7. Gaussian Processes - Regression

Motivation

- So far we have handled learning techniques that use a parametric representation of a function
- However: what if the parametric model is incorrect or hard to find?
- Gaussian Processes model functions implicitly without parameters (“data-driven”)
- **Idea:** represent the data using a large multi-variate Normal distribution
- Only requirement: all sub-sets of data points are assumed to be jointly Normal distributed

Before We Start…

- For the following derivation we first need some math:
- Question: What is the inverse of a block matrix such as this?

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

Before We Start…

- For the following derivation we first need some math:
- Question: What is the inverse of a block matrix such as this?

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}^{-1} = \begin{pmatrix} (M/D)^{-1} & -(M/D)^{-1}BD^{-1} \\ -D^{-1}C(M/D)^{-1} & D^{-1} + D^{-1}C(M/D)^{-1}BD^{-1} \end{pmatrix}$$

- We have proven the matrix inversion lemma:

$$(A + BCD)^{-1} = A^{-1} - A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1}$$

Kernelized Regression (Rep.)

We can use that to compute:

$$\mathbf{w} = (\lambda I_D + \Phi^T \Phi)^{-1} \Phi^T \mathbf{t} = \Phi^T (\lambda I_N + \Phi \Phi^T)^{-1} \mathbf{t}$$

by defining: $K = \Phi \Phi^T \qquad \mathbf{a} = (\lambda I_N + K)^{-1} \mathbf{t}$

we get:

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

$$= \mathbf{k}(\mathbf{x})^T (K + \lambda I_N)^{-1} \mathbf{t}$$

This means that the predicted output is a **linear combination** of the training outputs, where the coefficients depend on the similarities to the training input.

Gaussian Marginals and Conditionals

We can also use the result for the following:

Assume we have two variables $\mathbf{x}_a$ and $\mathbf{x}_b$ that are jointly Gaussian distributed, i.e. $\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \Sigma)$

with

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{pmatrix} \quad \boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{pmatrix} \quad \Sigma = \begin{pmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{pmatrix}$$

Question: how can we compute the conditional?

$$p(\mathbf{x}_a \mid \mathbf{x}_b)$$

Gaussian Marginals and Conditionals

We can also use the result for the following:

Assume we have two variables $\mathbf{x}_a$ and $\mathbf{x}_b$ that are jointly Gaussian distributed, i.e. $\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \Sigma)$

with

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{pmatrix} \quad \boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{pmatrix} \quad \Sigma = \begin{pmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{pmatrix}$$

The conditional distribution is $p(\mathbf{x}_a \mid \mathbf{x}_b) = \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_{a|b}, \Sigma_{a|b})$

where $\boldsymbol{\mu}_{a|b} = \boldsymbol{\mu}_a + \Sigma_{ab}\Sigma_{bb}^{-1}(\mathbf{x}_b - \boldsymbol{\mu}_b)$

and $\Sigma_{a|b} = \Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}$ **“Schur Complement”**

The marginal is $p(\mathbf{x}_a) = \mathcal{N}(\mathbf{x}_a \mid \boldsymbol{\mu}_a, \Sigma_{aa})$

Definition

Definition: A **Gaussian process** is a collection of random variables, any finite number of which have a joint Gaussian distribution.

The number of random variables can be **infinite**!

This means: a GP is a Gaussian distribution over **functions**!

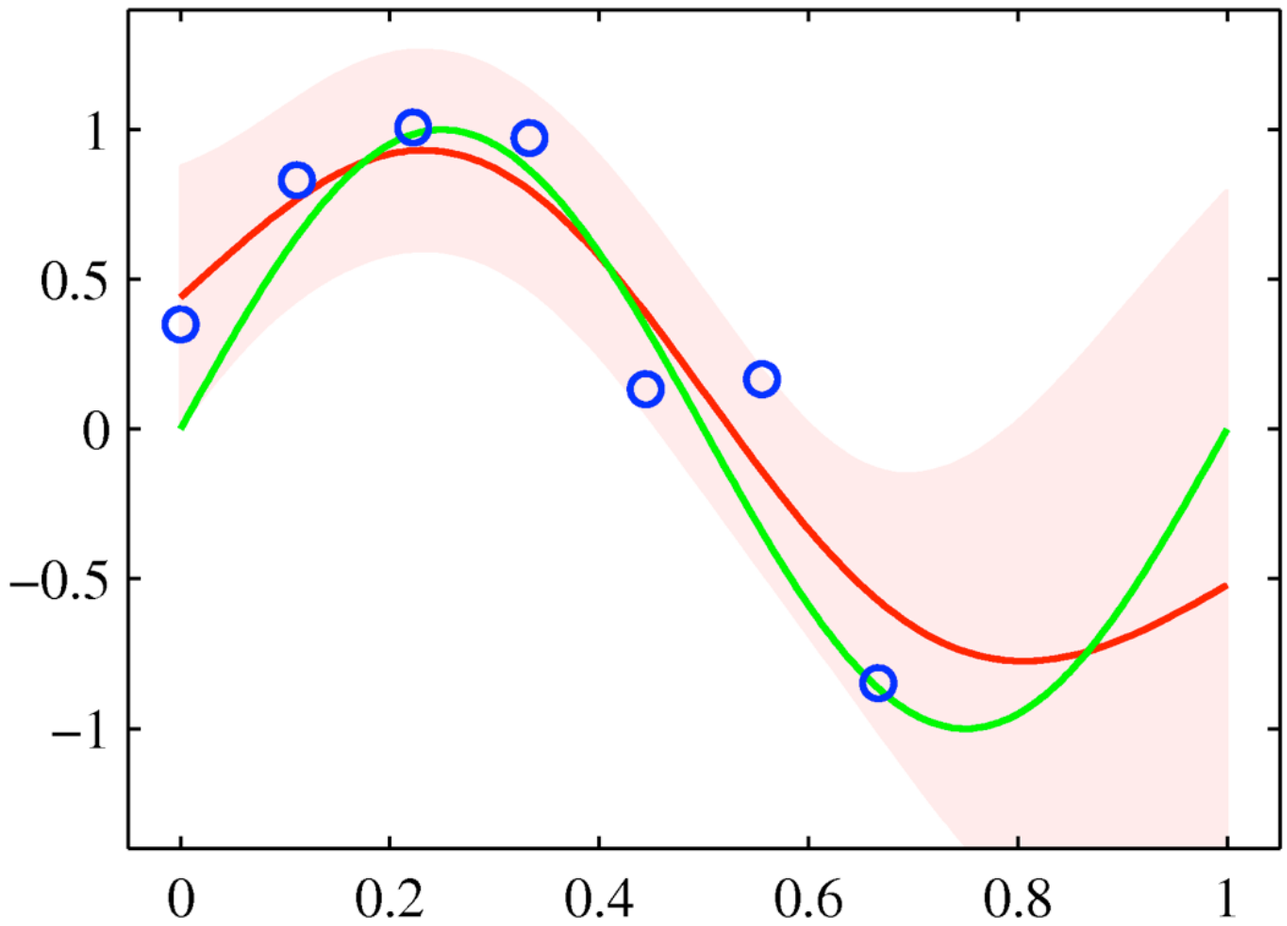
To specify a GP we need:

mean function: $m(\mathbf{x}) = \mathbb{E}[y(\mathbf{x})]$

covariance function:

$$k(\mathbf{x}_1, \mathbf{x}_2) = \mathbb{E}[y(\mathbf{x}_1) - m(\mathbf{x}_1)y(\mathbf{x}_2) - m(\mathbf{x}_2)]$$

Example



- green line: sinusoidal data source
- blue circles: data points with Gaussian noise
- red line: mean function of the Gaussian process
- shaded red area: 2σ confidence interval

How Can We Handle Infinity?

Idea: split the (infinite) number of random variables into a finite and an infinite subset.

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_f \\ \mathbf{x}_i \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu}_f \\ \boldsymbol{\mu}_i \end{pmatrix}, \begin{pmatrix} \Sigma_f & \Sigma_{fi} \\ \Sigma_{fi}^T & \Sigma_i \end{pmatrix}\right)$$

finite part

infinite part

From the marginalization property we get:

$$p(\mathbf{x}_f) = \int p(\mathbf{x}_f, \mathbf{x}_i) d\mathbf{x}_i = \mathcal{N}(\mathbf{x}_f \mid \boldsymbol{\mu}_f, \Sigma_f)$$

This means we can use finite vectors.

A Simple Example

In Bayesian linear regression, we had $y(\mathbf{x}) = \phi(\mathbf{x})^T \mathbf{w}$ with prior probability $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \Sigma_p)$. This means:

$$\mathbb{E}[y(\mathbf{x})] = \phi(\mathbf{x})^T \mathbb{E}[\mathbf{w}] = \mathbf{0}$$

$$\mathbb{E}[y(\mathbf{x}_1) y(\mathbf{x}_2))] = \phi(\mathbf{x}_1)^T \mathbb{E}[\mathbf{w}\mathbf{w}^T] \phi(\mathbf{x}_2) = \phi(\mathbf{x}_1)^T \Sigma_p \phi(\mathbf{x}_2)$$

Any number of function values $y(\mathbf{x}_1), \ldots, y(\mathbf{x}_N)$ is jointly Gaussian with zero mean.

The covariance function of this process is

$$k(\mathbf{x}_1, \mathbf{x}_2) = \phi(\mathbf{x}_1)^T \Sigma_p \phi(\mathbf{x}_2)$$

In general, any valid kernel function can be used.

The Covariance Function

The most used covariance function (kernel) is:

$$k(\mathbf{x}_p, \mathbf{x}_q) = \sigma_f^2 \exp(-\frac{1}{2l^2}(\mathbf{x}_p - \mathbf{x}_q)^2) + \sigma_n^2 \delta_{pq}$$

signal variance

length scale

noise variance

It is known as “squared exponential”, “radial basis function” or “Gaussian kernel”.

Other possibilities exist, e.g. the exponential kernel:

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

This is used in the “Ornstein-Uhlenbeck” process.

Sampling from a GP

Just as we can sample from a Gaussian distribution, we can also generate samples from a GP. **Every sample will then be a function!**

Process:

1. Choose a number of input points $\mathbf{x}_1^*, \ldots, \mathbf{x}_M^*$
2. Compute the covariance matrix K where

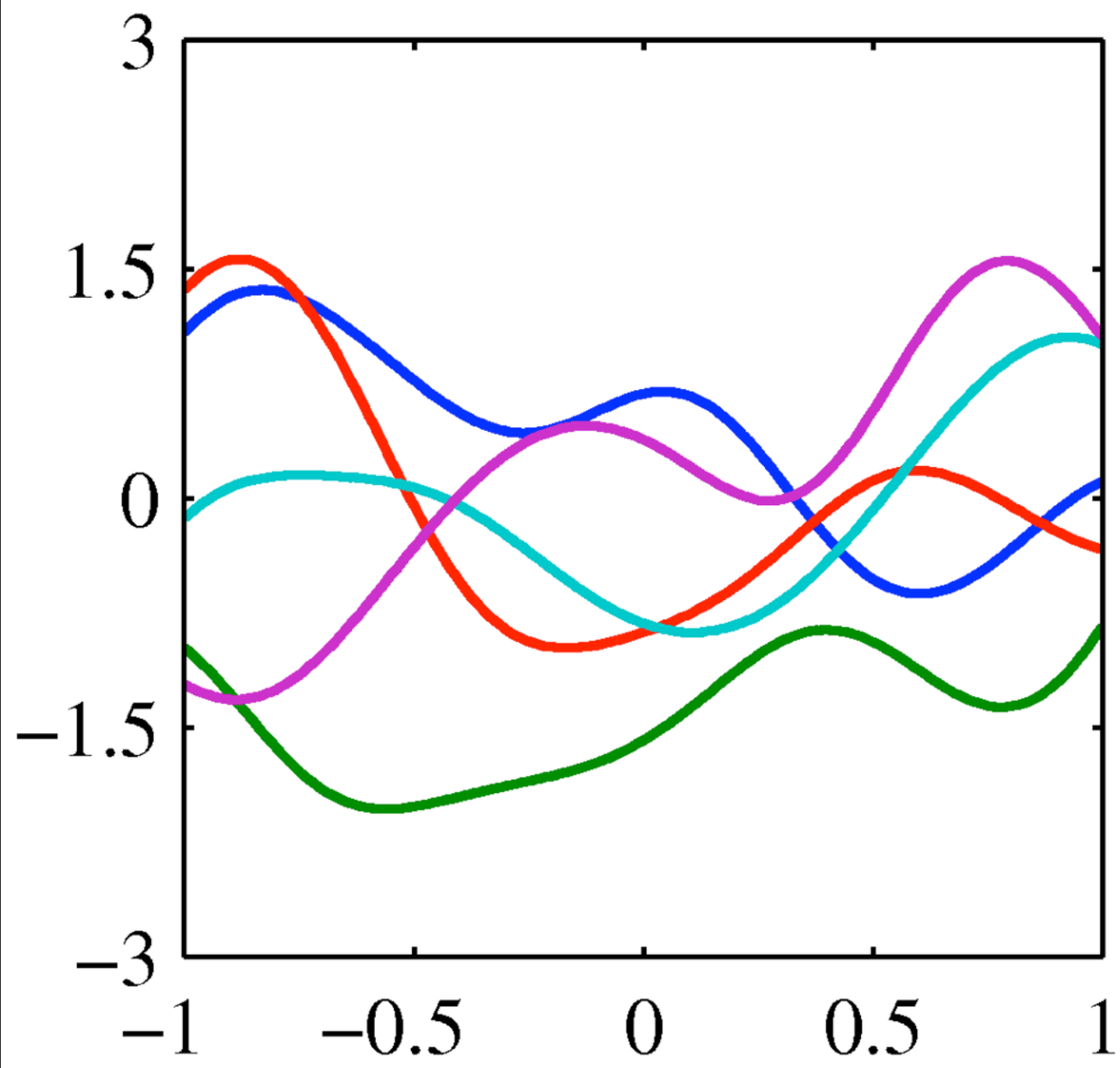
$$K_{ij} = k(\mathbf{x}_i^*, \mathbf{x}_j^*)$$

3. Generate a random Gaussian vector from

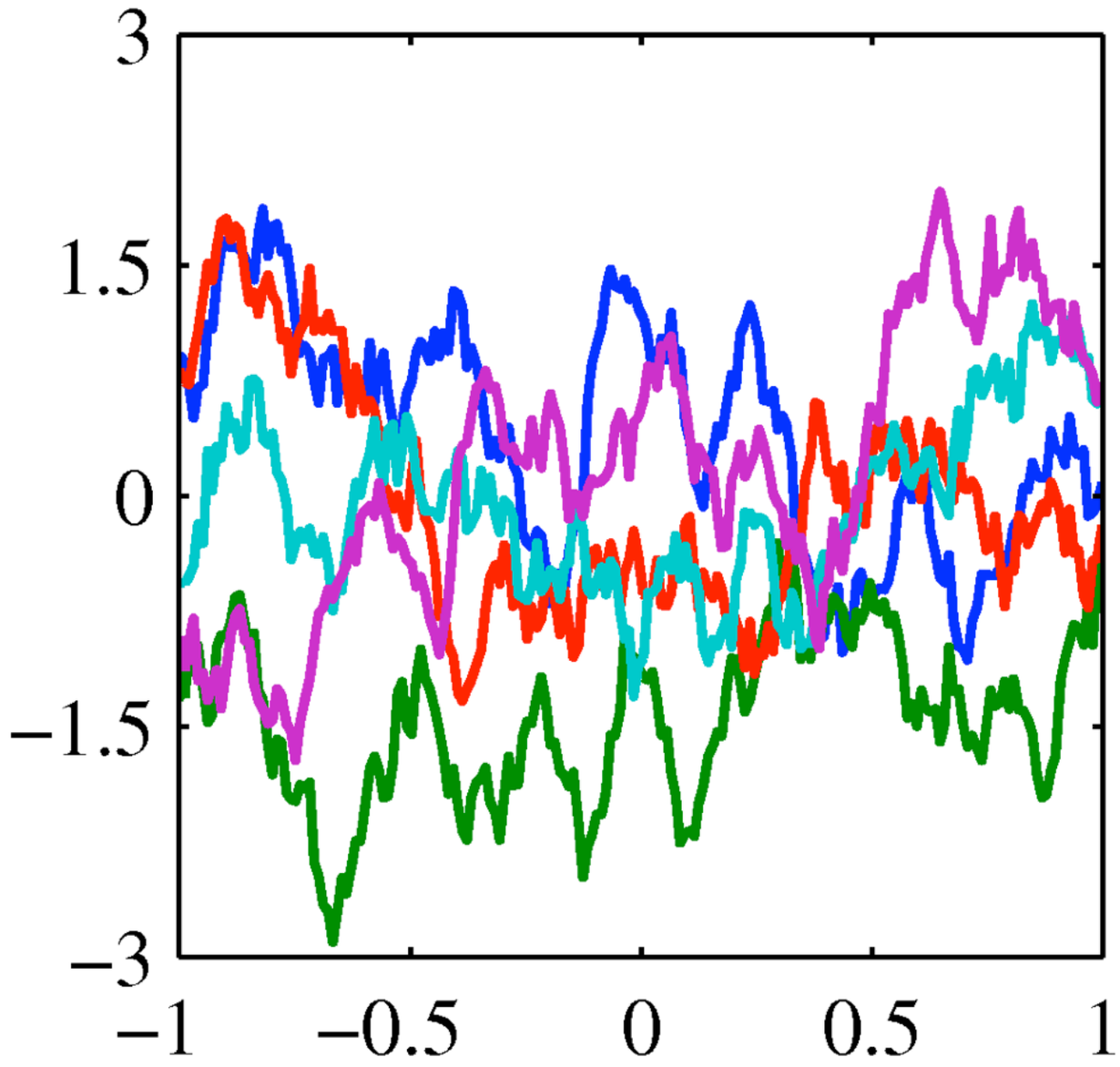
$$\mathbf{y}_* \sim \mathcal{N}(\mathbf{0}, K)$$

4. Plot the values $\mathbf{x}_1^*, \ldots, \mathbf{x}_M^*$ versus $y_1^*, \ldots, y_M^*$

Sampling from a GP



Squared exponential kernel



Exponential kernel

Prediction with a Gaussian Process

Most often we are more interested in predicting new function values for given input data.

We have:

training data $\mathbf{x}_1, \ldots, \mathbf{x}_N \quad t_1, \ldots, t_N$

test input $\mathbf{x}_1^*, \ldots, \mathbf{x}_M^*$

And we want test outputs $y_1^*, \ldots, y_M^*$

The joint probability is

$$\begin{pmatrix} \mathbf{y} \\ \mathbf{y}_* \end{pmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{pmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{pmatrix}\right)$$

and we need to compute $p(\mathbf{y}^* \mid \mathbf{x}^*, X, \mathbf{y})$.

Prediction with a Gaussian Process

In the case of only one test point $\mathbf{x}^*$ we have

$$K(X, \mathbf{x}^*) = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_*) \\ \vdots \\ k(\mathbf{x}_N, \mathbf{x}_*) \end{pmatrix} = \mathbf{k}_*$$

Now we compute the conditional distribution

$$p(y^* \mid \mathbf{x}^*, X, \mathbf{y}) = \mathcal{N}(y_* \mid \mu_*, \Sigma_*)$$

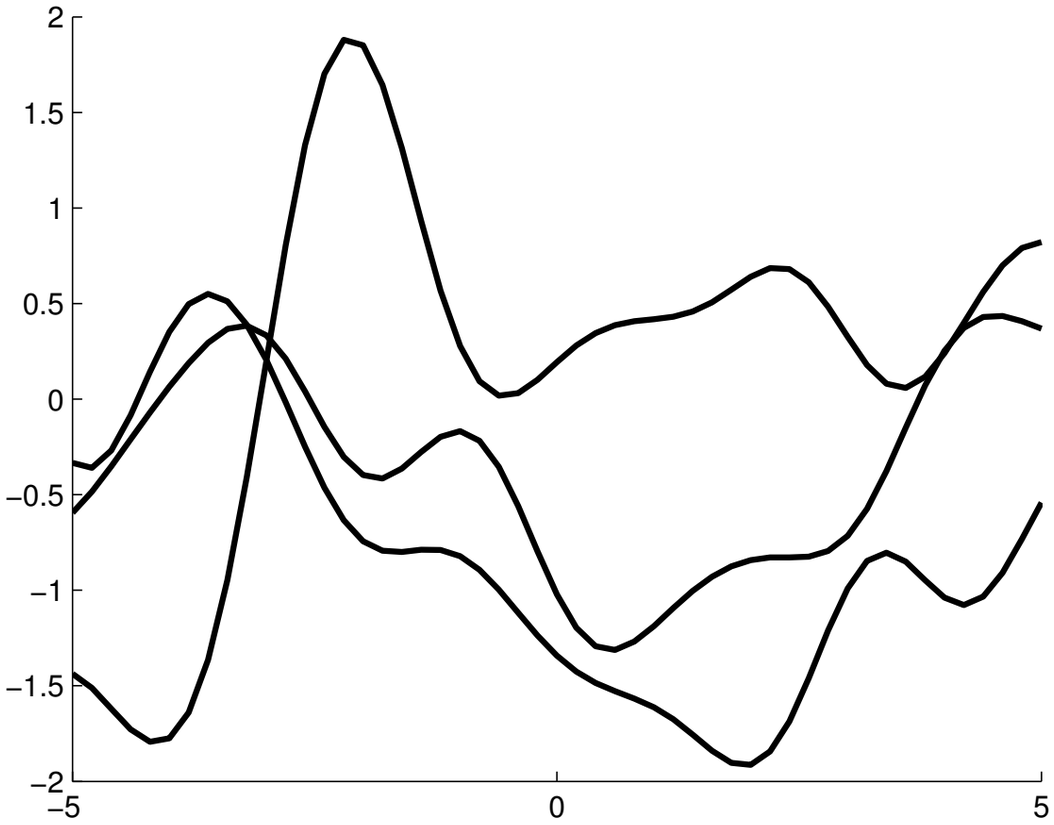
where

$$\mu_* = \mathbf{k}_*^T K^{-1} \mathbf{y}$$

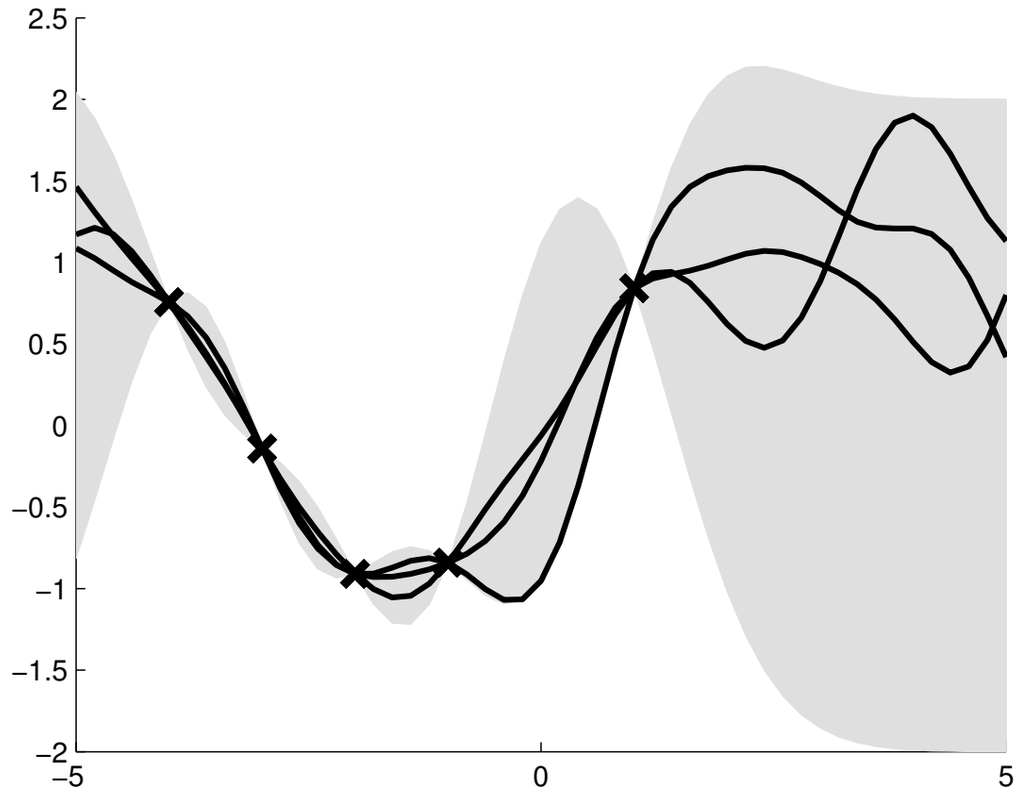
$$\Sigma_* = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^T K^{-1} \mathbf{k}_*$$

This defines the **predictive distribution.**

Example



Functions sampled from
a Gaussian Process prior



Functions sampled from the
predictive distribution

The predictive distribution is itself a Gaussian process.

It represents the posterior after observing the data.

The covariance is low in the vicinity of data points.

Prediction With Noisy Observations

- Assume we have observations with noise
- We can model this with a **latent function** f
- Thus: y is a noisy variant of f
- The formulation only differs slightly:

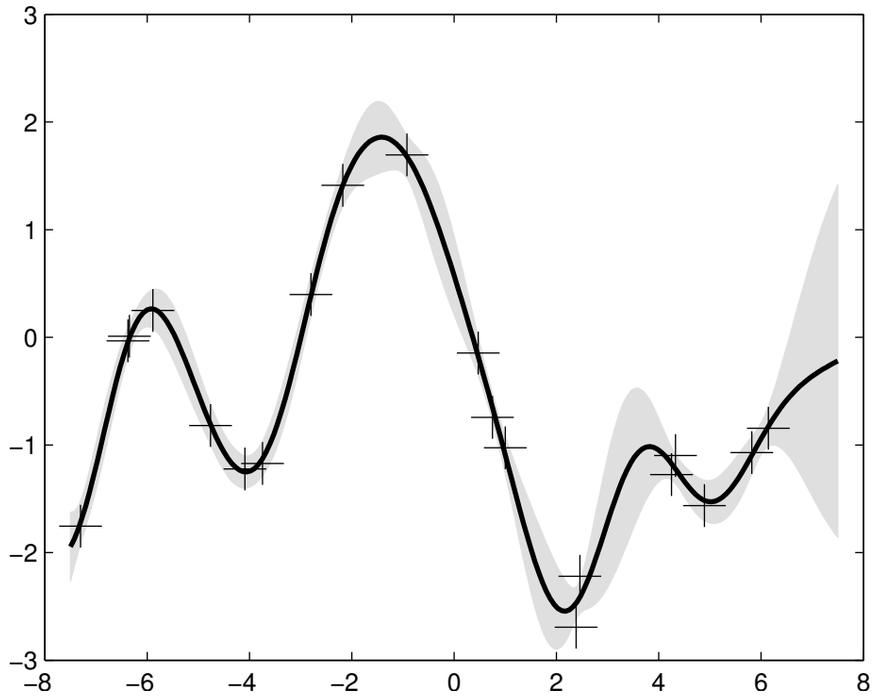
$$p(f^* \mid \mathbf{x}^*, X, \mathbf{y}) = \mathcal{N}(f_* \mid \mu_*, \Sigma_*)$$

noise-free function value — test data — training data — noisy training output

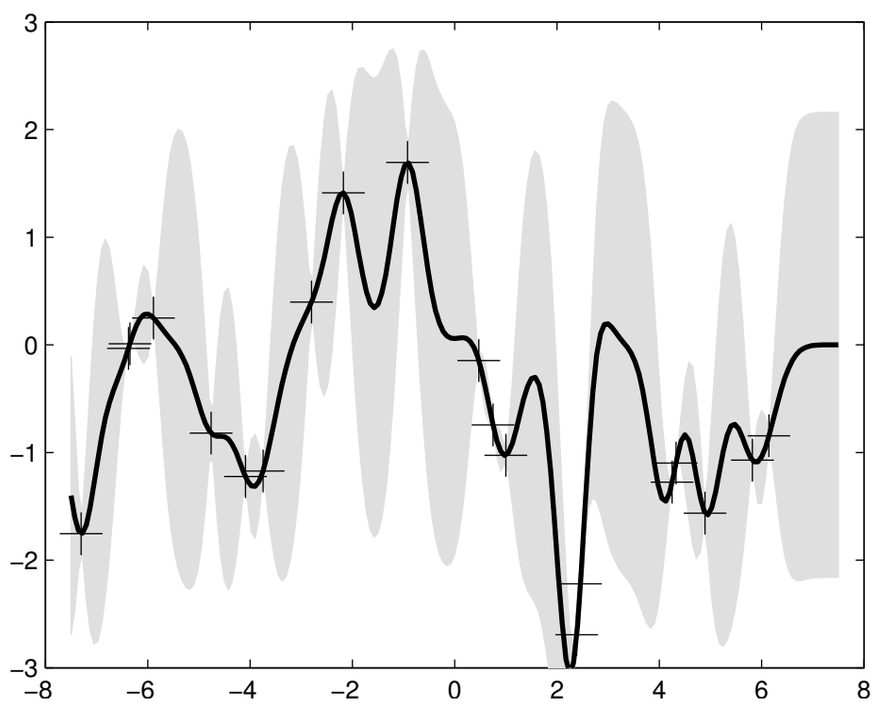
- We assume that the noise is Gaussian:

$$p(\mathbf{y} \mid \mathbf{f}) = \mathcal{N}(\mathbf{f}, \sigma_y^2 I)$$

Varying the Hyperparameters



$$l = \sigma_f = 1, \quad \sigma_n = 0.1$$

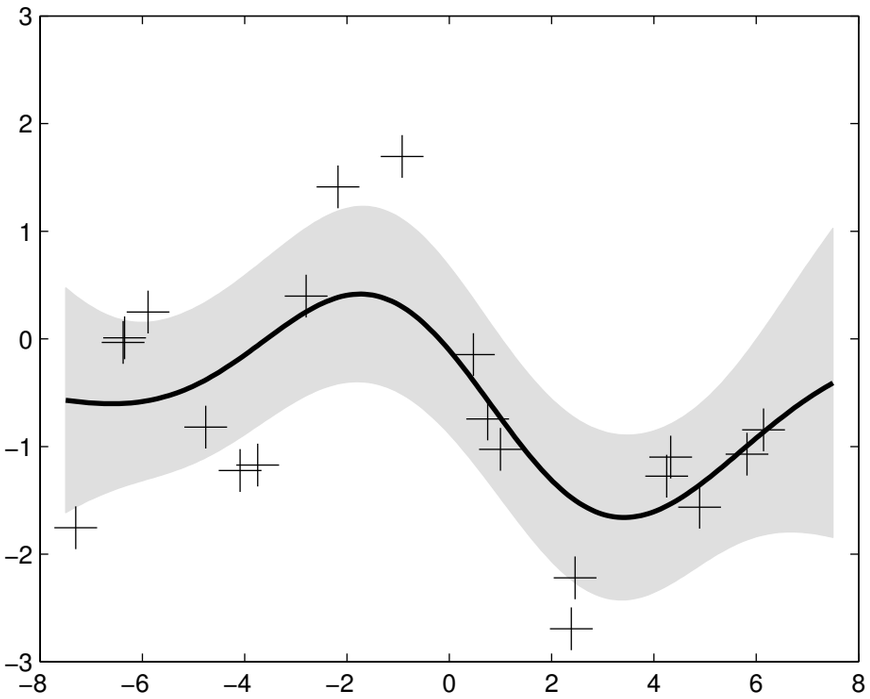


$$l = 0.3,$$

$$\sigma_f = 1.08,$$

$$\sigma_n = 0.0005$$

- 20 data samples
- GP prediction with different kernel hyper parameters



$$l = 3$$

$$\sigma_f = 1.16$$

$$\sigma_n = 0.89$$

Varying the Hyperparameters

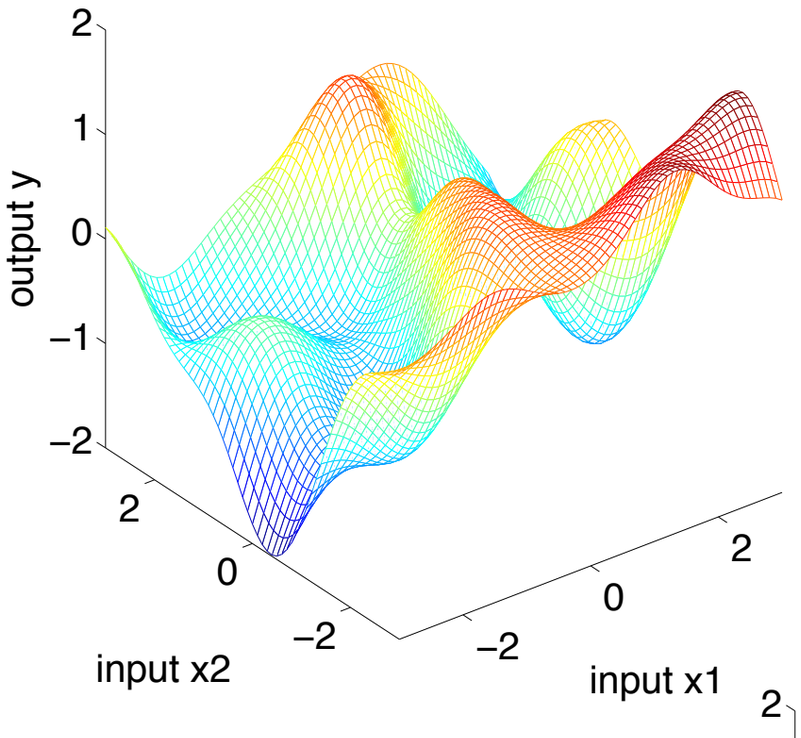
The squared exponential covariance function can be generalized to

$$k(\mathbf{x}_p, \mathbf{x}_q) = \sigma_f^2 \exp(-\frac{1}{2}(\mathbf{x}_p - \mathbf{x}_q)^T M(\mathbf{x}_p - \mathbf{x}_q)) + \sigma_n^2 \delta_{pq}$$

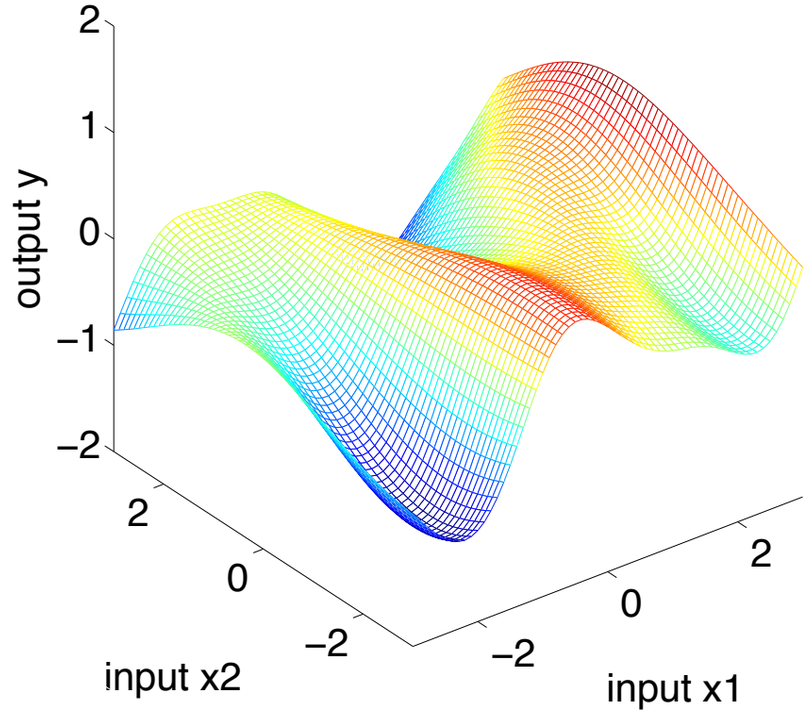
where M can be:

- $M = l^{-2} I$: this is equal to the above case
- $M = \mathrm{diag}(l_1, \ldots, l_D)^{-2}$: every feature dimension has its own length scale parameter
- $M = \Lambda\Lambda^T + \mathrm{diag}(l_1, \ldots, l_D)^{-2}$: here Λ has less than D columns

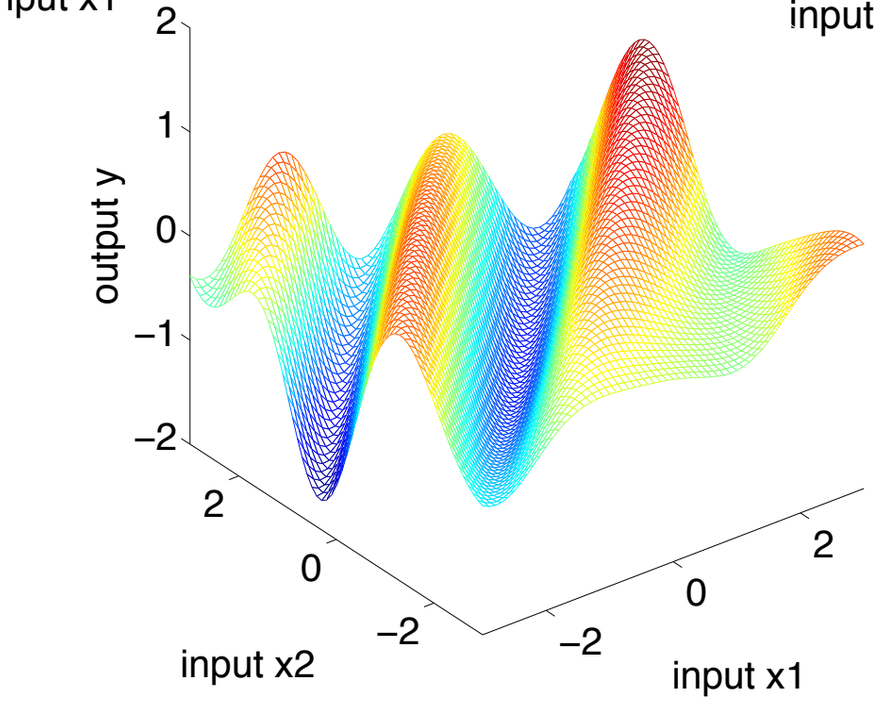
Varying the Hyperparameters



$$M = I$$



$$M = \mathrm{diag}(1, 3)^{-2}$$



$$M = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} + \mathrm{diag}(6, 6)^{-2}$$

Implementation

Algorithm 1: GP regression

Data: training data $(X, \mathbf{y})$, test data $\mathbf{x}_*$

Input: Hyper parameters σ_f^2, l, σ_n^2

$K_{ij} \leftarrow k(\mathbf{x}_i, \mathbf{x}_j)$

$L \leftarrow \texttt{cholesky}(K + \sigma_y^2 I)$

$\boldsymbol{\alpha} \leftarrow L^T \backslash (L \backslash \mathbf{y})$

Training Phase

$\mathbb{E}[f_*] \leftarrow \mathbf{k}_*^T \boldsymbol{\alpha}$

$\mathbf{v} \leftarrow L \backslash \mathbf{k}_*$

$\texttt{var}[f_*] \leftarrow k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^T \mathbf{v}$

Test Phase

$\log p(\mathbf{y} \mid X) \leftarrow -\frac{1}{2}\mathbf{y}^T \boldsymbol{\alpha} - \sum_i \log L_{ii} - \frac{N}{2} \log(2\pi)$

- Cholesky decomposition is numerically stable
- Can be used to compute inverse efficiently

Estimating the Hyperparameters

To find optimal hyper parameters we need the **marginal likelihood:**

$$p(\mathbf{y} \mid X) = \int p(\mathbf{y} \mid \mathbf{f}, X) p(\mathbf{f} \mid X) d\mathbf{f}$$

$$p(\mathbf{y} \mid \mathbf{f}) = \prod_i \mathcal{N}(y_i \mid f_i, \sigma_y^2) \qquad p(\mathbf{f} \mid X) = \mathcal{N}(\mathbf{f} \mid 0, K)$$

Estimating the Hyperparameters

To find optimal hyper parameters we need the **marginal likelihood:**

$$p(\mathbf{y} \mid X) = \int p(\mathbf{y} \mid \mathbf{f}, X) p(\mathbf{f} \mid X) d\mathbf{f}$$

$$p(\mathbf{y} \mid \mathbf{f}) = \prod_i \mathcal{N}(y_i \mid f_i, \sigma_y^2) \qquad p(\mathbf{f} \mid X) = \mathcal{N}(\mathbf{f} \mid 0, K)$$

This results in

$$p(\mathbf{y} \mid X) = \mathcal{N}(\mathbf{y} \mid 0, K + \sigma_y^2 I)$$

Estimating the Hyperparameters

To find optimal hyper parameters we need the **marginal likelihood:**

$$p(\mathbf{y} \mid X) = \int p(\mathbf{y} \mid \mathbf{f}, X) p(\mathbf{f} \mid X) d\mathbf{f}$$

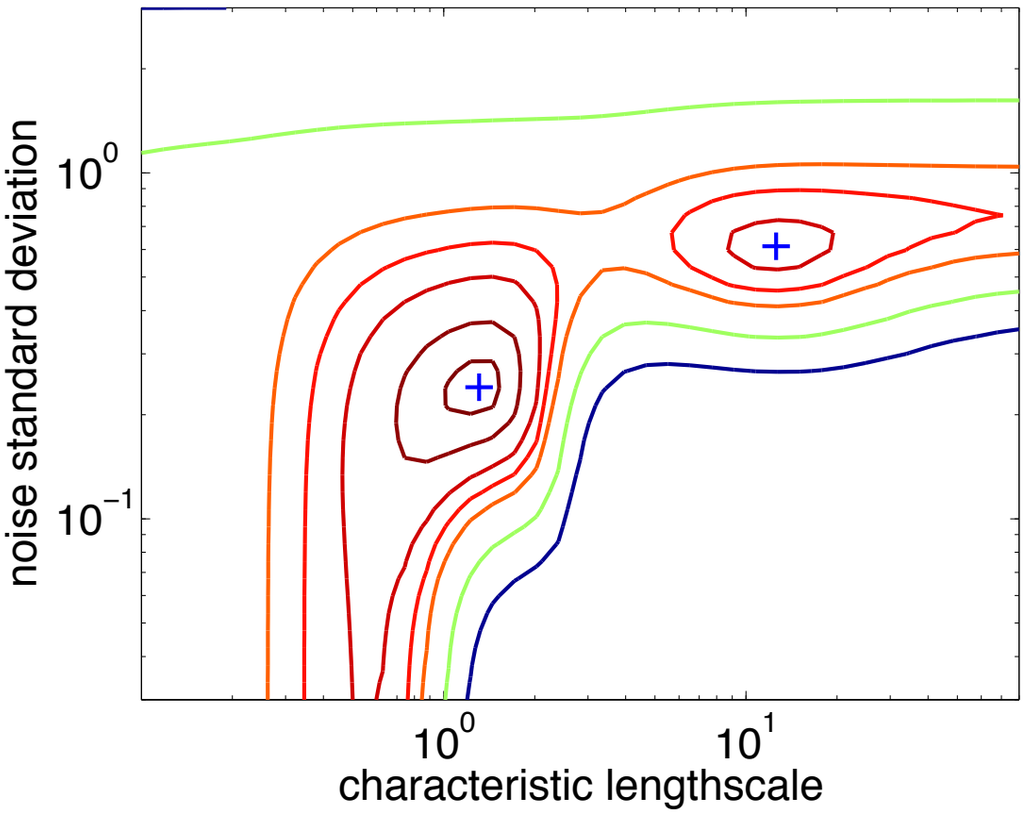
$$p(\mathbf{y} \mid \mathbf{f}) = \prod_i \mathcal{N}(y_i \mid f_i, \sigma_y^2) \qquad p(\mathbf{f} \mid X) = \mathcal{N}(\mathbf{f} \mid 0, K)$$

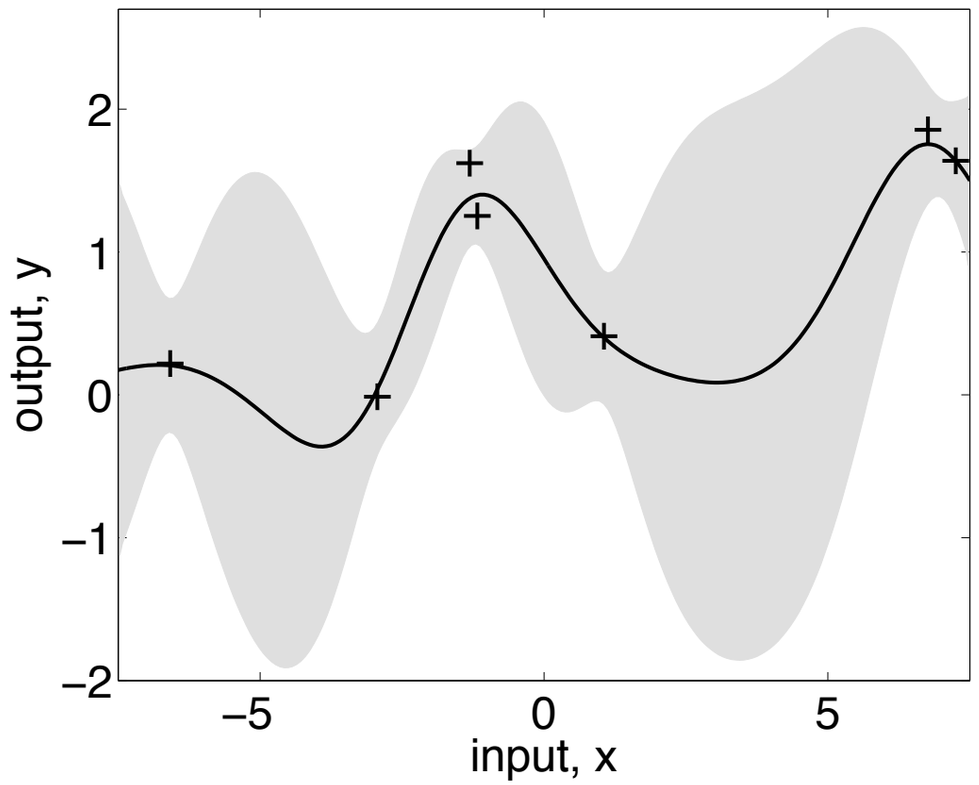
This results in

$$p(\mathbf{y} \mid X) = \mathcal{N}(\mathbf{y} \mid 0, K + \sigma_y^2 I)$$

We take the logarithm, compute the derivative and set it to 0. This is the **training** step.

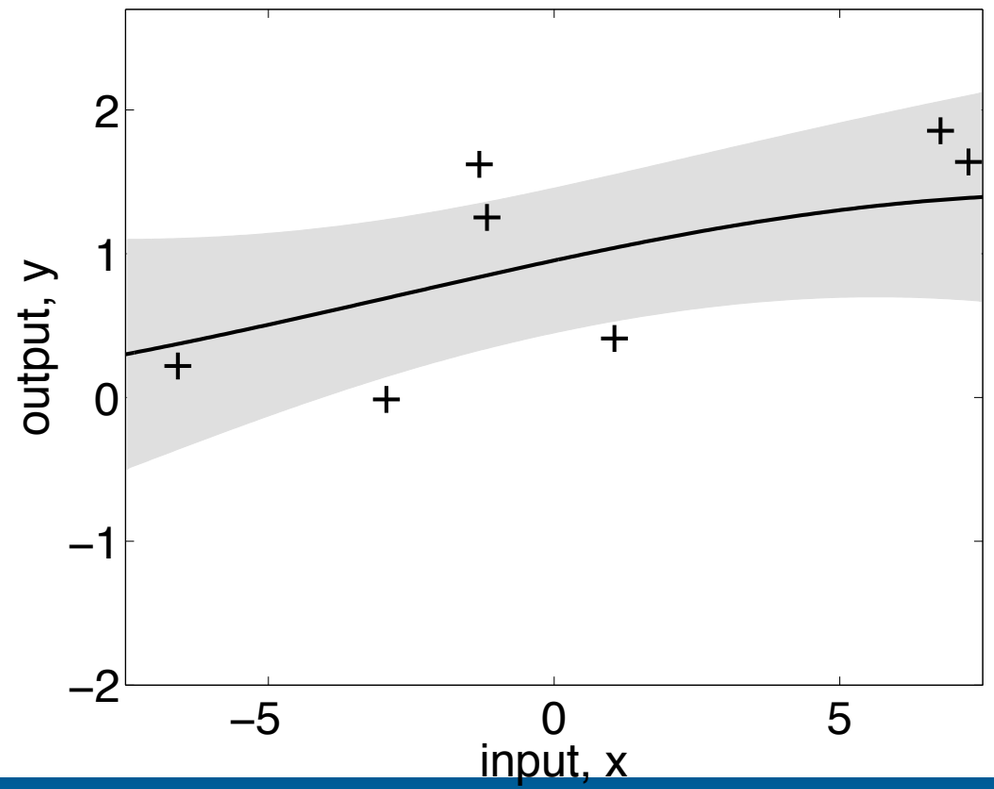
Estimating the Hyperparameters





The log marginal likelihood is not necessarily concave, i.e. it can have local maxima.

The local maxima can correspond to sub-optimal solutions.



Automatic Relevance Determination

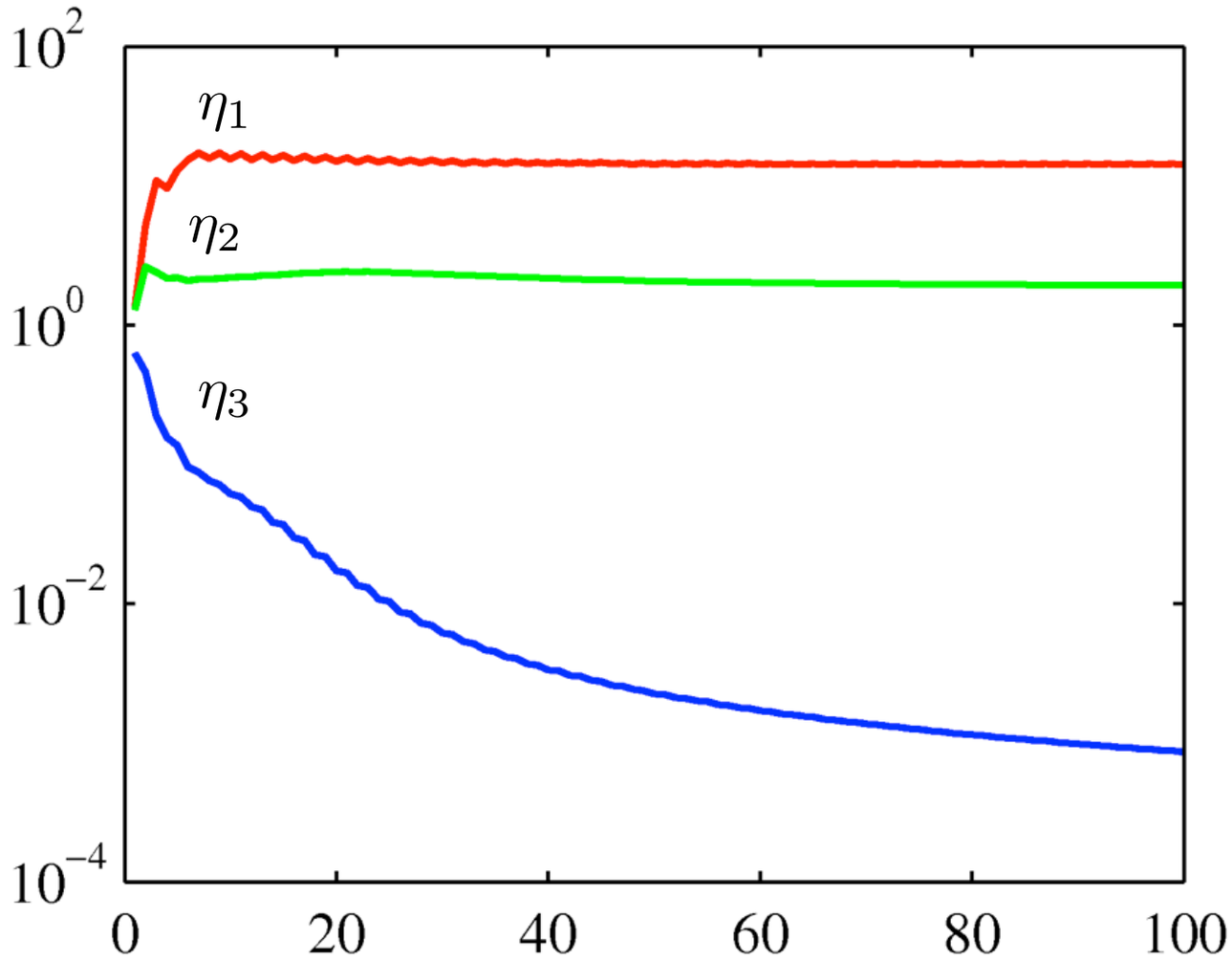
- We have seen how the covariance function can be generalized using a matrix M
- If M is diagonal this results in the kernel function

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

- We can interpret the η_i as weights for each feature dimension
- Thus, if the length scale $l_i = 1/\eta_i$ of an input dimension is large, the input is less relevant
- During training this is done automatically

Automatic Relevance Determination

3-dimensional data, parameters η_1 η_2 η_3 as they evolve during training



During the optimization process to learn the hyper-parameters, the reciprocal length scale for one parameter decreases, i.e.:

This hyper parameter is not very relevant!

Gaussian Processes - Classification

Gaussian Processes For Classification

In regression we have $y \in \mathbb{R}$, in binary classification we have $y \in \{-1; 1\}$

To use a GP for classification, we can apply a **sigmoid** function to the posterior obtained from the GP and compute the class probability as:

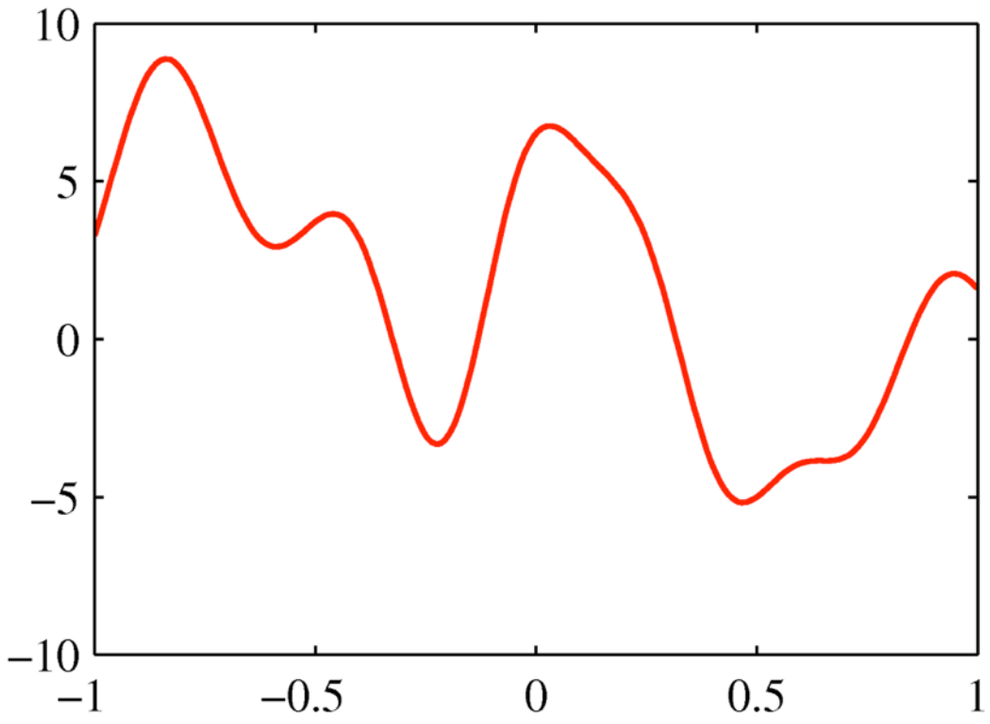
$$p(y = +1 \mid \mathbf{x}) = \sigma(f(\mathbf{x}))$$

If the sigmoid function is symmetric: $\sigma(-z) = 1 - \sigma(z)$ then we have $p(y \mid \mathbf{x}) = \sigma(yf(\mathbf{x}))$.

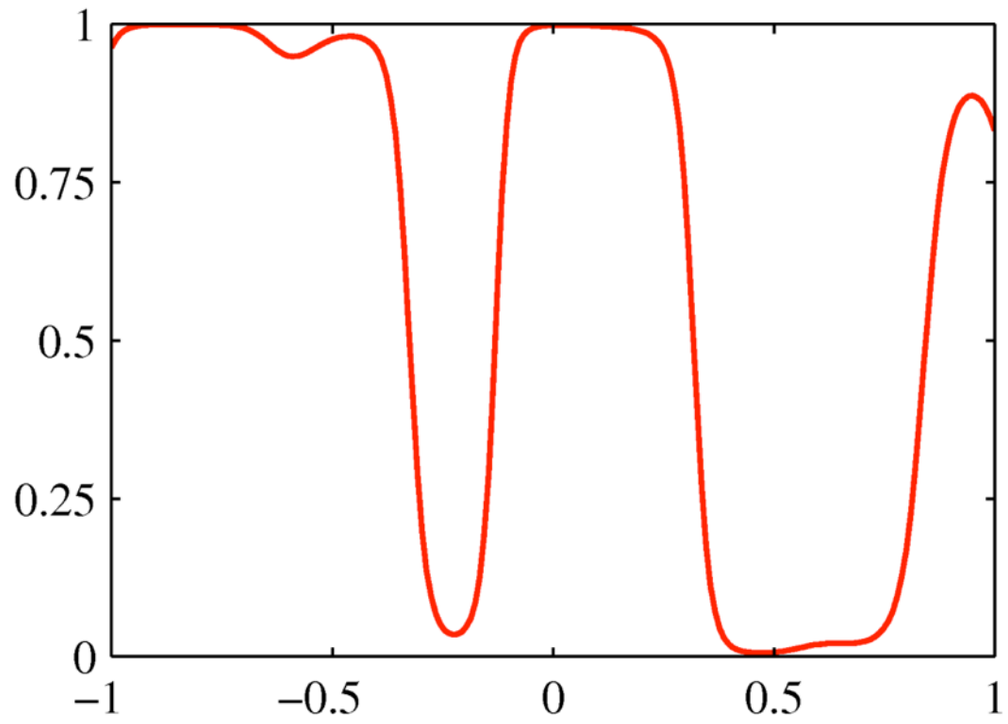
A typical type of sigmoid function is the logistic sigmoid:

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

Application of the Sigmoid Function



Function sampled from a Gaussian Process

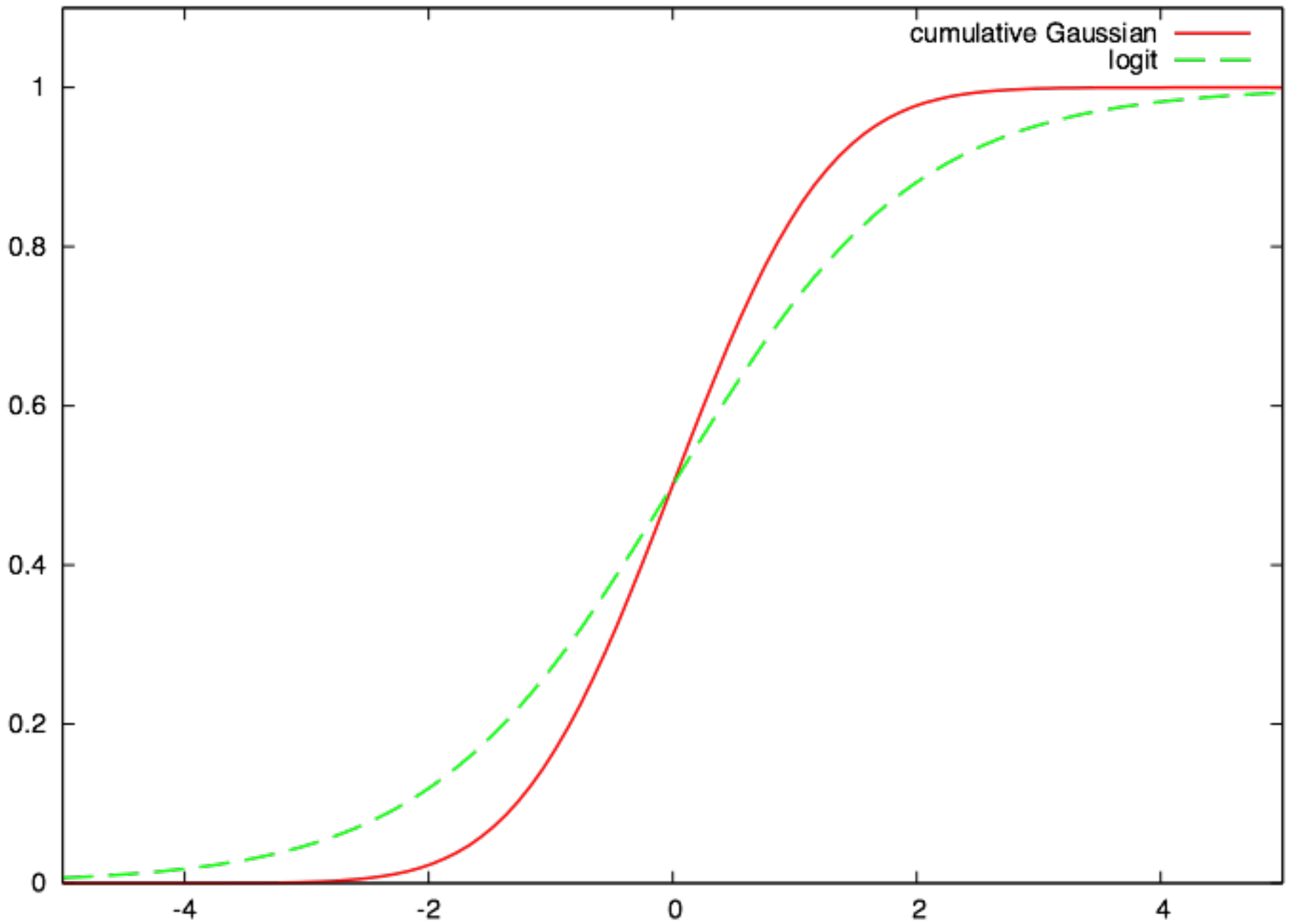


Sigmoid function applied to the GP function

Another symmetric sigmoid function is the **cumulative Gaussian:**

$$\Phi(z) = \int_{-\infty}^{z} \mathcal{N}(x \mid 0, 1) dx$$

Visualization of Sigmoid Functions



The cumulative Gaussian is slightly steeper than the logistic sigmoid

The Latent Variables

In regression, we directly estimated f as

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$$

and values of f where observed in the training data. Now only labels +1 or -1 are observed and f is treated as a set of **latent variables.**

A major advantage of the Gaussian process classifier over other methods is that it **marginalizes** over all latent functions rather than maximizing some model parameters.

Class Prediction with a GP

The aim is to compute the predictive distribution

$$p(y_* = +1 \mid X, \mathbf{y}, \mathbf{x}_*) = \int p(y_* \mid f_*) p(f_* \mid X, \mathbf{y}, \mathbf{x}_*) df_*$$

$\sigma(f_*)$

Class Prediction with a GP

The aim is to compute the predictive distribution

$$p(y_* = +1 \mid X, \mathbf{y}, \mathbf{x}_*) = \int p(y_* \mid f_*) p(f_* \mid X, \mathbf{y}, \mathbf{x}_*) df_*$$

we marginalize over the latent variables from the training data:

$$p(f_* \mid X, \mathbf{y}, \mathbf{x}_*) = \int p(f_* \mid X, \mathbf{x}_*, \mathbf{f}) p(\mathbf{f} \mid X, \mathbf{y}) d\mathbf{f}$$

predictive distribution of the latent variable (from regression)

Class Prediction with a GP

The aim is to compute the predictive distribution

$$p(y_* = +1 \mid X, \mathbf{y}, \mathbf{x}_*) = \int p(y_* \mid f_*) p(f_* \mid X, \mathbf{y}, \mathbf{x}_*) df_*$$

we marginalize over the latent variables from the training data:

$$p(f_* \mid X, \mathbf{y}, \mathbf{x}_*) = \int p(f_* \mid X, \mathbf{x}_*, \mathbf{f}) p(\mathbf{f} \mid X, \mathbf{y}) d\mathbf{f}$$

we need the posterior over the latent variables:

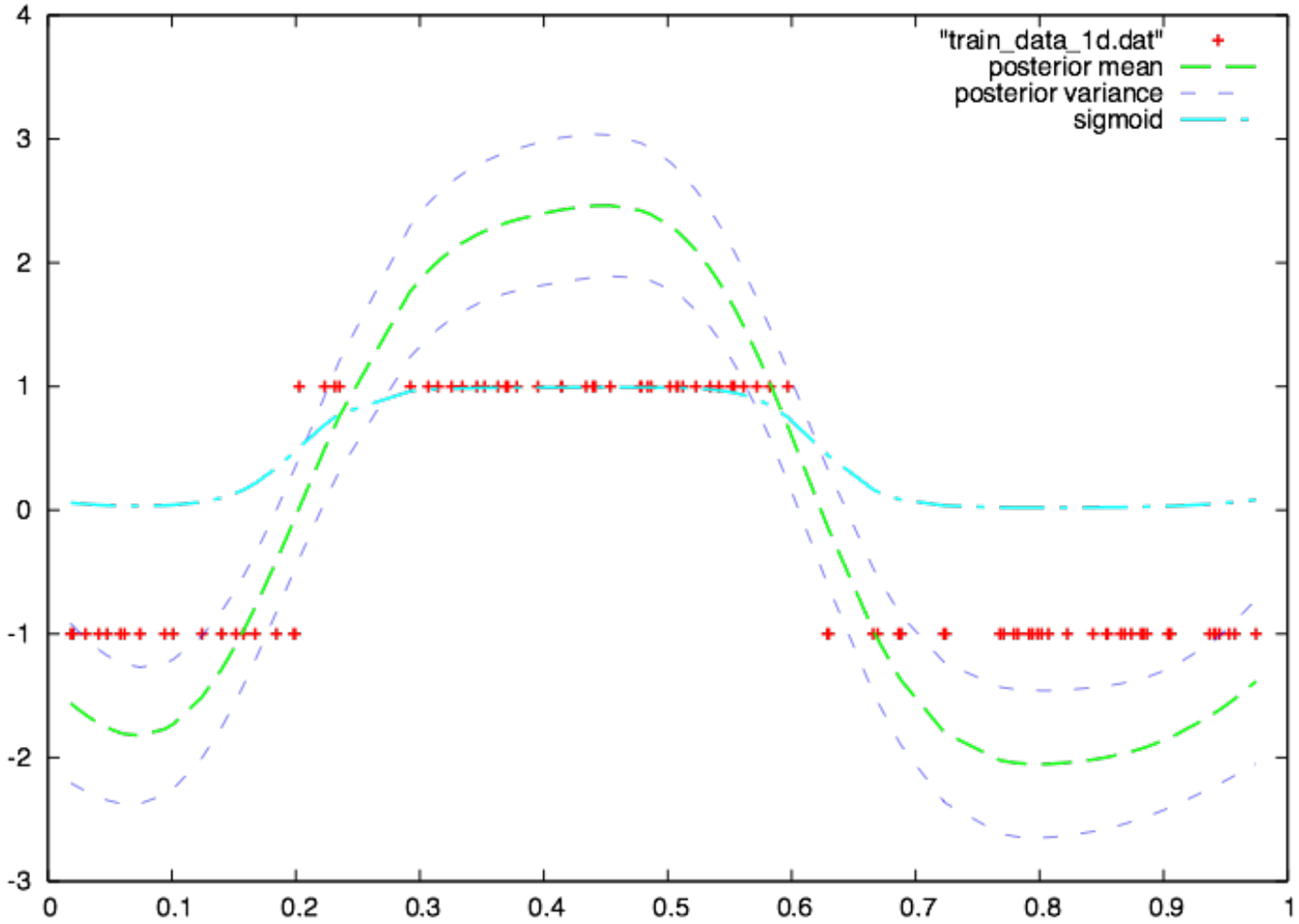
$$p(\mathbf{f} \mid X, \mathbf{y}) = \frac{p(\mathbf{y} \mid \mathbf{f}) p(\mathbf{f} \mid X)}{p(\mathbf{y} \mid X)}$$

likelihood (sigmoid)

prior

normalizer

A Simple Example



- Red: Two-class training data
- Green: mean function of $p(\mathbf{f} \mid X, \mathbf{y})$
- Light blue: sigmoid of the mean function

But There Is A Problem...

$$p(\mathbf{f} \mid X, \mathbf{y}) = \frac{p(\mathbf{y} \mid \mathbf{f}) p(\mathbf{f} \mid X)}{p(\mathbf{y} \mid X)}$$

- The likelihood term is not a Gaussian!
- This means, we can not compute the posterior in closed form.
- There are several different solutions in the literature, e.g.:
  - Laplace approximation
  - Expectation Propagation
  - Variational methods

Laplace Approximation

$$p(\mathbf{f} \mid X, \mathbf{y}) \approx q(\mathbf{f} \mid X, \mathbf{y}) = \mathcal{N}(\mathbf{f} \mid \hat{\mathbf{f}}, A^{-1})$$

where $\hat{\mathbf{f}} = \arg\max_{\mathbf{f}} p(\mathbf{f} \mid X, \mathbf{y})$

and $A = -\nabla\nabla \log p(\mathbf{f} \mid X, \mathbf{y})|_{\mathbf{f}=\hat{\mathbf{f}}}$

second-order Taylor expansion

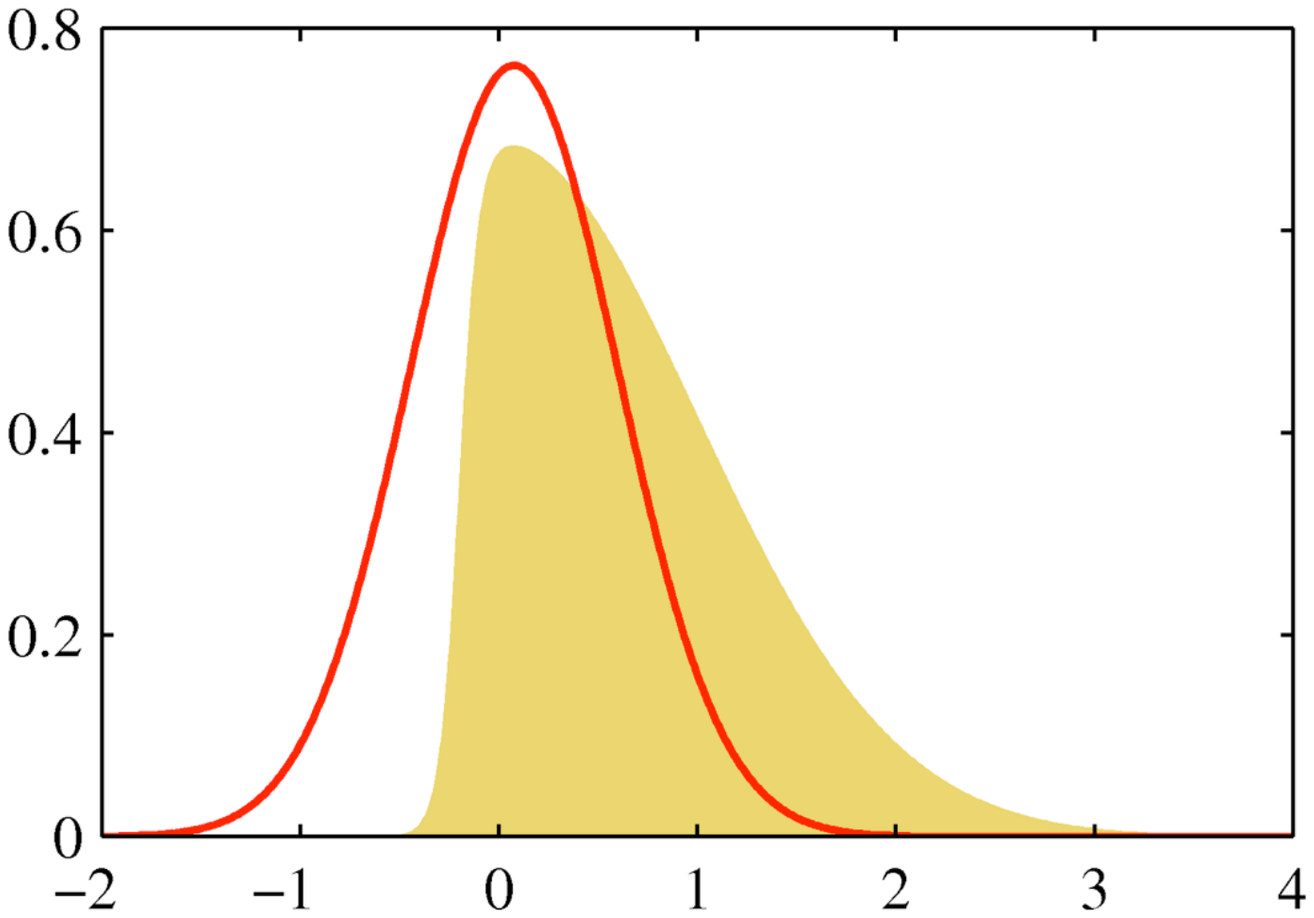
To compute $\hat{\mathbf{f}}$ an iterative approach using Newton's method has to be used.

The Hessian matrix A can be computed as

$$A = K^{-1} + W$$

where $W = -\nabla\nabla \log p(\mathbf{y} \mid \mathbf{f})$ is a diagonal matrix which depends on the sigmoid function.

Laplace Approximation



- Yellow: a non-Gaussian posterior
- Red: a Gaussian approximation, the mean is the mode of the posterior, the variance is the negative second derivative at the mode

Predictions

Now that we have $p(\mathbf{f} \mid X, \mathbf{y})$ we can compute:

$$p(f_* \mid X, \mathbf{y}, \mathbf{x}_*) = \int p(f_* \mid X, \mathbf{x}_*, \mathbf{f}) p(\mathbf{f} \mid X, \mathbf{y}) d\mathbf{f}$$

From the regression case we have:

$$p(f_* \mid X, \mathbf{x}_*, \mathbf{f}) = \mathcal{N}(f_* \mid \mu_*, \Sigma_*)$$

where $\mu_* = \mathbf{k}_*^T K^{-1} \mathbf{f}$ $\quad \Sigma_* = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^T K^{-1} \mathbf{k}_*$

Linear in $\mathbf{f}$

This reminds us of a property of Gaussians that we saw earlier!

Gaussian Properties (Rep.)

If we are given this:

I. $$p(\mathbf{x}) = \mathcal{N}(\mathbf{x} \mid \mu, \Sigma_1)$$

II. $$p(\mathbf{y} \mid \mathbf{x}) = \mathcal{N}(\mathbf{y} \mid A\mathbf{x} + \mathbf{b}, \Sigma_2)$$

Then it follows (properties of Gaussians):

III. $$p(\mathbf{y}) = \mathcal{N}(\mathbf{y} \mid A\mu + \mathbf{b}, \Sigma_2 + A\Sigma_1 A^T)$$

IV. $$p(\mathbf{x} \mid \mathbf{y}) = \mathcal{N}(\mathbf{x} \mid \Sigma(A^T \Sigma_2^{-1}(\mathbf{y} - \mathbf{b}) + \Sigma_1^{-1}\mathbf{y}), \Sigma)$$

where

$$\Sigma = (\Sigma_1^{-1} + A^T \Sigma_s^{-1} A)^{-1}$$

Applying this to Laplace

$$\mathbb{E}[f_* \mid X, \mathbf{y}, \mathbf{x}_*] = \mathbf{k}(\mathbf{x}_*)^T K^{-1} \hat{\mathbf{f}}$$

$$\mathbb{V}[f_* \mid X, \mathbf{y}, \mathbf{x}_*] = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^T (K + W^{-1})^{-1} \mathbf{k}_*$$

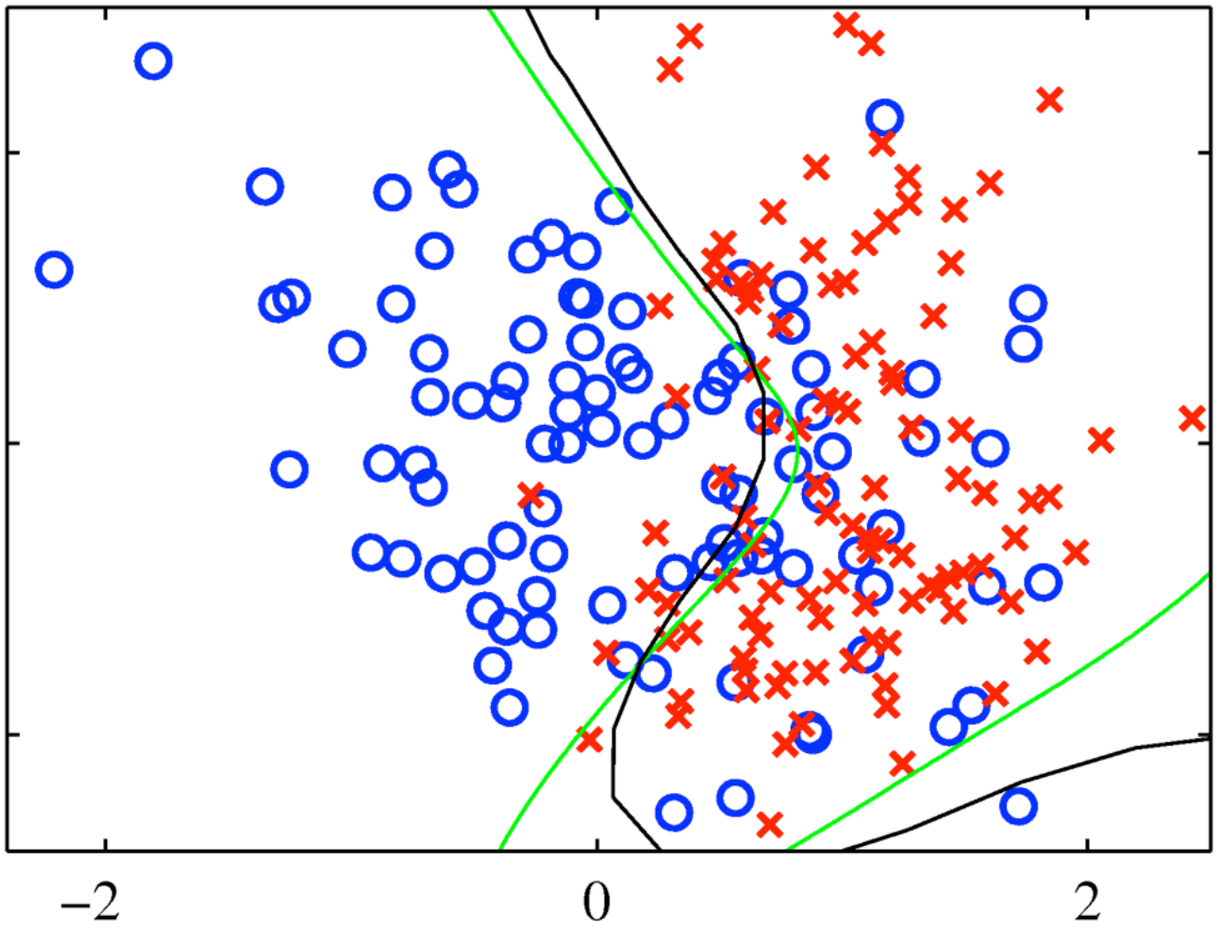
It remains to compute

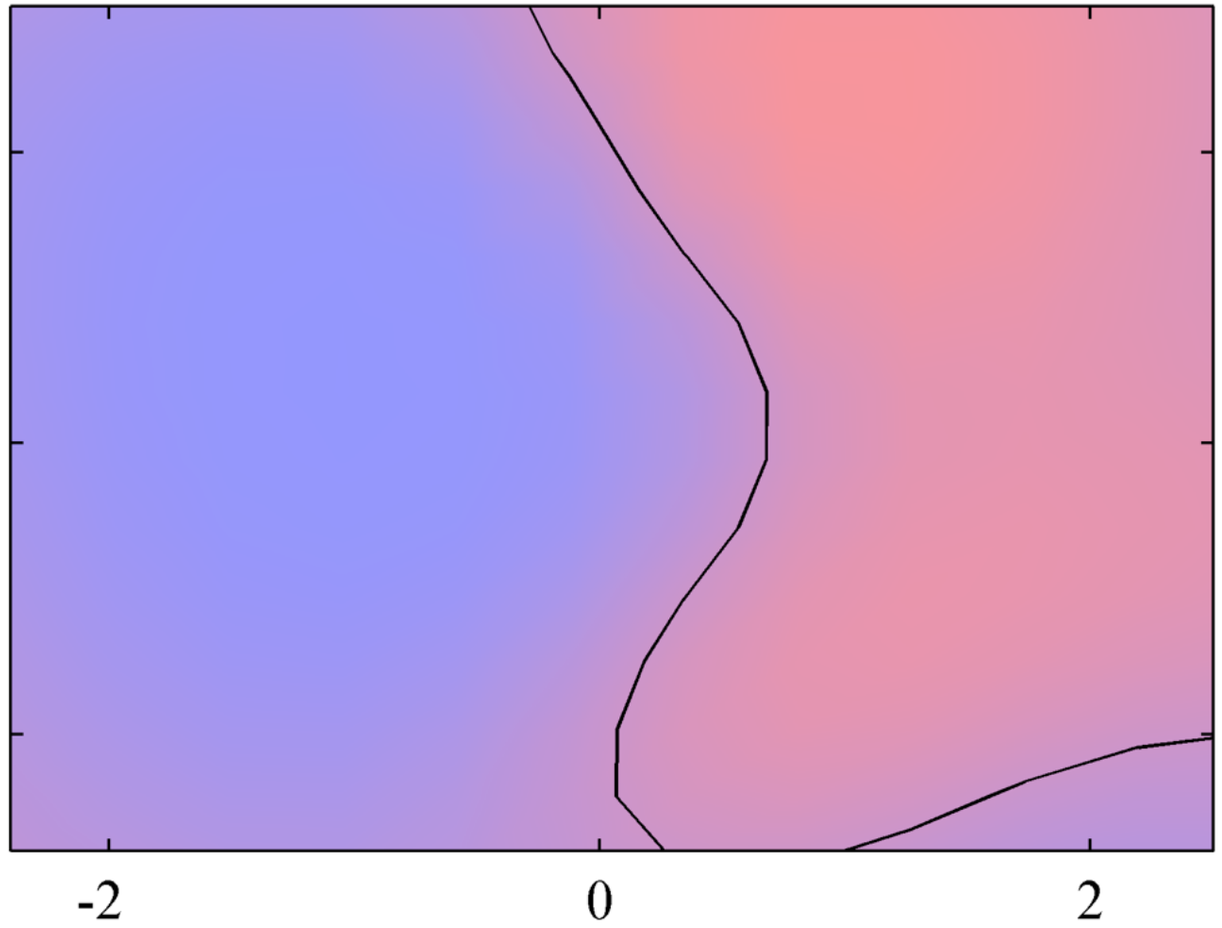
$$p(y_* = +1 \mid X, \mathbf{y}, \mathbf{x}_*) = \int p(y_* \mid f_*) p(f_* \mid X, \mathbf{y}, \mathbf{x}_*) df_*$$

Depending on the kind of sigmoid function we

- can compute this in closed form (cumulative Gaussian sigmoid)
- have to use sampling methods or analytical approximations (logistic sigmoid)

A Simple Example





- Two-class problem (training data in red and blue)
- Green line: optimal decision boundary
- Black line: GP classifier decision boundary
- Right: posterior probability

Properties of the GP Classifier

- The biggest benefit of the GPC is that it gives a "well-calibrated" estimate of the uncertainty:
  - The prediction is based on marginalization over all possible models, not on a particular one
  - Other techniques use the distance to the decision boundary, which is often not correct
- The biggest disadvantage of the GPC is its memory and run time complexity
- However: There are sparse versions of the GPC, such as the Informative Vector Machine (IVM), that uses a similar idea than the SVM

Summary

- Gaussian Processes are Normal distributions over functions
- To specify a GP we need a covariance function (kernel) and a mean function
- For regression we can compute the predictive distribution in closed form
- For classification, we use a sigmoid and have to approximate the latent posterior
- More on Gaussian Processes: http://videolectures.net/epsrcws08_rasmussen_lgp/