TP 2 – AOS1

Bayesian linear regression, Gaussian process regression partial solution

1 Introduction

This practical session aims at applying Bayesian linear regression in a first step, and Gaussian processes in a second step. You will need several libraries for this purpose.

```
import numpy as np
import scipy as sp
import scipy.stats as spst
import matplotlib.pyplot as plt
```

2 Bayesian linear regression

2.1 Practice

1 Create a function for generating synthetic outputs according to a sample of inputs and a user-defined model (i.e. a given functional relation). Generate training and test datasets.

```
def model(x, sign=1):
    f = 3 + 2*x
    y = f + spst.norm(0,sign).rvs(size=x.shape[0])
    return y

# training data
Xtr = spst.uniform(-5,10).rvs(size=5)
ytr = model(Xtr)
# prediction (test) data
Xpr = spst.uniform(-5,10).rvs(size=1)
ypr = model(Xpr)
```

2 Program a function which makes predictions given training data `Xtr` and `ytr`, a noise covariance matrix Σ_n, a prior matrix Σ_p, and the set of prediction (test) instances `Xpr`. Return the associated credibility intervals via vectors `ypr_inf` and `ypr_sup`.

You may use the `np.linalg.inv` function for this purpose. Optimizing the function (using, e.g., Cholesky decomposition) is not mandatory.

```
def predGLR(Xpr, Xtr, ytr, Sign, Sigp):
    Xtr = np.concatenate([np.ones(Xtr.shape), Xtr], axis=1)
    Xpr = np.concatenate([np.ones(Xpr.shape), Xpr], axis=1)
    ...

    return [ypr, ypr_cov, ypr_inf, ypr_sup]
```

```python
def predGLR(Xpr, Xtr, ytr, Sign, Sigp, alph=0.05):
    if len(Xtr.shape)==1:
        Xtr = Xtr.reshape(-1,1)
        Xpr = Xpr.reshape(-1,1)
    ytr = ytr.reshape(-1,1)
    Xtr = np.concatenate((np.ones((Xtr.shape[0],1)), Xtr), axis=1)
    Xpr = np.concatenate((np.ones((Xpr.shape[0],1)), Xpr), axis=1)

    Sinv = np.linalg.inv(Sign)
    A = Xtr.T@Sinv@Xtr + np.linalg.inv(Sigp)
    Ainv = np.linalg.inv(A)

    ypr_ave = Xpr@Ainv@Xtr.T@Sinv@ytr
    ypr_cov = np.diag(Xpr@Ainv@Xpr.T).reshape(-1,1)

    ypr_inf, ypr_sup = spst.norm.interval(1-alph/2, ypr_ave,
                                          np.sqrt(ypr_cov.reshape(len(ypr_cov),1)))

    return ypr_ave, ypr_cov, ypr_inf, ypr_sup
```

3 Use the Gaussian LR model on the generated data, for several levels of noise and several covariance priors. Represent the credibility intervals obtained using the following code.

```python
fig, ax = plt.subplots()
ax.plot(Xtr, ytr, 'k+', label='test data')
ax.plot(Xpl, ypl_ave, label='estimates')
ax.fill_between(Xpl, ypl_inf.reshape(-1), ypl_sup.reshape(-1),
                 color='lightblue', label='credibility interval')
ax.legend()
```

```python
# noise and prior matrices
Sign = np.diag(np.repeat(sign**2,Xtr.shape[0]))
Sigp = np.diag(np.repeat(1e10,2))

# confidence level for prediction region
alph = 0.995

# predictions on test data
ypr_ave, ypr_cov, ypr_inf, ypr_sup = predGLR(Xpr, Xtr, ytr, Sign, Sigp, alph)

# define plot data
Xpl = np.linspace(-5,5,100)

# predict on the plot data
ypl_ave, ypl_cov, ypl_inf, ypl_sup = predGLR(Xpl, Xtr, ytr, Sign, Sigp, alph)

# display the results
fig, ax = plt.subplots()
ax.plot(Xtr, ytr, 'k+', label='test data')
ax.plot(Xpl, ypl_ave, label='estimates')
ax.fill_between(Xpl, ypl_inf.reshape(-1), ypl_sup.reshape(-1),
                 color='lightblue', label='credibility interval')
ax.legend()
```

Notice how the prior covariance matrix "influences" the training (the smaller the value on the diagonal of Σ_p, the more the model will deviate from the ML solution).
Notice also that the lower/upper bounds on the credibility intervals over the predicted outputs are quadratic in $\mathbf{x}$: the further we are from the training sample, the higher the uncertainty on $f(\mathbf{x})$ is.

2.2 Theory

4 Show that the ML estimates for the weights are obtained by

$$\widehat{\mathbf{w}} = \left(X^\top X\right)^{-1} X^\top \mathbf{y},$$

where X stands for the training input matrix (with training instances $\mathbf{x}_i$ stored row-wise), and $\mathbf{y}$ for the vector of associated outputs y_i.

Maximizing the (log-)likelihood amounts to solve

$$\mathbf{w} = \arg\min J(\mathbf{w}) = \|\mathbf{y} - X\mathbf{w}\|^2,$$

i.e. to solve an ordinary least-squares problem. The vector of first-order derivatives writes as

$$\begin{aligned}\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} &= \frac{\partial}{\partial \mathbf{w}}\left(\mathbf{w}^\top X^\top X\mathbf{w} - 2\mathbf{w}^\top X^\top \mathbf{y} + \mathbf{y}^\top \mathbf{y}\right), \\ &= 2X^\top X\mathbf{w} - 2X^\top \mathbf{y},\end{aligned}$$

and the matrix of second-order derivatives as

$$\frac{\partial^2 J(\mathbf{w})}{\partial \mathbf{w} \partial \mathbf{w}^\top} = 2X^\top X.$$

The matrix of second-order derivatives is obviously positive definite: therefore, the objective function $J(\mathbf{w})$ is convex, and setting the vector of first-order derivatives to zero yields its global minimum:

$$\begin{aligned}\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = 0 \;&\Leftrightarrow\; 2X^\top X\mathbf{w} - 2X^\top \mathbf{y} = 0, \\ &\Leftrightarrow\; \mathbf{w} = \left(X^\top X\right)^{-1} X^\top \mathbf{y},\end{aligned}$$

since $X^\top X$ is positive definite and therefore invertible.

5 We study here the distribution of the ML estimator $\widehat{\mathbf{w}}$ of the parameter vector $\mathbf{w}$.

5 a Show that for any Gaussian random vector $\mathbf{U} \sim \mathcal{N}(\mathbf{a}, B)$, then the random vector $\mathbf{V} = \mathbf{c} + D\mathbf{U}$ is such that $\mathbf{V} \sim \mathcal{N}(\mathbf{c} + D\mathbf{a}, DBD^\top)$.

A linear combination of Gaussian random vectors is a Gaussian random vector: $\mathbf{V} \sim \mathcal{N}(\mathbb{E}[\mathbf{V}], \mathrm{Var}(\mathbf{V}))$. In addition, the expectation satisfies

$$\mathbb{E}[\mathbf{V}] = \mathbb{E}[\mathbf{c} + D\mathbf{U}] = \mathbf{c} + D\,\mathbb{E}[\mathbf{U}];$$

whereas for the variance,

$$\mathrm{Var}(\mathbf{V}) = \mathrm{Var}(\mathbf{c} + D\mathbf{U}) = D\,\mathrm{Var}(\mathbf{U})\,D^\top.$$

5 b Show that the ML estimator $\widehat{\mathbf{w}}$ [1] of the parameter vector $\mathbf{w}$ is distributed as

$$\widehat{\mathbf{w}} \sim \mathcal{N}\left(\mathbf{w}, \sigma_n^2\left(X^\top X\right)^{-1}\right).$$

[1] Note that this notation does not make a distinction between the estimator and its realization.

We apply the previous equality with $\mathbf{U} := \mathbf{Y}$ (the realization of which is the output vector $\mathbf{y}$), $\mathbf{a} := X\mathbf{w}$, and $B := \sigma_n^2 \operatorname{Id}_p$, our linear combination of Gaussian random vectors being thus

$$\mathbf{V} := \widehat{\mathbf{w}} = D\mathbf{Y} = \left(X^\top X\right)^{-1} X^\top \mathbf{Y}.$$

We recall that $\mathbb{E}[\mathbf{Y}] = X\mathbf{w}$ and $\operatorname{Var}(\mathbf{Y}) = \sigma_n^2 \operatorname{Id}_n$: thus,

$$\begin{aligned} & \widehat{\mathbf{w}} \sim \mathcal{N}\left(\left(X^\top X\right)^{-1}\left(X^\top X\right)\mathbf{w}, \left(X^\top X\right)^{-1} X^\top \left(\sigma_n^2 \operatorname{Id}_n\right)\left(\left(X^\top X\right)^{-1} X^\top\right)^\top\right), \\ \Leftrightarrow \quad & \widehat{\mathbf{w}} \sim \mathcal{N}\left(\mathbf{w}, \sigma_n^2 \left(X^\top X\right)^{-1} X^\top \operatorname{Id}_n X \left(X^\top X\right)^{-1}\right), \\ \Leftrightarrow \quad & \widehat{\mathbf{w}} \sim \mathcal{N}\left(\mathbf{w}, \sigma_n^2 \left(X^\top X\right)^{-1}\right). \end{aligned}$$

3 Gaussian process regression

We consider the `scikit-learn` implementation of Gaussian process regression.

```
from sklearn.gaussian_process import GaussianProcessRegressor as GPR
from SKlearn.gaussian_process.kernels import RBF, ConstantKernel as C
from sklearn.gaussian_process.kernels import DotProduct as DP, WhiteKernel as WK
```

3.1 Practice

6 We first consider the noise-free case.

6 a Take a function such as e.g. $f(x) = x\cos(x)$, with $x \in \mathbb{R}$. Generate training, prediction and plot data accordingly, without noise.

Note that GPR requires the training, test and plot data to be defined as $n \times p$ input matrices.

```
def model(x, sign=1):
    f = x * np.cos(x)
    y = f + spst.norm(0,sign).rvs(size=x.shape)
    return y

Xtr = spst.uniform(0,10).rvs(size=20)
ytr = model(Xtr, 0)

Xpr = spst.uniform(0,10).rvs(size=20)
ypr = model(Xpr, 0)

Xpl = np.linspace(0,10,1000)

Xtr = Xtr.reshape(-1,1)
Xpr = Xpr.reshape(-1,1)
Xpl = Xpl.reshape(-1,1)
```

6 b Build the GPR model; fit the model to the training data, make predictions, and display.

We build the GPR model by first creating a kernel and then instantiating the model.

```
# instantiation of the GPR model and training
kern = C(1.0, (1e-3, 1e3)) * RBF(10, (1e-2, 1e2))
GP_0 = GPR(kernel=kern, alpha=0, n_restarts_optimizer=10)
GP_0.fit(Xtr, ytr)
# predictions for the plot instances
ypl_ave, ypl_std = GP_0.predict(Xpl, return_std=True)

# display
fig, ax = plt.subplots(nrows=3)
fig.set_figheight(10)
ax[0].plot(Xtr, ytr, 'k+')
ax[0].plot(Xpl, ypl_ave)
ax[0].fill_between(x=Xpl.reshape(-1),
                   y1=ypl_ave+spst.norm.ppf(0.025)*ypl_std,
                   y2=ypl_ave+spst.norm.ppf(0.975)*ypl_std,
                   color='lightblue', interpolate=True)
```

7 We now introduce noise in the model outputs.

```
ytr_noi = model(Xtr.ravel(), 1)
```

7 a Train the model with the noisy data *assuming they are noise-free*, and display the results.

```
# model instantiation and training
GP_1 = GPR(kernel=kern, alpha=0, n_restarts_optimizer=10)
GP_1.fit(Xtr, ytr_noi)
# predictions for the plot instances
ypl_ave_1, ypl_std_1 = GP_1.predict(Xpl, return_std=True)

# display
ax[1].plot(Xtr, ytr_noi, 'k+')
ax[1].plot(Xpl, ypl_ave_1)
ax[1].fill_between(x=Xpl.reshape(-1),
                   y1=ypl_ave_1+spst.norm.ppf(0.025)*ypl_std_1,
                   y2=ypl_ave_1+spst.norm.ppf(0.975)*ypl_std_1,
                   color='lightblue', interpolate=True)
```

7 b Display the outputs of a model which assumes the presence of noise in the training data, with a fixed amount of noise (using the `GPR` parameter `alpha`, set for instance to `alpha=1`).

```
# model instantiation and training
GP_2 = GPR(kernel=kern, alpha=1, n_restarts_optimizer=10)
GP_2.fit(Xtr, ytr_noi)
# predictions for the plot instances
ypl_ave_2, ypl_std_2 = GP_2.predict(Xpl, return_std=True)

# display
ax[2].plot(Xtr, ytr_noi, 'k+')
ax[2].plot(Xpl, ypl_ave_2)
ax[2].fill_between(x=Xpl.reshape(-1),
                   y1=ypl_ave_2+spst.norm.ppf(0.025)*ypl_std_2,
                   y2=ypl_ave_2+spst.norm.ppf(0.975)*ypl_std_2,
                   color='lightblue', interpolate=True)
```

Part of the variability in the outputs is now captured by the noise component. As a result, the GPR model fit to the data is more regular, and arguably closer to the actual model $f(x) = x\cos(x)$.