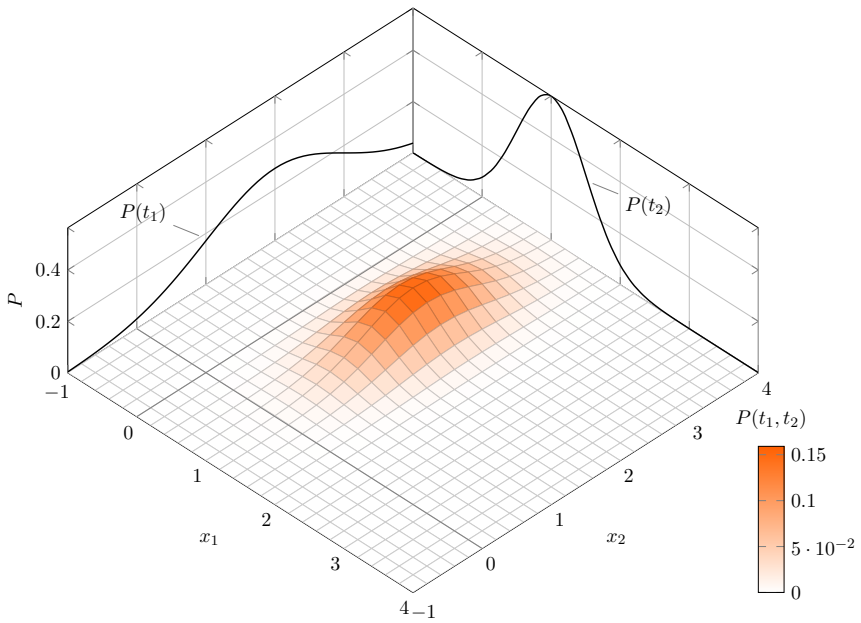
- Goal: evaluating a predictive distribution

$$p\left(t_{N+1}|\mathbf{t}_N, \mathbf{x}_{N+1}, \mathbf{x}_1, \ldots, \mathbf{x}_N\right)$$

- Mechanism of GP regression
  - $t_1/t_2$: training/test point
  - Red curve: $p\left(t_1, t_2\right)$
  - Blue line: conditioning on $t_1$
  - Green curve: $p\left(t_2|t_1\right)$
  - Predictive distribution for $t_{N+1}$: Gaussian whose mean and variance both depend on $\mathbf{x}_{N+1}$





- Begin by writing down the joint distribution $p(\mathbf{t}_{N+1})$

$$\begin{aligned} p(\mathbf{t}_{N+1}) &= \mathcal{N}(\mathbf{t}_{N+1}|\mathbf{0}, \mathbf{C}_{N+1}) \\ \mathbf{C}_{N+1} &= \begin{pmatrix} \mathbf{C}_N & \mathbf{k} \\ \mathbf{k} & c \end{pmatrix} \\ \mathbf{k} &= [k(\mathbf{x}_1, \mathbf{x}_{N+1}), \ldots, k(\mathbf{x}_N, \mathbf{x}_{N+1})]^\top \\ c &= k(\mathbf{x}_{N+1}, \mathbf{x}_{N+1}) + \beta^{-1} \end{aligned}$$

- Conditional distribution

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

$$\begin{aligned} m(\mathbf{x}_{N+1}) &= \mathbf{k}^\top \mathbf{C}_N^{-1} \mathbf{t}_N \\ \sigma^2(\mathbf{x}_{N+1}) &= c - \mathbf{k}^\top \mathbf{C}_N^{-1} \mathbf{k} \end{aligned}$$

(Chapter 2.3.1: Conditional Gaussian distributions)



$$\begin{aligned} m(\mathbf{x}_{N+1}) &= \mathbf{k}^\top \mathbf{C}_N^{-1} \mathbf{t}_N \\ \sigma^2(\mathbf{x}_{N+1}) &= c - \mathbf{k}^\top \mathbf{C}_N^{-1} \mathbf{k} \end{aligned}$$

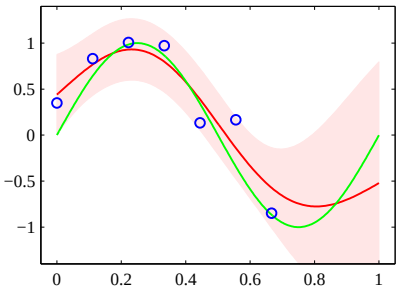
- $\mathbf{k}$: a function of the test point $\mathbf{x}_{N+1}$
- $p(\mathbf{t}_{N+1})$: Gaussian whose mean and variance both depend on $\mathbf{x}_{N+1}$

$$\begin{aligned} m(\mathbf{x}_{N+1}) &= \mathbf{k}^\top \underbrace{\mathbf{C}_N^{-1} \mathbf{t}_N}_{\equiv \mathbf{a}_N} \\ &= \sum_{n=1}^{N} a_n k(\mathbf{x}_n, \mathbf{x}_{N+1}) \end{aligned}$$

- If the kernel function $k(\mathbf{x}_n, \mathbf{x}_{N+1})$ depends only on the distance $\|\mathbf{x}_n - \mathbf{x}_{N+1}\|$, then we obtain an expansion in radial basis functions.

## Illustration of Gaussian process regression applied to sinusoidal data

- Green: sinusoidal function from which data points (blue) are obtained
- Red: mean of the Gaussian process predictive distribution
- Shaded region: ±two standard deviations
- Notice how the uncertainty increases in the region to the right of the data points.






# Computational Complexity

- Inversion of a matrix
  - Function space (*i.e.*, GP): a matrix $\mathbf{C}_N$ of size $N \times N \Rightarrow O\left(N^3\right)$
  - Parameter space: a matrix $\mathbf{S}_N$ of size $M \times M \Rightarrow O\left(M^3\right)$
- Vector-matrix multiply
  - Function space (*i.e.*, GP): $O\left(N^2\right)$
  - Parameter space: $O\left(M^2\right)$
- When $M < N$ basis function, more efficient to work in the basis function framework
- However, an advantage of a GP viewpoint is that it allows to consider covariance functions that can be expressed in terms of *an infinite number of basis functions*.

# Learning the Hyperparameters

- Predictions of a Gaussian process model will depend on the choice of covariance function.
- In practice, rather than fixing the covariance function, we may prefer to use a parametric family of functions and then infer the parameter values from data.
- Parameters govern such things as
  - length scale of the correlations: region over which variables are correlated
  - precision of the noise
- Parameters correspond to the hyperparameters in a standard parametric model.



- Techniques for learning hyperparameters are based on the evaluation of likelihood function $p(\mathbf{t}_N|\theta)$
  - $\theta$: hyperparameters of a Gaussian process model
- (Point estimate) maximizing the log-likelihood function

$$\ln p(\mathbf{t}_N|\theta) = -\frac{1}{2}\ln|\mathbf{C}_N| - \frac{1}{2}\mathbf{t}_N^\top \mathbf{C}_N^{-1}\mathbf{t}_N - \frac{N}{2}\ln(2\pi)$$

  - Using gradient-based optimization methods

$$\frac{\partial}{\partial\theta_i}\ln p(\mathbf{t}_N|\theta) = -\frac{1}{2}\mathrm{Tr}\left(\mathbf{C}_N^{-1}\frac{\partial\mathbf{C}_N}{\partial\theta_i}\right) + \frac{1}{2}\mathbf{t}_N^\top\mathbf{C}_N^{-1}\frac{\partial\mathbf{C}_N}{\partial\theta_i}\mathbf{C}_N^{-1}\mathbf{t}_N$$

  - Since $\ln p(\mathbf{t}_N|\theta)$ is in general a non-convex function, it can have multiple maxima.

- Bayesian treatment by introducing $p(\theta)$
- MAP: to maximize the log posterior using gradient-based methods
- Fully Bayesian treatment: evaluate marginals over $\theta$ weighted by the product of the prior $p(\theta)$ and the likelihood function $p(\mathbf{t}|\theta)$
  - In general, exact marginalization will be intractable, must resort to approximations



- We have assumed that the contribution to the predictive variance arising from the additive noise, governed by the parameter $\beta$, is a constant.
- From some problems, known as *heteroscedastic*, the noise variance itself will also depend on $\mathbf{x}$.
- To model this, we can extend the Gaussian process framework by introducing a second Gaussian process to represent the dependence of $\beta$ on the input $\mathbf{x}$. [Goldberg *et al.*, 1998]
- Because $\beta$ is a variance, and hence nonnegative, we use the Gaussian process to model $\ln \beta(\mathbf{x})$.

# Automatic Relevance Determination

- Maximum likelihood can be used to determine a value for the correlation length-scale parameter in a GP.

- Usefully be extended by incorporating a separate parameter for each input variable [Rasmussen and Williams, 2006]
  - **relative importance of different inputs to be inferred from the data**

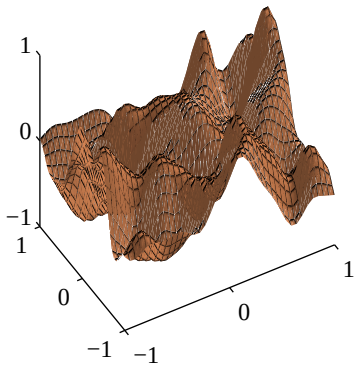
- Consider a Gaussian process with a two-dimensional input space $\mathbf{x} = (x_1, x_2)$, having a kernel function of the form

$$k\left(\mathbf{x}, \mathbf{x}'\right) = \theta_0 \exp\left\{-\frac{1}{2}\sum_{i=1}^{2} \eta_i \left(x_i - x_i'\right)^2\right\}$$

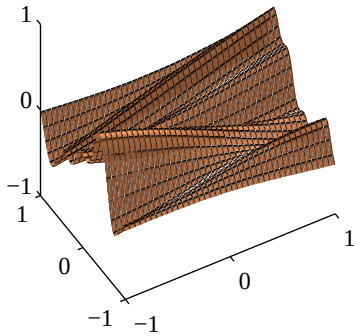


- As particular parameter $\eta_i$ becomes small, the function $y\left(\mathbf{x}\right)$ becomes relatively insensitive to the corresponding input variable $x_i$.

Samples from the ARD prior for GP




$\eta_1 = \eta_2 = 1$

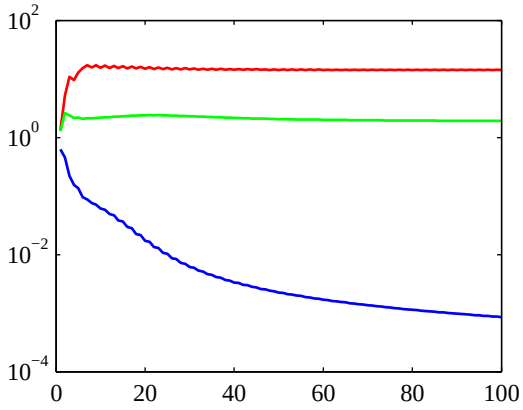



$\eta_1 = 1,\ \eta_2 = 0.01$

- By adapting these parameters to a dataset using maximum likelihood
  - Possible to detect input variables that have little effect on the predictive distribution
  - Small values of $\eta_i \Rightarrow$ discarding the corresponding variables

## Illustration of ARD in a Gaussian process

- Three inputs: $x_1$, $x_2$, and $x_3$
  - $x_1$: sampled from a Gaussian
  - $x_2 = x_1 + \epsilon$ ($\epsilon$: noise)
  - $x_3$: sampled from an independent Gaussian distribution
- Target $t = \sin(2\pi x_1) + \epsilon$
  - $x_1$: good predictor of $t$
  - $x_2$: more noisy predictor of $t$
  - $x_3$: only chance correlation with $t$




- $\eta_1$: converges to a relatively large value
- $\eta_2$: converges to a much smaller value
- $\eta_3$: becomes very small indicating that $x_3$ is irrelevant for predicting $t$



- ARD framework is incorporated into the exponential-quadratic kernel

$$k(\mathbf{x}_n, \mathbf{x}_m) == \theta_0 \exp\left\{-\frac{\theta_1}{2}\sum_{i=1}^{D}\eta_i (x_{ni} - x_{mi})^2\right\} + \theta_2 + \theta_3 \mathbf{x}_n^\top \mathbf{x}_m$$

  - $D$: dimensionality of the input space

# Gaussian Process for Classification

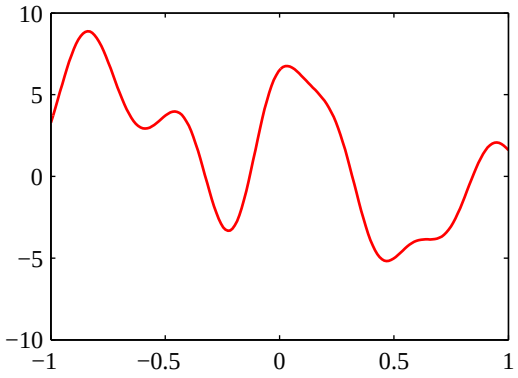
- Gaussian process model makes predictions that lie on the entire real axis.
- In a probabilistic approach to classification
  - Model the posterior probabilities of the target variables for a new input vector, given a set of training data
  - Transforming the output of the Gaussian process using an appropriate nonlinear activation function



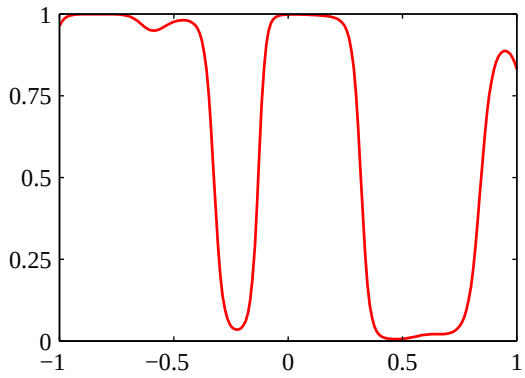
Two-class problem with a target variable $t \in \{0, 1\}$

- Define a Gaussian process over a function $a(\mathbf{x})$
- Transform $a(\mathbf{x})$ using a logistic sigmoid $y = \sigma(a)$
- Resulting in a non-Gaussian stochastic process over functions $y(\mathbf{x})$, where $y \in \{0, 1\}$

$$p(t|a) = \sigma(a)^t (1 - \sigma(a))^{1-t}$$




Sample from a Gaussian process prior over function $a(\mathbf{x})$




Result of transforming the sample using a logistic sigmoid function



- Goal: evaluating a predictive distribution $p(t_{N+1}|\mathbf{t}_N, \mathbf{x}_{N+1}, \mathbf{x}_1, \ldots, \mathbf{x}_N)$
- Introduce a Gaussian process prior over the vector $\mathbf{a}_{N+1}$

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

  - $\mathbf{a}_{N+1} = [a(\mathbf{x}_1), \ldots, a(\mathbf{x}_{N+1})]^\top$
  - $\mathbf{C}_{N+1}$: no longer includes a noise term (because we assume that all of the training data points are correctly labelled)
  - However, for numerical reasons, it is convenient to introduce a noise-like term governed by a parameter $\nu$, typically fixed in advance, that ensures that the covariance matrix is positive definite.

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

- In turn, defines a non-Gaussian process over $\mathbf{t}_{N+1}$
- By conditioning on the training data $\mathbf{t}_N$, we obtain the required predictive distribution.

- Two-class problems, it is sufficient to predict $p(t_{N+1}=1|\mathbf{t}_N)$

$$p(t_{N+1}=1|\mathbf{t}_N)=\int \underbrace{p(t_{N+1}=1|a_{N+1})}_{\sigma(a_{N+1})} p(a_{N+1}|\mathbf{t}_N)\,da_{N+1}$$

- Analytically intractable: approximation techniques
  - Laplace approximation
  - Variational inference [Gibbs & McKay, 2000]
    - Making use of the local variational bound on logistic sigmoid
    - Allows product of sigmoid functions to be approximated by a product of Gaussians, thereby allowing the marginalization over $\mathbf{a}_N$ to be performed analytically
  - Expectation Propagation (EP) [Opper & Winther, 2000; Minka, 2001; Seeger, 2003]
    - Because the true posterior distribution is unimodal, the EP can give good results.



# Laplace Approximation

- Seek a Gaussian approximation to the posterior distribution over $a_{N+1}$

$$\begin{aligned}
p(a_{N+1}|\mathbf{t}_N) &= \int p(a_{N+1},\mathbf{a}_N|\mathbf{t}_N)\,d\mathbf{a}_N \\
&= \frac{1}{p(\mathbf{t}_N)}\int p(a_{N+1},\mathbf{a}_N)\,p(\mathbf{t}_N|a_{N+1},\mathbf{a}_N)\,d\mathbf{a}_N \\
&= \frac{1}{p(\mathbf{t}_N)}\int p(a_{N+1}|\mathbf{a}_N)\,p(\mathbf{a}_N)\,p(\mathbf{t}_N|\mathbf{a}_N)\,d\mathbf{a}_N \\
&= \int p(a_{N+1}|\mathbf{a}_N)\,p(\mathbf{a}_N|\mathbf{t}_N)\,d\mathbf{a}_N
\end{aligned}$$

$$p\left(a_{N+1}|\mathbf{t}_N\right) = \int p\left(a_{N+1}, \mathbf{a}_N\right) p\left(\mathbf{t}_N|\mathbf{a}_N\right) d\mathbf{a}_N$$

$$p\left(a_{N+1}|\mathbf{a}_N\right) = \mathcal{N}\left(m\left(a_{N+1}\right), \sigma^2\left(a_{N+1}\right)\right)$$

$$\begin{aligned} m\left(a_{N+1}\right) &= \mathbf{k}^\top \mathbf{C}_N^{-1} \mathbf{a}_N \\ \sigma^2\left(a_{N+1}\right) &= c - \mathbf{k}^\top \mathbf{C}_N^{-1} \mathbf{k} \end{aligned}$$

- Need to evaluate the integral by finding a Laplace approximation for the posterior distribution $p\left(\mathbf{t}_N|\mathbf{a}_N\right)$
- Use the standard result for the convolution of two Gaussian distributions



- Prior $p\left(\mathbf{a}_N\right) = \mathcal{N}\left(\mathbf{0}, \mathbf{C}_N\right)$
- Assuming independence of the data points

$$p\left(\mathbf{t}_N|\mathbf{a}_N\right) = \prod_{n=1}^{N} \sigma\left(a_n\right)^{t_n} \left(1 - \sigma\left(a_n\right)\right)^{1-t_n} = \prod_{n=1}^{N} e^{a_n t_n} \sigma\left(-a_n\right)$$

- Obtaining the Laplace approximation by Taylor expanding the logarithm of $p\left(\mathbf{t}_N|\mathbf{a}_N\right)$, which up to an additive normalization constant is given by

$$\begin{aligned} \Psi\left(\mathbf{a}_N\right) &= \ln p\left(\mathbf{a}_N\right) + \ln p\left(\mathbf{t}_N|\mathbf{a}_N\right) \\ &= -\frac{1}{2}\mathbf{a}_N^\top \mathbf{C}_N^{-1} \mathbf{a}_N - \frac{N}{2}\ln\left(2\pi\right) - \frac{1}{2}\ln\left|\mathbf{C}_N\right| + \mathbf{t}_N^\top \mathbf{a}_N \\ &\quad - \sum_{n=1}^{N} \ln\left(1 + e^{a_n}\right) + \text{const.} \end{aligned}$$

- Find the mode of the posterior distribution
  - Evaluating the gradient of $\Psi(\mathbf{a}_N)$

$$\nabla\Psi(\mathbf{a}_N) = \mathbf{t}_N - \boldsymbol{\sigma}_N - \mathbf{C}_N^{-1}\mathbf{a}_N$$

  - $\boldsymbol{\sigma}_N = [\sigma(a_n)]^\top$: depends nonlinearly on $\mathbf{a}_N$
  - Thus, cannot simply find the mode by setting the gradient to zero

- Resort to an iterative scheme based on the Newton-Raphson method, which gives rise to an Iterative Reweighted Least Squares (IRLS) algorithm
- Requiring the second derivatives of $\Psi(\mathbf{a}_N)$, which also require for the Laplace approximation

$$\nabla\nabla\Psi(\mathbf{a}_N) = -\mathbf{W}_N - \mathbf{C}_N^{-1}$$

  - $\mathbf{W}_N = \text{diag}\left(\sigma(a_n)^{t_n}(1-\sigma(a_n))^{1-t_n}\right)$



- Hessian matrix $\mathbf{H} = -\nabla\nabla\Psi(\mathbf{a}_N)$: positive definite
  - $\mathbf{C}_N$: positive definite by construction
  - $\mathbf{W}_N$: positive definite
    - $\because$ $\sigma(a_n)^{t_n}(1-\sigma(a_n))^{1-t_n}$ lies in the range $(0, 1/4)$

- The posterior distribution $p(\mathbf{a}_N|\mathbf{t}_N)$ is log convex and therefore has a single mode that is the global maximum.

- However, since $\mathbf{H}$ is a function of $\mathbf{a}_N$, the posterior is not Gaussian.

Using the Newton-Raphson formula (*i.e.*, $\mathbf{a}_N^{\text{new}} = \mathbf{a}_N^{\text{old}} - \mathbf{H}^{-1}\nabla\Psi\left(\mathbf{a}_N^{\text{old}}\right)$)

- Iterative update equation for $\mathbf{a}_N$ until convergence $\mathbf{a}_N^\star$

$$\mathbf{a}_N^{\text{new}} = \mathbf{C}_N\left(\mathbf{I} + \mathbf{W}_N\mathbf{C}_N\right)^{-1}\left\{\mathbf{t}_N - \boldsymbol{\sigma}_N + \mathbf{W}_N\mathbf{a}_N\right\}$$

- At the mode, the gradient $\nabla\Psi\left(\mathbf{a}_N\right)$ will vanish, and hence $\mathbf{a}_N^\star$ will satisfy

$$\mathbf{a}_N^\star = \mathbf{C}_N\left(\mathbf{t}_N - \boldsymbol{\sigma}_N\right)$$

- After finding the mode $\mathbf{a}_N^\star$, evaluate the Hessian matrix $\mathbf{H}$

$$\mathbf{H} = -\nabla\nabla\Psi\left(\mathbf{a}_N\right) = \mathbf{W}_N + \mathbf{C}_N^{-1}$$

- This defines the Gaussian approximation to the posterior distribution $p\left(\mathbf{a}_N|\mathbf{t}_N\right)$

$$q\left(\mathbf{a}_N\right) = \mathcal{N}\left(\mathbf{a}_N|\mathbf{a}_N^\star, \mathbf{H}^{-1}\right)$$



$$p\left(a_{N+1}|\mathbf{t}_N\right) = \int p\left(a_{N+1}|\mathbf{a}_N\right)p\left(\mathbf{a}_N|\mathbf{t}_N\right)d\mathbf{a}_N$$

$$p\left(a_{N+1}|\mathbf{a}_N\right) = \mathcal{N}\left(m\left(a_{N+1}\right), \sigma^2\left(a_{N+1}\right)\right)$$

$$\begin{aligned} m\left(a_{N+1}\right) &= \mathbf{k}^\top\mathbf{C}_N^{-1}\mathbf{a}_N \\ \sigma^2\left(a_{N+1}\right) &= c - \mathbf{k}^\top\mathbf{C}_N^{-1}\mathbf{k} \end{aligned}$$

$$p\left(\mathbf{a}_N|\mathbf{t}_N\right) \approx q\left(\mathbf{a}_N\right) = \mathcal{N}\left(\mathbf{a}_N|\mathbf{a}_N^\star, \mathbf{H}^{-1}\right)$$

$$\Downarrow$$

$$\begin{aligned} \mathbb{E}\left[a_{N+1}|\mathbf{t}_N\right] &= \mathbf{k}^\top\left(\mathbf{t}_N - \boldsymbol{\sigma}_N\right) \\ \text{Var}\left[a_{N+1}|\mathbf{t}_N\right] &= c - \mathbf{k}^\top\left(\mathbf{W}_N^{-1} + \mathbf{C}_N\right)^{-1}\mathbf{k} \end{aligned}$$

- Now, we have a Gaussian distribution for $p(a_{N+1}|\mathbf{t}_N)$, it is possible to approximate the integration

$$\begin{aligned} p(t_{N+1}=1|\mathbf{t}_N) &= \int \underbrace{p(t_{N+1}=1|a_{N+1})}_{\sigma(a_{N+1})} p(a_{N+1}|\mathbf{t}_N)\, da_{N+1} \\ &= \int \sigma(a_{N+1})\, \mathcal{N}(a_{N+1}|\mathbb{E}[a_{N+1}|\mathbf{t}_N], \mathrm{Var}[a_{N+1}|\mathbf{t}_N])\, da \\ &\approx \sigma(\kappa(\mathrm{Var}[a_{N+1}|\mathbf{t}_N])\, \mathbb{E}[a_{N+1}|\mathbf{t}_N]) \end{aligned}$$

(Refer to Eq. (4.153) in Chapter 4.5.2)



## Learning hyperparameters from data

- Determining the parameters $\theta$ of the covariance function
- By maximizing the likelihood function given by $p(\mathbf{t}_N|\theta)$

$$p(\mathbf{t}_N|\theta) = \int p(\mathbf{t}_N|\mathbf{a}_N)\, p(\mathbf{a}_N|\theta)\, d\mathbf{a}_N$$

  - Expressions for the log likelihood and its gradient
  - If desired, suitable regularization can be added, leading to a penalized maximum likelihood solution.

- Log-likelihood (based on Laplace approximation)

$$\ln p\left(\mathbf{t}_N|\theta\right) = \underbrace{\ln p\left(\mathbf{a}_N^\star|\theta\right) + \ln p\left(\mathbf{t}_N|\mathbf{a}_N^\star\right)}_{\equiv\Psi\left(\mathbf{a}_N^\star\right)} - \frac{1}{2}\ln\left|\mathbf{W}_N + \mathbf{C}_N^{-1}\right| + \frac{N}{2}\ln\left(2\pi\right)$$

$$\left(\because Z = \int f(z)dz = f(z_0)\int \exp\left\{-\frac{A}{2}\left(z-z_0\right)^2\right\}dz = f(z_0)\frac{(2\pi)^{1/2}}{A^{1/2}}\right)$$

- Gradient of $\ln p\left(\mathbf{t}_N|\theta\right)$ w.r.t. the parameter vector $\theta$

$$\frac{\partial \ln p\left(\mathbf{t}_N|\theta\right)}{\partial\theta_j} = \frac{1}{2}\mathbf{a}_N^{\star\top}\mathbf{C}_N^{-1}\frac{\partial\mathbf{C}_N}{\partial\theta_j}\mathbf{C}_N^{-1}\mathbf{a}_N^\star - \frac{1}{2}\mathrm{Tr}\left[\left(\mathbf{I}+\mathbf{C}_N\mathbf{W}_N\right)^{-1}\mathbf{W}_N\frac{\partial\mathbf{C}_N}{\partial\theta_j}\right]$$

- Changes in $\theta$ will cause changes in $\mathbf{a}_N^\star$, leading to additional terms in the gradient
  - Dependence of the covariance matrix $\mathbf{C}_N$ on $\theta$
  - Dependence of $\mathbf{a}_N^\star$ on $\theta$



- Laplace approximation has been constructed such that $\Psi\left(\mathbf{a}_N\right)$ has zero gradient at $\mathbf{a}_N = \mathbf{a}_N^\star$, and so $\Psi\left(\mathbf{a}_N\right)$ gives no contribution to the gradient as a result of its dependence on $\mathbf{a}_N^\star$.
- Contribution to the derivative w.r.t. a component $\theta_j$ of $\theta$

$$-\frac{1}{2}\sum_{n=1}^{N}\frac{\partial\ln\left|\mathbf{W}_N+\mathbf{C}_N^{-1}\right|}{\partial a_n^\star}\frac{\partial a_n^\star}{\theta_j}$$

$$= -\frac{1}{2}\sum_{n=1}^{N}\left[\left(\mathbf{I}+\mathbf{C}_N\mathbf{W}_N\right)^{-1}\mathbf{C}_N\right]_{nn}\sigma_n^\star\left(1-\sigma_n^\star\right)\left(1-2\sigma_n^\star\right)\frac{\partial a_n^\star}{\partial\theta_j}$$

where $\sigma_n^\star = \sigma\left(a_n^\star\right)$

- Derivative of $a_n^\star$ w.r.t. $\theta_j$

$$\begin{aligned}\frac{\partial a_n^\star}{\partial\theta_j} &= \frac{\partial\mathbf{C}_N}{\partial\theta_j}\left(\mathbf{t}_N-\boldsymbol{\sigma}_N\right) - \mathbf{C}_N\mathbf{W}_N\frac{\partial a_n^\star}{\partial\theta_j}\\ &= \left(\mathbf{I}+\mathbf{W}_N\mathbf{C}_N\right)^{-1}\frac{\partial\mathbf{C}_N}{\partial\theta_j}\left(\mathbf{t}_N-\boldsymbol{\sigma}_N\right)\end{aligned}$$

$$\frac{\partial \ln p\left(\mathbf{t}_N|\theta\right)}{\partial \theta_j} = \frac{1}{2}\mathbf{a}_N^{\star\top}\mathbf{C}_N^{-1}\frac{\partial \mathbf{C}_N}{\partial \theta_j}\mathbf{C}_N^{-1}\mathbf{a}_N^{\star} - \frac{1}{2}\mathrm{Tr}\left[\left(\mathbf{I}+\mathbf{C}_N\mathbf{W}_N\right)^{-1}\mathbf{W}_N\frac{\partial \mathbf{C}_N}{\partial \theta_j}\right]$$

$$-\frac{1}{2}\sum_{n=1}^{N}\frac{\partial \ln\left|\mathbf{W}_N+\mathbf{C}_N^{-1}\right|}{\partial a_n^{\star}}\frac{\partial a_n^{\star}}{\theta_j}$$

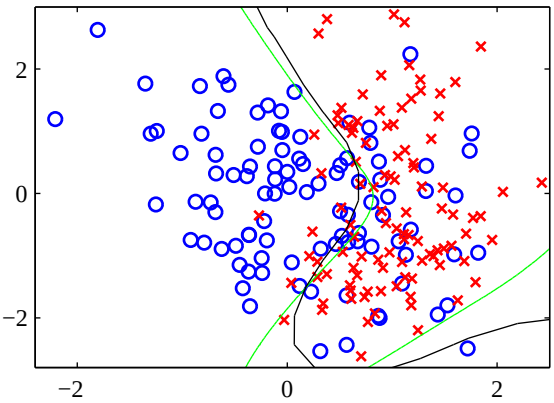
$$= -\frac{1}{2}\sum_{n=1}^{N}\left[\left(\mathbf{I}+\mathbf{C}_N\mathbf{W}_N\right)^{-1}\mathbf{C}_N\right]_{nn}\sigma_n^{\star}\left(1-\sigma_n^{\star}\right)\left(1-2\sigma_n^{\star}\right)\frac{\partial a_n^{\star}}{\partial \theta_j}$$

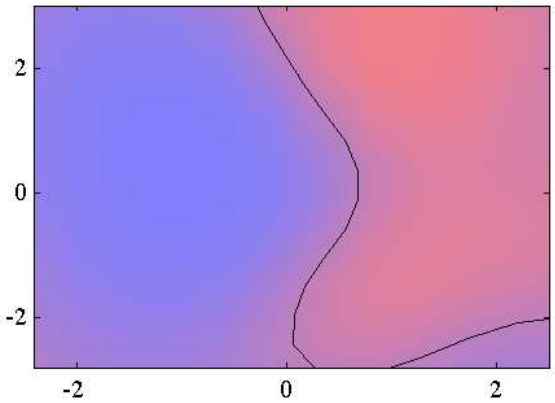
$$\frac{\partial a_n^{\star}}{\partial \theta_j} = \left(\mathbf{I}+\mathbf{W}_N\mathbf{C}_N\right)^{-1}\frac{\partial \mathbf{C}_N}{\partial \theta_j}\left(\mathbf{t}_N-\boldsymbol{\sigma}_N\right)$$

By combining these, we can evaluate the gradient of the log likelihood function and determine a value for $\theta$ with standard nonlinear optimization algorithms.



## Illustration of the use of a Gaussian process for classification




- Green: true distribution
- Black: decision boundary from the Gaussian process classifier
- Predicted posterior probability

# Connection to Neural Networks

- NN: the number $M$ of hidden units determine the range of functions that can be represented
  - For sufficiently large $M$, a 2-layer network can approximate any given function with arbitrary accuracy.
- Bayesian NN: prior distribution over the parameter vector $\mathbf{w}$ produces a prior distribution over functions from $y(\mathbf{w})$
- In the limit $M \to \infty$, the distribution of functions generated by NN will tend to a Gaussian process [Neal, 1996].
  - However, in this limit, the output variables of the NN become independent.



- NN outputs share the hidden units thus borrowing statistical strength from each other
  - The weights associated with each hidden unit are influenced by all of the output variables not just by one of them.
  - This property is lost in the Gaussian process limit.
- Nonstationary kernel functions
  - cannot be expressed as a function of the difference $\mathbf{x} - \mathbf{x}'$, as a consequence of the Gaussian weight prior being centered on zero which breaks translation invariance in weight space
- By working directly with the covariance function
  - we have implicitly marginalized over the distribution of weights
  - If the weight prior is governed by hyperparameters, then their values will determine the length scales of the distribution over functions.
  - cannot marginalize out the hyperparameters analytically, and must instead resort to approximation techniques