MDA Part 2, Assignment 3

Marinos Marios

January 11, 2021

Exercise 1

Below on each figure is displayed 5 samples using the formulas in 1.1, 1.2, etc. The hyper parameters for each kernel was :

- Squared Exponential Kernel : length scale = 0.8, v_0 signal variance = 1
- Polynomial Kernel : $\alpha = 1$ and polynomial degree = 2
- Neuronal Kernel : $\Sigma = 1.5$

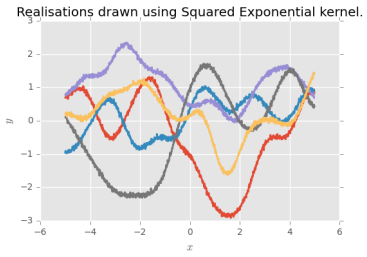

Figure 1: Realisations drawn using Exponential squared kernel.

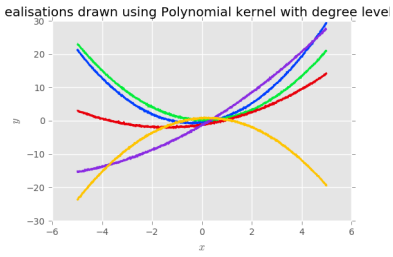

Figure 2: Realisations drawn using Polynomial kernel.

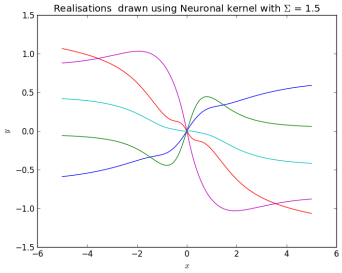

Figure 3: Realisations drawn using Neuronal kernel.

For all of the kernels a small "noise" = 0.015 multiplied with the identity matrix is added to final Covariance matrix K to regularize.

$$Squared\ Exponential\ Kernel\ :\ K(x,\bar{x}) = v_0 exp\left(-\frac{\|x-\bar{x}\|^2}{2\lambda^2}\right) \quad [1.1]$$

$$Polynomial\ Kernel\ :\ K(x,y) = \alpha(1+x^T y)^u \quad [1.2]$$

$$Neuronal\ Kernel\ :\ K(x,\bar{x}) = \frac{2}{\pi} arcsin\left(\frac{\alpha(x,\bar{x})}{\sqrt{(1+\alpha(x,x))(1+\alpha(\bar{x},\bar{x}))}}\right) \quad [1.3]$$

$$where\ \alpha(x,\bar{x}) = 2\bar{x}^T \Sigma x \quad [1.4]$$

Exercise 2

In Figure 4 the plot of the predictive density for $x \in [-3, 3]$ can be found. Also, the predictive value for $x = 0$ is equal to 0.00251256 based on the model created using the 4 points (training data). The shaded part shows the predictive value of $x \in [-3, 3]$ ± 1.960 *standard deviation* which means we have a confidence interval of 95% to include uncertainty in our model. Finally, sklearn by default has the optimizer set to empirical Bayes (type II maximum likelihood), so by setting the parameter n_restarts_optimizer=25 it means that it will optimize it 25 diff times to find the best hyperparameters for RBF kernel using empirical Bayes.

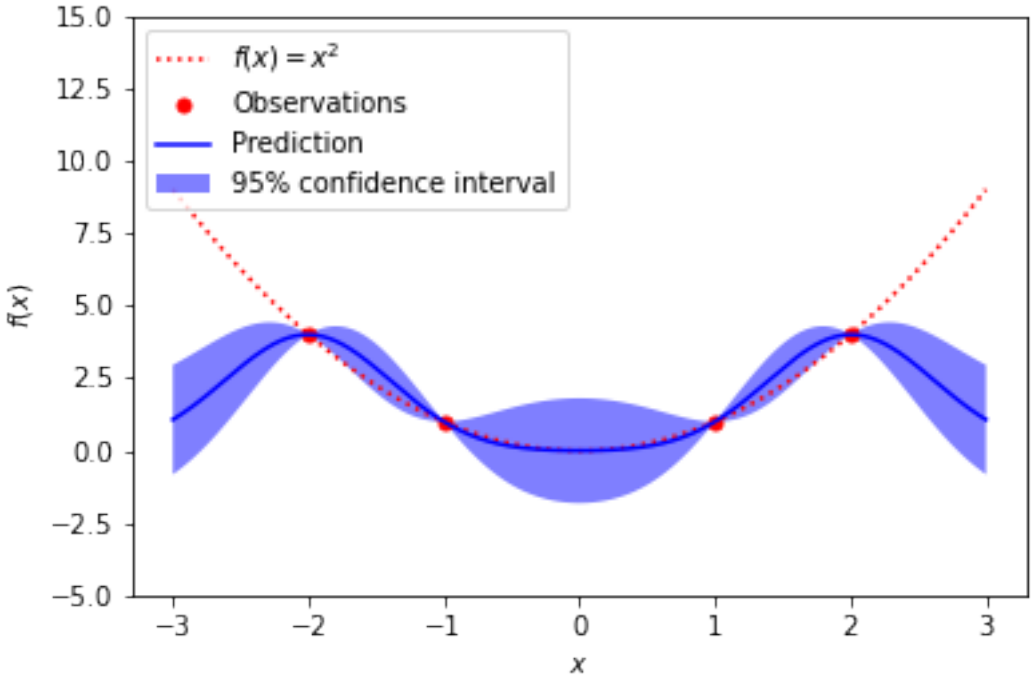

Figure 4: Plot of predictive density for $x \in [-3, 3]$ with 95% confidence interval ($\pm 1.960 std$).

A Code Exercise 1

```
import numpy as np
import matplotlib.pyplot as plt
import scipy
import itertools
np.set_printoptions(suppress=True)

def Squared_Exponential(X1, X2, multiple_stability, l=0.8, sigma_f=1.0):
    """
    Isotropic squared exponential kernel.
    Args:
        X1: Array of m points (m x d).
        X2: Array of n points (n x d).
        multiple_stability : factor to multiple with the identity matrix to have
    Returns:
        (m x n) matrix.
        ||x - x'||^2 = x^2 + x'^2 - 2 X @ X.T
    """
    sqdist = np.sum(X1**2, 1).reshape(-1, 1) + np.sum(X2**2, 1) - 2 * X1 @ X2.T
    K = sigma_f * np.exp(-sqdist / (2* l**2))
    # add a bit of noise for numerical stability
    Noise_K = K + multiple_stability*np.eye(X1.shape[0])
    return Noise_K

def Polynomial_kernel(X1, X2, multiple_stability, degree, alpha = 1):
    """
        X1: Array of m points (m x d).
        X2: Array of n points (n x d).
        multiple_stability : factor to multiple with the identity matrix to have

        Returns the covariance matrix using the polynomial kernel.
    """
    K = alpha * (1 + X1 @ X2.T) ** degree
    Noise_K = K + multiple_stability*np.eye(X1.shape[0])
    return Noise_K

def a(X1, X2, sigma):
    return 2 * X2.T * sigma @ X1

def Neuronal(X1,X2, sigma):
    numerator =a(X1, X2, sigma)
    denominator = np.sqrt((1+a(X1, X1, sigma))*(1+a(X2, X2, sigma)))
    return (2/np.pi) * np.arcsin(numerator / denominator)

def draw_Samples(samples, mu, cov):
    samples = 5
    realizations = np.random.multivariate_normal(mu.ravel(), cov, samples)
```

```
    # Plot GP mean, confidence interval and samples
    for i in range(samples):
        plt.plot(X, realizations[i])
    plt.xlabel('$x$')
    plt.ylabel('$y$')

if __name__ == "__main__":

    # Finite number of points, n x 1 atrix
    # Define constants such as mu matrix, X data and stability term.
    X = np.linspace(-5, 5, 1001).reshape(-1, 1)
    # Mean and covariance of the prior
    mu = np.zeros(X.shape)

    stability_term = 0.0015

    # using squared exponential kernel.

    cov = Squared_Exponential(X, X, stability_term)
    # Draw five samples from the prior.
    plt.style.use('seaborn-bright')
    draw_Samples(5, mu, cov)
    plt.title('Realisations drawn using Squared Exponential kernel.')
    # plt.savefig("example.png")
    # Using polynomial kernel.

    cov = Polynomial_kernel(X, X, stability_term, 2)
    print(cov.shape)
    # Draw five samples from the prior.
    plt.style.use('_classic_test_patch')
    draw_Samples(5, mu, cov)
    plt.title('Realisations drawn using Polynomial kernel with degree level 2.')
    # plt.savefig("example.png")
    sigma = 1.5

    cov = [Neuronal(i, j, sigma) for (i, j) in itertools.product(X, X)]

    cov = np.array(cov).reshape(X.shape[0], X.shape[0])
    # Draw five samples from the prior.
    plt.style.use('_classic_test_patch')

    # plt.savefig("example.png")
    draw_Samples(5, mu, cov)
    plt.title('Realisations  drawn using Neuronal kernel with $\Sigma$ = 1.5')
    plt.savefig('Neuronal_kernel.png')
```

B Code Exercise 2

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

from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF
np.set_printoptions(suppress=True)
np.random.seed(1)


def f(x):
    """The function to predict."""
    return x ** 2

if __name__ == '__main__':
    X = np.atleast_2d([-1, 1, -2, 2]).T
    y = f(X).ravel()
    # Mesh the input space for evaluations of the real function, the prediction a
    # its MSE
    x = np.atleast_2d(np.linspace(-3, 3, 1000)).T

    # Instantiate a Gaussian Process model
    gp = GaussianProcessRegressor(kernel=RBF(1), n_restarts_optimizer=25)

    # Fit the GPR with the X data provided.
    gp.fit(X, y)


    # predict from -3 to 3 to get the posterior mean.
    y_pred, sigma = gp.predict(x, return_std=True)
    # Plot the function, the prediction and the 95% confidence interval based on
    # predictive space. (x -3 to 3.)
    plt.figure()
    plt.plot(x, f(x), 'r:', label=r'$f(x) = x^{2}$')
    plt.plot(X, y, 'r.', markersize=10, label='Observations')
    plt.plot(x, y_pred, 'b-', label='Prediction')
    plt.fill(np.concatenate([x, x[::-1]]),
            np.concatenate([y_pred - 1.960 * sigma,
                            (y_pred + 1.960 * sigma)[::-1]]),
            alpha=.5, fc='b', ec='None', label='95% confidence interval')
    plt.xlabel('$x$')
    plt.ylabel('$f(x)$')
    plt.ylim(-5, 15)
    plt.legend(loc='upper left')
```