SML Assignment 1

Yi Zhang & Anton Kuznietsov
July 4, 2021

1 Task 1

1.1 1a Linear Features

1.The ridge coefficient is a representative positive value that controls how much Tikhonov regulation is applied. It's used to prevent overfitting and increase numerical stability.

2. To find the least squares solution **w**.

$$\mathbf{w}^* = arg_w min|\mathbf{X}\mathbf{w} - \mathbf{y}|^2 + \lambda|\mathbf{w}|^2 \tag{1}$$

Take the derivative and the gradient is equal to zero

$$\frac{\delta}{\delta \mathbf{w}}|\mathbf{X}\mathbf{w} - \mathbf{y}|^2 + \lambda|\mathbf{w}|^2 = 2\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{X}^T\mathbf{y} + 2\lambda\mathbf{w} = 0 \tag{2}$$

So **w** is represented by

$$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \tag{3}$$

3.Code in Python:

Listing 1: python code

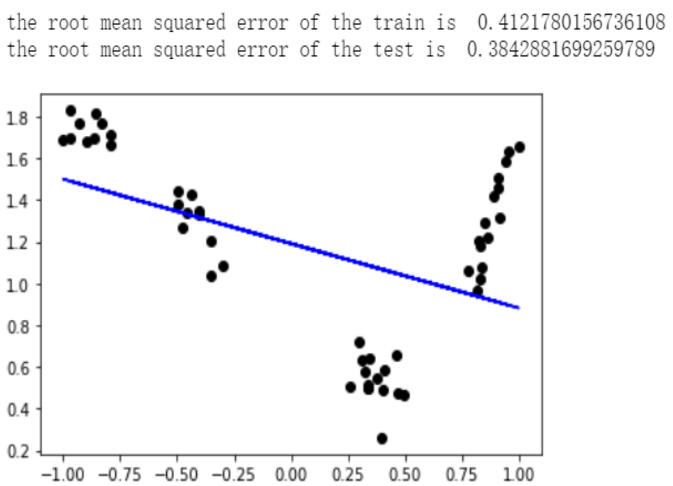
```
#!/usr/bin/env python
# coding: utf-8

# In[2]:


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

train_data=np.loadtxt("lin_reg_train.txt")#get data from lin_reg_train
test_data=np.loadtxt("lin_reg_test.txt")#get data from lin_reg_test




def get_xy(data):# according fomula of w,we need get the matrix of x and y
    N=len(data)      # the number of data
    x=np.empty((N,2))
    y=np.empty((N,1))
    for i in range(0,N):
```

```
        x[i][0]=data[i][0] # read the first column from data and give to the first column of x
        x[i][1]=1          #give the bias for x
        y[i][0]=data[i][1] # read the second column from data and give to the y
    return x,y

def CI(c):                 #use the ridge coefficient
    I=np.zeros([2,2])
    for j in range(0,2):
        I[j][j]=c
    return I

def get_w(x,y,ci):                   #according formula,get w
    x_transpose = np.transpose(x)    #get transpose matrix of x
    x_x=np.matmul(x_transpose, x)    # do the multiplication of x and transpose of x
    x_ci=x_x+ci
    x_inverse = np.linalg.inv(x_ci) # get the inverse
    x_y=np.matmul(x_transpose, y)
    w=np.matmul(x_inverse, x_y)
    return w

def predicted_value(x,w):          #get the predicted value
    x_transpose=np.empty((1,2))
    y=np.empty((len(x),1))
    for i in range(0,len(x)):
        x_transpose=x[i]
        y[i]=np.matmul(x_transpose,w) #y=x*w
    return y


def RMSE(y_pre,y):
    N=len(y_pre)
    sum=0
    for i in range(0,N):
        sum=sum+pow((y_pre[i]-y[i]),2)  # square and add the difference between the predicted value and

    result=math.sqrt(sum/N) # get the average and sqrt
    return result

def plot(x_real,y_real,y_predict):
    plt.scatter(x_real,y_real,c='#000000')
    plt.plot(x_real,y_predict,'b-')
    plt.show()

x,y=get_xy(train_data)   # extract x and y from train data
ci=CI(0.01)     #use the ridge coefficient
w=get_w(x,y,ci)    #return the value of w
print("the value of w is ",w)

y_pre=predicted_value(x,w)
RMSE_train=RMSE(y_pre,y)
print(" the root mean squared error of the train is ",RMSE_train)

x_test,y_test=get_xy(test_data)# extract x and y from train data
y_test_pre=predicted_value(x_test,w)
RMSE_test=RMSE(y_test_pre,y_test)
print(" the root mean squared error of the test is ",RMSE_test)

x_train_real=np.empty((len(x),1))
for i in range (len(x)):
    x_train_real[i]=x[i][0]
```

```
plot(x_train_real,y,y_pre)


# In[ ]:




# In[ ]:




# In[ ]:
```

4.
the root mean squared error of the train is 0.4121780156736108
the root mean squared error of the test is 0.3842881699259789

5.



1.2 1b Polynomial Features

1.Code in Python:

Listing 2: python code

```
#!/usr/bin/env python
# coding: utf-8
```

```
# In[10]:


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

train_data=np.loadtxt("lin_reg_train.txt")#get data from lin_reg_train
test_data=np.loadtxt("lin_reg_test.txt")#get data from lin_reg_test


def get_x_y(data,degree):         # according fomula of w,we need get the matrix of x and y
    N=len(data)
    d=degree+1
    x=np.empty((N,d))
    y=np.empty((N,1))
    for j in range(0,d):
        for i in range(0,N):
            if j==0:
                x[i][j]=1        #the first column is equal to 1
            elif j==1:
                x[i][j]=data[i][0] # the second column is equal to the first column from data
            else:
                x[i][j]=pow(data[i][0],j)#the n. column is equal to the x^(n-1)
    for i in range(0,N):
         y[i][0]=data[i][1] # read the second column from data and give to the y
    return x,y

def C_I(c,degree):
    d=degree+1
    I=np.zeros([d,d])
    for j in range(0,d):
        I[j][j]=c
    return I          #d+1 dimentions

def get_poly_w(x,y,ci):                     #according formula,get w
    x_transpose = np.transpose(x)  #get transpose matrix of x
    x_x=np.matmul(x_transpose, x)  # do the multiplication of x and transpose of x
    x_ci=x_x+ci
    x_inverse = np.linalg.inv(x_ci) # get the inverse
    x_y=np.matmul(x_transpose, y)
    w=np.matmul(x_inverse, x_y)
    return w


def predicted_poly_value(x,w,degree):#get the predicted value
    d=degree+1
    x_transpose=np.empty((1,d))
    y=np.empty((len(x),1))
    for i in range(0,len(x)):
        x_transpose=x[i]                #read each line
        y[i]=np.matmul(x_transpose,w)

    return y

def RMSE_poly(y_pre,y):
    N=len(y_pre)
    sum=0
    for i in range(0,N):
```

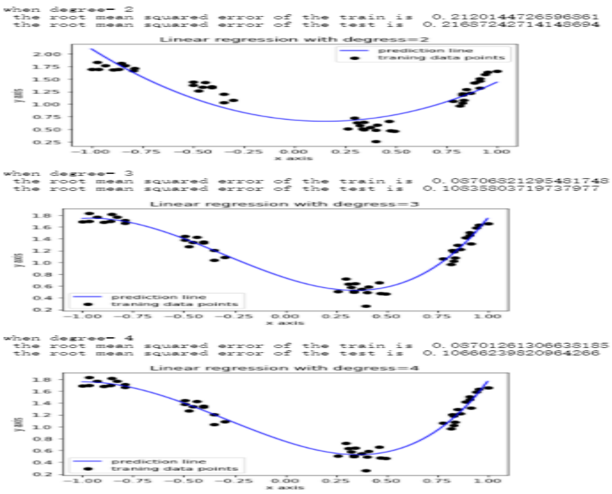
```
        sum=sum+pow((y_pre[i]-y[i]),2)# square and add the difference between the predicted value and t
    result=math.sqrt(sum/N)# get the average and sqrt
    return result


x_train_real=np.empty((len(train_data),1))
for i in range (len(train_data)):
    x_train_real[i]=train_data[i][0]# only have the real train data

for degree in range(2,5):
    x,y=get_x_y(train_data,degree)
    x_test,y_test=get_x_y(test_data,degree)
    ci=C_I(0.01,degree)    #use the ridge coefficient
    w_poly=get_poly_w(x,y,ci)   #return the value of w
    #print("when degree=",degree,"the value of w is  ",w_poly)

    y_pre=predicted_poly_value(x,w_poly,degree)
    y_pre_test=predicted_poly_value(x_test,w_poly,degree)
    RMSE_poly_train=RMSE_poly(y_pre,y)
    RMSE_poly_test=RMSE_poly(y_pre_test,y_test)
    print("when degree=",degree)
    print(" the root mean squared error of the train is ",RMSE_poly_train)
    print(" the root mean squared error of the test is ",RMSE_poly_test)

    #print(f"x shape:{x.shape}") #(50,3)
    fig, ax = plt.subplots()

    x_plot = np.linspace( np.min(x_train_real ), np.max(x_train_real) ,num=100)
    #print(f"x plot shape:{x_plot.shape}")
    x_plot = np.concatenate([x_plot.reshape(100, 1), np.ones(100).reshape(100, 1)], axis=1)
    #print(f"x plot shape:{x_plot.shape}")
    x_plot_m, _ = get_x_y(x_plot, degree)
    y_plot = predicted_poly_value(x_plot_m,w_poly,degree)

    ax.plot( x_plot.T[0], y_plot ,c = "blue" , label = 'prediction line')
    plt.scatter(x_train_real, y, c='#000000', label = 'traning data points')
    ax.set_title(f"Linear regression with degress={degree}")
    ax.set_xlabel('x axis')
    ax.set_ylabel('y axis')
    plt.legend()
    plt.show()
    #plot(x_train_real,y,y_pre)


# In[ ]:
```

In[ ]:

In[ ]:

2.when degree= 2
the root mean squared error of the train is 0.2120144726596861
the root mean squared error of the test is 0.21687242714148694
when degree= 3
the root mean squared error of the train is 0.08706821295481748
the root mean squared error of the test is 0.10835803719737977
when degree= 4
the root mean squared error of the train is 0.08701261306638185
the root mean squared error of the test is 0.10666239820964266

3.



4.Since the data model is linear in relation to the **w** parameters.For Bayesian linear regession, this is especially crucial.

1.3 1c Bayesian Linear Regression

1.

$$p(\mathbf{w}|\mathbf{X}, \mathbf{y}) = N(\boldsymbol{\mu}_n, \boldsymbol{\Lambda}_n^{-1}) \tag{4}$$

$$\boldsymbol{\mu}_n = \sigma^{-2}\boldsymbol{\Lambda}_n^{-1}\mathbf{X}^T\mathbf{y} \tag{5}$$

$$\boldsymbol{\Lambda}_n = \sigma^{-2}\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I} \tag{6}$$

2.

$$p(\mathbf{y}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{y}) = \int p(\mathbf{y}_*|\mathbf{X}_*, \mathbf{w})p(\mathbf{w}|\mathbf{X}, \mathbf{y})d\mathbf{w} \tag{7}$$

$$= N(\mathbf{X}_*\boldsymbol{\mu}_n, \sigma^2 + \mathbf{X}_*\boldsymbol{\Lambda}_n^{-1}\mathbf{X}_*^T) \tag{8}$$

3.Code in Python:

Listing 3: python code

```
#!/usr/bin/env python
# coding: utf-8

# In[4]:


import numpy as np
import math
import matplotlib.pyplot as plt
from scipy.stats import norm


B = 1/ (0.1 **2)
a = 0.01

train_data=np.loadtxt("lin_reg_train.txt")#get data from lin_reg_train
test_data=np.loadtxt("lin_reg_test.txt")#get data from lin_reg_test

def get_x__y(data):        # according fomula of w,we need get the matrix of x and y
    N=len(data)
    x=np.empty((2,N))
    y=np.empty((N,1))
    for i in range(0,N):
        x[0][i]=data[i][0] # read the first column from data and give to the first column of x
        x[1][i]=1          #give the bias for x
        y[i][0]=data[i][1] # read the second column from data and give to the y
    return x,y


def a_b(alpha,beta):                #get the λI
    c=alpha/beta
    I=np.zeros([2,2])
    for j in range(0,2):
        I[j][j]=c
    return I

def parameter_posterior(x,y,ci):        #get the w
    x_transpose = np.transpose(x)
    x_x=np.matmul(x,x_transpose)
    x_ci=x_x+ci
    x_inverse = np.linalg.inv(x_ci)
    x_y=np.matmul(x, y)
    w=np.matmul(x_inverse, x_y)
    return w

def predicted_value(x,w):           #predicte the value,according the formula y=xw
    x_transpose = np.transpose(x)
    x_i=np.empty((1,2))
    y=np.empty((len(x_transpose),1))
    for i in range(0,len(y)):
        x_i=x_transpose[i]
        y[i]=np.matmul(x_i,w)
    return y

def RMSE(y_pre,y):
    N=len(y_pre)
    sum=0
    for i in range(0,N):
        sum=sum+pow((y_pre[i]-y[i]),2)
```

```
    result=math.sqrt(sum/N)
    return result

def square(x_train, x_test, a, B):          #according formula, calculate the square
    x_transpose = np.transpose(x_train)
    x_test_transpose=np.transpose(x_test)
    x_x=np.matmul(x_train, x_transpose)
    B_xx=B*x_x                               #βΦΦ^T
    square=np.empty((len(x_test_transpose),1))
    aI=np.zeros([2,2])
    for j in range(0,2):
        aI[j][j]=a
    inverse=np.linalg.inv((aI+B_xx)) #α(I + βΦΦ^T−)^1
    for i in range(0,len(square)):
        x=x_test_transpose[i]
        x_t=np.transpose(x)
        square[i]=(1/B)+np.matmul((np.matmul(x,inverse)),x_t)  #β1/+ φ(□x) α(I + βΦΦ^T−)^1 φ (□x)

    return square

def Gaussian_Distribution(mean, square, y_data): #realize the formula of Parametric Density Models
    p=np.empty((len(mean),1))
    for i in range(0,len(square)):
        p1=1/(math.sqrt(2*math.pi*square[i]))
        p2=((−1)*pow((y_data[i]−mean[i]),2))/(2*square[i])
        p[i]=p1*math.exp(p2)

    return p                           #get the probability

def average_log_likelihood(p):
    for i in range(len(p)):
        if i==0:
            sumy=np.log(p[i]) # the first one only need calculate the exponential
        else:
            sumy=sumy+np.log(p[i])# get the sum of all exponential result

    average=sumy/len(p)
    return average



x_train_bayesian, y_train_bayesian=get_x__y(train_data)
test_x, test_y=get_x__y(test_data)

ci_bayesian=a_b(a,B)
w_posterior=parameter_posterior(x_train_bayesian, y_train_bayesian, ci_bayesian)#use train data to get w
test_predicted_value=predicted_value(test_x, w_posterior)  # get the predicted value of test data
#print(test_predicted_value)
test_p=Gaussian_Distribution(test_predicted_value, square(x_train_bayesian, test_x, a, B), test_y)
#print(test_p)
log_l_test=average_log_likelihood(test_p)
print("the log−likelihood of the test is", log_l_test)

#print("parameter of posterior to predicte test data ", w_posterior)
#print("predicted value are ", test_predicted_value)
print("RMSE of test data is", RMSE(test_predicted_value, test_y))

w_posterior_train=parameter_posterior(x_train_bayesian, y_train_bayesian, ci_bayesian)
train_predicted_value=predicted_value(x_train_bayesian, w_posterior_train)
```

```
train_p=Gaussian_Distribution(train_predicted_value,square(x_train_bayesian,x_train_bayesian,a,B),y_tra

log_l_train=average_log_likelihood(train_p)
print("the log-likelihood of the train is",log_l_train)
#print("parameter of posterior to predicte train data  ",w_posterior_train)
#print("predicted value are ",train_predicted_value)
print("RMSE of train data is",RMSE(train_predicted_value,y_train_bayesian))

fig,ax = plt.subplots()
x_ = np.linspace(np.min(x_train_bayesian[0]), np.max(x_train_bayesian[0]), num=100).reshape(100, 1)
x_ = np.concatenate([x_, np.ones(100).reshape(100, 1)], axis=1)
y_ = predicted_value(x_.T,w_posterior)

sig_ = square(x_.T,x_.T,a,B)
sig_ = np.sqrt( sig_ )

ax.scatter(x_.T[0], y_ ,c='blue', label='prediction')
ax.scatter(x_train_bayesian[0],y_train_bayesian ,c='black', label='original train data points')
for i in range(3):
    plt.fill_between(x_.T[0], y_.reshape(100) + sig_.reshape(100) * ((i + 1.)),
                     y_.reshape(100)  - sig_.reshape(100) * ((i + 1.)),
                     color="b", alpha=0.2)
ax.set_title(f"Bayesian Linear Regression ")
ax.set_xlabel('x axis')
ax.set_ylabel('y axis')

plt.legend()
plt.show()


# In[ ]:




# In[ ]:




# In[ ]:
```

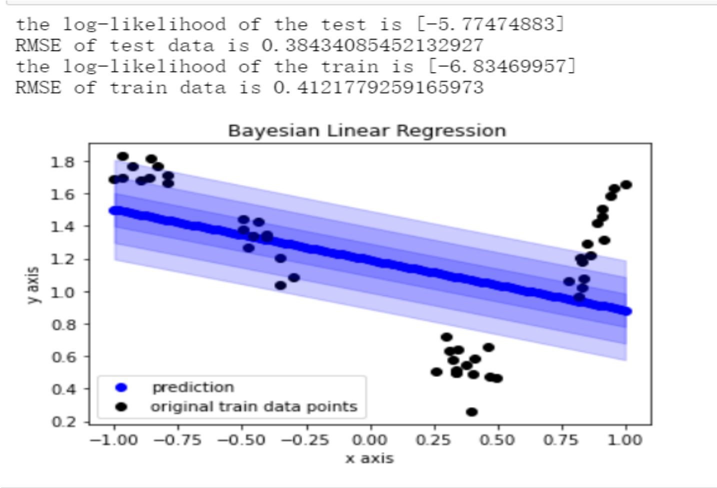
4.

RMSE of train data is 0.4121779259165973
RMSE of test data is 0.38434085452132927

5.

the log-likelihood of the train is [-6.83469957]
the log-likelihood of the test is [-5.77474883]

6.



7.We obtain the ideal value for the parameters **w** in linear regression by bringing the gradient of the squared error loss function to zero. Under Gaussian assumptions, this is comparable to the maximum likelihood point estimate probabilistically.In Bayesian linear regression, instead of utilizing the greatest likelihood estimate, we use Bayes'rule to obtain the full posterior distribution of the parameters **w**.Linear regression computes a single vector for **w**,Bayesian linear regression computes a full probability distribution for **w**.

1.4 1d Squared Exponential Features

1.Code in Python:

Listing 4: python code

```
#!/usr/bin/env python
# coding: utf-8

# In[5]:


import numpy as np
import math
import matplotlib.pyplot as plt
from scipy.stats import norm


B = 1/ (0.1 **2)
a = 0.01

train_data=np.loadtxt("lin_reg_train.txt")#get data from lin_reg_train
test_data=np.loadtxt("lin_reg_test.txt")#get data from lin_reg_test

def get_x__y(data):         # according fomula of w,we need get the matrix of x and y
    N=len(data)
    x=np.empty((2,N))
    y=np.empty((N,1))
    for i in range(0,N):
        x[0][i]=data[i][0]
        x[1][i]=1
        y[i][0]=data[i][1] # read the second column from data and give to the y
    return x,y
```

```
def Map(data):
    X_data = data[:, 0]
    y = data[:, 1]
    N = data.shape[0]
    k_dim = 20
    X_ =np.zeros([ X_data.shape[0], 20 ])
    for i in range(  X_data.shape[0]  ):
        for j in range( k_dim ):
            X_[i][j] =np.power( np.e ,  ( (-0.5 * 10) * ( ( X_data[i] - (j+1)*0.1 ) **2 ) ) )

    b = np.ones(N)
    X_ = np.concatenate([X_, b.reshape(N, 1)], axis=1)
    X_= X_.T
    return X_,y



def a_b(alpha,beta):                    #use the ridge coefficient
    c=alpha/beta
    I=np.zeros([21,21])
    for j in range(0,21):
        I[j][j]=c
    return I

def parameter_posterior(x,y,ci):
    x_transpose = np.transpose(x)
    x_x=np.matmul(x,x_transpose)
    x_ci=x_x+ci
    x_inverse = np.linalg.inv(x_ci)
    x_y=np.matmul(x, y)
    w=np.matmul(x_inverse, x_y)
    return w

def predicted_value(x,w):
    x_transpose = np.transpose(x)
    x_i=np.empty((1,2))
    y=np.empty((len(x_transpose),1))
    for i in range(0,len(y)):
        x_i=x_transpose[i]
        # print(f"x:{x.shape}")
        # print(f"w:{w.shape}")
        y[i]=np.matmul(x_i,w)
    return y

def RMSE(y_pre,y):
    N=len(y_pre)
    sum=0
    for i in range(0,N):
        sum=sum+pow((y_pre[i]-y[i]),2)

    result=math.sqrt(sum/N)
    return result

def square(x_train,x_test,a,B):
    x_transpose = np.transpose(x_train)
    x_test_transpose=np.transpose(x_test)
    x_x=np.matmul(x_train,x_transpose)
    B_xx=B*x_x
```

```
    square=np.empty((len(x_test_transpose),1))
    aI=np.zeros([21,21])
    for j in range(0,21):
        aI[j][j]=a
    inverse=np.linalg.inv((aI+B_xx))
    for i in range(0,len(square)):
        x=x_test_transpose[i]
        x_t=np.transpose(x)
        square[i]=(1/B)+np.matmul((np.matmul(x,inverse)),x_t)

    return square

def Gaussian_Distribution(mean,square,y_data):
    p=np.empty((len(mean),1))
    for i in range(0,len(square)):
        p1=1/(math.sqrt(2*math.pi*square[i]))
        p2=((-1)*pow((y_data[i]-mean[i]),2))/(2*square[i])
        p[i]=p1*math.exp(p2)
        #print(f"mean :{mean[i] }")
        #print(f"sigma:{square[i]}")


        #gauss = norm(loc=mean[i], scale=np.sqrt(square[i]))  # loc: mean □□□ scale: standard deviation
        #current_p_of_y = gauss.pdf(y_data[i])
        #print(current_p_of_y)
    return p

def average_log_likelihood(p): # get likelihood
    for i in range(len(p)):
        if i==0:
            sumy=np.log(p[i]) # the first one only need calculate the exponential
        else:
            sumy=sumy+np.log(p[i])# get the sum of all exponential result
        #print(f"p = {p[i]}")
    #print(f"sumy:{sumy}")
    average=sumy/len(p)
    return average



x_train_bayesian_ori,y_train_bayesian_ori=get_x__y(train_data)


x_train_bayesian,y_train_bayesian=Map(train_data)
test_x,test_y=Map(test_data)


ci_bayesian=a_b(a,B)
w_posterior=parameter_posterior(x_train_bayesian,y_train_bayesian,ci_bayesian)
test_predicted_value=predicted_value(test_x,w_posterior)
test_p=Gaussian_Distribution(test_predicted_value,square(x_train_bayesian,test_x,a,B),test_y)
log_l_test=average_log_likelihood(test_p)
print("the log-likelihood of the test is",log_l_test)

#print("parameter of posterior to predicte test data ",w_posterior)
#print("predicted value are ",test_predicted_value)
print("RMSE of test data is",RMSE(test_predicted_value,test_y))

w_posterior_train=parameter_posterior(x_train_bayesian,y_train_bayesian,ci_bayesian)
train_predicted_value=predicted_value(x_train_bayesian,w_posterior_train)
```

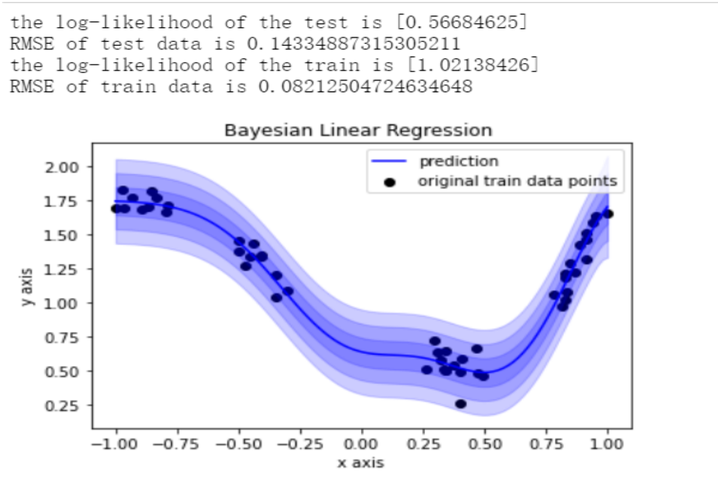
```
train_p=Gaussian_Distribution(train_predicted_value,square(x_train_bayesian,x_train_bayesian,a,B),y_tra

log_l_train=average_log_likelihood(train_p)
print("the log-likelihood of the train is",log_l_train)
#print("parameter of posterior to predicte train data  ",w_posterior_train)
#print("predicted value are ",train_predicted_value)
print("RMSE of train data is",RMSE(train_predicted_value,y_train_bayesian))

fig, ax = plt.subplots()
x_ = np.linspace(np.min(x_train_bayesian_ori[0]), np.max(x_train_bayesian_ori[0]), num=100).reshape(100
x_ = np.concatenate([x_, np.ones(100).reshape(100, 1)], axis=1)
x_maped, _ = Map(x_)
y_ = predicted_value(x_maped,w_posterior)


sig_p = square(x_maped,x_maped,a,B)
sig_p = np.sqrt( sig_p )

ax.plot(x_.T[0], y_ ,c='blue', label='prediction')
ax.scatter(x_train_bayesian_ori[0],y_train_bayesian_ori ,c='black', label='original train data points')
for i in range(3):
    plt.fill_between(x_.T[0], y_.reshape(100) + sig_p.reshape(100) * ((i + 1.)),
                     y_.reshape(100) - sig_p.reshape(100) * ((i + 1.)),
                     color="b", alpha=0.2)
ax.set_title(f"Bayesian Linear Regression ")
ax.set_xlabel('x axis')
ax.set_ylabel('y axis')

plt.legend()
plt.show()


# In[ ]:




# In[ ]:




# In[ ]:
```

2.
RMSE of train data is 0.08212504724634648
RMSE of test data is 0.14334887315305211

3.
the log-likelihood of the train is [1.02138426]

the log-likelihood of the test is [0.56684625]

4.



5.SE features are same as Gaussian basis functions, with α representing the mean and β representing the precision.

1.5 1e Cross validation

Cross validation: The learning set is randomly divided into n sets. The algorithm learns from n-1 sets and tests on the omitted set. This is done n time.
pro:
1) Cross Validation reduces overfitting,because the dataset is split into numerous folds and the algorithm is trained on each fold separately. This avoids the training dataset from being overfitted by our model. As a result, the model is able to generalize.
2) Cross Validation aids in the discovery of the ideal value of hyperparameters in order to improve the algorithm's efficiency.

cons:
1) Increase training time.Cross-validation greatly increases training time. Previously we trained the model on only one training set, but in cross-validation we had to train the model on multiple training sets.
2) Cross Validation requires expensive Computation. Cross Validation is computationally quite costly in terms of processing power.

1.6 reference

http://theprofessionalspoint.blogspot.com/2019/02/advantages-and-disadvantages-of-cross.html

References

[1] Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020, pp. 99–100.

[2] K. B. Petersen and M. S. Pedersen. *The Matrix Cookbook*. Version 20121115. Technical University of Denmark, 2012, p. 17. URL: `http://www2.compute.dtu.dk/pubdb/pubs/3274-full.html`.

[3] K. B. Petersen and M. S. Pedersen. *The Matrix Cookbook*. Version 20121115. Technical University of Denmark, 2012, p. 22. URL: `http://www2.compute.dtu.dk/pubdb/pubs/3274-full.html`.