8. Kernel Methods and Gaussian Processes

Motivation

- Usually learning algorithms assume that some kind of feature function is given
- Reasoning is then done on a feature vector of a given (finite) length
- But: some objects are hard to represent with a fixed-size feature vector, e.g. text documents, molecular structures, evolutionary trees
- Idea: use a way of measuring similarity **without** the need of features, e.g. the edit distance for strings
- This we will call a **kernel function**

Dual Representation

Many problems can be expressed using a **dual** formulation. Example (linear regression):

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(\mathbf{w}^T\phi(\mathbf{x}_n) - t_n)^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \qquad \phi(\mathbf{x}_n) \in \mathbb{R}^M$$

Dual Representation

Many problems can be expressed using a **dual** formulation. Example (linear regression):

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(\mathbf{w}^T\phi(\mathbf{x}_n) - t_n)^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \qquad \phi(\mathbf{x}_n) \in \mathbb{R}^M$$

if we write this in vector form, we get

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}\Phi^T\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \quad \mathbf{t} \in \mathbb{R}^N$$

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

Dual Representation

Many problems can be expressed using a **dual** formulation. Example (linear regression):

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(\mathbf{w}^T\phi(\mathbf{x}_n) - t_n)^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \qquad \phi(\mathbf{x}_n) \in \mathbb{R}^M$$

if we write this in vector form, we get

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}\Phi^T\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \quad \mathbf{t} \in \mathbb{R}^N$$

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

and the solution is

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

Dual Representation

Many problems can be expressed using a **dual** formulation, including linear regression.

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}\Phi^T\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

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

However, we can express this result in a different way using the **matrix inversion lemma:**

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

Dual Representation

Many problems can be expressed using a **dual** formulation. Example (linear regression):

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}\Phi^T\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

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

However, we can express this result in a different way using the **matrix inversion lemma:**

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

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

Dual Representation

Many problems can be expressed using a **dual** formulation. Example (linear regression):

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}\Phi^T\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

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

$$\mathbf{w} = \Phi^T\underbrace{(\Phi\Phi^T + \lambda I_N)^{-1}\mathbf{t}}_{=:\mathbf{a}}$$

“Dual Variables”

Dual Representation

Many problems can be expressed using a **dual** formulation. Example (linear regression):

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}\Phi^T\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

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

$$\mathbf{w} = \Phi^T\underbrace{(\Phi\Phi^T + \lambda I_N)^{-1}\mathbf{t}}_{=:\mathbf{a}}$$

“Dual Variables”

Plugging $\mathbf{w} = \Phi^T\mathbf{a}$ into $J(\mathbf{w})$ gives:

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^T\underbrace{\Phi\Phi^T}_{=:K}\Phi\Phi^T\mathbf{a} - \mathbf{a}^T\Phi\Phi^T\mathbf{t} + \mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^T\Phi\Phi^T\mathbf{a}$$

Dual Representation

Many problems can be expressed using a **dual** formulation. Example (linear regression):

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}\Phi^T\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^T KK\mathbf{a} - \mathbf{a}^T K\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^T K\mathbf{a} \quad K = \Phi\Phi^T$$

This is called the **dual formulation**.

Note: $\mathbf{a} \in \mathbb{R}^N \quad \mathbf{w} \in \mathbb{R}^M$

Dual Representation

Many problems can be expressed using a **dual** formulation. Example (linear regression):

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}\Phi^T\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

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

This is called the **dual formulation**.

The solution to the dual problem is:

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

Dual Representation

Many problems can be expressed using a **dual** formulation. Example (linear regression):

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}\Phi^T\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

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

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

This we can use to make **predictions**:

$$f(\mathbf{x}^*) = \mathbf{w}^T\phi(\mathbf{x}^*) = \mathbf{a}^T\Phi\phi(\mathbf{x}^*) = \mathbf{k}(\mathbf{x}^*)^T(K + \lambda I_N)^{-1}\mathbf{t}$$

(now $\mathbf{x}^*$ is unknown and $\mathbf{a}$ is given from training)

Dual Representation

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

where:

$$\mathbf{k}(\mathbf{x}^*) = \begin{pmatrix} \phi(\mathbf{x}_1)^T \phi(\mathbf{x}^*) \\ \vdots \\ \phi(\mathbf{x}_N)^T \phi(\mathbf{x}^*) \end{pmatrix} \quad K = \begin{pmatrix} \phi(\mathbf{x}_1)^T \phi(\mathbf{x}_1) & \dots & \phi(\mathbf{x}_1)^T \phi(\mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ \phi(\mathbf{x}_N)^T \phi(\mathbf{x}_1) & \dots & \phi(\mathbf{x}_N)^T \phi(\mathbf{x}_N) \end{pmatrix}$$

Thus, f is expressed only in terms of **dot products** between different pairs of $\phi(\mathbf{x})$, or in terms of the **kernel function**

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

Representation using the Kernel

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

Now we have to invert a matrix of size $N \times N$, before it was $M \times M$ where $M < N$, but:

By expressing everything with the kernel function, we can deal with very high-dimensional or even **infinite**-dimensional feature spaces!

Idea: Don't use features at all but simply define a **similarity** function expressed as the kernel!

Constructing Kernels

The straightforward way to define a kernel function is to first find a basis (=feature) function $\phi(\mathbf{x})$ and to define:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

This means, k is an inner product in some space $\mathcal{H}$, i.e:

1. Symmetry: $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_j), \phi(\mathbf{x}_i) \rangle = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$
2. Linearity: $\langle a(\phi(\mathbf{x}_i) + \mathbf{z}), \phi(\mathbf{x}_j) \rangle = a\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle + a\langle \mathbf{z}, \phi(\mathbf{x}_j) \rangle$
3. Positive definite: $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_i) \rangle \geq 0$, equal if $\phi(\mathbf{x}_i) = \mathbf{0}$

Can we find conditions for k under which there is a (possibly infinite dimensional) basis function into $\mathcal{H}$, where k is an inner product?

Constructing Kernels

Theorem (Mercer): If k is

1. symmetric, i.e. $k(\mathbf{x}_i, \mathbf{x}_j) = k(\mathbf{x}_j, \mathbf{x}_i)$ and
2. positive definite, i.e.

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

"Gram Matrix"

is positive definite, then there exists a mapping $\phi(\mathbf{x})$ into a feature space $\mathcal{H}$ so that k can be expressed as an inner product in $\mathcal{H}$.

This means, we don't need to find $\phi(\mathbf{x})$ explicitly!

We can directly work with k "Kernel Trick"

Examples of Valid Kernels

- Polynomial Kernel:

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + c)^d \quad c > 0 \quad d \in \mathbb{N}$$

- Gaussian Kernel:

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

- Kernel for sets:

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

- Matern kernel:

$$k(r) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu} r}{l} \right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu} r}{l} \right) \qquad r = \|\mathbf{x}_i - \mathbf{x}_j\|, \nu > 0, l > 0$$

A Simple Example

Define a kernel function as

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}')^2 \qquad \mathbf{x}, \mathbf{x}' \in \mathbb{R}^2$$

This can be written as:

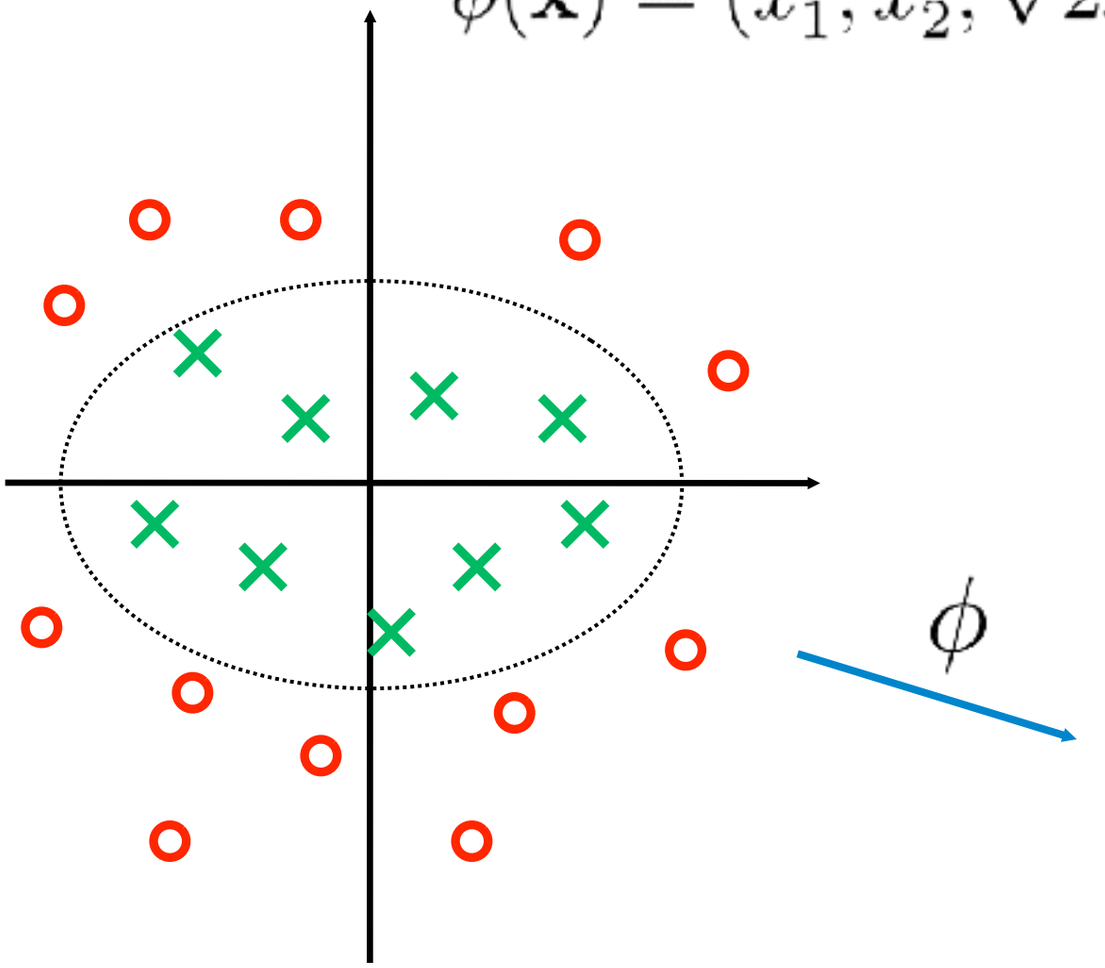
$$\begin{aligned}(x_1 x_1' + x_2 x_2')^2 &= x_1^2 x_1'^2 + 2 x_1 x_1' x_2 x_2' + x_2^2 x_2'^2 \\ &= (x_1^2, x_2^2, \sqrt{2} x_1 x_2)(x_1'^2, x_2'^2, \sqrt{2} x_1' x_2')^T \\ &= \phi(\mathbf{x})^T \phi(\mathbf{x}')\end{aligned}$$

It can be shown that this holds in general for

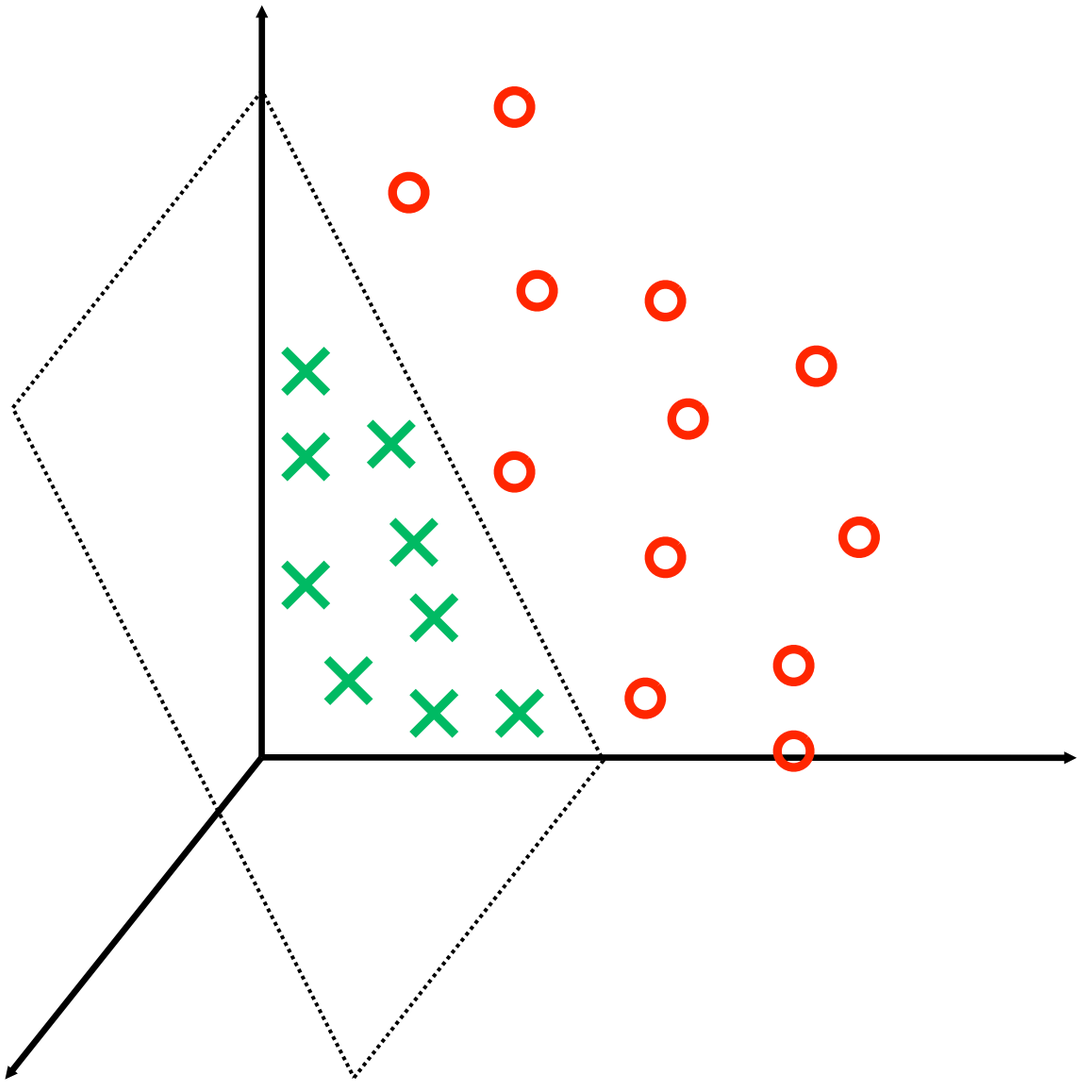
$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j)^d$$

Visualization of the Example

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



ϕ



Decision boundary becomes a hyperplane

Original decision boundary is an ellipse

Application Examples

Kernel Methods can be applied for many different problems, e.g.:

- Density estimation (unsupervised learning)
- Regression
- Principal Component Analysis (PCA)
- Classification

Most important Kernel Methods are

- Support Vector Machines
- Gaussian Processes

Kernelization

- Many existing algorithms can be converted into kernel methods
- This process is called “kernelization”

Idea:

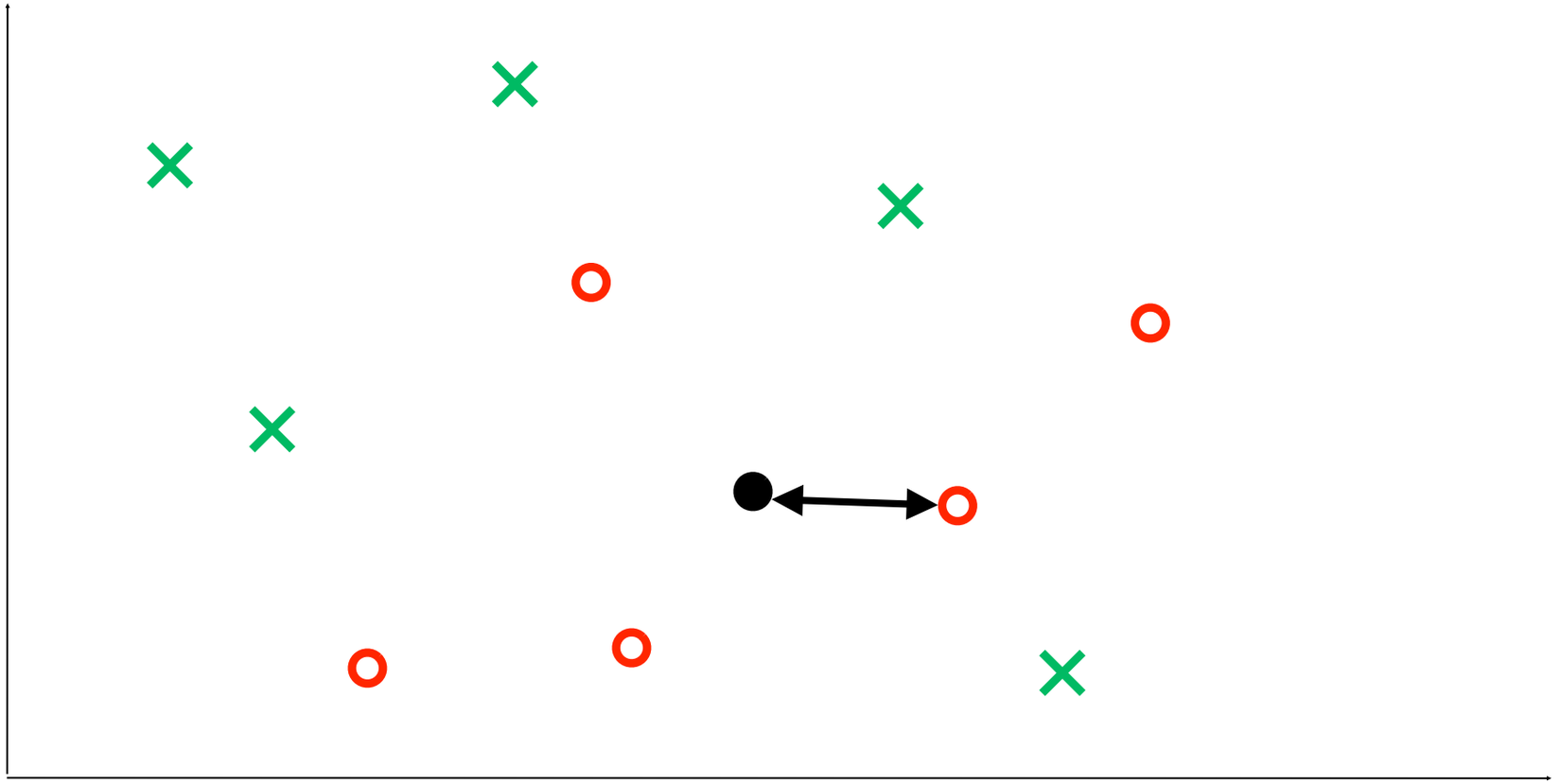
- express similarities of data points in terms of an inner product (dot product)
- replace all occurrences of that inner product by the kernel function

This is called the **kernel trick**

Example: Nearest Neighbor

- The NN classifier selects the label of the nearest neighbor in Euclidean distance

$$\|\mathbf{x}_i - \mathbf{x}_j\|^2 = \mathbf{x}_i^T\mathbf{x}_i + \mathbf{x}_j^T\mathbf{x}_j - 2\mathbf{x}_i^T\mathbf{x}_j$$



Example: Nearest Neighbor

- The NN classifier selects the label of the nearest neighbor in Euclidean distance

$$\|\mathbf{x}_i - \mathbf{x}_j\|^2 = \mathbf{x}_i^T\mathbf{x}_i + \mathbf{x}_j^T\mathbf{x}_j - 2\mathbf{x}_i^T\mathbf{x}_j$$

- We can now replace the dot products by a valid Mercer kernel and we obtain:

$$d(\mathbf{x}_i, \mathbf{x}_j)^2 = k(\mathbf{x}_i, \mathbf{x}_i) + k(\mathbf{x}_j, \mathbf{x}_j) - 2k(\mathbf{x}_i, \mathbf{x}_j)$$

- This is a **kernelized** nearest-neighbor classifier
- We do not explicitly compute feature vectors!

Back to Linear Regression (Rep.)

We had the primal and the dual formulation:

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\Phi^T\Phi\mathbf{w} - \mathbf{w}\Phi^T\mathbf{t} + \frac{1}{2}\mathbf{t}^T\mathbf{t} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

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

with the dual solution:

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

This we can use to make **predictions**:

$$f(\mathbf{x}^*) = \mathbf{w}^T\phi(\mathbf{x}^*) = \mathbf{a}^T\Phi\phi(\mathbf{x}^*) = \mathbf{k}(\mathbf{x}^*)^T(K + \lambda I_N)^{-1}\mathbf{t}$$

Note: This is (only) the **most likely** prediction!

Observations

- We have found a way to predict function values of y for new input points $\mathbf{x}^*$
- As we used regularized regression, we can equivalently find the **predictive distribution** by marginalizing out the parameters $\mathbf{w}$

Questions:

- Can we find a closed form for that distribution?
- How can we model the **uncertainty** of our prediction?
- Can we use that for classification?

Gaussian Marginals and Conditionals

First, we need some formulae:

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

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

Then the cond. distribution $p(\mathbf{x}_a \mid \mathbf{x}_b) = \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_{a|b}, \Sigma_{a|b})$

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**!

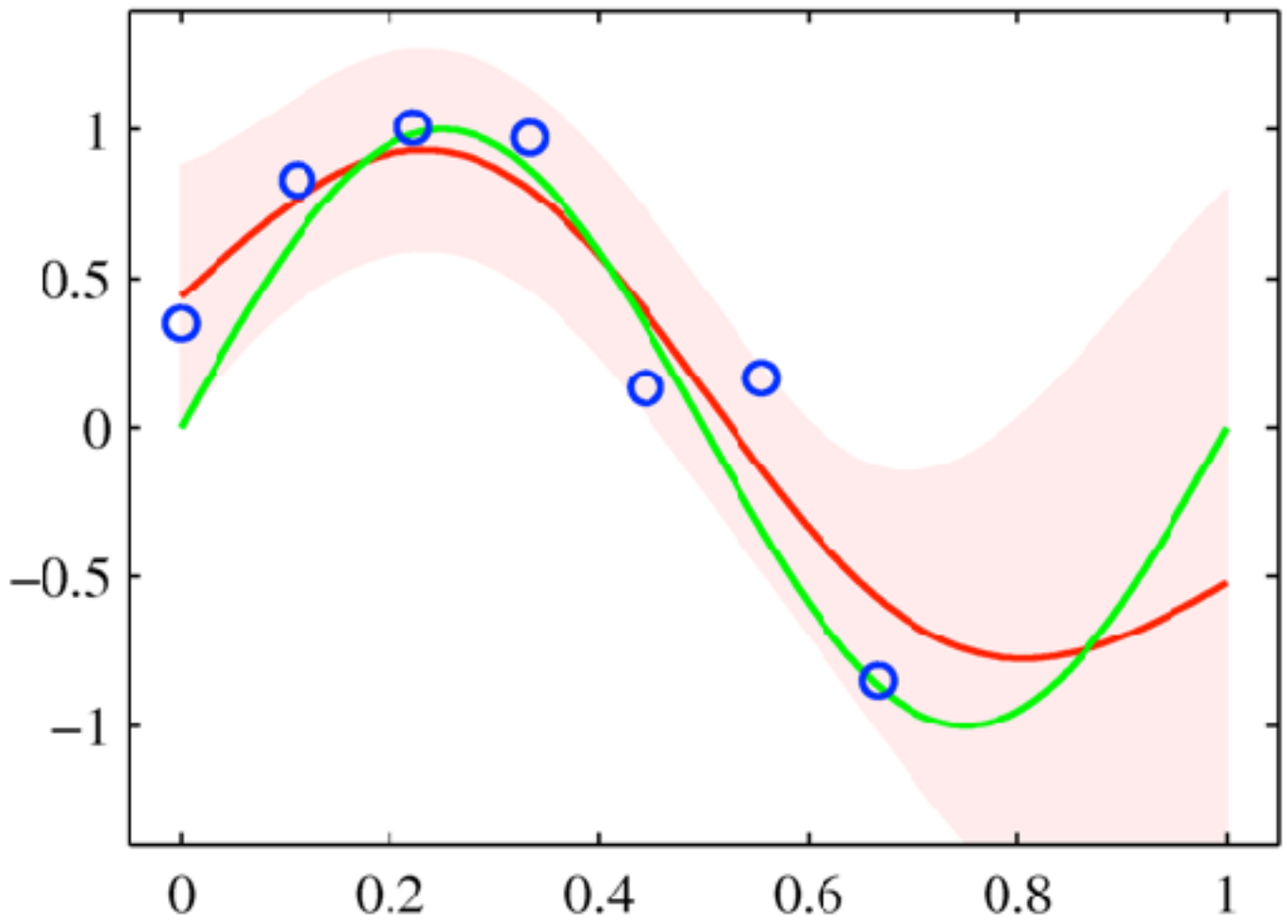
To specify a GP we need:

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

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.

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

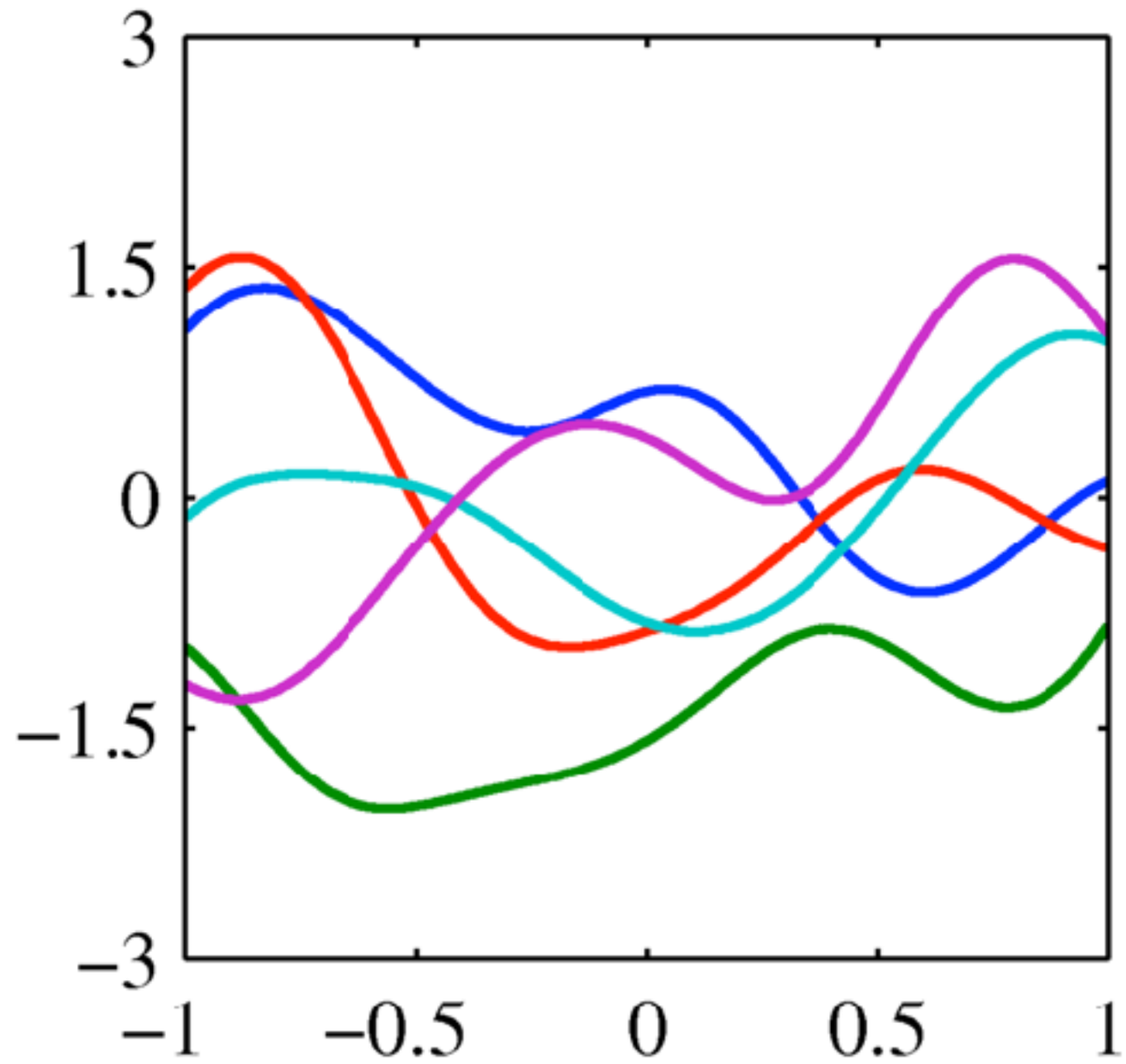
$$K_{ij} = k(\mathbf{x}_i^*, \mathbf{x}_j^*)$$

3. Generate a random Gaussian vector from

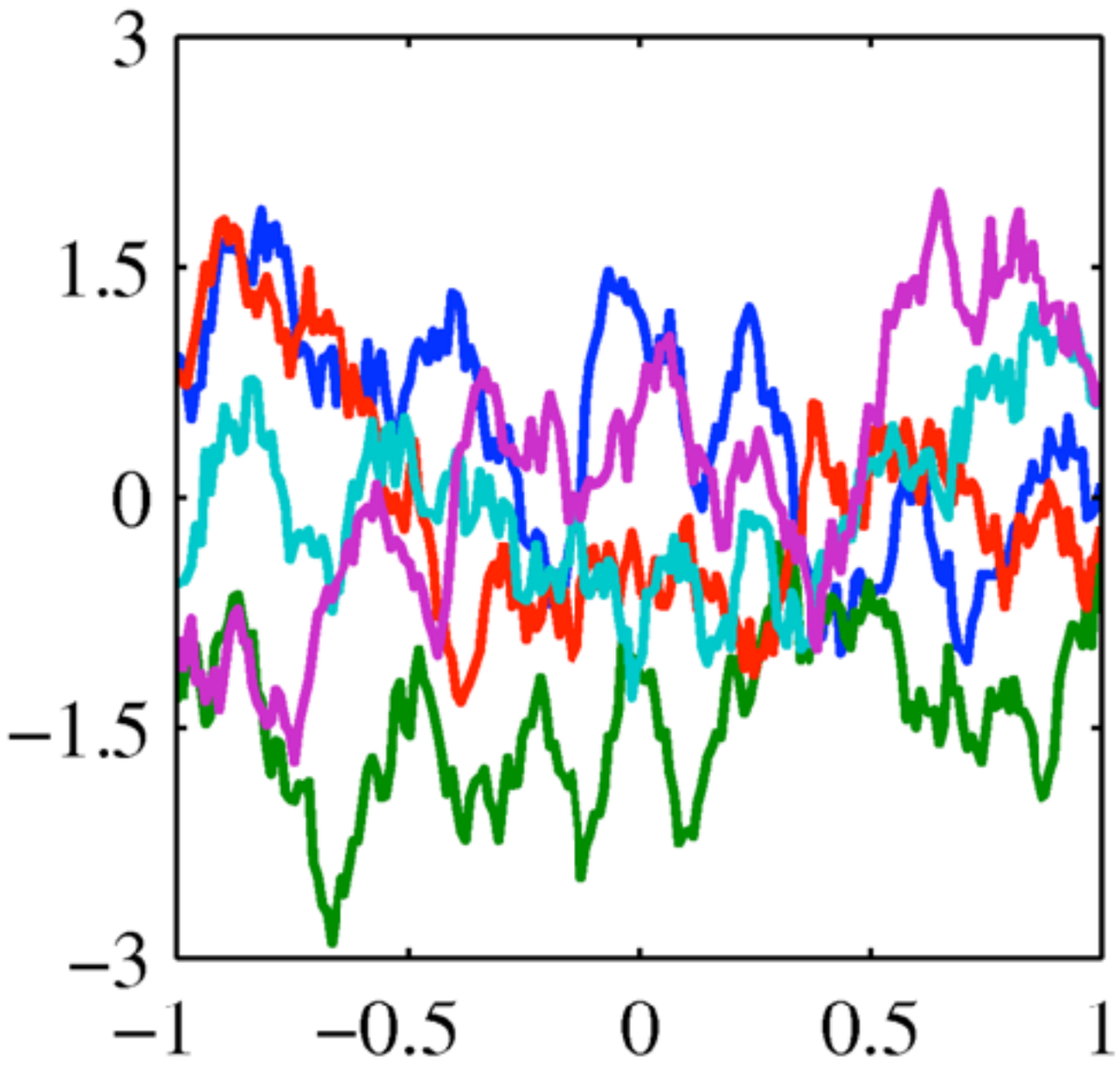
$$\mathbf{y}_* \sim \mathcal{N}(\mathbf{0}, K)$$

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 $\mathbf{x}_1, \ldots, \mathbf{x}_N \quad y_1, \ldots, y_N$

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

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

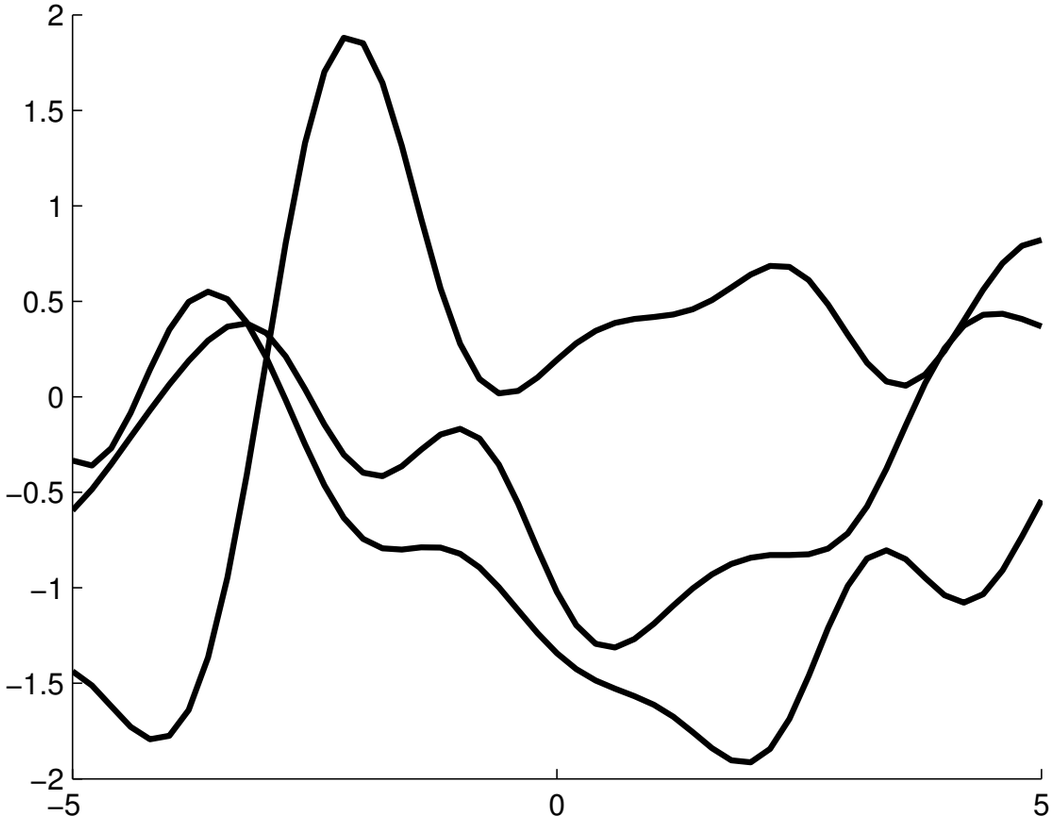
where

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

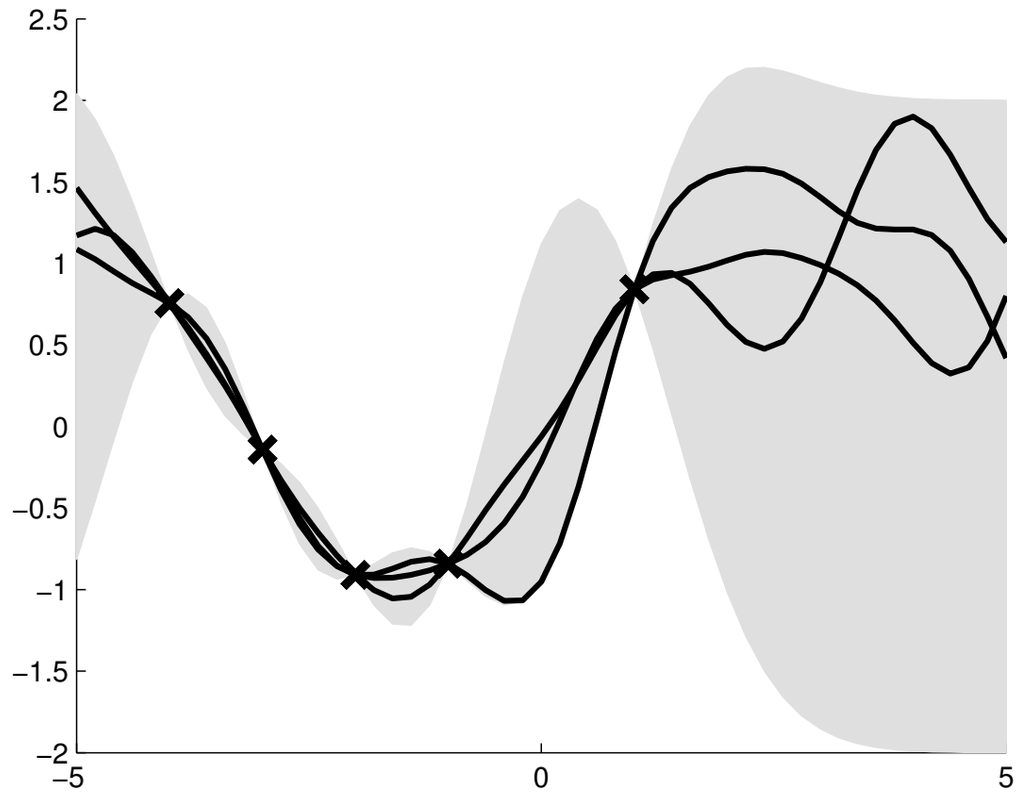
$$\Sigma_* = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^T K^{-1} \mathbf{k}_*$$

This defines the **predictive distribution.**

Example



Functions sampled from
a Gaussian Process prior



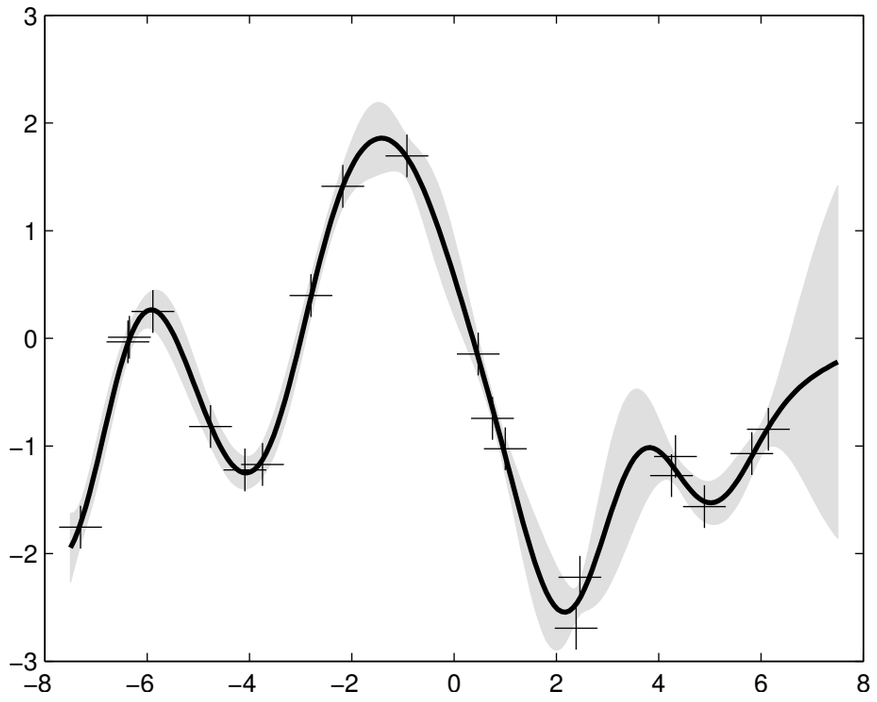
Functions sampled from the
predictive distribution

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.

Varying the Hyperparameters



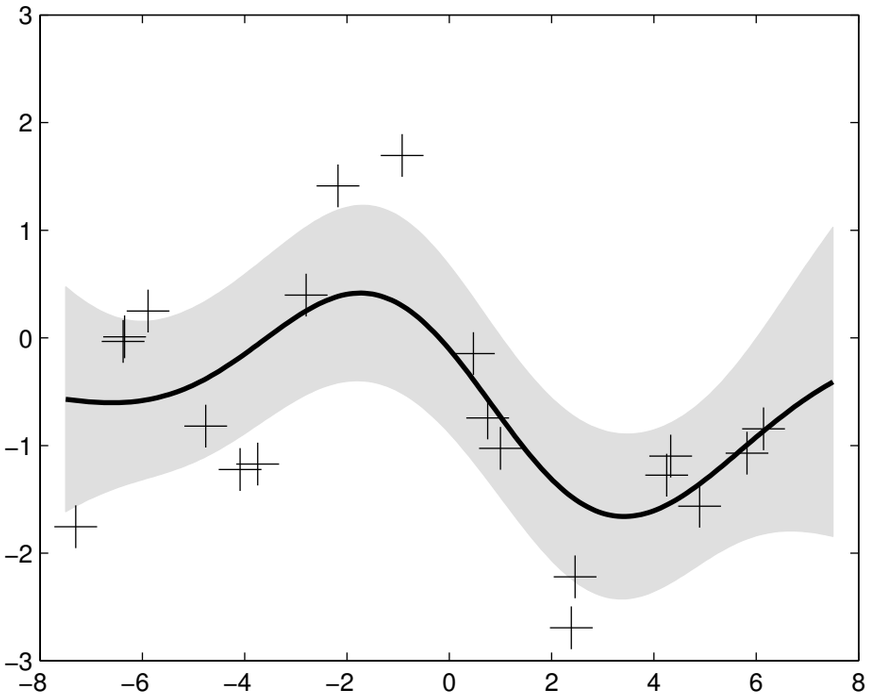
$$l = 0.3,$$

$$\sigma_f = 1.08,$$

$$\sigma_n = 0.0005$$

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

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



$$l = 3$$

$$\sigma_f = 1.16$$

$$\sigma_n = 0.89$$

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

$K_{ij} \leftarrow k(\mathbf{x}_i, \mathbf{x}_j)$
$L \leftarrow \texttt{cholesky}(K + \sigma_n^2 I)$
$\boldsymbol{\alpha} \leftarrow L^T \backslash (L \backslash \mathbf{y})$

Precomputed during Training

$\mathbb{E}[f_*] \leftarrow \mathbf{k}_*^T \boldsymbol{\alpha}$
$\mathbf{v} \leftarrow L \backslash \mathbf{k}_*$
$\texttt{var}[f_*] \leftarrow k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^T \mathbf{v}$

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

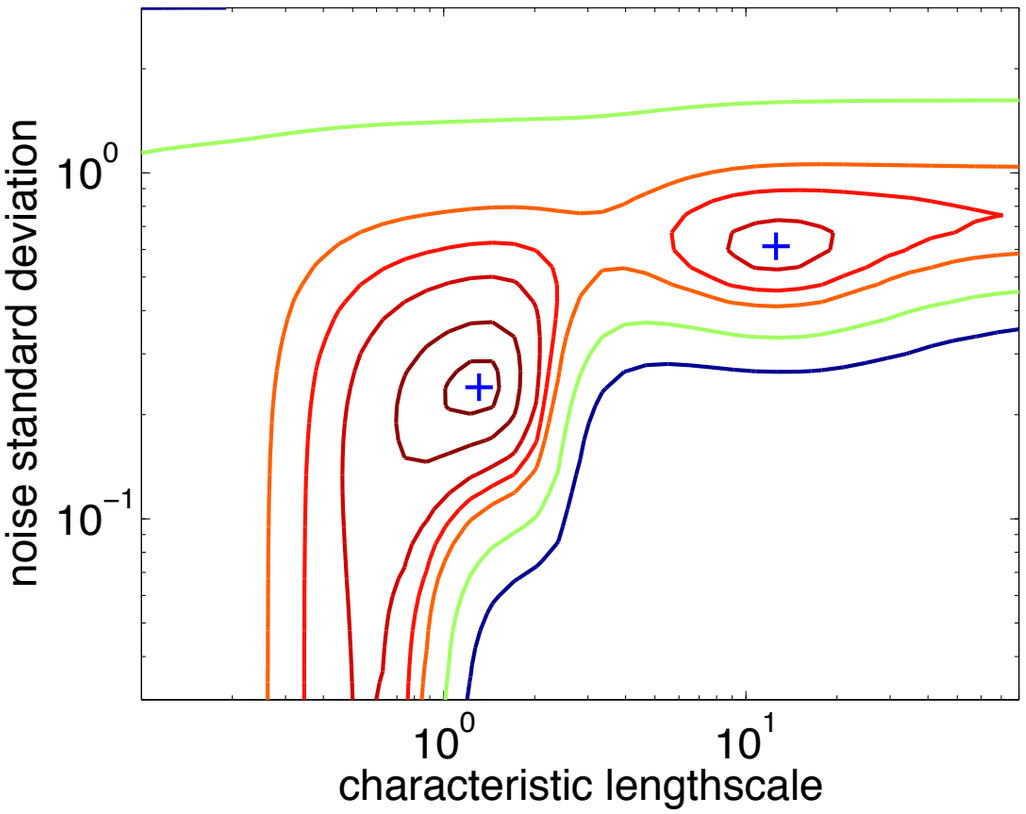
To find optimal hyper parameters we need the **marginal likelihood:**

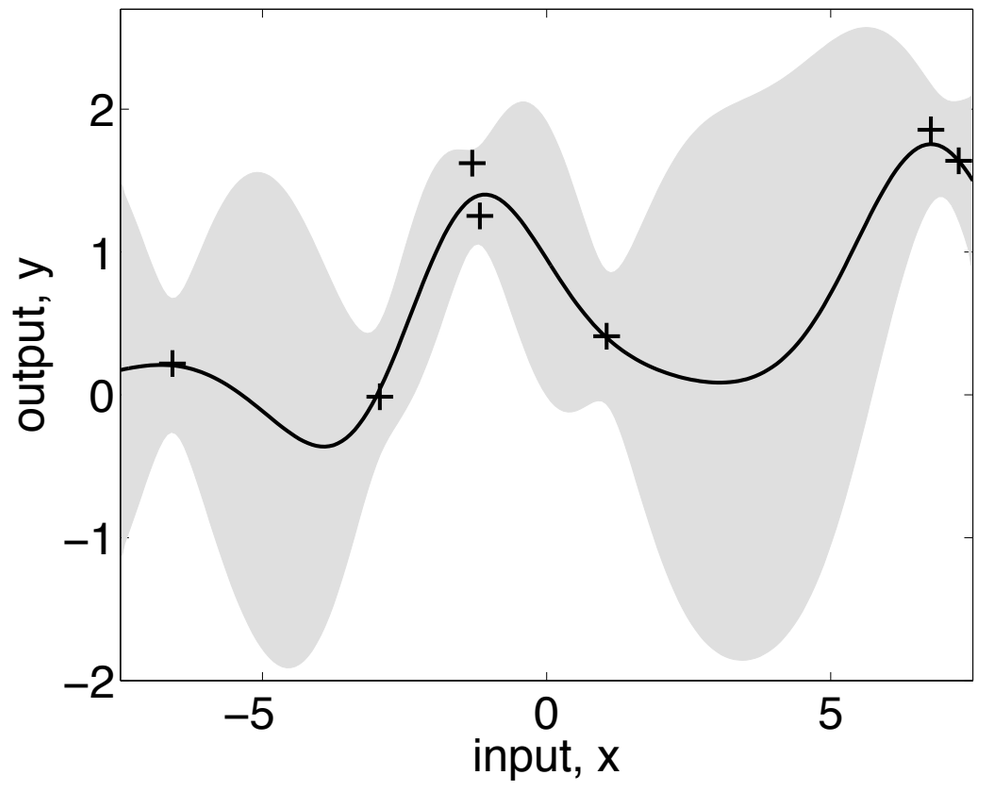
$$p(\mathbf{y} \mid X) = \int p(\mathbf{y} \mid \mathbf{f}, X) p(\mathbf{f} \mid X) d\mathbf{f}$$

This expression implicitly depends on the hyper parameters, but $\mathbf{y}$ and X are given from the training data. It can be computed in closed form, as all terms are Gaussians.

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

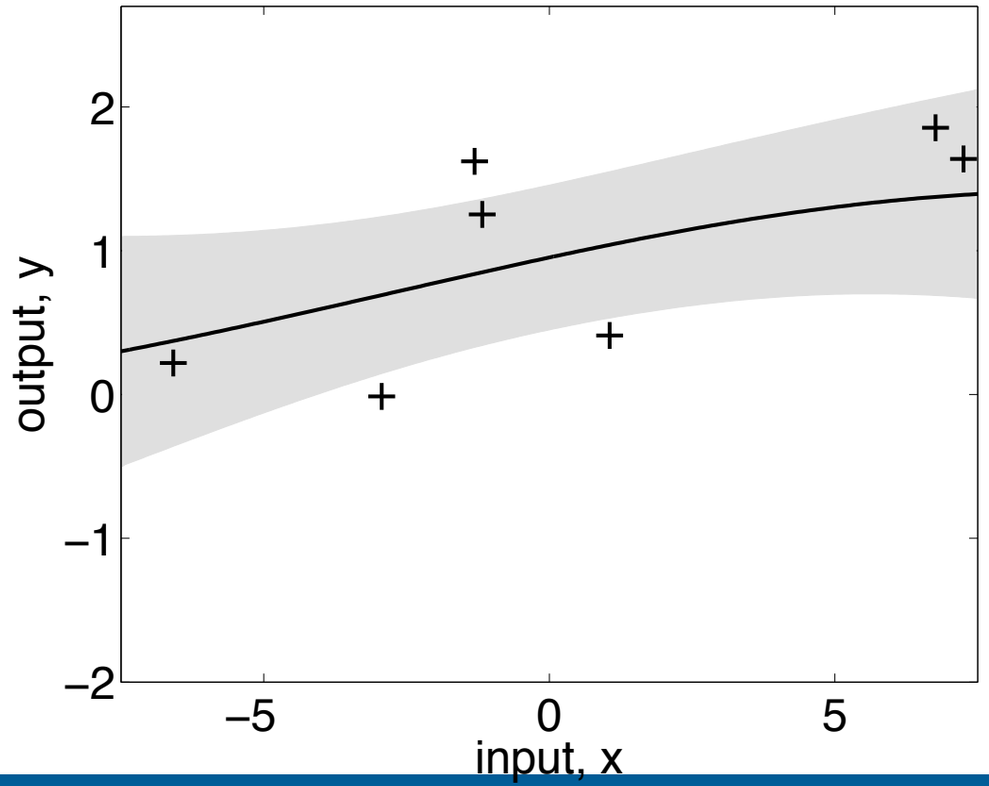
Estimating the Hyperparameters





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

The local maxima can correspond to sub-optimal solutions.



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:

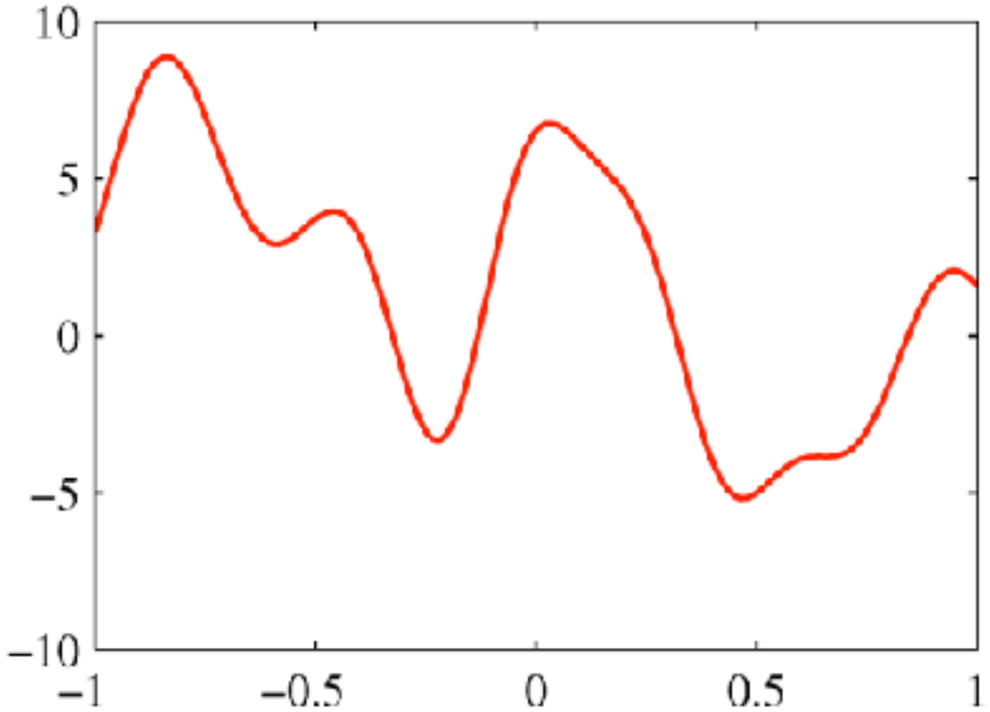
$$p(y = +1 \mid \mathbf{x}) = \sigma(f(\mathbf{x}))$$

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

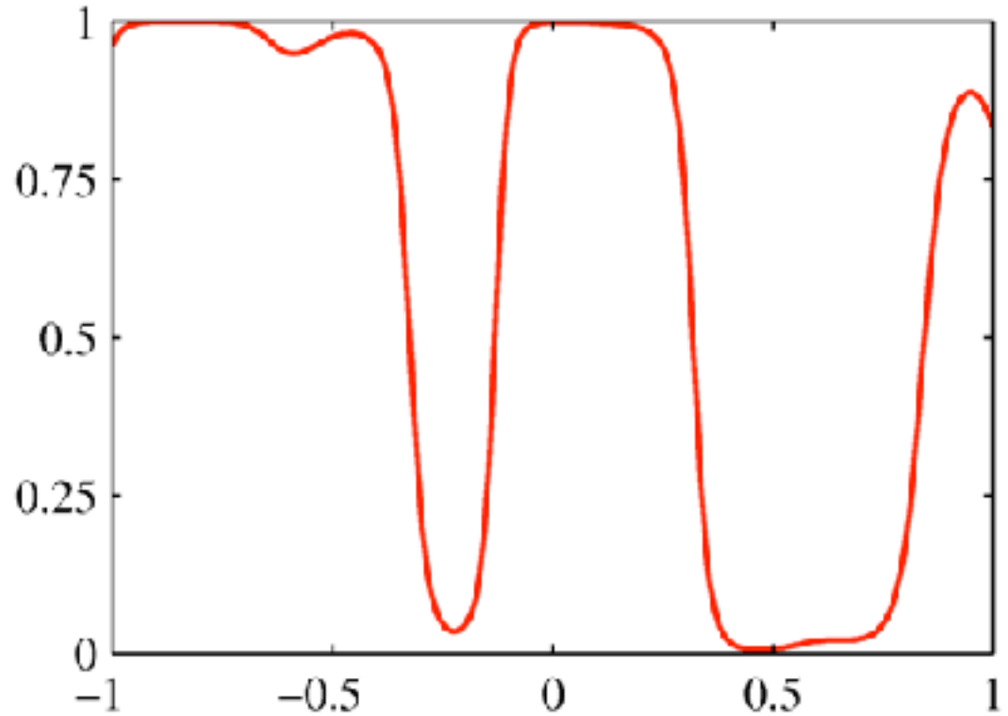
A typical type of sigmoid function is the logistic sigmoid:

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

Application of the Sigmoid Function



Function sampled from a Gaussian Process

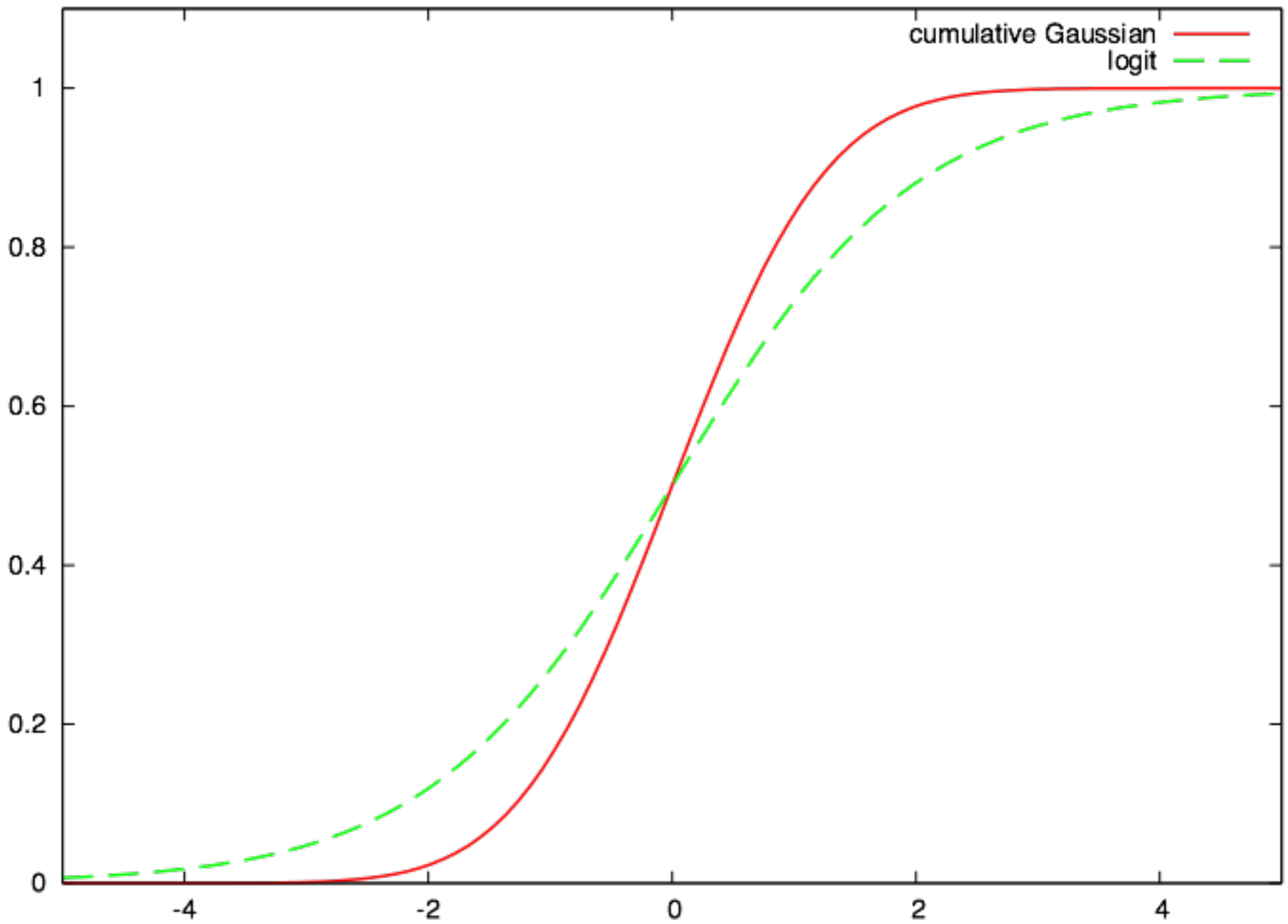


Sigmoid function applied to the GP function

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:

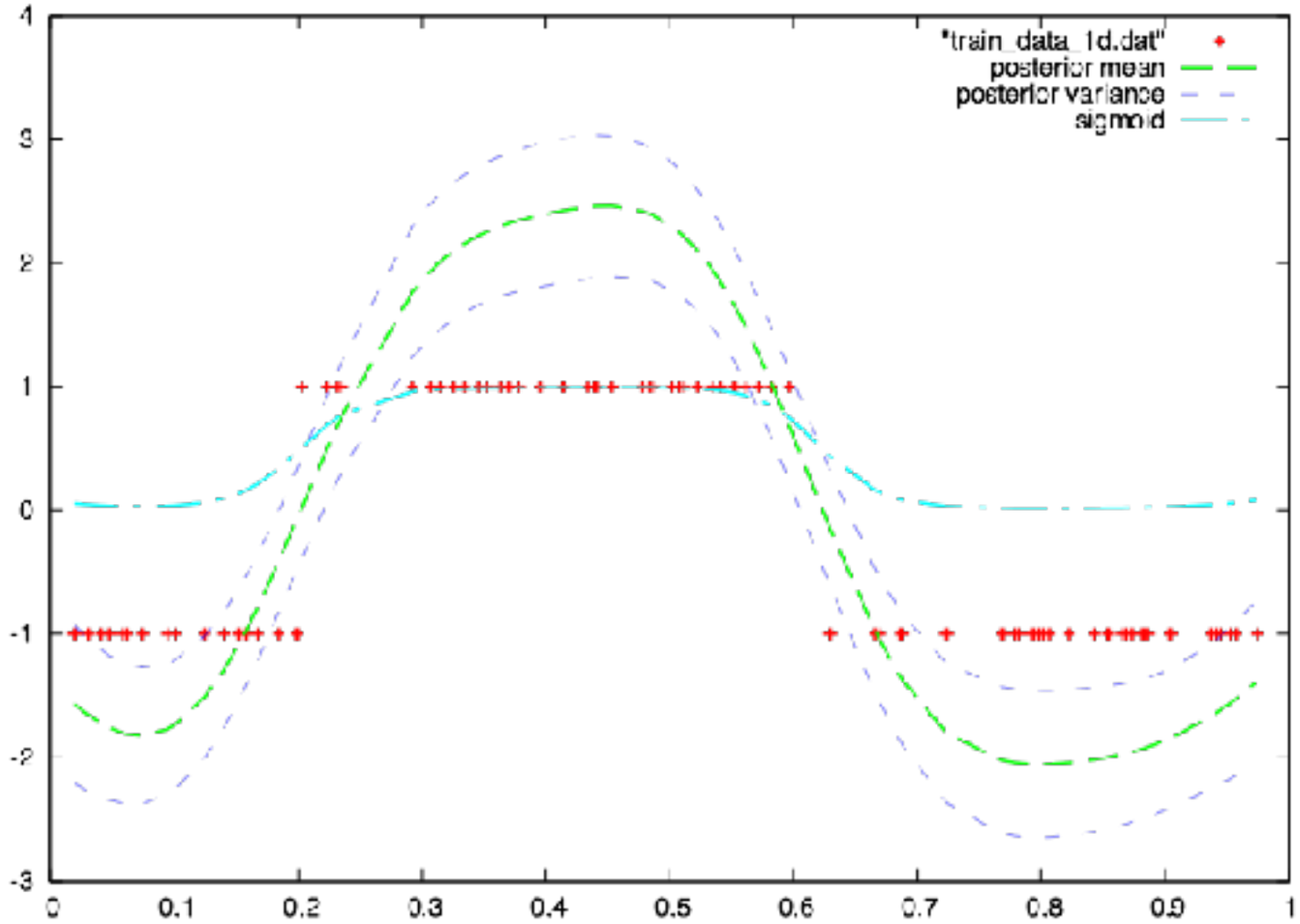
$$p(\mathbf{f} \mid X, \mathbf{y}) = \frac{p(\mathbf{y} \mid \mathbf{f}) p(\mathbf{f} \mid X)}{p(\mathbf{y} \mid X)}$$

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

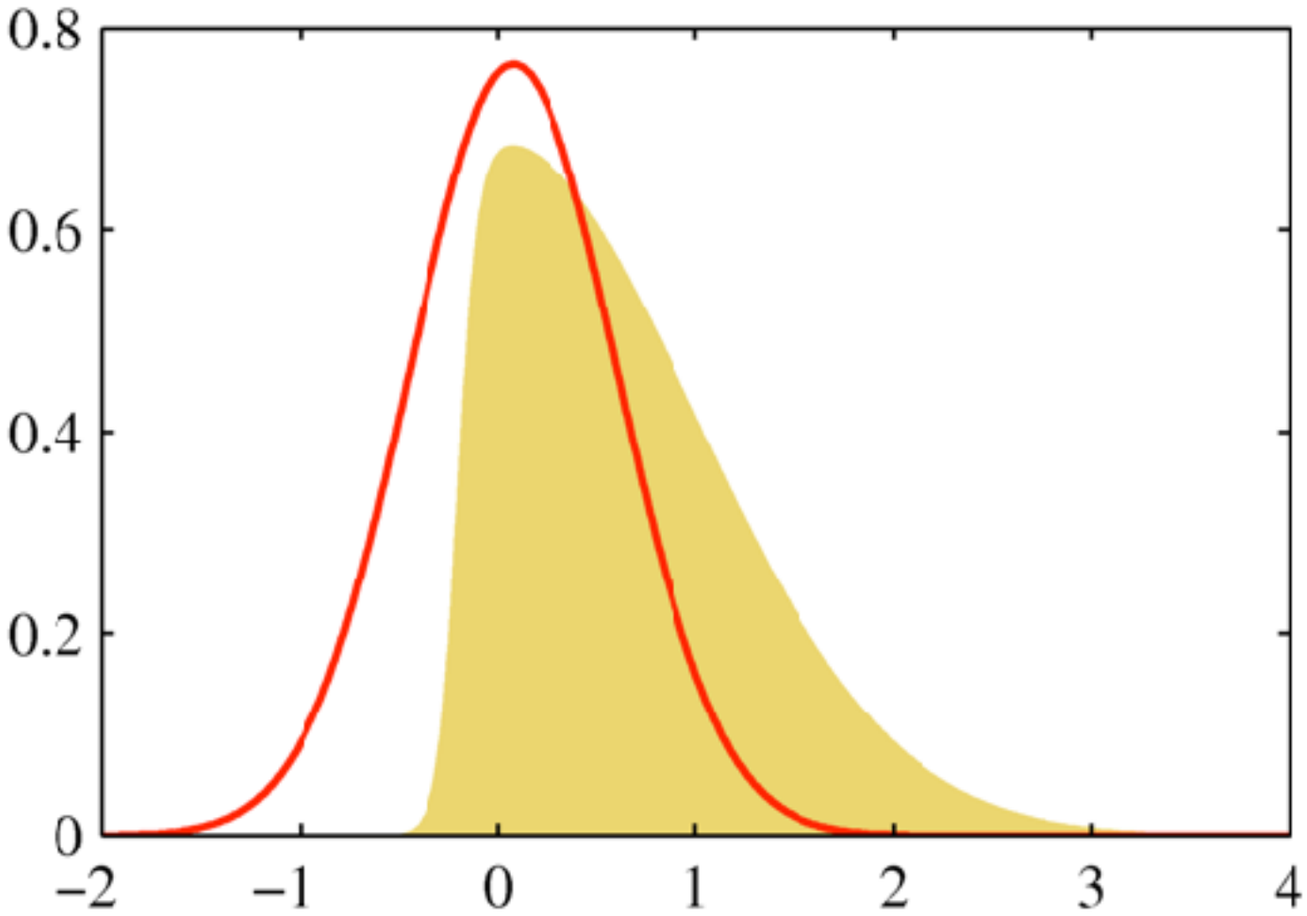
To compute $\hat{\mathbf{f}}$ an iterative approach using Newton's method has to be used.

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

Applying this to Laplace

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

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

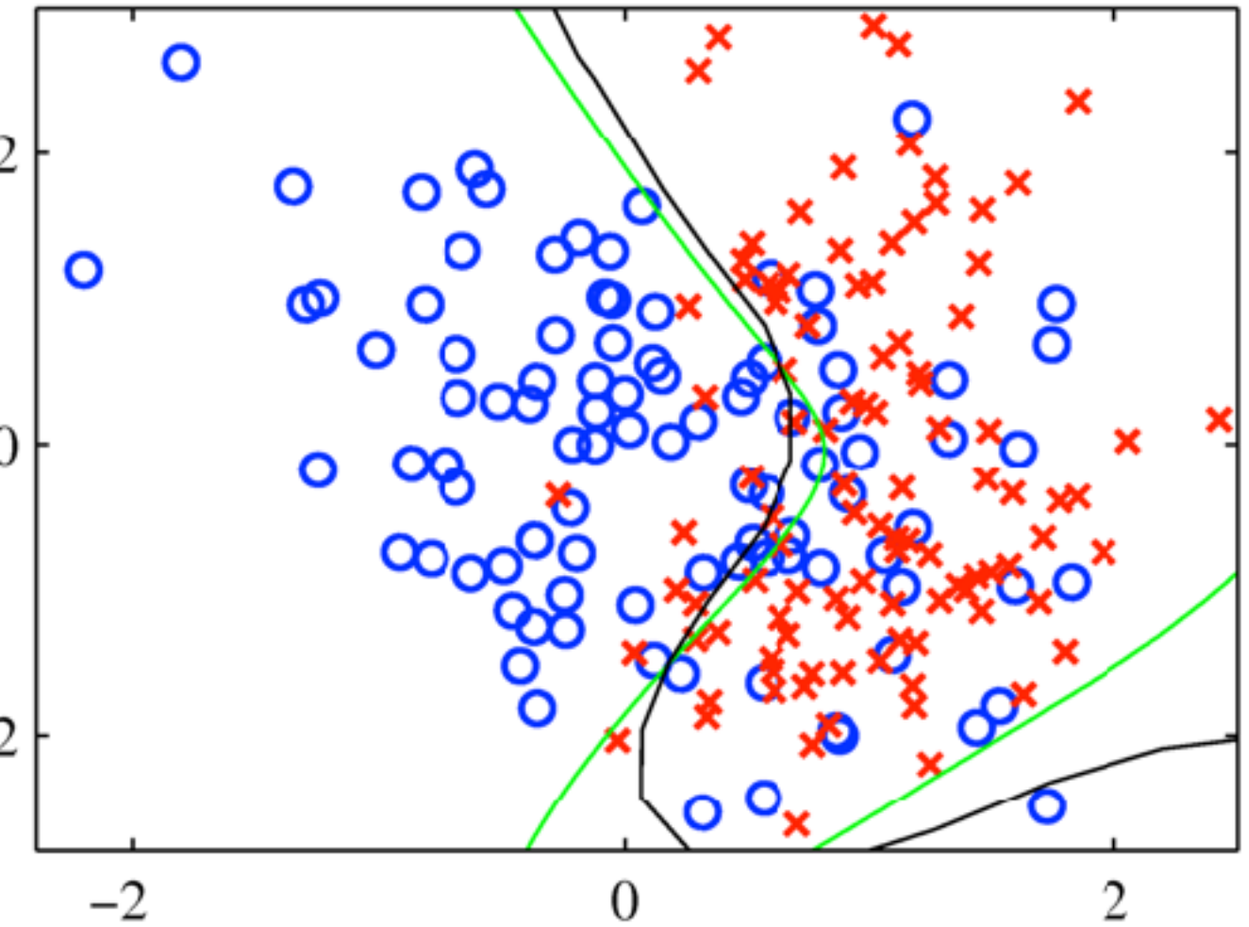
It remains to compute

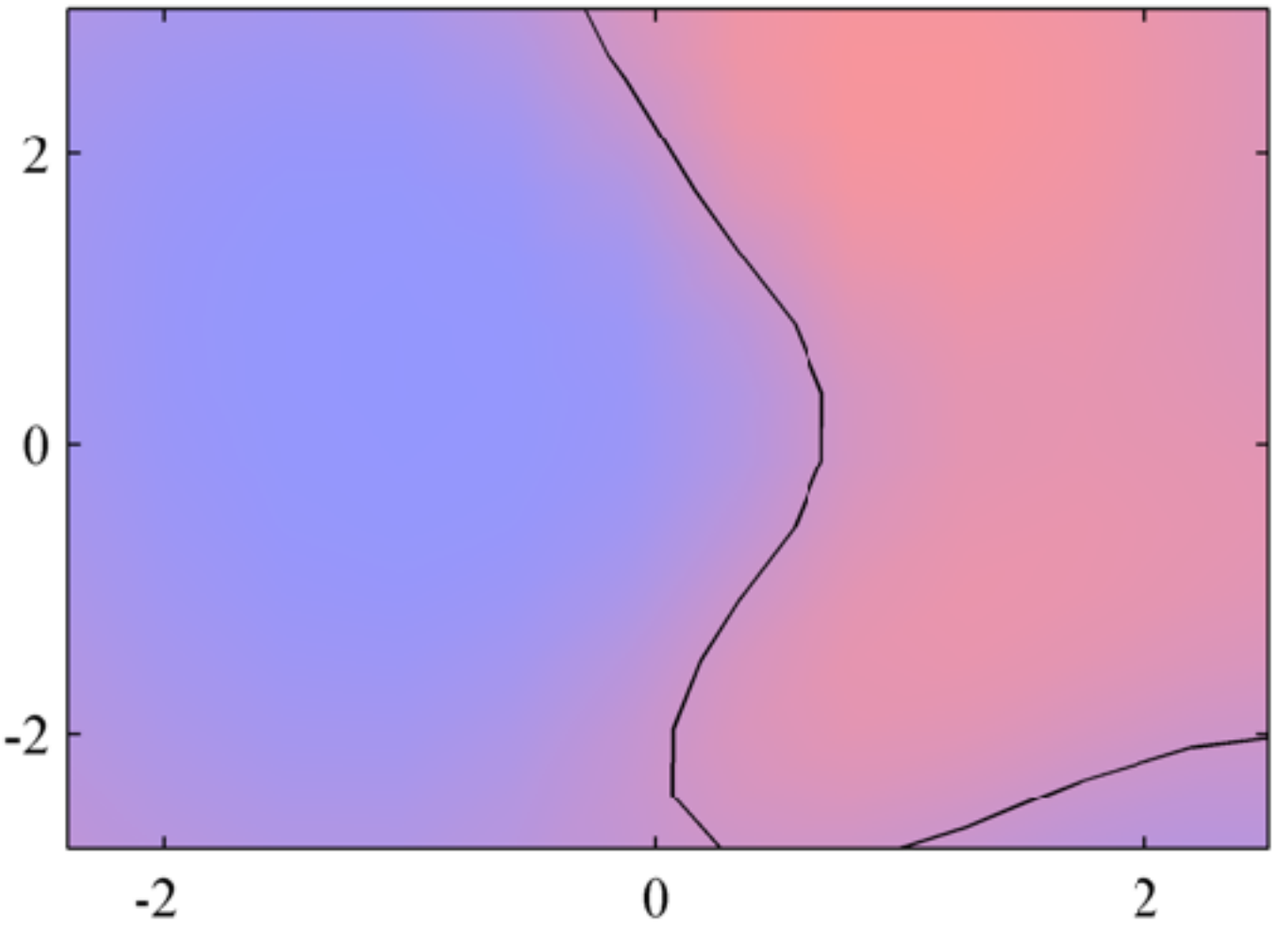
$$p(y_* = +1 \mid X, \mathbf{y}, \mathbf{x}_*) = \int p(y_* \mid f_*) p(f_* \mid X, \mathbf{y}, \mathbf{x}_*) df_*$$

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

Summary

- Kernel methods solve problems by implicitly mapping the data into a (high-dimensional) feature space
- The feature function itself is not used, instead the algorithm is expressed in terms of the kernel
- Gaussian Processes are Normal distributions over functions
- To specify a GP we need a covariance function (kernel) and a mean function
- More on Gaussian Processes: http://videolectures.net/epsrcws08_rasmussen_lgp/