NTB
Interstaatliche Hochschule
für Technik Buchs

FHO Fachhochschule Ostschweiz

Lab 9b

TSM MachLe MSE
FS 2019

Gaussian Processes

Machine Learning
WÜRC

Essentially, all models are wrong, but some are useful.
George E.T. Box

After this unit, ...

Lernziele/Kompetenzen

- you know the principle of maximum likelihood (`ML`) and and maximum a posteriori probability (`MAP`) and you know their difference.
- you know (K1), that both the *conditionals* $p(x|y)$ and the *marginals* $p(x)$ of a joint Gaussian $p(x, y)$ are again Gaussian.
- you know (K1) that a *Gaussian process* $\mathcal{GP}(\mu, k)$ is a generalization of a multivariate Gaussian distribution to infinitely many variables. A Gaussian process is a *prior* over *functions* $p(f)$ which can be used for Bayesian regression. Sampling from a Gaussian process means sampling *functions* (instead of samples of a random variable) out of a pool of functions characterized by a mean function μ and a covariance function $k(x, x')$.
- you know (K1), that every model relies on (explicit or implicit) *assumptions*. We discriminate *knowledge*, *assumptions* and *simplifying assumptions*. In Bayesian reasoning, assumptions are formulated as *prior distribution* $p(\theta)$ over the parameters θ of a model. Using Bayes rule, one can calculate the posterior parameter distribution $p(\theta|x, y)$ given the data (x, y) and the model assumptions.

$$\text{posterior} = p(\theta|x, y) = \frac{p(y|x, \theta) \cdot p(\theta)}{\int_\theta p(y|x, \theta) \cdot p(\theta) d\theta} = \frac{\text{likelihood} \cdot \text{prior}}{\text{marginal}} \tag{1}$$

- your are able to formulate *probabilitstic models* that use *priors* to express knowlege (or beliefs) about aspects of the model. You can formulate a *probabilistic model* for a process $f(x, \theta)$ with additive Gaussian noise ε. You can derive the *likelihood function* $p(y|x, \varepsilon, \theta)$ for this model given the parameters θ.
- you are able (K3) to *sample functions* from a Gaussian Process $\mathcal{GP}(\mu, k)$ with given mean $\mu(x)$ and covariance function $k(x, x')$ using the `GaussianProcessRegressor` of the class `sklearn.gaussian_process`.

- you are able (K3) to *fit* n-dimensional data using a Gaussian Process, i.e. you are able to *infer* hyperparameters of the model from given data using the `GaussianProcessRegressor` of the class `sklearn.gaussian_process`.
- you are able (K3) to *make predictions* using the `GaussianProcessRegressor` of the class `sklearn.gaussian_process`.
- you know (K1) that the *predictive distribution* which is used for making predictions for unknown data (x^*, y^*) can be calculated by *marginalizing* (integrating or averaging) over the parameter distribution.

$$p(y^*|x^*, x, y) = \int_\theta p(y^*|x^*, x, y, \theta) \cdot p(\theta|x, y) d\theta \tag{2}$$

- you know (K1) the most important covariance functions (kernels) $k(x, x')$, namely the *constant* kernel, the *Gaussian* kernel, the *RBF*-kernel (radial basis function), the *Dot-Product* kernel and the *sine-exponential* kernel.
- you are able (K3) to apply *kernel operations* (namely sum and product) in order to construct a probabilistic model adapted to a given dataset.

---

1. Medical Inference (Bayes Theorem) [M,I]

Breast cancer facts:

- 1 % of scanned women have breast cancer
- 80 % of women with breast cancer get positive mammography scans
- 9.6 % of women without breast cancer also get positive mammography scans

Question: A woman gets a scan, and it is positive. what is the probability that she has breast cancer?

Welche der folgenden Aussagen sind wahr und welche falsch?	**wahr**	**falsch**
a) less than 1 %	☐	☐
b) less than 10 %	☐	☐
c) around 80 %	☐	☐
d) around 90 %	☐	☐

2. Likelihood function, MAP and linear regression [L, II]

Assume a linear model $y = \theta_1 + \theta_2 \cdot x$ for observed data ppoints $\mathbf{X}, \mathbf{Y} = \{x_i, y_i\}$ $(i = 1 \dots N)$ superposed by Gaussian noise ε with zero mean value and variance σ_n^2. The parameters $\theta = \{\theta_1, \theta_2\}$ of the model are the offset θ_1 and the slope θ_2 and

$$\begin{aligned} y &= f(x|\theta) + \varepsilon \\ y &= \theta_1 + \theta_2 \cdot x + \varepsilon \\ \varepsilon &\sim \mathcal{N}(0, \sigma_n^2) \end{aligned}$$

a) Write down the *likelihood function* $p(\mathbf{Y}|\mathbf{X}, \theta)$.

b) Show that the *log-likelihood* takes the form of the sum of the squared errors $\mathrm{SSE}(\theta)$ between the model and the data points.

$$\begin{aligned} \log\left[p\left(\mathbf{Y}|\mathbf{X}, \theta\right)\right] &= -\frac{1}{2\sigma_n^2} \sum_{i=1}^{N} \left(y_i - f(x_i|\theta)\right)^2 - \frac{N}{2} \cdot \log\left(2\pi\sigma_n^2\right) && (3) \\ &= -\frac{1}{2\sigma_n^2} \left\|\mathbf{Y} - f(\mathbf{X}|\theta)\right\|^2 - \frac{N}{2} \cdot \log\left(2\pi\sigma_n^2\right) && (4) \\ &= -\frac{1}{2\sigma_n^2} \mathrm{SSE}(\theta) - \frac{N}{2} \cdot \log\left(2\pi\sigma_n^2\right) && (5) \end{aligned}$$

c) Show that in the case of a *linear* model in the parameter θ, we can write the sum of squared error $\mathrm{SSE}(\theta)$ in matrix form:

$$\begin{aligned} \mathrm{SSE}(\theta) &= \left(\mathbf{Y} - f(\mathbf{X}|\theta)\right)^T \cdot \left(\mathbf{Y} - f(\mathbf{X}|\theta)\right) \\ &= \left(\mathbf{Y} - \Phi \cdot \theta\right)^T \cdot \left(\mathbf{Y} - \Phi \cdot \theta\right) \end{aligned}$$

where

$$\theta = \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix} \qquad \Phi = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{pmatrix} = \left(\mathbb{1}_N \, \mathbf{X}\right) \qquad (6)$$

d) To calculate the model parameters $\hat{\theta}_{\mathrm{ML}}$ that maximize the likelihood function, we could set the derivative of the likelihood function $p(\mathbf{Y}|\mathbf{X}, \theta)$ with respect to θ to zero and solve for θ. But instead of calculating the derivative of the likelihood function, it is easier to calculate the derivative of the log-likelihood function. The logarithm is monotonous, so the postitions of the extrema are not changed. Since the loglikelihood is proporitonal to the negative sum of squared errors, $\mathrm{SSE}(\theta)$, we can also look for a *minimum* in $\mathrm{SSE}(\theta)$.

Show that minimizing $\mathrm{SSE}(\theta)$ leads - in case of a linear model - to the normal equations:

$$\hat{\theta}_{\mathrm{ML}} = \left[\Phi^T \Phi\right]^{-1} \Phi^T \mathbf{Y} \qquad (7)$$

3. Prior samples and posterior distributions from differnt kernels of a $\mathcal{GP}$ [A,II]

Open the Jupyter notebook `plot_gpr_prior_posterior.jpynb` and execute the first cell. The goal of this exercise is that you get familiar with the Gaussian process class `GaussianProcessRegressor` and the implemented `kernels` from `sckit-learn`.

a) Analyze the code and try to understand what is done at each line. At which line is the `posterior distribution` calculated for the given evidence (data). What is the prior used for calculating the posterior?

b) Have a look at the samples drawn from the given Gaussian processes for each kernel: `RBF, Matern, RationalQuadratic, ExpSineSquared, DotProduct and ConstantKernel`. Give an example for data that could be described by these covariance functions.
Hint:

- Check the kernel cookbook from `http://www.cs.toronto.edu/~duvenaud/cookbook/index.html`.
- Interested readers may have a look at the paper of Zoubin Ghahramani at `http://www.cs.toronto.edu/~duvenaud/cookbook/index.html`,
- or chapter 4 of C.E Rasmussens book available at `http://www.gaussianprocess.org/gpml/chapters/RW4.pdf`.

c) How would you model a periodic process with noise?

d) How would you set up a model for a polynomial regression using a Gaussian process?

e) For which cases would you use a Matérn kernel?

4. Model fitting, prediction and noise estimation using a $\mathcal{GP}$ [A,II]

Open the Jupyter notebook `FitGPModel_NoiseEstimation.jpynb` and execute the first cell. We generate a synthetic dataset of 300 samples that represent a noisy sine wave. The goal of this problem is to find an appropriate model for the data, to estimate the hyperparameters of the model using a Gaussian process, to make predictions and to get an estimate of the noise level in the data.

a) Plot the data in a scatter plot using `matplotlib.pyplot`. Try to estimate the noise level and the periodicity of the data. This will be used as staring point for the optimization of the *hyperparameters*.

```
nSamples=300;
rng = np.random.RandomState(0)
X = rng.uniform(0, 5, nSamples)[:, np.newaxis]
y = 0.5 * np.sin(3 * X[:, 0]) + rng.normal(0, 0.3, X.shape[0])
```

b) Generate a `kernel` for the covariance function of the Gaussian process as the sum of a `RBF`-kernel and a `WhiteKernel`. Import the *kernels* from `sklearn.gaussian_process`. Select appropriate length scales, noise levels and the corresponding lower and upper bounds for the optimizer.

```
from sklearn.gaussian_process import kernels
kernel = 1.0 * kernels.RBF(length_scale=..., length_scale_bounds=(...,
                                                    ...)) \
+ kernels.WhiteKernel(noise_level=..., noise_level_bounds=(..., ...))
```

c) Generate a Gaussian process `gp` as instance of the class `GaussianProcessRegressor` using the above `kernel` and fit the model to the data using the `.fit(X,y)` method.

```
gp = GaussianProcessRegressor(kernel=kernel,alpha=1e-5).fit(X, y)
```

d) Make *predictions* with the infered paramers of the model within the interval $X^* = [0, 10]$ using the `.predict` method of the `GaussianProcessRegressor`.
Set the option `return_cov=True` to get the covariance matrix.

```
X_ = np.linspace(0, 10, 100)
y_mean, y_cov = gp.predict(X_[:, np.newaxis], return_cov=True)
```

e) Plot the mean of the estimated model in the interval $X^* = [0, 10]$ using the `y_mean` output of the `.predict method`. Plot the confidence interval as $\pm$ one standard deviation from the mean value. The standard deviation can be obtained from `y_cov` by `stdv=np.sqrt(np.diag(y_cov))`. If you like, you can use `plt.fill_between` to paint the range within the standard deviation in gray. Explain how the model behaves in the interval where there were no data points available. Check and print the values of the *fitted hyperparameters*, especially the estimated noise level.

```
plt.fill_between(X_, y_mean - np.sqrt(np.diag(y_cov)),
y_mean + np.sqrt(np.diag(y_cov)),
alpha=0.5, color='k')
```

f) Repeat now the same process using the `ExpSineSquared kernel`. It has an additional parameter called `periodicity` that you have to specify including the lower and upper bounds.

```
kernel = 0.5 * kernels.ExpSineSquared(length_scale=..., periodicity=
                                      ...,
length_scale_bounds=(..., ...),
periodicity_bounds=(..., ...)) \
+ kernels.WhiteKernel(noise_level=..., noise_level_bounds=(..., ...))
```

Repeat the above steps fro b) to e) using this kernel.