Pattern Recognition and Machine Learning
Chapter 6: Kernel Methods

Vasil Khalidov Alex Kläser

December 13, 2007

Training Data: Keep or Discard?

- **Parametric** methods (linear/nonlinear) so far:
  - learn parameter vector $\boldsymbol{w}$ or posterior distribution $p(\boldsymbol{w}|\boldsymbol{t})$
  - discard training data $\boldsymbol{t}$

- **Non-parametric** methods:
  - **Parzen** probability density model: set of kernel functions centered on training data points
  - **Nearest neighbours** technique: closest example from the training set
  - **Memory-based** methods: similar examples from the training set

- **Kernel** methods:
  - prediction is based on linear combinations of a *kernel function* evaluated at the training data points

Kernel Function

- for models based on feature space mapping $\boldsymbol{\phi}(\boldsymbol{x})$:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}') \tag{6.1}$$

- symmetric function: $k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}', \boldsymbol{x})$

- simple example - *linear* kernel: $\boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{x}$
- *stationary* kernel: $k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x} - \boldsymbol{x}')$
- *homogeneous* kernel: $k(\boldsymbol{x}, \boldsymbol{x}') = k(\|\boldsymbol{x} - \boldsymbol{x}'\|)$

Algorithm, expressed in terms of scalar products can be reformulated using *kernel substitution*: PCA, nearest-neighbour classifiers, Fisher discriminant

Dual Representation

Many linear models for regression and classification can be reformulated in terms of a dual representation in which the kernel function arises naturally.

- linear regression model ($\lambda \geq 0$):

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) - t_n\right\}^2 + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w} \tag{6.2}$$

- set the gradient to zero:

$$\boldsymbol{w} = -\frac{1}{\lambda}\sum_{n=1}^{N}\left\{\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) - t_n\right\}\boldsymbol{\phi}(\boldsymbol{x}_n) = \boldsymbol{\Phi}^T\boldsymbol{a} \tag{6.3}$$

- substitute $\boldsymbol{w}$ and define the *Gram* matrix $\mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^T$:

$$K_{nm} = \boldsymbol{\phi}(\boldsymbol{x}_n)^T\boldsymbol{\phi}(\boldsymbol{x}_m) = k(\boldsymbol{x}_n, \boldsymbol{x}_m) \tag{6.6}$$

Solution for Dual Problem

- in terms of new parameter vector $\boldsymbol{a}$:

$$\boldsymbol{w} = \frac{1}{2}\boldsymbol{a}^T\mathbf{K}\mathbf{K}\boldsymbol{a} - \boldsymbol{a}^T\mathbf{K}\boldsymbol{t} + \frac{1}{2}\boldsymbol{t}^T\boldsymbol{t} + \frac{\lambda}{2}\boldsymbol{a}^T\mathbf{K}\boldsymbol{a} \tag{6.7}$$

- set the gradient to zero:

$$\boldsymbol{a} = (\mathbf{K} + \lambda\mathbf{I}_N)^{-1}\boldsymbol{t} \tag{6.8}$$

- prediction for a new input $\boldsymbol{x}$

$$y(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{a}^T\boldsymbol{\Phi}\boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x})^T(\mathbf{K} + \lambda\mathbf{I}_N)^{-1}\boldsymbol{t} \tag{6.9}$$

where

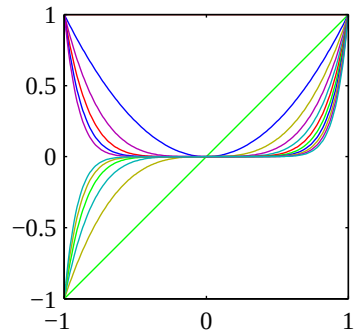
$$\boldsymbol{k}(\boldsymbol{x}) \quad = (k(\boldsymbol{x}_1, \boldsymbol{x}), ..., k(\boldsymbol{x}_N, \boldsymbol{x}))$$

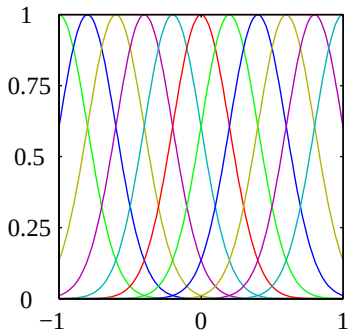
- inverting $N \times N$ matrix instead of $M \times M$

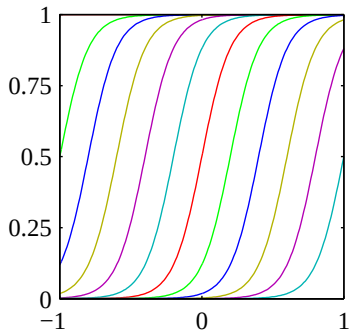
Constructing Kernels - First Approach

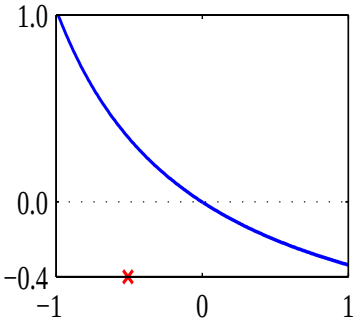
- choose feature space mapping $\boldsymbol{\phi}(\boldsymbol{x})$

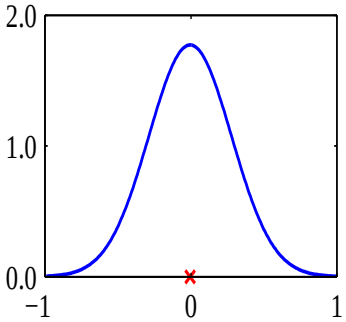
$$k(x, x') = \boldsymbol{\phi}(x)^T \boldsymbol{\phi}(x') = \sum_{i=1}^{M} \phi_i(x)\phi_i(x') \tag{6.10}$$

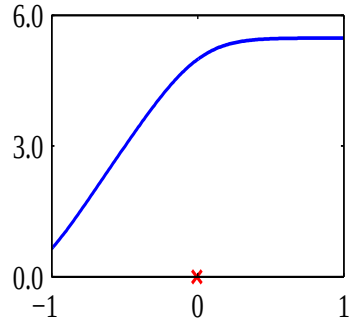












Constructing Kernels - Second Approach

- construct kernel function directly and verify its validity
- simple example

$$k(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x}^T \boldsymbol{z})^2 \tag{6.11}$$

in 2-D case corresponds to

$$k(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{z}) \tag{6.12}$$

with $\boldsymbol{\phi}(\boldsymbol{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)^T$

To test validity without having to construct the function $\boldsymbol{\phi}(\boldsymbol{x})$ explicitly, one can use the condition:

Function $k(\boldsymbol{x}, \boldsymbol{x}')$ is a valid kernel $\iff \mathbf{K} \geq 0 \qquad \forall\{x_n\}$

Combining Kernels

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

$$
\begin{aligned}
k(\boldsymbol{x}, \boldsymbol{x}') &= ck_1(\boldsymbol{x}, \boldsymbol{x}') && (6.13)\\
k(\boldsymbol{x}, \boldsymbol{x}') &= f(\boldsymbol{x})k_1(\boldsymbol{x}, \boldsymbol{x}')f(\boldsymbol{x}') && (6.14)\\
k(\boldsymbol{x}, \boldsymbol{x}') &= q(k_1(\boldsymbol{x}, \boldsymbol{x}')) && (6.15)\\
k(\boldsymbol{x}, \boldsymbol{x}') &= exp(k_1(\boldsymbol{x}, \boldsymbol{x}')) && (6.16)\\
k(\boldsymbol{x}, \boldsymbol{x}') &= k_1(\boldsymbol{x}, \boldsymbol{x}') + k_2(\boldsymbol{x}, \boldsymbol{x}') && (6.17)\\
k(\boldsymbol{x}, \boldsymbol{x}') &= k_1(\boldsymbol{x}, \boldsymbol{x}')k_2(\boldsymbol{x}, \boldsymbol{x}') && (6.18)\\
k(\boldsymbol{x}, \boldsymbol{x}') &= k_3(\boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}')) && (6.19)\\
k(\boldsymbol{x}, \boldsymbol{x}') &= \boldsymbol{x}^T \mathbf{A} \boldsymbol{x}' && (6.20)\\
k(\boldsymbol{x}, \boldsymbol{x}') &= k_a(\boldsymbol{x}_a, \boldsymbol{x}'_a) + k_b(\boldsymbol{x}_b, \boldsymbol{x}'_b) && (6.21)\\
k(\boldsymbol{x}, \boldsymbol{x}') &= k_a(\boldsymbol{x}_a, \boldsymbol{x}'_a)k_b(\boldsymbol{x}_b, \boldsymbol{x}'_b) && (6.22)
\end{aligned}
$$

with corresponding conditions on $c, f, q, \boldsymbol{\phi}, k_3, \mathbf{A}, \boldsymbol{x}_a, \boldsymbol{x}_b, k_a, k_b$

Examples

- Polynomial kernels:

  $k(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^T \boldsymbol{x}')^2$ contains only terms of degree 2
  $k(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^T \boldsymbol{x}' + c)^2$, $c > 0$
  $k(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^T \boldsymbol{x}')^M$ contains all monomials of order M
  $k(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^T \boldsymbol{x}' + c)^M$, $c > 0$

- *Gaussian* kernel:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\|\boldsymbol{x} - \boldsymbol{x}'\|^2 / 2\sigma^2) \tag{6.23}$$

  Note: can substitute $\boldsymbol{x}^T \boldsymbol{x}'$ with a nonlinear kernel $\kappa(\boldsymbol{x}, \boldsymbol{x}')$

- Kernel on nonvectorial space:

$$k(A_1, A_2) = 2^{|A_1 \cap A_2|} \tag{6.27}$$

- Sigmoidal kernel:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \tanh(a \boldsymbol{x}^T \boldsymbol{x}' + b) \tag{6.37}$$

Probabilistic generative models

- Kernel for generative model $p(\boldsymbol{x})$:

$$k(\boldsymbol{x}, \boldsymbol{x}') = p(\boldsymbol{x})p(\boldsymbol{x}') \tag{6.28}$$

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sum_i p(\boldsymbol{x}|i)p(\boldsymbol{x}'|i)p(i) \tag{6.29}$$

$$k(\boldsymbol{x}, \boldsymbol{x}') = \int p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{x}'|\boldsymbol{z})p(\boldsymbol{z})d\boldsymbol{z} \tag{6.30}$$

- Kernel for HMM:

$$k(\mathbf{X}, \mathbf{X}') = \sum_{\mathbf{Z}} p(\mathbf{X}|\mathbf{Z})p(\mathbf{X}'|\mathbf{Z})p(\mathbf{Z}) \tag{6.31}$$

$\mathbf{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_L\}$ - observations
$\mathbf{Z} = \{\boldsymbol{z}_1, ..., \boldsymbol{z}_L\}$ - hidden states

Fisher kernel

- parametric generative model $p(\boldsymbol{x}|\boldsymbol{\theta})$
- *Fisher score*:

$$\boldsymbol{g}(\boldsymbol{\theta}, \boldsymbol{x}) = \nabla_{\boldsymbol{\theta}} \ln p(\boldsymbol{x}|\boldsymbol{\theta}) \tag{6.32}$$

- *Fisher kernel* and information matrix:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{g}(\boldsymbol{\theta}, \boldsymbol{x})^T \mathbf{F}^{-1} \boldsymbol{g}(\boldsymbol{\theta}, \boldsymbol{x}') \tag{6.33}$$

$$\mathbf{F} = \mathbb{E}_{\boldsymbol{x}} \left[\boldsymbol{g}(\boldsymbol{\theta}, \boldsymbol{x}) \boldsymbol{g}(\boldsymbol{\theta}, \boldsymbol{x})^T | \boldsymbol{\theta} \right] \tag{6.34}$$

- *Note*: the kernel is invariant under $\boldsymbol{\theta} \rightarrow \boldsymbol{\psi}(\boldsymbol{\theta})$
- Simplify matrix calculation:

$$\mathbf{F} \simeq \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{g}(\boldsymbol{\theta}, \boldsymbol{x}_n) \boldsymbol{g}(\boldsymbol{\theta}, \boldsymbol{x}_n)^T \tag{6.35}$$

Or simply omit the Fisher information matrix

Radial Basis Functions

- By definition $\phi_j(\boldsymbol{x}) = h(\|\boldsymbol{x} - \boldsymbol{\mu}_j\|)$
- Originally were introduced for the problem of exact interpolation $f(\boldsymbol{x}_n) = t_n$:

$$f(\boldsymbol{x}) = \sum_{n=1}^{N} w_n h(\|\boldsymbol{x} - \boldsymbol{x}_n\|) \tag{6.38}$$

- Green's functions for an isotropic differential operator in regularizer
- Interpolation problem with noisy inputs

$$E = \frac{1}{2} \sum_{n=1}^{N} \int \{y(\boldsymbol{x}_n + \boldsymbol{\xi}) - t_n\}^2 \nu(\boldsymbol{\xi}) d\boldsymbol{\xi} \tag{6.39}$$

Interpolation With Noisy Inputs

- Sum of squares function

$$E = \frac{1}{2} \sum_{n=1}^{N} \int \{y(\boldsymbol{x}_n + \boldsymbol{\xi}) - t_n\}^2 \nu(\boldsymbol{\xi}) d\boldsymbol{\xi} \tag{6.39}$$

- Optimal value

$$y_(\boldsymbol{x}_n) = \sum_{n=1}^{N} t_n h(\boldsymbol{x} - \boldsymbol{x}_n) \tag{6.40}$$

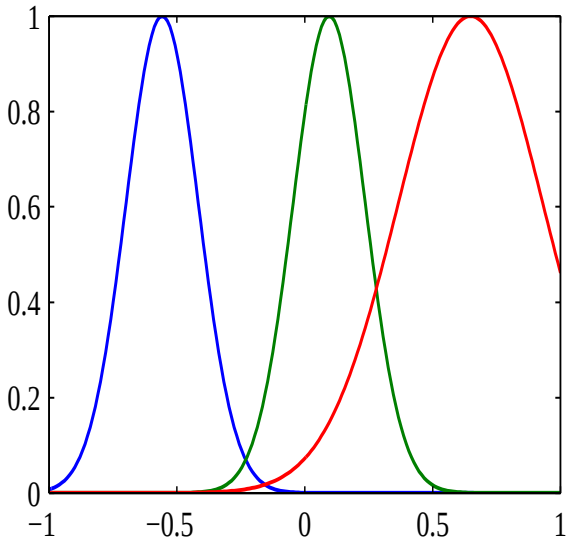
with basis functions given by normalized functions

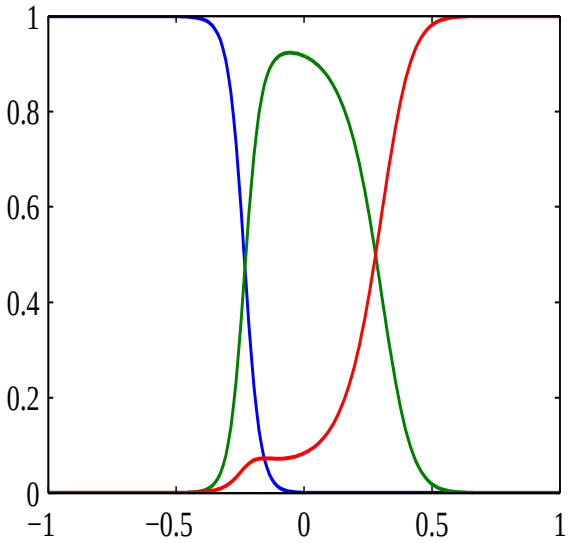
$$h(\boldsymbol{x} - \boldsymbol{x}_n) = \frac{\nu(\boldsymbol{x} - \boldsymbol{x}_n)}{\sum_{n=1}^{N} \nu(\boldsymbol{x} - \boldsymbol{x}_n)} \tag{6.41}$$

- If noise distribution $\nu(\boldsymbol{\xi})$ is isotropic, basis functions would be radial

Normalization effect

Avoids regions in an input space where all of the basis functions take small values





Reducing Size of the Basis

- Keep number of basis functions M smaller than input data size N
- Centers locations $\boldsymbol{\mu}_i$ are determined based on the input data $\{\boldsymbol{x}_n\}$ alone
- Coefficients $\{w_i\}$ are determined by least squares
- Choice of centers:
  - random
  - orthogonal least squares - greatest error reduction
  - using clustering algorithms

Parzen Density Estimator

- Prediction of linear regression model - linear combination of t_n with 'equivalent kernel' values

- Same result starting from Parzen density estimator

$$p(\boldsymbol{x}, t) = \frac{1}{N} \sum_{n=1}^{N} f(\boldsymbol{x} - \boldsymbol{x}_n, t - t_n) \tag{6.42}$$

- Regression function

$$\begin{aligned} y(\boldsymbol{x}) &= \mathbb{E}[t|\boldsymbol{x}] = \int_{-\infty}^{\infty} t p(t|\boldsymbol{x}) dt = \\ &= \frac{\int t p(\boldsymbol{x}, t) dt}{\int p(\boldsymbol{x}, t) dt} = \\ &= \frac{\sum_n \int t f(\boldsymbol{x} - \boldsymbol{x}_n, t - t_n) dt}{\sum_m \int f(\boldsymbol{x} - \boldsymbol{x}_m, t - t_m) dt} \end{aligned} \tag{6.43}$$

Nadaraya-Watson Model

Assume that the component density functions have zero mean so that

$$\int_{-\infty}^{\infty} t f(\boldsymbol{x}, t) dt = 0 \tag{6.44}$$

for all values of $\boldsymbol{x}$. Then by variable change

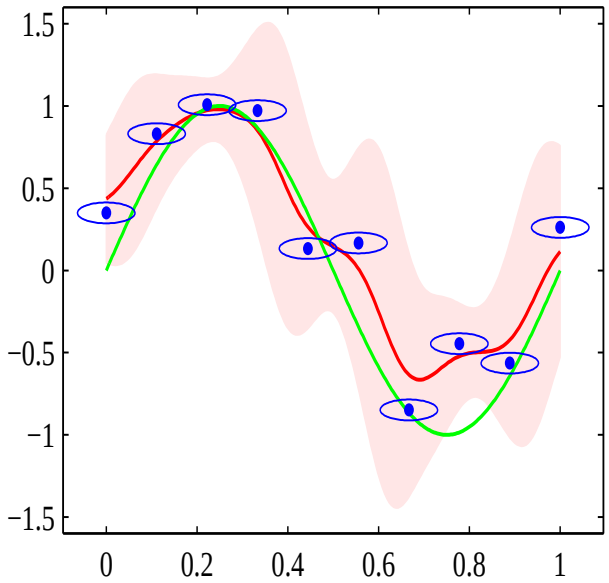
$$\begin{aligned} y(\boldsymbol{x}) &= \frac{\sum_n g(\boldsymbol{x} - \boldsymbol{x}_n) t_n}{\sum_m g(\boldsymbol{x} - \boldsymbol{x}_m)} = \\ &= \sum_n k(\boldsymbol{x}, \boldsymbol{x}_n) t_n \end{aligned} \tag{6.45}$$

with $g(\boldsymbol{x}) = \int_{-\infty}^{\infty} f(\boldsymbol{x}, t) dt$ and $k(\boldsymbol{x}, \boldsymbol{x}_n)$ given by

$$k(\boldsymbol{x}, \boldsymbol{x}_n) = \frac{g(\boldsymbol{x} - \boldsymbol{x}_n)}{\sum_m g(\boldsymbol{x} - \boldsymbol{x}_m)} \tag{6.46}$$

Illustration of the Nadaraya-Watson Model

single input variable x
$f(x,t)$ is a zero-mean isotropic Gaussian with variance σ^2



Conditional distribution

$$p(t|\boldsymbol{x}) = \frac{p(t,\boldsymbol{x})}{\int p(t,\boldsymbol{x})dt} = \frac{\sum_n f(\boldsymbol{x}-\boldsymbol{x}_n, t-t_n)}{\sum_m \int f(\boldsymbol{x}-\boldsymbol{x}_m, t-t_m)dt} \tag{6.47}$$

is given by a mixture of Gaussians

Gaussian processes: Key Idea

- The idea is similar to linear regression with a fixed set of basis functions (Chapter 3):

$$y(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) \tag{6.49}$$

- However, we forget about the parametric model
- Instead we take a infinite number of basis functions given by a probability distribution over functions
- Might loook difficult to handle, but it is not . . . we only have to consider the values of the functions at training/test data points

Linear regression revisited

Model defined by linear combination of M fixed basis functions:

$$y(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) \tag{6.49}$$

A Gaussian prior distribution over the weight vector $\mathbf{w}$

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I}) \tag{6.50}$$

governed by the hyperparameter α (precision) induces then a probability distribution over functions $y(\mathbf{x})$.

Linear regression revisited (2)

Evaluating $y(\mathbf{x})$ for a set of training data points $\mathbf{x}_1, \ldots, \mathbf{x}_N$, we have a joint distribution

$$\mathbf{y} = \mathbf{\Phi}\mathbf{w}, \quad \text{with elements} \quad y_n = y(\mathbf{x}_n) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n) \propto \mathcal{N}, \quad (6.51)$$

where $\mathbf{\Phi}$ is the design matrix with elements $\Phi_{nk} = \phi_k(\mathbf{x}_n)$.
$p(\mathbf{y})$ is Gaussian, and its mean and covariance can be shown to be

$$\mathbb{E}[\mathbf{y}] = \mathbf{0} \quad (6.52)$$

$$\text{cov}[\mathbf{y}] = \frac{1}{\alpha}\mathbf{\Phi}\mathbf{\Phi}^T = \mathbf{K} \quad (6.53)$$

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

$$K_{nm} = k(\mathbf{x}_n, \mathbf{x}_m) = \frac{1}{\alpha}\boldsymbol{\phi}(\mathbf{x}_n)^T \boldsymbol{\phi}(\mathbf{x}_m). \quad (6.54)$$

Up to now we *only* took data points + prior, but no target values.

Linear regression revisited: summary

- The model we have seen is one example for a Gaussian process
- "*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* $\mathbf{x}_1, \ldots, \mathbf{x}_N$ *jointly have a Gaussian distribution*"
- Key point: the joint distribution is definied completely by second-order statistics (mean, covariance)
- Note, usually the mean is taken to be zero, then we only need the covariance, i.e., the kernel-function:

$$\mathbb{E}[y(\mathbf{x}_n)y(\mathbf{x}_m)] = k(\mathbf{x}_n, \mathbf{x}_m) \tag{6.55}$$

- Actually, instead of choosing (a limited number of) basis functions, we can directly choose a kernel function (which may result in an infinite number of basis functions)

Gaussian process regression

To use Gaussian processes for regression, we need to model noise:

$$t_n = y_n + \epsilon_n \quad \text{with} \quad y_n = y(\mathbf{x_n}) \tag{6.57}$$

For noise processes with a Gaussian distribution we obtain

$$p(t_n|y_n) = \mathcal{N}(t_n|y_n, \beta^{-1}). \tag{6.58}$$

The joint distribution for $\mathbf{t} = (t_1, \ldots, t_n)^T$ and $\mathbf{y} = (y_1, \ldots, y_n)^T$ is then (since the noise is assumed to be independent)

$$p(\mathbf{t}|\mathbf{y}) = \mathcal{N}(\mathbf{t}|\mathbf{y}, \beta^{-1}\mathbf{I}_N). \tag{6.59}$$

Gaussian process regression (2)

And from the definition of a Gaussian process we have

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K}), \tag{6.60}$$

i.e., points that are more similar (given the kernel function k) will be stronger correlated. For the marginal distribution $p(\mathbf{t})$, we need to integrate over $\mathbf{y}$ (see Section 2.3.3):

$$p(\mathbf{t}) = \int p(\mathbf{t}|\mathbf{y})p(\mathbf{y})\mathrm{d}\mathbf{y} = \mathcal{N}(\mathbf{t}|\mathbf{0}, \mathbf{C}) \quad \text{with} \quad \mathbf{C} = \mathbf{K} + \beta^{-1}\mathbf{I}. \tag{6.61}$$

Making predictions

- So far we have a model for the joint probability distribution over sets of data points
- For predictions of a new input variable x_{N+1}, we need to evaluate the predictive distribution $p(t_{N+1}|\mathbf{t})$
- By partitioning (see Section 2.3.1) the joint Gaussian distribution over $\mathbf{x}_1, \ldots, \mathbf{x}_N, \mathbf{x}_{N+1}$, we obtain $p(t_{N+1}|\mathbf{t})$ given by its mean and covariance:

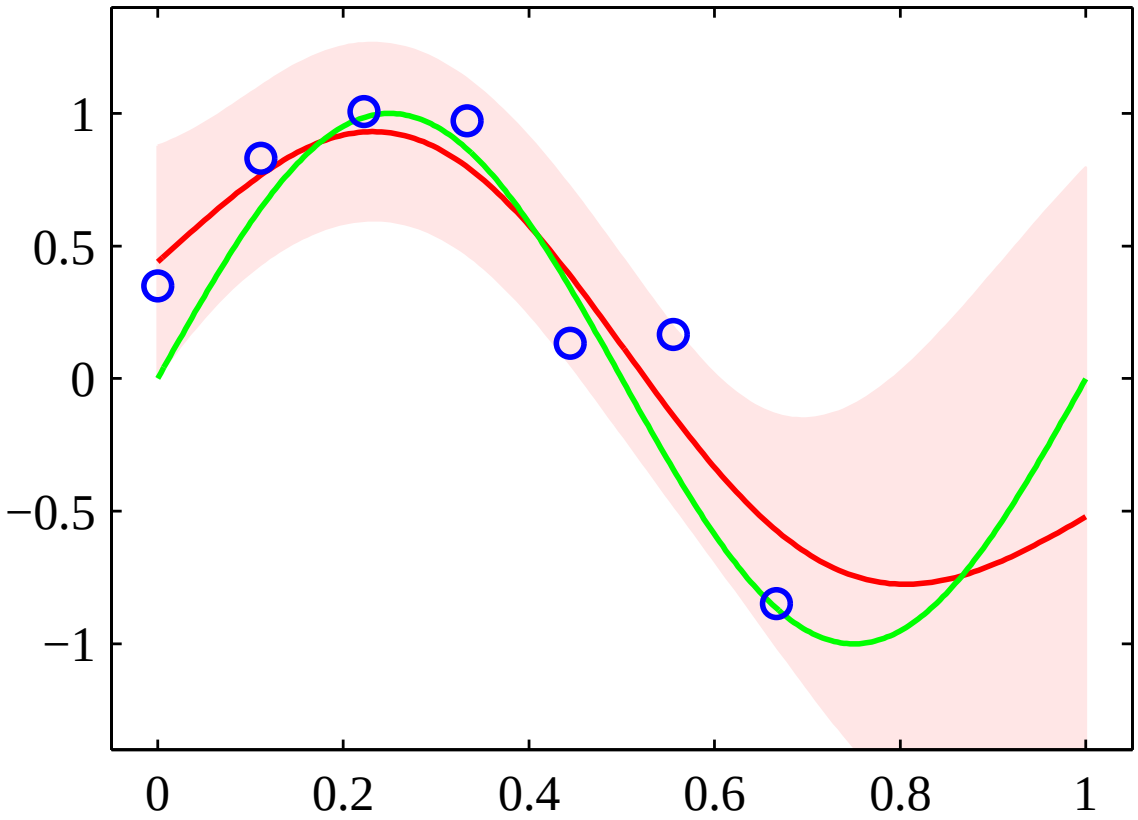
$$m(\mathbf{x}_{N+1}) = \mathbf{k}^T \mathbf{C}^{-1} \mathbf{t} \quad \text{with} \quad (6.66)$$

$$\mathbf{k} = (k(\mathbf{x}_1, \mathbf{x}_{N+1}), \ldots, k(\mathbf{x}_N, \mathbf{x}_{N+1}))^T$$

$$\sigma^2(\mathbf{x}_{N+1}) = c - \mathbf{k}^T \mathbf{C}^{-1} \mathbf{k} \quad \text{with} \quad (6.67)$$

$$c = k(\mathbf{x}_{N+1}, \mathbf{x}_{N+1}) + \beta^{-1}.$$

Making predictions (2)

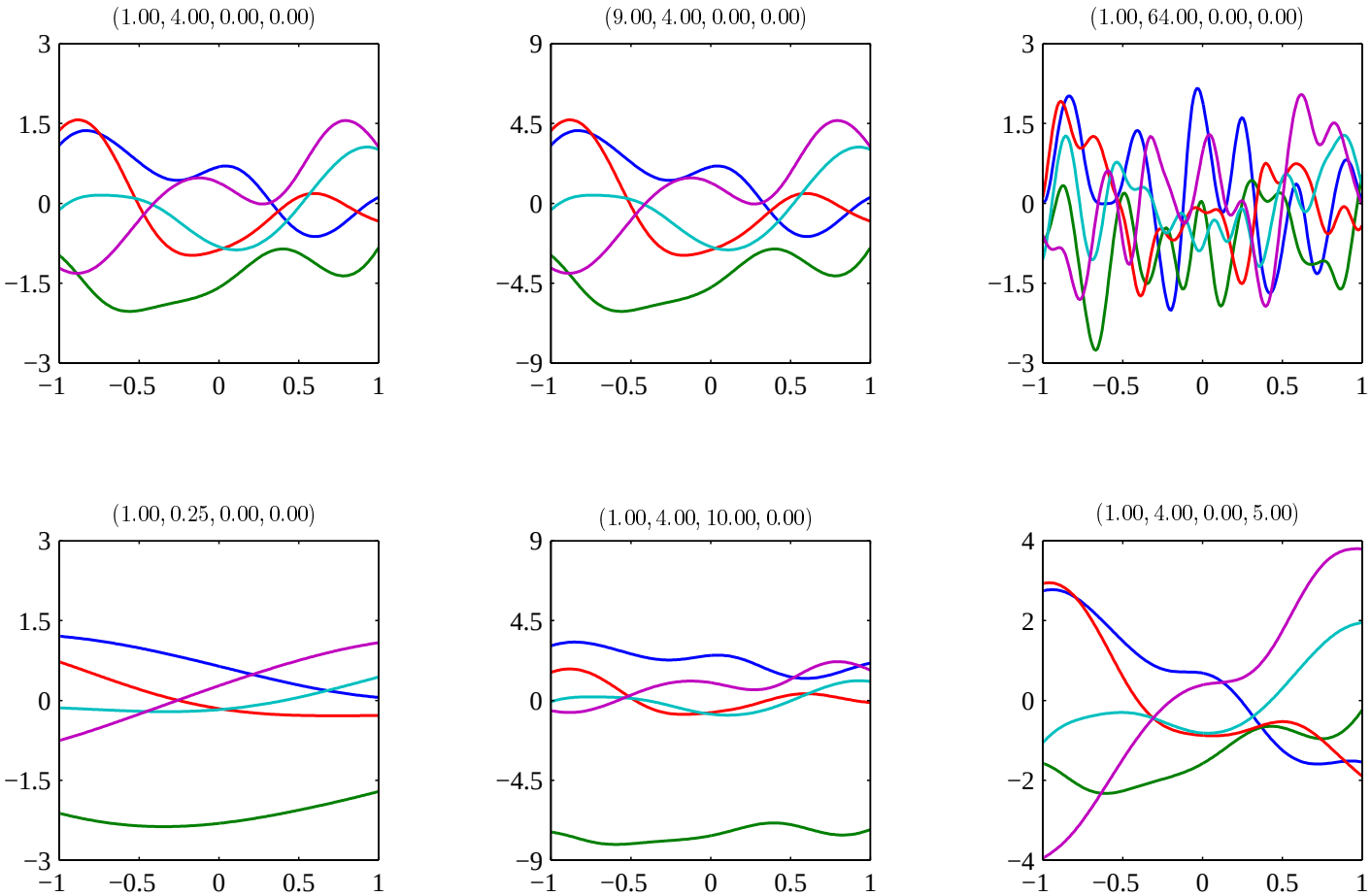


Green curve: original sinusoidal function; blue points: sampled training data points with additional noise; read line: mean estimate; shaded regions: $+/- 2\sigma$

Distribution over functions . . . hm?

Sample functions from prior $p(\mathbf{y})$ with common kernel function

$$k(\mathbf{x}_n, \mathbf{x}_m) = \theta_0 \exp\left(-\frac{\theta_1}{2} \parallel \mathbf{x}_n - \mathbf{x}_m \parallel^2\right) + \theta_2 + \theta_3 \mathbf{x}_n^T \mathbf{x}_m \quad (6.63)$$



Learning hyperparameters $\boldsymbol{\theta}$

- In practice, it can be preferable not to fix parameters, but to infer them from the data
- Parameters $\boldsymbol{\theta}$ are, e.g.: length scale of correlations, precision of noise (β)
- Simplest approach:
  - Maximizing $\boldsymbol{\theta}$ for the log-likelihood (maximum likelihood): $\ln p(\mathbf{t}|\boldsymbol{\theta})$
  - Problem: $\ln p(\mathbf{t}|\boldsymbol{\theta})$ is in general non-convex and can have multiple maxima
- Introduce a prior $p(\boldsymbol{\theta})$ and maximize the log-posterior (maximum a posteriori): $\ln p(\mathbf{t}|\boldsymbol{\theta}) + \ln p(\boldsymbol{\theta})$
- To be Bayesian, we need the actual distribution and have to marginalize; this is not tractable $\Rightarrow$ approximations
- The noise might not be additive but dependent on on $\mathbf{x}$
  - A second Gaussian process can be introduced to represent the dependency of β on $\mathbf{x}$

Automatic relevance detection (ARD)

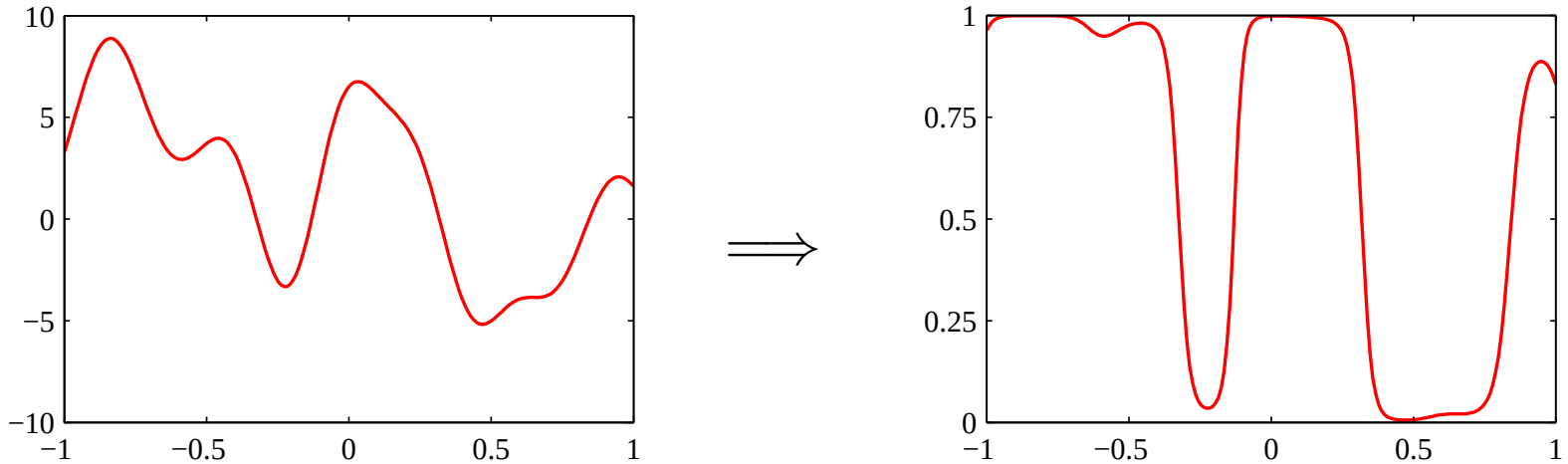
- An additional hyperparameter can be introduced for each input dimension, e.g.:

$$k(\mathbf{x}_n, \mathbf{x}_m) = \theta_0 \exp\left(-\frac{1}{2}\sum_{i=1}^{D}\eta_i(x_{ni} - x_{mi})^2\right) + \theta_2 + \theta_3 \mathbf{x}_n^T \mathbf{x}_m \tag{6.72}$$

- Hyperparameter optimization by maximum likelihood allows then a different weighting of each dimension
- Unrelevant dimensions (with small weights) can be detected and discarded

Gaussian process classification

- **Objective:** model posterior probabilities of the target variable for a new input
- **Problem:** we need to map values to interval $(0; 1)$
- **Solution:** use a Gaussian process together with a non-linear activation function (e.g., sigmoid)



Gaussian process classification (2)

Consider two-class problem with target values $t \in \{0, 1\}$. Define a Gaussian process over a function $a(\mathbf{x})$ and transform a using the logistic sigmoid to

$$y = \sigma(a(\mathbf{x})).$$

Similar to before, we need to predict the conditional distribution:

$$\begin{aligned} p(t_{N+1} = 1|\mathbf{t}) &= \int p(t_{N+1} = 1|a_{N+1}) p(a_{N+1}|\mathbf{t}) \mathrm{d}a_{N+1} \\ &= \int \sigma(a_{N+1}) p(a_{N+1}|\mathbf{t}) \mathrm{d}a_{N+1}. \end{aligned} \tag{6.76}$$

However, the integral is analytically intractable. Approximations can be of numerical or analytical nature.

Gaussian process classification (3)

- Problem 1: we need to compute a weird integral
- Solution 1: we know how to compute the convolution of an Gaussian and a sigmoid function (Eq. (4.153)) $\Rightarrow$ approximate the posterior distribution $p(a_{N+1}|\mathbf{t})$ as Gaussian
- Problem 2: Hm . . . but how do we approximate the posterior?
- Solution 2: the Laplace approximation (among others)

Laplace approximation

We can rewrite the posterior over a_{N+1} using Bayes's theorem:

$$p(a_{N+1}|\mathbf{t}) = \int p(a_{N+1}, \mathbf{a}|\mathbf{t})\mathrm{d}\mathbf{a}$$
$$= \ldots = \int p(a_{N+1}|\mathbf{a})p(\mathbf{a}|\mathbf{t})\mathrm{d}\mathbf{a}. \qquad (6.77)$$

We know how to compute mean and covariance for $p(a_{N+1}|\mathbf{a})$. Now we have to find a Gaussian approximation only for $p(\mathbf{a}|\mathbf{t})$. This is done noting that $p(\mathbf{a}|\mathbf{t}) \propto p(\mathbf{t}|\mathbf{a})p(\mathbf{a})$, thus

$$\boldsymbol{\Psi}(\mathbf{a}) = \ln p(\mathbf{a}|\mathbf{t}) = \ln p(\mathbf{t}|\mathbf{a}) + \ln p(\mathbf{a}) + \text{const.} \qquad (6.80)$$

Laplace approximation (2)

- We can use the iterative reweighted least squares (IRLS) algorithm (Sec. 4.3.3) to find the mode of $\mathbf{\Psi}(\mathbf{a})$ (first and second derivative have to be evaluated)
- It can be shown that $\mathbf{\Psi}(\mathbf{a})$ is convex and thus has only one mode :-)
- The mode position $\mathbf{a}^\star$ and the Hessian matrix $\mathbf{H}$ at this position define our Gaussian approximation

$$q(\mathbf{a}) = \mathcal{N}(\mathbf{a}|\mathbf{a}^\star, \mathbf{H}^{-1}) \tag{6.86}$$

- Now we can go back to the formulas and compute the integrals and finally also $p(t_{N+1}|\mathbf{t})$ from Eq. (6.76)

Learning hyperparameters

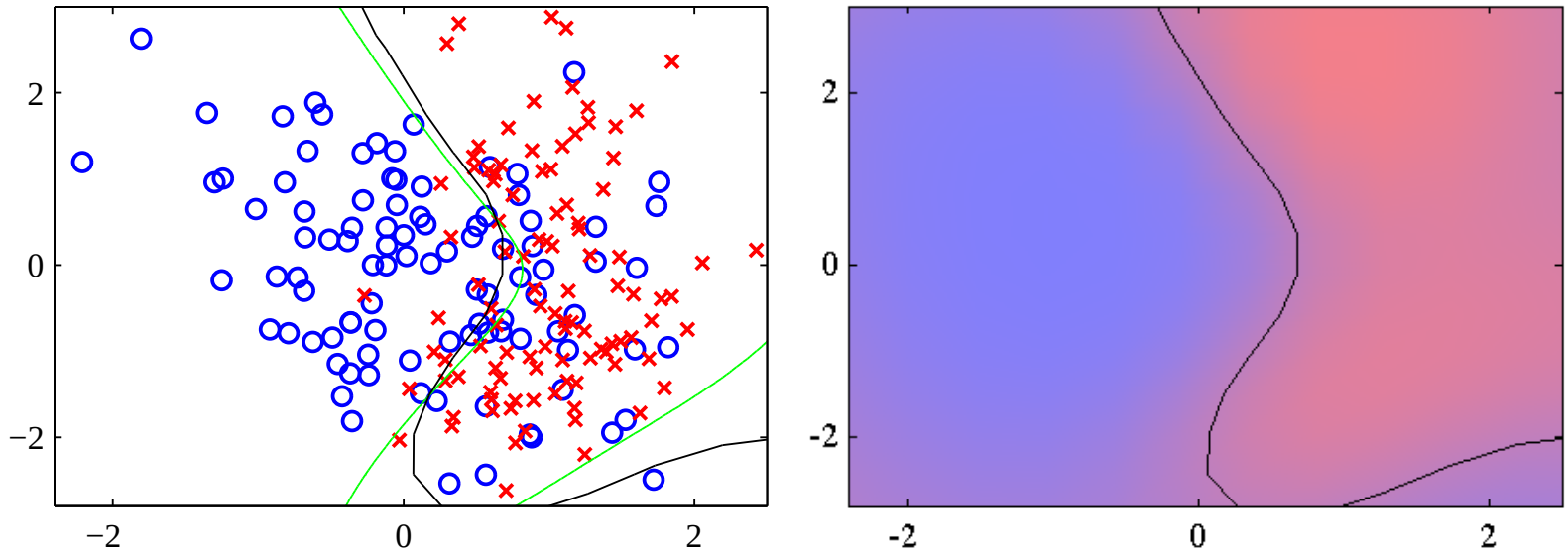
- To determine the parameters $\boldsymbol{\theta}$, we can maximize the likelihood function:

$$p(\mathbf{t}|\boldsymbol{\theta}) = \int p(\mathbf{t}|\mathbf{a})p(\mathbf{a}|\boldsymbol{\theta})\mathrm{d}\mathbf{a} \tag{6.89}$$

- Again, the integral is analytically intractable, so the Laplace approximation can be applied again
- We need an expression for the gradient of the logarithm of $p(\mathbf{t}|\boldsymbol{\theta})$
- Hm . . . this is a bit trickier, but nevertheless doable
- . . . by now we might be happy to skip the exact details :-)

Gaussian process classification (finishing up)

Just to finish up, and now we have a binary classifier based on a Gaussian process



Connection to neural networks

- Neural Networks
  - The range of representable functions is governed by the number M of hidden units
  - Within the maximum likelihood framework, they overfit as M comes close to the number of training samples
- Bayesian Neural Networks
  - The prior over $\mathbf{w}$ in conjunction with the network function $f(\mathbf{x}, \mathbf{w})$ produces a prior distribution over functions from $y(\mathbf{x})$
  - The distribution of functions will tend to a Gaussian process in the limit $M \to \infty$
  - The property that the outputs share hidden units (and thus 'borrow statistical strength' from each other) is lost in this limit
  - . . . and some other details