Equivalent Kernel

***Abstract*—This document contains theory involved in curve fitting.**

1 Objective

The objective is to use an equivalent kernel on a noisy data.

2 Generate Dataset

Create a sinusoidal function of the form

$$y = A \sin 2\pi x + n(t) \tag{2.0.1}$$

n(t) is the random noise that is included in the training set. This set consists of N samples of input data i.e. x expressed as shown below

$$x = \left(x_1, x_2, .., x_N\right)^T \tag{2.0.2}$$

which give the corresponding values of y denoted as

$$y = \left(y_1, y_2, .., y_N\right)^T \tag{2.0.3}$$

The corresponding values of y are generated from the Eq (2.0.1).The first term $A \sin 2\pi x$ is computed directly and then random noise samples having a normal(Gaussian) distribution are added inorder to get the corresponding values of y.

```
#Generate the sine curve
noise = 0.5
A = 2
# Noisy training data
X_train = np.arange(−3, 4, 1).reshape(−1, 1)
Y_train = A∗np.sin(2∗np.pi∗X_train) + noise ∗
    np.random.randn(∗X_train.shape)
```

The generated input matrix would look like

$$\mathbf{F} = \begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{N-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{N-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{N-1} \\ \vdots & & \vdots & & \vdots \\ 1 & \dots & \dots & \dots & x_N^{N-1} \end{pmatrix} \tag{2.0.4}$$

3 Polynomial Curve Fitting

The goal is to find the best line that fits into the pattern of the training data shown in the graph. We shall fit the data using a polynomial function of the form,

$$y\left(w, x\right) = \sum_{j=0}^{M} w_j x^j \tag{3.0.1}$$

(3.0.2)

M is the order of the polynomial The polynomial coefficient are collectively denoted by the vector **w**.The proposed vector **w** of the model referring to Eq (2.0.4) is given by

$$\hat{\mathbf{w}} = \left(\mathbf{F}^T\mathbf{F}\right)^{-1}\mathbf{F}^T y \tag{3.0.3}$$

4 Equivalent Kernel

For Bayesian treatment of linear regression, we introduce a prior probability distribution over the model parameters **w**.

The corresponding conjugate prior is therefore given by a Gaussian distribution of the form

$$p(\mathbf{w}) = N(\mathbf{w}|\mathbf{m_0}, \mathbf{S_0}) \tag{4.0.1}$$

having mean $\mathbf{m_0}$ and covariance $\mathbf{S_0}$.

We now compute the posterior distribution, which is proportional to the product of the likelihood function and the prior which takes the form

$$p(\mathbf{w}|t) = N(\mathbf{w}|\mathbf{m_N}, \mathbf{S_N}) \tag{4.0.2}$$

Since the prior is Gaussian, so would be the posterior. In Eq (4.0.2),

$$\mathbf{m}_N = \mathbf{S}_N\left(\mathbf{S}_0^{-1}\mathbf{m}_0 + \beta f^T t\right) \tag{4.0.3}$$

$$\mathbf{S}_N^{-1} = \mathbf{S}_0^{-1} + \beta f^T f \tag{4.0.4}$$

Specifically, we consider a zero - mean isotropic Gaussian governed by a single precision parameter α so that

$$p(\mathbf{w}|\alpha) = N(\mathbf{w}|0, \alpha^{-1}\mathbf{I}) \tag{4.0.5}$$

Then the corresponding posterior distribution over $\mathbf{w}$ is then given as

$$\mathbf{m}_N = \beta \mathbf{S}_N f^T t \tag{4.0.6}$$

$$\mathbf{S}_N^{-1} = \alpha \mathbf{I} + \beta f^T f \tag{4.0.7}$$

The predictive mean can be written in the form

$$\begin{aligned} y(\mathbf{x}, \mathbf{m}_N) &= \mathbf{m}_N f(x) \\ &= \beta f(\mathbf{x})^T S_N f^T t \\ &= \sum_{n=1}^{N} \beta f(\mathbf{x})^T S_N f(\mathbf{x}_n) t_n \end{aligned} \tag{4.0.8}$$

The mean of the predictive distribution at a point $\mathbf{x}$ is given by a linear combination of the training set target variables t_n , so

$$y(\mathbf{x}, \mathbf{m}_N) = \sum_{n=1}^{N} k(\mathbf{x}, \mathbf{x}^{'}) t_n \tag{4.0.9}$$

where

$$k(\mathbf{x}, \mathbf{x}_n) = \beta f(x)^T S_N f(\mathbf{x}^{'}) \tag{4.0.10}$$

Eq (4.0.10) is known as the smoother matrix of the equivalent kernel.

Further insight into the role of the equivalent kernel can be obtained by considering the covariance between $y(\mathbf{x})$ and $y(\mathbf{x}^{'})$ given by

$$cov[y(\mathbf{x}), y(\mathbf{x}^{'})] = cov[f(\mathbf{x})^T \mathbf{w}, \mathbf{w}^T f(\mathbf{x}^{'})] \tag{4.0.11}$$

$$= f(\mathbf{x})^T S_N f(\mathbf{x}^{'}) \tag{4.0.12}$$

$$= \beta^{-1} k(\mathbf{x}, \mathbf{x}^{'}) \tag{4.0.13}$$

we define a localized kernel directly and use this to make predictions for new input vectors $\mathbf{x}$.

This leads to practical frameworks for regression known as Gaussian Processes.

An effective kernel should satisfy

$$\sum_{n=1}^{N} k(\mathbf{x}, \mathbf{x}_n) = 1 \tag{4.0.14}$$

for all values of x.

Eq (4.0.10) satisfies the general equation of kernel, that can be expressed in the form an inner product with respect to a vector $f(\mathbf{x})$ of non linear functions given by

$$k(\mathbf{x}, \mathbf{z}) = \psi(\mathbf{x})^T \psi(\mathbf{z}) \tag{4.0.15}$$

where

$$\psi(\mathbf{x}) = \beta^{\frac{1}{2}} S_N^{\frac{1}{2}} f(\mathbf{x}) \tag{4.0.16}$$

5 Implementation using numPy

Here, we will use the squared exponential kernel, also known as Gaussian kernel. given by

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp -\frac{1}{2l^2}(\mathbf{x}_i - \mathbf{x}_j)^T (\mathbf{x}_i - \mathbf{x}_j) \tag{5.0.1}$$

The length l controls the smoothness of the function and σ_f the vertical variation

We define the kernel Eq (5.0.1),

```
#Kernel
def kernel(X1, X2, l=1.0, sigma_f=1.0):
    sqdist = np.sum(X1**2, 1).reshape(-1, 1) +
        np.sum(X2**2, 1) - 2 * np.dot(X1, X2.T
        )
    return sigma_f**2 * np.exp(-0.5 / l**2 *
        sqdist)
```

We use a function for plotting the samples

```
def plot_gp(mu, cov, X, X_train=None,
    Y_train=None, samples=[]):
    X = X.ravel()
    mu = mu.ravel()
    uncertainty = 1.96 * np.sqrt(np.diag(cov))

    plt.fill_between(X, mu + uncertainty, mu -
        uncertainty, alpha=0.1)
    plt.plot(X, mu, label='Mean')
    for i, sample in enumerate(samples):
        plt.plot(X, sample, lw=1, ls='--', label=
            f'Sample {i+1}')
    if X_train is not None:
        plt.plot(X_train, Y_train, 'rx')
    plt.legend()
```

The following code draws three random samples and plots it together with the zero mean.

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

X = np.arange(-5, 5, 0.2).reshape(-1, 1)

mu = np.zeros(X.shape)
cov = kernel(X, X)

# Draw three samples from the prior
```

```
samples = np.random.multivariate_normal(mu.
    ravel(), cov, 3)

# Plot GP mean, confidence interval and samples
plot_gp(mu, cov, X, samples=samples)
```

The Plot would look as below shown in Fig 0

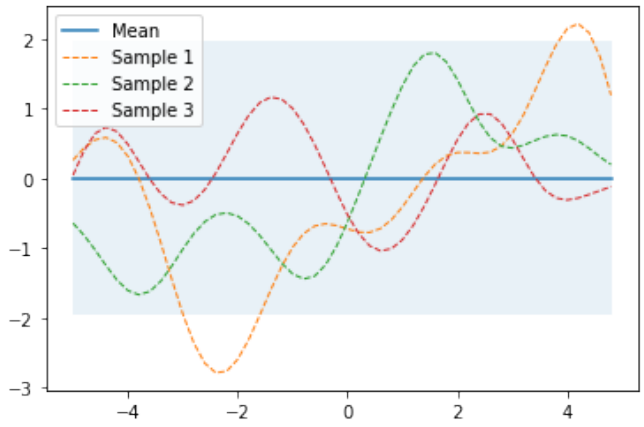

Fig. 0

We define the posterior as

```
from numpy.linalg import inv

def posterior(X_s, X_train, Y_train, l=1.0,
    sigma_f=1.0, sigma_y=1e-8):
    K = kernel(X_train, X_train, l, sigma_f) +
        sigma_y**2 * np.eye(len(X_train))
    K_s = kernel(X_train, X_s, l, sigma_f)
    K_ss = kernel(X_s, X_s, l, sigma_f) + 1e
        -8 * np.eye(len(X_s))
    K_inv = inv(K)

    # Equation (7)
    mu_s = K_s.T.dot(K_inv).dot(Y_train)

    # Equation (8)
    cov_s = K_ss - K_s.T.dot(K_inv).dot(K_s
        )

    return mu_s, cov_s
```

Now , the prediction from noisy training data

```
noise = 0.5
A = 2
# Noisy training data
X_train = np.arange(-3, 4, 1).reshape(-1, 1)
Y_train = A*np.sin(2*np.pi*X_train) + noise *
    np.random.randn(*X_train.shape)

# Compute mean and covariance of the posterior
    distribution
mu_s, cov_s = posterior(X, X_train, Y_train,
    sigma_y=noise)

samples = np.random.multivariate_normal(mu_s.
    ravel(), cov_s, 3)
plot_gp(mu_s, cov_s, X, X_train=X_train,
    Y_train=Y_train, samples=samples)
```

Th plot would look like in Fig 0

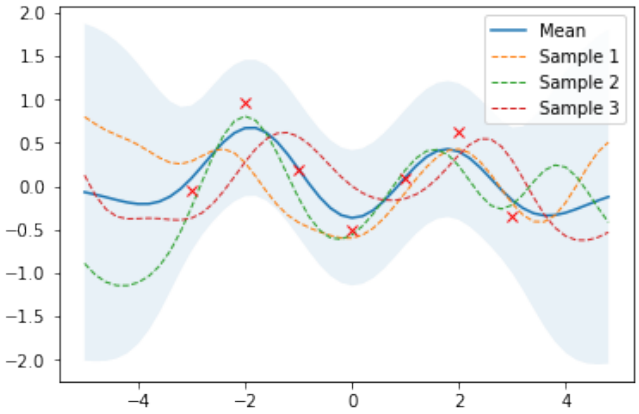

Fig. 0

Python code:

https://github.com/Hrithikraj2/EE4015_IDP/blob/main/Assignment_6/Assignment_6.ipynb