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:

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

by defining:

\[
    K = \Phi\Phi^T \quad a = (\lambda I_N + K)^{-1} t
\]

we get:

\[
    y(x) = \phi(x)^T w = \phi(x)^T \Phi^T a = k(x)^T (K + \lambda I_N)^{-1} t
\]

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 $x_a$ and $x_b$ that are jointly Gaussian distributed, i.e. $\mathcal{N}(x | \mu, \Sigma)$ with

$$
\begin{align*}
\mathbf{x} &= \begin{pmatrix} x_a \\ x_b \end{pmatrix} \\
\mu &= \begin{pmatrix} \mu_a \\ \mu_b \end{pmatrix} \\
\Sigma &= \begin{pmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{pmatrix}
\end{align*}
$$

Question: how can we compute the conditional?

$$p(x_a | x_b)$$
Gaussian Marginals and Conditionals

We can also use the result for the following:
Assume we have two variables $x_a$ and $x_b$ that are jointly Gaussian distributed, i.e. $\mathcal{N}(x | \mu, \Sigma)$ with

$$
x = \begin{pmatrix} x_a \\ x_b \end{pmatrix} \quad \mu = \begin{pmatrix} \mu_a \\ \mu_b \end{pmatrix} \quad \Sigma = \begin{pmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{pmatrix}
$$

The conditional distribution is $p(x_a | x_b) = \mathcal{N}(x | \mu_{a|b}, \Sigma_{a|b})$ where

$$
\mu_{a|b} = \mu_a + \Sigma_{ab} \Sigma_{bb}^{-1} (x_b - \mu_b)
$$

and

$$
\Sigma_{a|b} = \Sigma_{aa} - \Sigma_{ab} \Sigma_{bb}^{-1} \Sigma_{ba}
$$

"Schur Complement"

The marginal is $p(x_a) = \mathcal{N}(x_a | \mu_a, \Sigma_{aa})$
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: \( m(x) = \mathbb{E}[y(x)] \)
covariance function:
\[
k(x_1, x_2) = \mathbb{E}[y(x_1) - m(x_1)y(x_2) - m(x_2)]
\]
Example

- green line: sinusoidal data source
- blue circles: data points with Gaussian noise
- red line: mean function of the Gaussian process
- shaded red area: $2\sigma$ confidence interval
How Can We Handle Infinity?

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

\[
x = \begin{pmatrix} x_f \\ x_i \end{pmatrix} \sim \mathcal{N} \left( \begin{pmatrix} \mu_f \\ \mu_i \end{pmatrix}, \begin{pmatrix} \Sigma_f & \Sigma_{fi} \\ \Sigma_{fi}^T & \Sigma_i \end{pmatrix} \right)
\]

From the marginalization property we get:

\[
p(x_f) = \int p(x_f, x_i) \, dx_i = \mathcal{N}(x_f \mid \mu_f, \Sigma_f)
\]

This means we can use finite vectors.
A Simple Example

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

\[
E[y(x)] = \phi(x)^T E[w] = 0
\]

\[
E[y(x_1)y(x_2))] = \phi(x_1)^T E[ww^T] \phi(x_2) = \phi(x_1)^T \Sigma_p \phi(x_2)
\]

Any number of function values \( y(x_1), \ldots, y(x_N) \) is jointly Gaussian with zero mean.

The covariance function of this process is

\[
k(x_1, x_2) = \phi(x_1)^T \Sigma_p \phi(x_2)
\]

In general, any valid kernel function can be used.
The Covariance Function

The most used covariance function (kernel) is:

\[ k(x_p, x_q) = \sigma_f^2 \exp\left(-\frac{1}{2l^2}(x_p - x_q)^2\right) + \sigma_n^2 \delta_{pq} \]

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

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

\[ k(x_p, x_q) = \exp\left(-\theta|x_p - x_q|\right) \]

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 $x_1^*, \ldots, x_M^*$
2. Compute the covariance matrix $K$ where
   $$K_{ij} = k(x_i^*, x_j^*)$$
3. Generate a random Gaussian vector from
   $$y^* \sim \mathcal{N}(0, K)$$
4. Plot the values $x_1^*, \ldots, x_M^*$ 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 $x_1, \ldots, x_N$  $t_1, \ldots, t_N$
- test input $x_1^*, \ldots, x_M^*$

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

The joint probability is

$$
\begin{pmatrix}
  y \\
  y_*
\end{pmatrix} \sim \mathcal{N}
\begin{pmatrix}
  0, \\
  K(X, X^*)
\end{pmatrix}
$$

and we need to compute $p(y^* | x^*, X, y)$. 
Prediction with a Gaussian Process

In the case of only one test point \( x^* \) we have

\[
K(X, x^*) = \begin{pmatrix}
k(x_1, x^*) \\
\vdots \\
k(x_N, x^*)
\end{pmatrix} = k_*
\]

Now we compute the conditional distribution

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

where

\[
\mu_* = k^T_* K^{-1} y
\]

\[
\Sigma_* = k(x_*, x_*) - k^T_* K^{-1} k_*
\]

This defines the \textbf{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:

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

• We assume that the noise is Gaussian:

\[
p(y \mid f) = \mathcal{N}(f, \sigma_y^2 I)
\]
Varying the Hyperparameters

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

\[ l = \sigma_f = 1, \quad \sigma_n = 0.1 \]

\[ l = 0.3, \quad \sigma_f = 1.08, \quad \sigma_n = 0.0005 \]

\[ l = 3, \quad \sigma_f = 1.16, \quad \sigma_n = 0.89 \]
Varying the Hyperparameters

The squared exponential covariance function can be generalized to

\[ k(x_p, x_q) = \sigma_f^2 \exp\left(-\frac{1}{2}(x_p - x_q)^T M (x_p - x_q)\right) + \sigma_n^2 \delta_{pq} \]

where \( M \) can be:

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

\[ M = I \]

\[ M = \text{diag}(1, 3)^{-2} \]

\[ M = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} + \text{diag}(6, 6)^{-2} \]
Implementation

Algorithm 1: GP regression

Data: training data \((X, y)\), test data \(x_*\)

Input: Hyper parameters \(\sigma_f^2, l, \sigma_n^2\)

\[ K_{ij} \leftarrow k(x_i, x_j) \]
\[ L \leftarrow \text{cholesky}(K + \sigma_y^2 I) \]
\[ \alpha \leftarrow L^T(L\backslash y) \]
\[ \mathbb{E}[f_*] \leftarrow k^T_*\alpha \]
\[ \mathbf{v} \leftarrow L\backslash k_* \]
\[ \text{var}[f_*] \leftarrow k(x_*, x_*) - \mathbf{v}^T\mathbf{v} \]
\[ \log p(y \mid X) \leftarrow -\frac{1}{2}y^T\alpha - \sum_i \log L_{ii} - \frac{N}{2} \log(2\pi) \]

- 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:

\[ p(y \mid X) = \int p(y \mid f, X)p(f \mid X)df \]

\[ p(y \mid f) = \prod_i \mathcal{N}(y_i \mid f_i, \sigma_y^2) \quad p(f \mid X) = \mathcal{N}(f \mid 0, K) \]
Estimating the Hyperparameters

To find optimal hyperparameters we need the marginal likelihood:

\[ p(y \mid X) = \int p(y \mid f, X)p(f \mid X)df \]

\[ p(y \mid f) = \prod_i \mathcal{N}(y_i \mid f_i, \sigma_y^2) \quad \quad \quad \quad \quad \quad p(f \mid X) = \mathcal{N}(f \mid 0, K) \]

This results in

\[ p(y \mid X) = \mathcal{N}(y \mid 0, K + \sigma_y^2 I) \]
Estimating the Hyperparameters

To find optimal hyper parameters we need the marginal likelihood:

\[ p(y \mid X) = \int p(y \mid f, X)p(f \mid X)df \]

\[ p(y \mid f) = \prod_{i} \mathcal{N}(y_i \mid f_i, \sigma_y^2) \]
\[ p(f \mid X) = \mathcal{N}(f \mid 0, K) \]

This results in

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

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

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

• We can interpret the $\eta_i$ 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
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:

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

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

A typical type of sigmoid function is the logistic sigmoid:

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$
Another symmetric sigmoid function is the cumulative Gaussian:

\[ \Phi(z) = \int_{-\infty}^{z} \mathcal{N}(x \mid 0, 1) \, dx \]
The cumulative Gaussian is slightly steeper than the logistic sigmoid.
The Latent Variables

In regression, we directly estimated $f$ as

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

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

\[ p(y_\ast = +1 \mid X, y, x_\ast) = \int p(y_\ast \mid f_\ast) p(f_\ast \mid X, y, x_\ast) df_\ast \]

\[ \sigma(f_\ast) \]
Class Prediction with a GP

The aim is to compute the predictive distribution

\[ p(y_\ast = +1 \mid X, y, x_\ast) = \int p(y_\ast \mid f_\ast)p(f_\ast \mid X, y, x_\ast) df_\ast \]

we marginalize over the latent variables from the training data:

\[ p(f_\ast \mid X, y, x_\ast) = \int p(f_\ast \mid X, x_\ast, f)p(f \mid X, y) df \]

predictive distribution of the latent variable (from regression)
Class Prediction with a GP

The aim is to compute the predictive distribution

\[ p(y_\ast = +1 \mid X, y, x_\ast) = \int p(y_\ast \mid f_\ast) p(f_\ast \mid X, y, x_\ast) df_\ast \]

we marginalize over the latent variables from the
training data:

\[ p(f_\ast \mid X, y, x_\ast) = \int p(f_\ast \mid X, x_\ast, f)p(f \mid X, y) df \]

we need the posterior over the latent variables:

\[ p(f \mid X, y) = \frac{p(y \mid f)p(f \mid X)}{p(y \mid X)} \]

likelihood (sigmoid) prior normalizer
A Simple Example

• Red: Two-class training data
• Green: mean function of $p(f \mid X, y)$
• Light blue: sigmoid of the mean function
But There Is A Problem...

\[ p(f \mid X, y) = \frac{p(y \mid f)p(f \mid X)}{p(y \mid X)} \]

- 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

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

where \( \hat{f} = \arg \max_f p(f \mid X, y) \)

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

To compute \( \hat{f} \) an iterative approach using Newton’s method has to be used.

The Hessian matrix \( A \) can be computed as

\[ A = K^{-1} + W \]

where \( W = -\nabla \nabla \log p(y \mid f) \) is a diagonal matrix which depends on the sigmoid function.
• 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 \( p(f \mid X, y) \) we can compute:

\[
p(f^* \mid X, y, x_*) = \int p(f^* \mid X, x^*, f)p(f \mid X, y)df
\]

From the regression case we have:

\[
p(f^* \mid X, x^*, f) = \mathcal{N}(f^* \mid \mu^*, \Sigma^*)
\]

where \( \mu^* = \mathbf{k}^T K^{-1}f \) \hspace{1cm} \( \Sigma^* = k(x^*, x^*) - \mathbf{k}^T K^{-1} \mathbf{k} \)

This reminds us of a property of Gaussians that we saw earlier!
Gaussian Properties (Rep.)

If we are given this:

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

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

Then it follows (properties of Gaussians):

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

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

where

\[
\Sigma = (\Sigma_1^{-1} + A^T\Sigma_2^{-1}A)^{-1}
\]
Applying this to Laplace

\[ \mathbb{E}[f_* \mid X, y, x_*] = k(x_*)^T K^{-1} \hat{f} \]
\[ \nabla[f_* \mid X, y, x_*] = k(x_*, x_*) - k_*^T (K + W^{-1})^{-1} k_* \]

It remains to compute

\[ p(y_* = +1 \mid X, y, x_*) = \int p(y_* \mid f_*) p(f_* \mid X, y, x_*) df_* \]

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

We can also use the result for the following:
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:
  [Link](http://videolectures.net/epsrcws08_rasmussen_lgp/)