Peergrade 1

Nicolaj Filrup Rasmussen

11 September 2018

2.12

Verify that $U(\boldsymbol{x}|a,b)$ is normalised and find expected value and variance

x is defined in the interval from a to b. Therefore normalisation can be checked for by seeing that the following integral equals 1.

$$\int_a^b \frac{1}{b-a}dx = \frac{b-a}{b-a} = 1$$

The expected value of a continuous random variable is found by

$$E[x] = \int_a^b xf(x)dx$$

This can be done as follows:

$$E[x] = \int_a^b x\frac{1}{b-a}dx = \left[\frac{x^2}{2(b-a)}\right]_a^b = \frac{b^2-a^2}{2(b-a)} = \frac{a+b}{2}$$

Lastly variance for a random variable x is defined as

$$var[x] = E[x^2] - E[x]^2$$

We find this as:

$$var[x] = \frac{a^2+ab+b^2}{3} - \frac{(a+b)^2}{4} = \frac{(b-a)^2}{12}$$

3.7

We know:

$$p(\mathbf{t}|\mathbf{w}) \propto \prod_{n=1}^{N} \mathcal{N}\left(t_n|\mathbf{w}^T\phi(x_n), \beta^{-1}\right)$$

$$p(\mathbf{w}) \propto \mathcal{N}(\mathbf{w}|\mathbf{m_0}, \mathbf{S_0})$$

Using Bayes rule, we can now derive (3.49).

$$p(\mathbf{w}|\mathbf{t}) \propto \left(\prod_{n=1}^{N} \mathcal{N}\left(t_n|\mathbf{w}^T\phi(x_n), \beta^{-1}\right)\right) \mathcal{N}(\mathbf{w}|\mathbf{m_0}, \mathbf{S_0})$$

$$\propto \exp\left(-\frac{\beta}{2}(\mathbf{t}-\mathbf{\Phi w})^T(\mathbf{t}-\mathbf{\Phi w})\right)\exp\left(-\frac{1}{2}(\mathbf{w}-\mathbf{m_0})^T\mathbf{S}_0^{-1}(\mathbf{w}-\mathbf{m_0})\right)$$

$$\exp\left(-\frac{1}{2}(\mathbf{w}^T(\mathbf{S}_0^{-1}+\beta\mathbf{\Phi}^T\mathbf{\Phi})\mathbf{w}-\beta\mathbf{t}^T\mathbf{\Phi w}-\beta\mathbf{w}^T\mathbf{\Phi}^T\mathbf{t}+\beta\mathbf{t}^T\mathbf{t}\mathbf{m}_0^T\mathbf{S}_0^{-1}\mathbf{w}-\mathbf{w}^T\mathbf{S}_0^{-1}\mathbf{m}_0+\mathbf{m}_0^T\mathbf{S}_0^{-1}\mathbf{m}_0)\right)$$

$$=\exp\left(-\frac{1}{2}(\mathbf{w}^T(\mathbf{S}_0^{-1}+\beta\mathbf{\Phi}^T\mathbf{\Phi})\mathbf{w}-(\mathbf{S}_0^{-1}\mathbf{m}_0+\beta\mathbf{\Phi}^T\mathbf{t})^T\mathbf{w}-\mathbf{w}^T(\mathbf{S}_0^{-1}\mathbf{m}_0+\beta\mathbf{\Phi}^T\mathbf{t})+\beta\mathbf{t}^T\mathbf{t}+\mathbf{m}_0^T\mathbf{S}_0^{-1}\mathbf{m}_0)\right)$$

$$\exp\left(-\frac{1}{2}(\mathbf{w}-\mathbf{m}_N)^T\mathbf{S}_N^{-1}(\mathbf{w}-\mathbf{m}_N)\right)\exp\left(-\frac{1}{2}(\beta\mathbf{t}^T\mathbf{t}+\mathbf{m}_0^T\mathbf{S}_0^{-1}\mathbf{m}_0-\mathbf{m}_N^T\mathbf{S}_N^{-1}\mathbf{m}_N)\right)$$

The second exponential is independent of w and can be absorbed into the normalisation factor.

3.17

Show that the evidence function for the Bayesian Linear Regression Model can be written in the form

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

From (3.79) we have

$$E(\mathbf{w})=\beta E_D(\mathbf{w})+\alpha E_W(\mathbf{w})=\frac{\beta}{2}||\mathbf{t}-\mathbf{\Phi w}||^2+\frac{\alpha}{2}\mathbf{w}^T\mathbf{w}$$

We know that

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

From (3.11) we have

$$p(\mathbf{t}|\mathbf{w},\beta)=\left(\frac{\beta}{2\pi}\right)^{N/2}\exp(-\beta E_D(\mathbf{w}))$$

From (3.52) we have

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

This can be rewritten as follows:

$$\mathcal{N}(\mathbf{w}|\mathbf{0},\alpha^{-1}\mathbf{I})=|2\pi\alpha^{-1}\mathbf{I}|^{-1/2}\exp\left(-\frac{1}{2}\mathbf{w}^T(\alpha^{-1}\mathbf{I})^{-1}\mathbf{w}\right)$$

$$=(2\pi\alpha^{-1})^{-M/2}\exp\left(-\frac{1}{2}\mathbf{w}^T\alpha\mathbf{I}^{-1}\mathbf{w}\right)$$

$$= \left(\frac{\alpha}{2\pi}\right)^{-M/2} \exp\left(-\frac{\alpha}{2}\mathbf{w}^T\mathbf{w}\right)$$

We can now insert this into the original equation to get

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

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

Programming 2.2

This is the code used to solve P2.2.

The values chosen for the hyperparameters were:

$$M = 15$$
$$\alpha = 0.01$$
$$\beta = 0.07$$

One could argue that the fit should be jumping more. There are a lot of points with very high values. But because they do not show a very clear trend, I am not comfortable making the fit any more volatile than this.

```
import csv
import numpy
import math
import matplotlib.pyplot as plt
#"Month","Clearwater River at Kamiah, Idaho. 1911 ? 1965"
def radial_basis(x, x_j, s):
    '''
    An implementation of a radial basis function
    '''
    phi = math.exp(-((x - x_j)**2 / (2 * s**2)))
    return(phi)
def Post_Cov(phi, alpha, beta):
    '''
    Computes the posterior covariance
    '''
    S = (alpha * numpy.identity(phi.shape[1], dtype=float) + beta * phi.T * phi).I
    return S
def Post_Mean(phi, S, beta, T):
    '''
    Computes the posterior mean
    '''
    m = beta * S * phi.T @ T
    return m
def Pred_mean(m, phi_X):
    '''
    Computes the prediction mean
    '''
```

```python
    return m @ phi_X
def Pred_stdDev(phi_x, S, beta):
    '''
    Computes the prediction standard deviation
    '''
    sigma = 1/beta + phi_x @ S @ phi_x
    return sigma
def main():
    with open('clearwater.csv', 'r') as f:
        #Read in file
        reader = list(csv.reader(f, delimiter = ','))[1:]
        #Make into numpy array.
        data = numpy.array(reader)

    #Using int sequence instead of dates
    T = numpy.array(data[:,1], dtype=float)
    N = len(T)
    x = numpy.array([i for i in range(N)])
    #Prep for Radial basis functions
    M = 15
    x_min = min(x)
    x_max = max(x)
    s = (x_max - x_min)/(M-1)
    xjs = numpy.linspace(x_min, x_max, M-1)
    #matrix of ones
    phi = numpy.ones((N, M), dtype=float)

    #Filling matrix, keeping one column of ones
    for i in range(N):
        for j in range(1, M):
            phi[i, j] = radial_basis(x[i], xjs[j-1], s)
    phi = numpy.matrix(phi, dtype=float)

    #Hyperparameters
    alpha = 0.01
    beta = 0.07
    #Computes posterior distribution
    S_N = Post_Cov(phi, alpha, beta)
    m_N = Post_Mean(phi, S_N, beta, T)

    #1000 uniformly distributed points for plotting
    x_test = numpy.linspace(0, N, num = 1000)

    #Make prediction (mean and stddev)
    m_pred = [float(Pred_mean(m_N, numpy.append(1, numpy.array([radial_basis(x_i, x_j, s) \
    for x_j in xjs])))) for x_i in x_test]
    sigma_pred = [float(Pred_stdDev(numpy.append(1, numpy.array([radial_basis(x_i, x_j, s) \
    for x_j in xjs])), S_N, beta)) for x_i in x_test]
    #Prep for shaded area
    #plus minus one stdDev
    upper = [sum(x) for x in zip(m_pred, sigma_pred)]
    lower = [i - j for i, j in zip(m_pred, sigma_pred)]
```

```
    #Plotting
    plt.plot(x_test, m_pred, color = 'red')
    plt.scatter(x, T, color = 'blue')
    plt.fill_between(x_test, lower, upper, facecolor='brown', alpha=0.5)
    plt.show()
if __name__ == '__main__':
    main()
```