Gaussian Process Regression

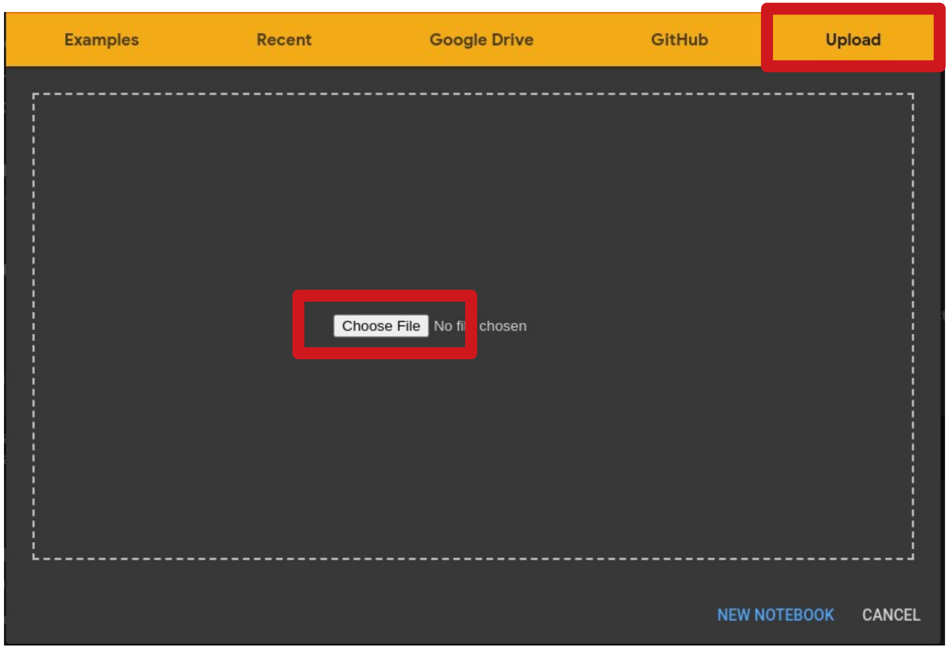
Lee Hong Jung, Jun Ho Woong
RLLAB

Overview

- [Step 1] Define Gaussian RBF
- [Step 2] Random Data Generation
- [Step 3] Prior Distribution
- [Step 4] Posterior Distribution
- [Step 5] Optimization
- [Extra] Easy Implementation with Scikit-learn

[Step 0] Setup

- Download zipfile from eTL
- Go to “colab.research.google.com”
- Select file “Gaussian_Process_Regression.ipynb” in the “upload” tab



[Step 1] Define Gaussian RBF

- Define kernel function as follows:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp\left(-\frac{1}{2l^2}(\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{x}_i - \mathbf{x}_j)\right)$$

- The kernel function used for the covariance of the Gaussian process is:

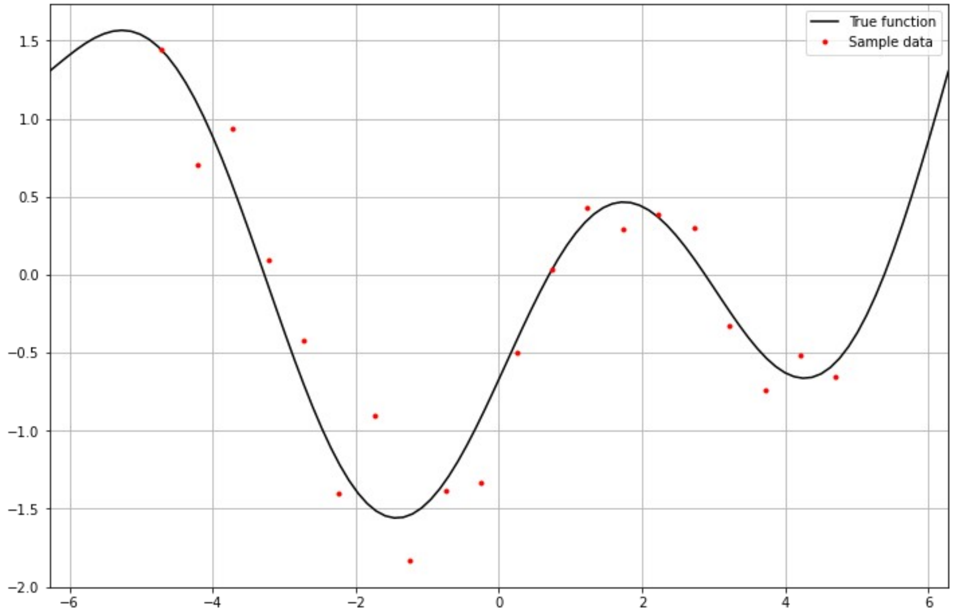
$$K(\mathbf{X}, \mathbf{Y}) = \begin{bmatrix} K(\mathbf{x}_1, \mathbf{y}_1) & K(\mathbf{x}_1, \mathbf{y}_2) & \cdots & K(\mathbf{x}_1, \mathbf{y}_n) \\ K(\mathbf{x}_2, \mathbf{y}_1) & K(\mathbf{x}_2, \mathbf{y}_2) & \cdots & K(\mathbf{x}_2, \mathbf{y}_n) \\ \vdots & \vdots & \ddots & \vdots \\ K(\mathbf{x}_m, \mathbf{y}_1) & K(\mathbf{x}_m, \mathbf{y}_2) & \cdots & K(\mathbf{x}_m, \mathbf{y}_n) \end{bmatrix}$$

[Step 2] Random Data Generation

- Define true function as follows:

$$f(x) = \sin(x) + 0.05x^2$$

- For simplicity, we assume the Gaussian process model has zero mean. Thus, we offset ‘y’ values by their mean value.



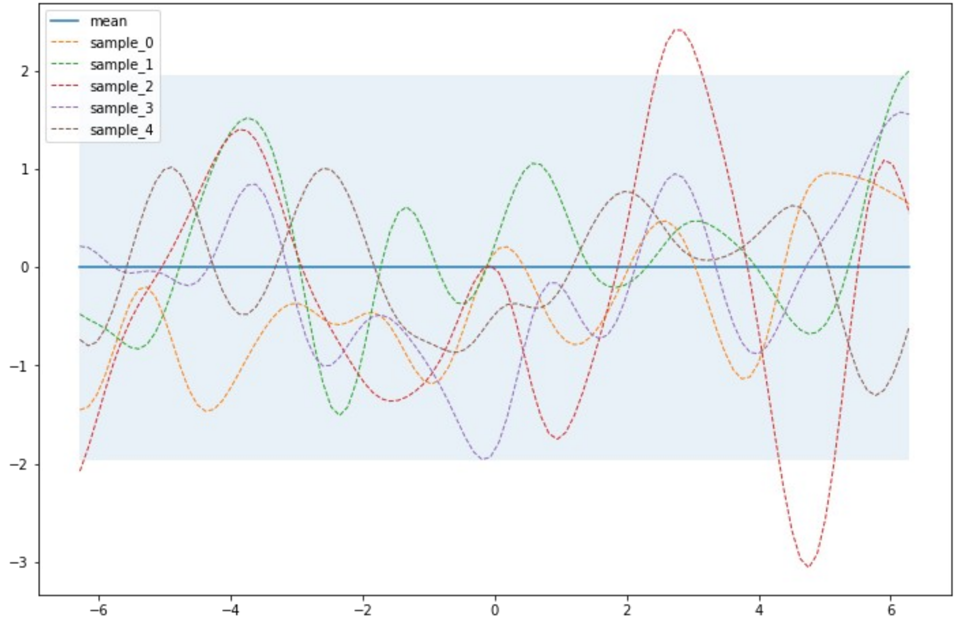
[Step 3] Prior Distribution

- We visualize how the prior distribution looks like without any training data.

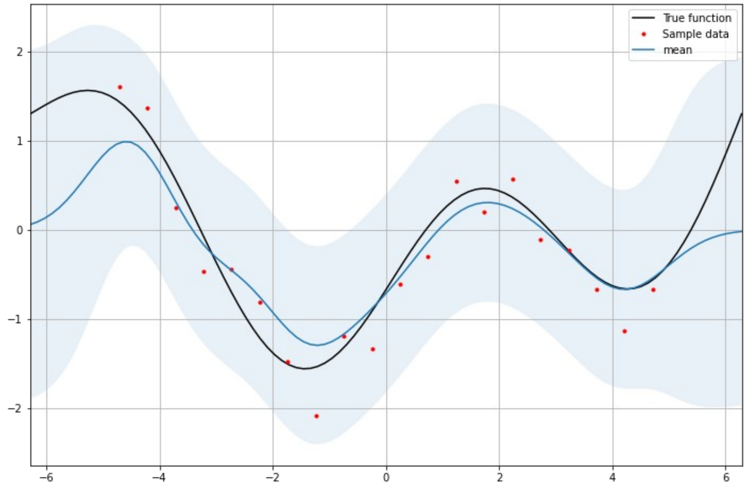
$$\mathbf{f} \sim \mathcal{N}\left(0, K(\mathbf{X}, \mathbf{X})\right)$$

- The blue shaded area corresponds to the 95% confidence interval, i.e., the sampled functions from the Gaussian process **will be** inside the blue shaded area with a **probability of 0.95**.

- Try rerunning this cell with different number of samples!



[Step 4] Posterior Distribution



- Recall, the joint distribution is given as follows:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{pmatrix} K(\mathbf{X},\mathbf{X}) + \sigma_y^2 \mathbb{I} & K(\mathbf{X},\mathbf{X}_*) \\ K(\mathbf{X}_*,\mathbf{X}) & K(\mathbf{X}_*,\mathbf{X}_*) \end{pmatrix}\right)$$

- The conditional distribution is given as follows:

$$\mathbf{f}_* \mid \mathbf{X}_*, \mathbf{X}, \mathbf{y} \sim \mathcal{N}(K(\mathbf{X}_*,\mathbf{X})(K(\mathbf{X},\mathbf{X}) + \sigma_y^2 \mathbb{I})^{-1}\mathbf{y},$$
$$K(\mathbf{X}_*,\mathbf{X}_*) - K(\mathbf{X}_*,\mathbf{X})(K(\mathbf{X},\mathbf{X}) + \sigma_y^2 \mathbb{I})^{-1} K(\mathbf{X},\mathbf{X}_*))$$

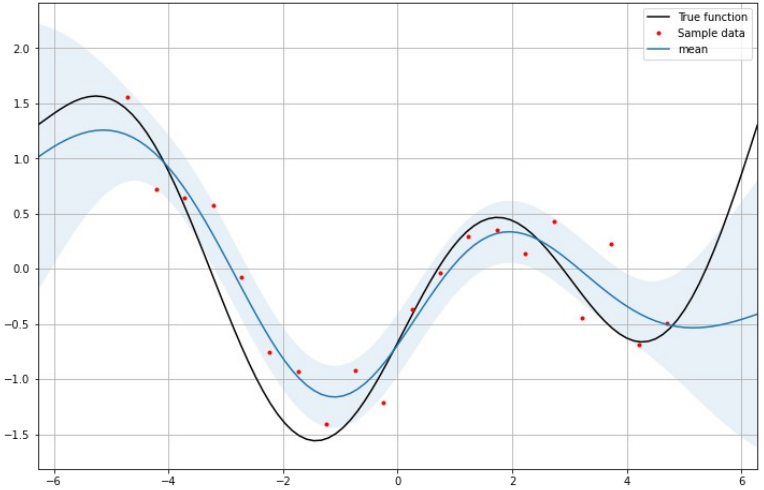
- Implement the posterior distribution function!

[Step 5] Optimization

- We can obtain better results by optimizing the hyperparameters, i.e., minimizing the negative log-likelihood with respect to its parameters!

$$NLL = \frac{1}{2}\mathbf{y}^T(K + \sigma_y^2\mathbb{I})^{-1}\mathbf{y} + \frac{1}{2}\log|K + \sigma_y^2\mathbb{I}|$$

- Implement the loss function!



[Extra] Implementation with sklearn

```python
# Easy implementation with scikit-learn package
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import ConstantKernel, RBF

# Set initial hyperparameters
init_lambda = 10. # l
init_beta = 1. # sigma_f
init_sigma = 0.04 # sigma_y

# Initialize GaussianRBF kernel and GPR model
kernel = ConstantKernel(init_beta, (1e-3, 1e3)) * RBF(init_lambda, (1e-3, 1e3))
gp = GaussianProcessRegressor(kernel=kernel, alpha=init_sigma, n_restarts_optimizer=9)

# Reshape arrays into 2d arrays
x_gp = x_true.reshape(-1, 1)
y_gp = y_true.reshape(-1, 1)
X_train_gp = x_data.reshape(-1, 1)
Y_train_gp = y_data.reshape(-1, 1)

# Optimize parameters and obtain results
gp.fit(X_train_gp, Y_train_gp)
Y_pred_sk, std_pred_sk = gp.predict(x_gp, return_std=True)
Y_pred_sk = Y_pred_sk.flatten()
std_pred_sk = std_pred_sk.flatten()

# Plot results
plt.figure(figsize=(12, 8))
plt.plot(x_true, y_true, 'k-', label='True function')
plt.plot(x_data, y_data, 'r.', label='Sample data')
plt.plot(x_true, Y_pred_sk, 'b-', label='GPR')
plt.fill_between(x_true, Y_pred_sk-1.96*std_pred_sk, Y_pred_sk+1.96*std_pred_sk, color='grey', alpha=0.5)
plt.legend()
plt.grid()
plt.xlim([-2*np.pi, 2*np.pi])
plt.show()
```