Exercises 9

Gaussian processes in scikit-learn

Exercise 9.1 GP posterior
Repeat problem 8.2 using `GaussianProcessRegressor` from `sklearn.gaussian_process`, only plotting the credibility regions based on the posterior predictive variance (not drawing samples). Some useful hints:

- As all supervised machine learning methods in `sklearn`, you first have to construct an object from the model class (in this case `GaussianProcessRegressor`), and thereafter train it on data by using its member function `fit()`. To obtain predictions, use the member function `predict()`. To the latter, you will either have to pass `return_std=True` or `return_cov=True` in order to obtain information about the posterior predictive variance.
- When you construct the model, you have to define a kernel. The kernels are available in `sklearn.gaussian_process.kernels`, where the squared exponential/RBF kernel is available as `RBF`
- The function `fit()` automatically optimizes the hyperparameters. To turn that feature off, you have to pass the argument `optimizer=None` to `GaussianProcessRegressor`.
- To include the measurement noise, you can formulate it as part of the kernel by using the kernel `WhiteKernel`

Exercise 9.2 Learning hyperparameters
Until now, we have made GP regression using predefined hyperparameters, such as the lengthscale ℓ and noise variance σ^2. In this exercise, we will estimate ℓ and σ^2 from the data by maximizing the marginal likelihood. That is done automatically by the `fit()`-function in scikit-learn. Use, as before, the RBF kernel and measurement noise together, this time with the data

$$\mathbf{X} = \begin{bmatrix} -5 & -3 & 0 & 0.1 & 1 & 4.9 & 5 \end{bmatrix}^T, \qquad \mathbf{y} = \begin{bmatrix} 0 & -0.5 & 1 & 0.7 & 0 & 1 & 0.7 \end{bmatrix}^T$$

(a) You still have to provide an initial value of the hyperparameters. Try $\ell = 1$ and $\sigma_n^2 = 0.1$. What hyperparameters do you get when optimizing? Plot the corresponding mean and credibility regions.

(b) Try instead to initialize with $\ell = 10$ and $\sigma_n^2 = 1$. What do you get now?

(c) Try to explain what happens by making a grid over different hyperparameter values, and inspect the marginal likelihood for each point in that grid. The `GaussianProcessRegressor` class has a member function `log_marginal_likelihood()` which you may use. (Do not forget to turn off the hyperparameter optimization!)

Exercise 9.3 Modeling CO_2 levels

The amount of carbon dioxide in the atmosphere has been measured continuously at the Mauna Loa observatory, Hawaii. In this problem, you should use a Gaussian process to model the data from 1958 to 2003, and see how well that model can be used for predicting the data from 2004-2019. They present their latest data at their homepage `https://www.esrl.noaa.gov/gmd/ccgg/trends/`, but for your convenience you can use the data in a more convenient format available here `https://github.com/gpschool/labs/raw/2019/.resources/mauna_loa` and the following code snippet

```
import pickle

with open("mauna_loa", "rb") as fid:
    data = pickle.load(fid)

x = data['X'].flatten()
y = data['Y'].flatten()

xtest = data['Xtest'].flatten()
ytest = data['Ytest'].flatten()
```

Here, `x` and `y` contains your training data.

Start exploring some simple kernels, and thereafter you may have a look page 118-122 of [6], `http://www.gaussianprocess.org/gpml/chapters/RW5.pdf`, for some inspiration on how to design a more bespoke kernel for this problem.

Solutions 9

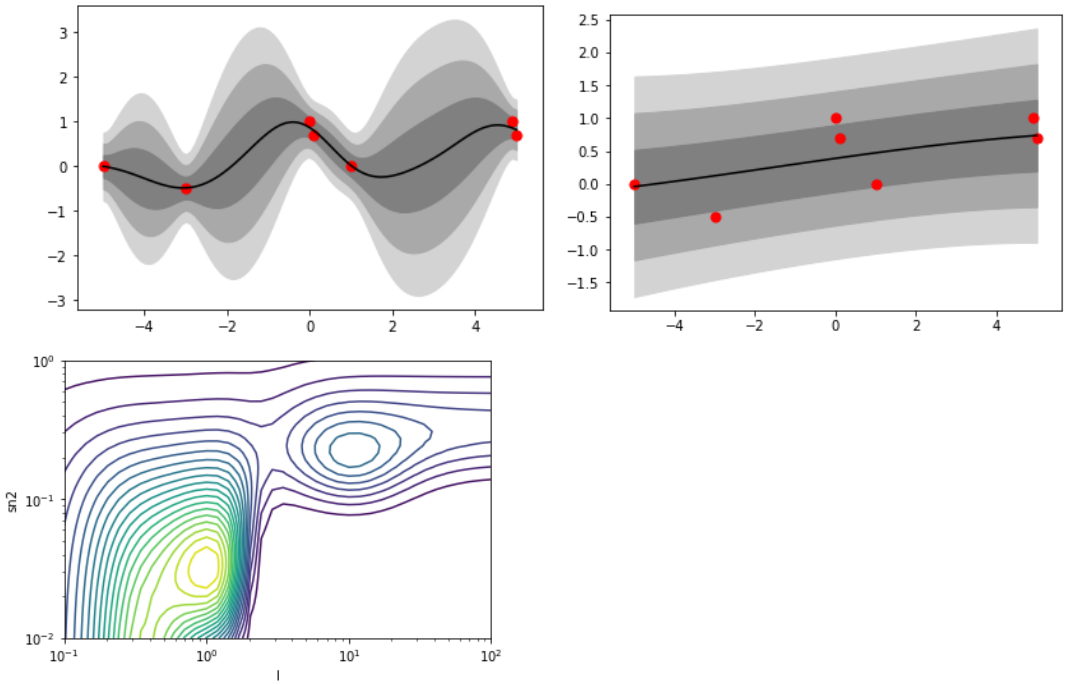
Gaussian processes in scikit-learn

Solution to Exercise 9.1

```
import numpy as np
from matplotlib import pyplot as plt

from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, WhiteKernel

n = 5
x=np.array([-4,-3,-1,0,2]) # Observed inputs
f=np.array([-2,0,1,2,-1]) # Observed function values

m = 101
xs = np.linspace(-5,5,m) # Test input vector

# Without noise
kernel = RBF(length_scale=np.sqrt(2))
gp = GaussianProcessRegressor(kernel=kernel,optimizer=None).fit(x[:, np.newaxis], f)
f_mean, f_std = gp.predict(xs[:, np.newaxis], return_std=True)
plt.plot(xs, f_mean, 'k')
plt.fill_between(xs, f_mean - 3*f_std,f_mean + 3*f_std,color='lightgray')
plt.fill_between(xs, f_mean - 2*f_std,f_mean + 2*f_std,color='darkgray')
plt.fill_between(xs, f_mean - 1*f_std,f_mean + 1*f_std,color='gray')
plt.scatter(x, f, c='r', s=50)
plt.show()

# With noise
kernel = RBF(length_scale=np.sqrt(2)) + WhiteKernel(noise_level=0.1)
gp = GaussianProcessRegressor(kernel=kernel,optimizer=None).fit(x[:, np.newaxis], f)
f_mean, f_std = gp.predict(xs[:, np.newaxis], return_std=True)
plt.plot(xs, f_mean, 'k')
plt.fill_between(xs, f_mean - 3*f_std,f_mean + 3*f_std,color='lightgray')
plt.fill_between(xs, f_mean - 2*f_std,f_mean + 2*f_std,color='darkgray')
plt.fill_between(xs, f_mean - 1*f_std,f_mean + 1*f_std,color='gray')
plt.scatter(x, f, c='r', s=50)
plt.show()

# Different length scale
kernel = RBF(length_scale=0.5) + WhiteKernel(noise_level=0.1)
gp = GaussianProcessRegressor(kernel=kernel,optimizer=None).fit(x[:, np.newaxis], f)
f_mean, f_std = gp.predict(xs[:, np.newaxis], return_std=True)
plt.plot(xs, f_mean, 'k')
plt.fill_between(xs, f_mean - 3*f_std,f_mean + 3*f_std,color='lightgray')
plt.fill_between(xs, f_mean - 2*f_std,f_mean + 2*f_std,color='darkgray')
plt.fill_between(xs, f_mean - 1*f_std,f_mean + 1*f_std,color='gray')
plt.scatter(x, f, c='r', s=50)
plt.show()
```

Solution to Exercise 9.2

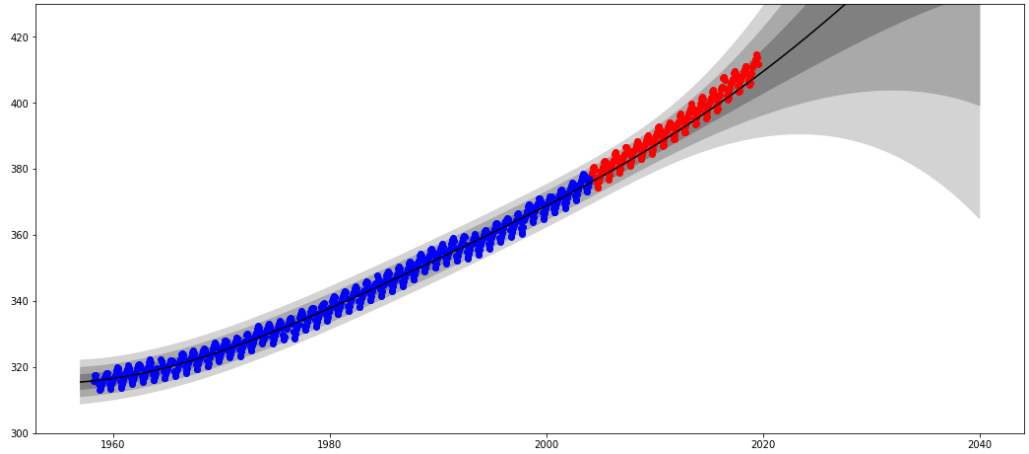
```
import numpy as np
from matplotlib import pyplot as plt

from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, WhiteKernel

n = 5
x=np.array([-5,-3,0,.1,1,4.9,5]) # Observed inputs
y=np.array([0,-.5,1,.7,0,1,.7]) # Observed function values

m = 101
xs = np.linspace(-5,5,m) # Test input vector

# Optimize hyperparameters
kernel = RBF(length_scale=1) + WhiteKernel(noise_level=0.1)
gp = GaussianProcessRegressor(kernel=kernel).fit(x[:, np.newaxis], y)
print(gp.kernel_)
f_mean, f_std = gp.predict(xs[:, np.newaxis], return_std=True)
plt.plot(xs, f_mean, 'k')
plt.fill_between(xs, f_mean - 3*f_std,f_mean + 3*f_std,color='lightgray')
plt.fill_between(xs, f_mean - 2*f_std,f_mean + 2*f_std,color='darkgray')
plt.fill_between(xs, f_mean - 1*f_std,f_mean + 1*f_std,color='gray')
plt.scatter(x, y, c='r', s=50)
plt.show()

# Optimize hyperparameters with another initalization
kernel = RBF(length_scale=10) + WhiteKernel(noise_level=1)
gp = GaussianProcessRegressor(kernel=kernel).fit(x[:, np.newaxis], y)
print(gp.kernel_)
f_mean, f_std = gp.predict(xs[:, np.newaxis], return_std=True)
plt.plot(xs, f_mean, 'k')
plt.fill_between(xs, f_mean - 3*f_std,f_mean + 3*f_std,color='lightgray')
plt.fill_between(xs, f_mean - 2*f_std,f_mean + 2*f_std,color='darkgray')
plt.fill_between(xs, f_mean - 1*f_std,f_mean + 1*f_std,color='gray')
plt.scatter(x, y, c='r', s=50)
plt.show()


# Make a grid of different hyperparameter values to explore the marginal likelihood landscape
lv = np.logspace(-1,2,40)
sn2v = np.logspace(-2,0,40)

L,SN2 = np.meshgrid(lv,sn2v)

xv = np.linspace(-6,6,100)

margloglik = np.zeros(L.size)

for i in range(L.size):

    l = L.flatten()[i]
    sn2 = SN2.flatten()[i]

    # With noise
    kernel = RBF(length_scale=l) + WhiteKernel(noise_level=sn2)
    gp = GaussianProcessRegressor(kernel=kernel,optimizer=None).fit(x[:, np.newaxis], y)

    margloglik[i] = gp.log_marginal_likelihood()

plt.contour(lv,sn2v,np.exp(margloglik.reshape(L.shape)-np.max(margloglik)),levels=20)
plt.xscale('log')
plt.yscale('log')
plt.xlabel('l')
plt.ylabel('sn2')
plt.show()
```



Solution to Exercise 9.3

A first modeling attempt is given by

```
import pickle

with open("mauna_loa", "rb") as fid:
    data = pickle.load(fid)

x = data['X'].flatten()
y = data['Y'].flatten()

xtest = data['Xtest'].flatten()
ytest = data['Ytest'].flatten()


import numpy as np
from matplotlib import pyplot as plt

from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, WhiteKernel


xs = np.linspace(1957,2040,200)

kernel = 100*RBF(length_scale=100) + WhiteKernel(noise_level=1)
gp = GaussianProcessRegressor(kernel=kernel).fit(x[:,np.newaxis], y)
print(gp.kernel_)

f_mean, f_std = gp.predict(xs[:,np.newaxis], return_std=True)

plt.figure(figsize=(18,8))
plt.plot(xs, f_mean, 'k')
plt.fill_between(xs, f_mean - 3*f_std,f_mean + 3*f_std,color='lightgray')
plt.fill_between(xs, f_mean - 2*f_std,f_mean + 2*f_std,color='darkgray')
plt.fill_between(xs, f_mean - 1*f_std,f_mean + 1*f_std,color='gray')
plt.scatter(x, y, c='b')
plt.scatter(xtest, ytest, c='r')
plt.ylim((300,430))
plt.show()
```



More elaborate models are implemented here: `https://scikit-learn.org/stable/auto_examples/gaussian_process/plot_gpr_co2.html`
and exercise 7 here: `https://nbviewer.jupyter.org/github/gpschool/labs/blob/2019/2019/.answers/lab_1.ipynb` (however using another GP package)

Bibliography

[1] David Barber. *Bayesian reasoning and machine learning*. Cambridge University Press, 2012.

[2] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[3] Kevin B Korb and Ann E Nicholson. *Bayesian artificial intelligence*. CRC press, 2010.

[4] Steffen L Lauritzen and David J Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, 50(2):157–194, 1988.

[5] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[6] Carl E. Rrasmussen. *Gaussian Processes for machine learning*. MIT press, 2006.