Evidence Function

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

1 Objective

The objective is to evaluate the evidence function.

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
def g(X, noise_variance):
    '''Sinusoidal function plus noise'''
    return 0.5 + np.sin(2 * np.pi * X) + noise(X.
        shape, noise_variance)
```

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,

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

$$\tag{3.0.2}$$

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 Evidence Function

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),

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

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

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 **w** is then given as

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

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

For Baye's theorem, the posterior distribution for α and β is given by

$$p(\alpha,\beta|\mathbf{t}) = p(\mathbf{t}|\alpha,\beta)p(\alpha,\beta) \tag{4.0.8}$$

The marginal likelihood function $p(\mathbf{t}|\alpha,\beta)$ is given by integrating over the weight parameters $\mathbf{w}$

$$p(\mathbf{t}|\alpha,\beta) = \int p(\mathbf{t}|\mathbf{w},\beta)p(\mathbf{w}|\alpha)d\mathbf{w} \tag{4.0.9}$$

From the standard form for the normalization coefficient of a Gaussian, The evidence function takes the form

$$p(\mathbf{t}|\alpha,\beta) = \left(\frac{\beta}{2\pi}\right)^{\frac{N}{2}} \left(\frac{\alpha}{2\pi}\right)^{\frac{M}{2}} \int \exp -E(\mathbf{w})d\mathbf{w} \tag{4.0.10}$$

where M is the dimensionality of $\mathbf{w}$ and we know,

$$E(\mathbf{w}) = \beta E_D(\mathbf{w}) + \alpha E_W(\mathbf{w}) \tag{4.0.11}$$

$$= \frac{\beta}{2} \|\mathbf{t} - \mathbf{f_w}\|^2 + \frac{\alpha}{2}\mathbf{w}^T\mathbf{w} \tag{4.0.12}$$

Eq (4.0.12) is equal up to a constant of proportionality to the regularized sum of squares error function. Now, we complete the square over $\mathbf{w}$

$$E(\mathbf{w}) = E(\mathbf{m}_N) + \frac{1}{2}(\mathbf{w} - \mathbf{m}_N)^T \mathbf{A} (\mathbf{w} - \mathbf{m}_N) \tag{4.0.13}$$

where

$$\mathbf{A} = \alpha\mathbf{I} + \beta\mathbf{f}^T\mathbf{f} \tag{4.0.14}$$

$$E(\mathbf{m}_N) = \frac{\beta}{2} \left\|\mathbf{t} - \mathbf{f}_{\mathbf{m}_N}\right\|^2 + \frac{\alpha}{2}\mathbf{m}_N^T\mathbf{m}_N \tag{4.0.15}$$

Here A corresponds to the matrix of second derivatives of the error function

$$A = \nabla\nabla E(\mathbf{w}) \tag{4.0.16}$$

which is known as Hessian matrix and $\mathbf{m}_N$ is defined as

$$\mathbf{m}_N = \beta\mathbf{A}^{-1}\mathbf{f}^T\mathbf{t} \tag{4.0.17}$$

Clearly, $A = S_N^{-1}$ which represents the mean of the posterior distribution.

The integral over $\mathbf{w}$ can now be evaluated simply by appealing to the standard result for the normalization coefficient of a multivariate Gaussian giving

$$\int \exp -E(\mathbf{w})d\mathbf{w} = \exp -E(\mathbf{m}_N) \int \exp -\frac{1}{2}(\mathbf{w} - \mathbf{m}_N)^T \mathbf{A} (\mathbf{w} - \mathbf{m}_N)d\mathbf{w} = (2\pi)^{\frac{M}{2}} |\mathbf{A}|^{-\frac{1}{2}} \exp -E(\mathbf{m}_N) \tag{4.0.18}$$

We can now write the log of the marginal likelihood in the form

$$ln\,p(\mathbf{t}|\alpha,\beta) = \frac{M}{2}ln\alpha + \frac{N}{2}ln\beta - E(\mathbf{m}_N) - \frac{1}{2}ln|\mathbf{A}| - \frac{N}{2}ln(2\pi) \tag{4.0.19}$$

Eq (4.0.19) is the required expression for the evidence function.

5 Implementation

Function *posterior* computes the mean and co variance matrix of the posterior distribution and function *posterior_predictive* computes the mean and the variances of the posterior predictive distribution.

```
import numpy as np

def posterior(Phi, t, alpha, beta, return_inverse=
    False):
    """Computes mean and covariance matrix of
        the posterior distribution."""
    S_N_inv = alpha * np.eye(Phi.shape[1]) +
        beta * Phi.T.dot(Phi)
    S_N = np.linalg.inv(S_N_inv)
    m_N = beta * S_N.dot(Phi.T).dot(t)
    if return_inverse:
        return m_N, S_N, S_N_inv
    else:
        return m_N, S_N

def posterior_predictive(Phi_test, m_N, S_N,
    beta):
    """Computes mean and variances of the
        posterior predictive distribution."""
    y = Phi_test.dot(m_N)
    # Only compute variances (diagonal elements
        of covariance matrix)
    y_var = 1 / beta + np.sum(Phi_test.dot(S_N
        ) * Phi_test, axis=1)
```

```
    return y, y_var
```

For fitting a linear model to a sinusoidal dataset we transform input x with *gaussian_basis_function* and later with *polynomial_basis_function*.

These non-linear basis functions are necessary to model the non-linear relationship between input x and target t.

```
def gaussian_basis_function(x, mu, sigma=0.1):
    return np.exp(-0.5 * (x - mu) ** 2 / sigma
        ** 2)

def polynomial_basis_function(x, power):
    return x ** power
```

The following code shows how to fit a Gaussian basis function model to a noisy sinusoidal dataset.

```
N_list = [3, 8, 20]

beta = 25.0
alpha = 2.0

# Training observations in [-1, 1)
X = np.random.rand(N_list[-1], 1)

# Training target values
t = g(X, noise_variance=1/beta)

# Test observations
X_test = np.linspace(0, 1, 100).reshape(-1, 1)

# Function values without noise
y_true = g(X_test, noise_variance=0)

# Design matrix of test observations
Phi_test = expand(X_test, bf=
    gaussian_basis_function, bf_args=np.
    linspace(0, 1, 9))
```

The following block defines the log of marginal likelihood

```
def log_marginal_likelihood(Phi, t, alpha, beta):
    """Computes the log of the marginal
        likelihood."""
    N, M = Phi.shape

    m_N, _, S_N_inv = posterior(Phi, t, alpha,
        beta, return_inverse=True)

    E_D = beta * np.sum((t - Phi.dot(m_N)) **
        2)
    E_W = alpha * np.sum(m_N ** 2)

    score = M * np.log(alpha) + \
            N * np.log(beta) - \
            E_D - \
            E_W - \
            np.log(np.linalg.det(S_N_inv)) - \
            N * np.log(2 * np.pi)

    return 0.5 * score
```

The 10 polynomial basis function models of degrees 0-9 are compared based on the log marginal likelihood. Some plots are shown below in increasing order of polynomial dgree

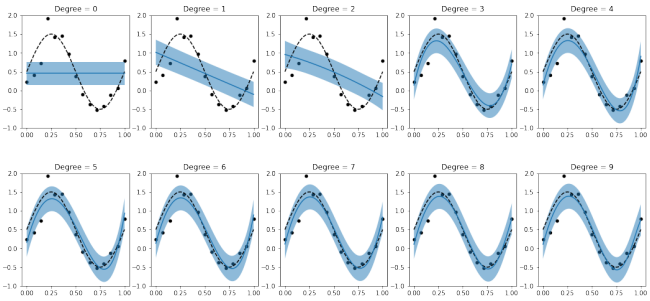

Fig. 0

From the above figure , we can observe that polynomial models of higher degree do not overfit to the data.

The following block plots the log of marginal likelihood vs Polynomial degree

```
mlls = []

for d in degrees:
    mll = log_marginal_likelihood(Phi[:,:d+1], t,
        alpha=alpha, beta=beta)
    mlls.append(mll)

degree_max = np.argmax(mlls)

plt.plot(degrees, mlls)
plt.axvline(x=degree_max, ls='--', c='k', lw=1)
plt.xticks(range(0, 10))
plt.xlabel('Polynomial degree')
plt.ylabel('Log marginal likelihood');
```

Python code:

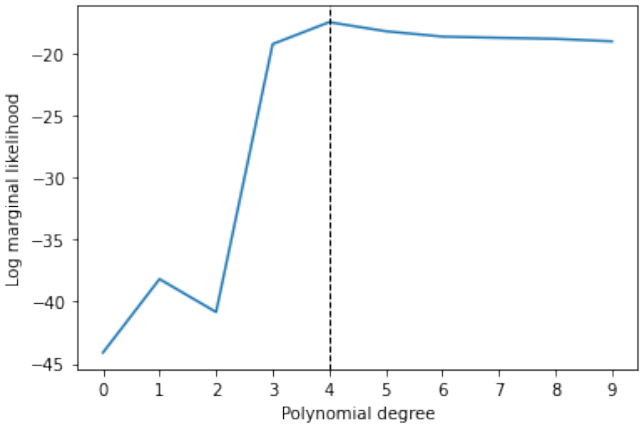

Fig. 0

https://github.com/Hrithikraj2/EE4015_IDP/blob/main/Assignment_7/Assignment_7.ipynb