Lab 9

FTP MachLe MSE
HS 2024

Machine Learning
WÜRC

Gaussian Processes

Essentially, all models are wrong, but some are useful.
GEORGE E.T. BOX

After this unit, ...

Lernziele/Kompetenzen

- you have repeated the *basic rules of probability theory.*
- you know the difference between a *joint* and a *conditional probability* distribution.
- you know how to apply *Bayes Theorem* to calculate the *posterior* probability distribution for simple discrete examples. You can name the *prior* probability distribution, the *likelihood function*, the *evidence*, and you know how to *marginalize* over a joint probability distribution.
- 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.

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

- you know the basic properties of a *multivariate Gaussian* probability distribution. You can plot a 2D Gaussian probability distribution given the *mean vector* $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$.
- you can explain the *naïve Bayes classifier* to your classmates and to your teacher.
- you know (K1), that both the *conditionals* $p(x|y)$ and the *marginals* $p(x)$ of a joint Gaussian distribution $p(x, y)$ 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 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) 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. Hamburger and Bayes Rule [A, II]

Consider the following fictitious scientic information: Doctors find that people with Kreuzfeld-Jacob disease (KJ) almost invariably ate hamburgers (Hamburger Eater, HE) , thus $p(\mathsf{HE}|\mathsf{KJ}) = 0.9$. The probability of an individual having KJ is currently rather low, about one in $100'000$.

a) Assuming eating lots of hamburgers is rather widespread, say $p(\mathsf{HE}) = 0.5$, what is the probability that a hamburger eater will have Kreuzfeld-Jacob disease? Determine the prior, the likelihood function and the posterior probability.

b) If the fraction of people eating hamburgers was rather small, $p(\mathsf{HE}) = 0,001$, what is the probability that a regular hamburger eater will have Kreuzfeld-Jacob disease?

2. Naïve Bayes Classifier [A, II]

In order to reduce my email load, I decide to implement a machine learning algorithm to decide whether or not I should read an email, or simply file it away instead. To train my model, I obtain the following data set of binary-valued *features* about each email, including whether I know the author or not, whether the email is long or short, and whether it has any of several key words, along with my final decision about whether to read it ($y = +1$ for 'read', $y = -1$ for 'discard').

x_1	x_2	x_3	x_4	x_5	y
know `author`?	is `long`?	has `research`	has `grade`	has `lottery`	$\Longrightarrow$ read?
0	0	1	1	0	-1
1	1	0	1	0	-1
0	1	1	1	1	-1
1	1	1	1	0	-1
0	1	0	0	0	-1
1	0	1	1	1	+1
0	0	1	0	0	+1
1	0	0	0	0	+1
1	0	1	1	0	+1
1	1	1	1	1	-1

a) Compute all the probabilities necessary for a naïve Bayes classifier, i.e. the class probability $p(y)$ and all the individual feature probabilities $p(x_i|y)$, for each class y and feature x_i.

b) Which class would be predicted for $x = \{00000\}$?
What about for $x = \{11010\}$?

c) Compute the *posterior probability* that $y = +1$ given the observation $x = \{00000\}$. Also compute the posterior probability that $y = +1$ given the observation $x = \{11010\}$.

d) Why should we probably not use a *'joint' Bayes classifier* (using the joint probability of the features x, as opposed to the conditional independencies assumed by naïve Bayes) for these data?

e) Suppose that before we make our predictions, we lose access to my address book, so that we cannot tell whether the email author is known. Do we need to re-train the model to classify based solely on the other four features? If so, how? *Hint:* How do the parameters of a naïve Bayes model over only features $x_2, \ldots, x_5$ differ?

3. Weather in London [A,II]

The weather in London can be summarised as: if it rains one day there's a 70% chance it will rain the following day; if it's sunny one day there's a 40% chance it will be sunny the following day.

$$p(\textsf{today} = rain \mid \textsf{yesterday} = rain) = 70\%$$

$$p(\textsf{today} = sun \mid \textsf{yesterday} = sun) = 40\%$$

From these likelihoods, we can *infer* the following:

$$p(\textsf{today} = sun \mid \textsf{yesterday} = rain) = 30\%$$

$$p(\textsf{today} = rain \mid \textsf{yesterday} = sun) = 60\%$$

a) Assuming that the prior probability it rained yesterday is 0.5, what is the probability that it was raining yesterday given that it's sunny today?

b) If the weather follows the same pattern as above, day after day, what is the probability that it will rain on any day (based on an effectively infinte number of days of observing the weather)?

c) Use the result from b) above as a new prior probability of rain yesterday and recompute the proba- bility that it was raining yesterday given that it's sunny today.

4. Bivariate Gaussian Distribution [A, II]

The probability density function (pdf) of a multivariate normal with $\mathbf{x} = \begin{pmatrix} x_a \\ x_b \end{pmatrix}$ and $\boldsymbol{\mu} = \begin{pmatrix} \mu_a \\ \mu_b \end{pmatrix}$ is given by:

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}\sqrt{\det(\boldsymbol{\Sigma})}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}$$

Consider a bivariate normal distribution $p(\mathbf{x}) = p(x_a, x_b)$ with $\mu_a = 0$, $\mu_b = 2$, $\Sigma_{aa} = 2$, $\Sigma_{bb} = 1$ and $\Sigma_{ab} = \Sigma_{ba} = \frac{\sqrt{2}}{2}$.

a) Calculate the *precision matrix* $\boldsymbol{\Lambda}$, the *inverse* $\boldsymbol{\Sigma}^{-1}$ of the covariance matrix $\boldsymbol{\Sigma}$.

b) Write out the squared generalized distance expression, the *Mahalanobis distance*

$$\boldsymbol{\Delta} = (\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu}) \tag{2}$$

as a function of x_a and x_b.

c) Write out the *bivariate normal density* $p(\mathbf{x}) = p(x_a, x_b)$ (joint probability density).

d) Calculate the *eigenvalues* $\lambda_{1,2}$ and the *eigenvectors* $\mathbf{u}_{1,2}$ of the covariance matrix $\boldsymbol{\Sigma}$ using Python (`numpy.linalg.eig`).

e) Plot the joint probability distribution $p(x_a, x_b)$ using Python (`scipy.stats.multivariate_normal`) and sample $N = 10'000$ points from this distribution.

5. 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?

6. 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 from b) to e) using this kernel.