CourseWork1 for Supervised Learning

Yihang OU 18101589
MSc Web Science and Big Data Analytics
Chenhao LU 18053689
MSc Web Science and Big Data Analytics

14/November/2018

1 Part1

1.1 Linear Regression

1. For each of the polynomial bases of dimension k=1,2,3,4 fit the data set of Figure 1 fit over the four data points(1,3),(2,2),(3.0),(4,5)

All for Question1 code in Appendix A

(a) we superimposing four different curves in Figure 1

(b) the equation corresponding to k=1 is $y = 2.5$
the equation corresponding to k=2 is $y = 0.4x + 1.5$
the equation corresponding to k=3 is $y = 1.5x^2 - 7.1x + 9.1$
the equation corresponding to k=4 is $y = 1.3x^3 - 8.5x^2 + 15.17x - 5$

(c) the MSE when k=1 is 3.25
the MSE when k=2 is 3.05
the MSE when k=3 is 0.8
the MSE when k=4 is $3.4901491 * 10^{-23}$

2. In this part we will illustrate the phenomena of overfitting

All for Question2 code in Appendix B

(a) we plot 6 curves in Figure 2, respectively are $y = sin^2(2\pi x)$ and $k = 2, 5, 10, 14, 18$ and many scatter points is drawn in Figure 2 which from $g_\sigma(x) := sin^2(2\pi x) + \varepsilon$ and where ε is a random variable distributed normally with mean 0 and variance σ^2

(b) we plot the log of training error versus the polynomial dimension $k = 1, ..., 18$ in Figure 3

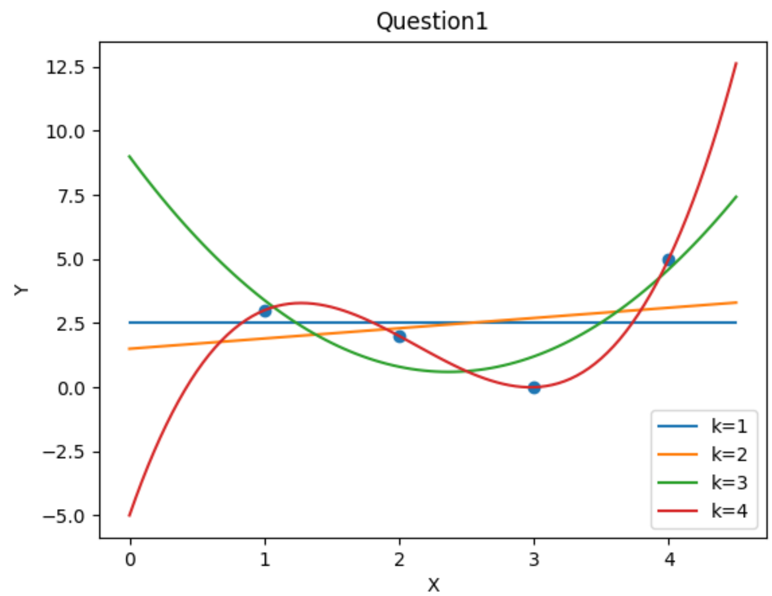

Figure 1: 1.a Linear Regression

(c) we generate a test set T of a thousand points, and plot the log of the test error versus the polynomial dimension $k = 1, ..., 18$ in Figure 4

(d) we random generate different training data and test data, we repeated items(b) and items(c) and run 100 times, we plot the $log(avg)$ of mse in Figure 5

3. Now use basis for (k=1,...,18)

$$sin(1\pi x), sin(2\pi x), sin(3\pi x), ..., sin(k\pi x)$$

Repeat the experiment in 2(b-d) with the above basis

All code for Question 3 in Appendix C

(a) like 2b, use above basis, we plot the mse with different k in Figure 6

(b) like 2c, use above basis, and plot the log of the test error versus the polynomial dimension $k = 1, ..., 18$ in Figure 7

(c) like 2d,we random generate different training data and test data, we repeated items(b) and items(c) and run 100 times, we plot the $log(avg)$ of mse in Figure 8

1

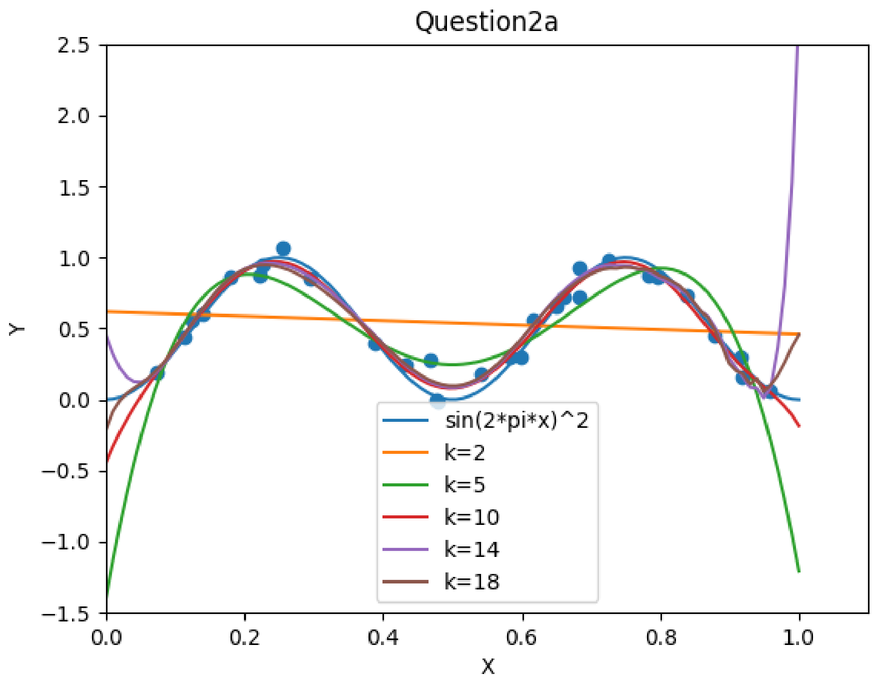

Figure 2: 2.a Question 2a

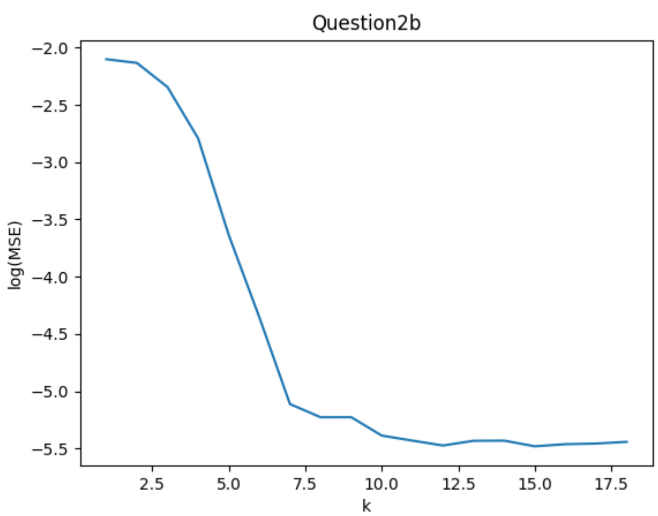

Figure 3: 2.b Question 2b

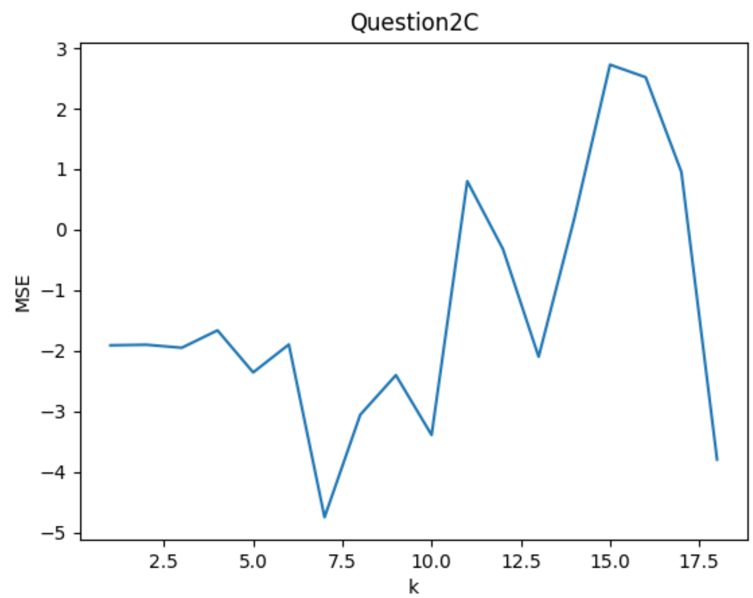

Figure 4: 2.c Question 2C

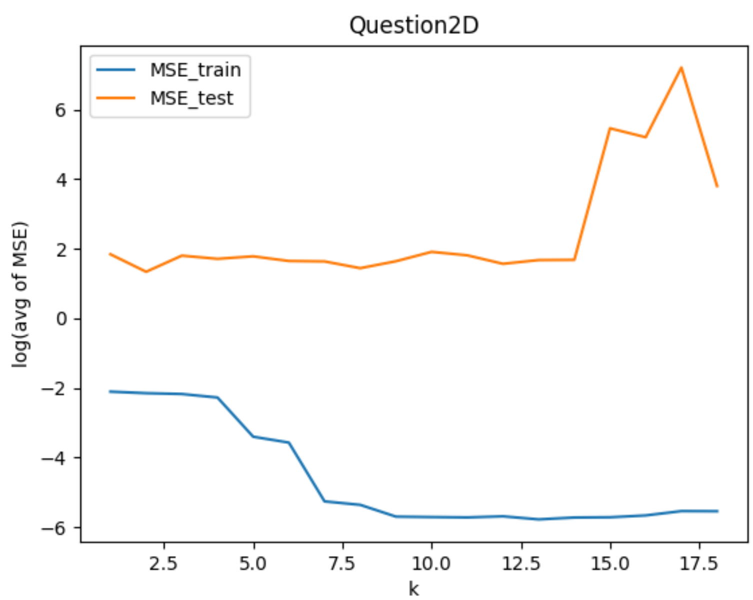

Figure 5: 2.d Question 2D

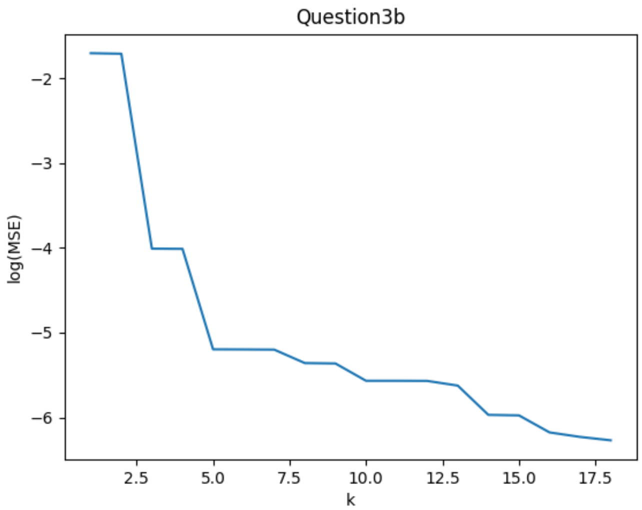

Figure 6: 3.b Question 3B

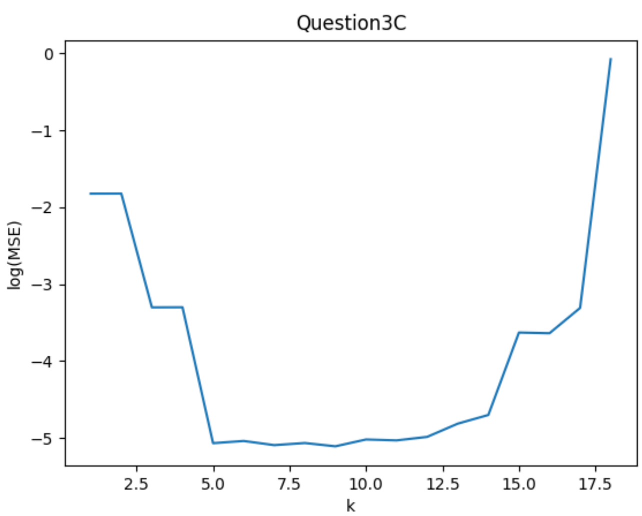

Figure 7: 3.c Question 3C

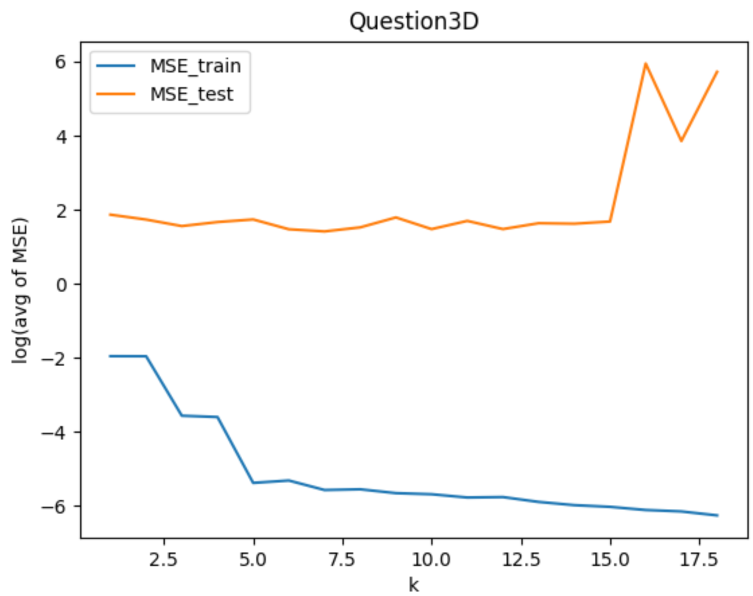

Figure 8: 3.d Question 3D

1.2 Boston Housing and kernels

4. Baseline versus full linear regression

All code for Question 4 in Appendix D

(a) Naive Regression Because we split the data set into 2/3 training set and 1/3 test set, the results are different each time. In one time,we calculate the MSE on the training set is 83.85412567690128, the MSE on the test set is 85.73318466671543.

(b) Give a simple interpretation of the constant function.
This constant function means we calculate the average of the y, the mean of y is we expect.

(c) Linear Regression with single attributes we plot the MSE of test set with each attributes in Figure 9.

(d) Linear Regression using all attributes
we using all of the data attributes at once. and we calculate the MSE of train is 23.296191. and the MSE of test is 21.810731, and the standard deviation of train MSE is 1.442286 and standard deviation of test MSE is 3.062832. As show in Table 1, Both in train set and test set, although some the MSE of single attribute is lower than the MSE of all attributes,

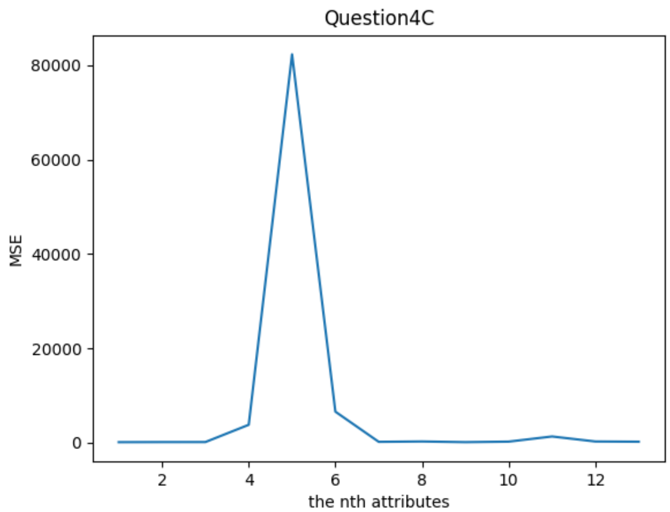

Figure 9: 4.c Question 4c

the standard deviation of MSE in all attributes is lower than single, so this method outperforms any of the individual regressors.

1.3 Kernelised ridge regression

5. Kernel Ridge Regression

All code for Question 5 in Appendix E and the function Question5D_1() and function Question5D_2() and function Question5D_3() in Appendix D use for Question 5(d)

1. using five-fold cross-validation to choose among all pairing of the values of γ and σ to select the best γ and σ Using five-fold cross-validation, we select the γ and σ when the cross-validation error is lowest, and the best γ is 2^{-26} and σ is 2^{13}, the results of each γ and σ and cross error stored in Q5_result.txt.

2. Plot the "cross-validation error we plot the "cross-validation error" as a function of γ and σ in Figure 10

3. Calculate the MSE on training and test sets for the best γ and σ The Best γ is 2^{-26} and σ is 2^{13}, MSE of training set is 39077.03642844244, MSE of test set is 18912.896638289247

4. Repeat "excise 4a,c,d" and "exercise 5c" over 20 random(2/3,1/3) splits of your data record the train/test error and the standard deviations of the

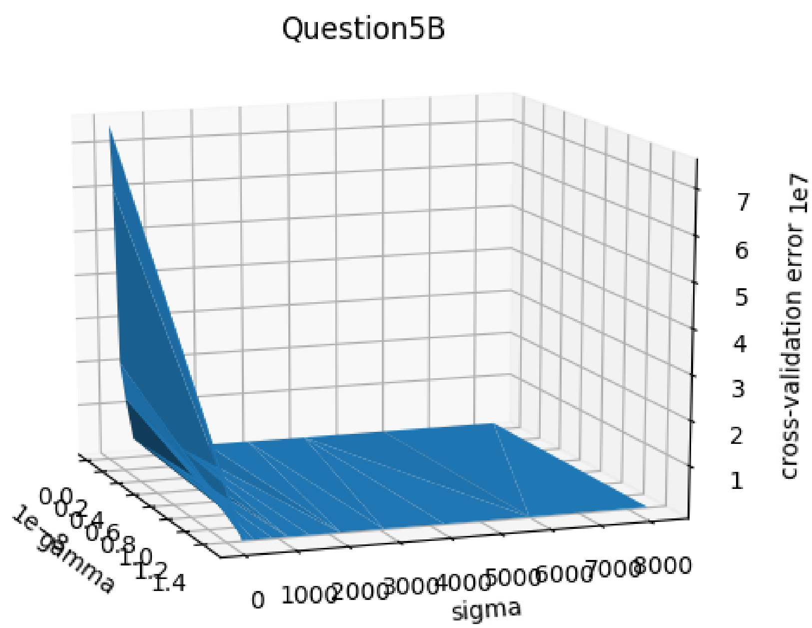

Figure 10: 5.b Question 5B

Method	MSE train	MSE test
Naive Regression	85.0215099 ± 5.300209	84.265901 ± 10.766989
Linear Regression(attribute 1)	3.828141 ± 17.11996	3.128918 ± 13.992950
Linear Regression(attribute 2)	4.115751 ± 18.406200	3.997990 ± 17.879556
Linear Regression(attribute 3)	3.338028 ± 14.928118	4.951336 ± 22.1430509
Linear Regression(attribute 4)	3.638778 ± 16.273110	$322.057991 \pm 1440.287124$
Linear Regression(attribute 5)	3.732333 ± 16.691502	$3784.435049 \pm 16924.508054$
Linear Regression(attribute 6)	1.773274 ± 7.930326	$502.199152 \pm 2245.902884$
Linear Regression(attribute 7)	3.577382 ± 15.998540	7.259205 ± 32.464154
Linear Regression(attribute 8)	3.795431 ± 16.973683	19.826130 ± 88.665151
Linear Regression(attribute 9)	3.472839 ± 15.53101	4.182220 ± 18.703456
Linear Regression(attribute 10)	3.269255 ± 14.620552	9.158062 ± 40.956102
Linear Regression(attribute 11)	3.411703 ± 15.257602	73.125048 ± 327.025158
Linear Regression(attribute 12)	3.739120 ± 16.721856	11.381800 ± 50.900960
Linear Regression(attribute 13)	2.142587 ± 9.581942	8.911042 ± 39.851392
Linear Regression(all attributes)	23.296191 ± 1.442286	21.810731 ± 3.062832
Kernel Ridge Regression	$23435.943881 \pm 3695.581415$	$45873.412581 \pm 2736.230894$

Table 1: MSE of train and test with different Methods

train/test errors and summarise these results in following type of table.
The results show in Table 1

2 Part II

6. Bayes estimator. In both of the following subquestions you will need to find the Bayes estimator with respect to the probability mass function $p(x, y)$ over (X, Y) where X and Y are finite thus $\sum_{x\in X}\sum_{y\in Y} p(x, y) =$ 1

(a) For this subquestion $Y = [k]$ and let $c \in [0, \infty)^k$ be a vector of k costs. Define $L_c : [k] \times [k] \to [0, \infty)$ as,

$$L_c(y, \hat{y}) := [y \neq \hat{y}]c_y$$

as the imbalanced classification loss function, i.e., if we don't predict the correct outcome y we suffer c_y loss. Derive the Bayes estimator.

Derivation:

Estimator: $\hat{y} = *arg\, min_y E([y \neq \hat{y}]c_y)$
$L(y, \hat{y}) = I(y \neq \hat{y}) = 0 \quad if \quad y = \hat{y}$
$L(y, \hat{y}) = I(y \neq \hat{y}) = c_y \quad if \quad y \neq \hat{y}$

The small loss on average is: $E(L(y, \hat{y})|x) = \sum_{y\in Y} L(y, \hat{y})\rho(\hat{y}|x)$

So that: E(L(y,$\hat{y}$)$|x) = \sum_{y\neq\hat{y}} \rho(y|x) = 1 - \rho(\hat{y}|x)$
$\hat{y} = *arg\, min_y E(L(y, \hat{y})|x) = *arg\, max_y \rho(y|x)$

Thus, y is taken to be the most likely value.

(b) For this subquestion $Y \subset \Re$. Let $L(y, \hat{y}) := |y - \hat{y}|$. Derive the Bayes estimator.

Derivation:

The median of $\rho(y|x)$ is the Bayes estimator with respect to absolute value loss

Proof: Choose $\hat{y}$ *to minimise* $E(L(y, \hat{y})|x) = \int_\theta |y - \hat{y}|\rho(y|x)d_y$
$= \int_{-\infty}^{\hat{y}} (\hat{y} - y)\rho(y|x)d_y + \int_{\hat{y}}^{\infty} (y - \hat{y})\rho(y|x)d_y$
So that, $0 = \frac{\partial}{\partial\hat{y}} E(L(y, \hat{y})|x) = \int_{-\infty}^{\hat{y}} \frac{\partial}{\partial\hat{y}} (\hat{y} - y)\rho(y|x)d_y + \int_{\hat{y}}^{\infty} \frac{\partial}{\partial\hat{y}} (y - \hat{y})\rho(y|x)d_y$

So that, $\int_{-\infty}^{\hat{y}} \rho(y|x)d_y = \int_{\hat{y}}^{\infty} \rho(y|x)d_y$

$2\int_{-\infty}^{\hat{y}} \rho(y|x)d_y = \int_{-\infty}^{\infty} \rho(y|x)d_y = 1$

Thus, $\int_{-\infty}^{\hat{y}} \rho(y|x)d_y = \frac{1}{2}$ *and* $\hat{y}$ *is the median of* $\rho(y|x)$.

7. Kernel modification Consider the function $K_c(x, z) := c + \sum_{i=1}^{n} x_i z_i$ **where** $x, z \in \Re^n$**.**

(a) For what values of $c \in \Re$ is K_c a positive semidefinite kernel? Given an argument supporting your claim.

According to Mercer's theorem, every semi-positive definite symmetric function is a kernel. And semi-positive definite symmetric functions correspond to a semi-positive definite symmetric Gram matrix.

Therefore, to show that $K_c(x, z)$ is a positive semidefinite kernel, it is equivalent to show that for any $x, z \in \Re^n$, the Gram matrix

$M(x, y) = c + x^2 c + xz$
$c + xzc + z^2$ is a positive semidefinite matrix.

Thus, $\det(M(x, y)) = (c + x^2)(c + z^2) - (c + xz)^2 = c(x - z)^2$

$\det(M(x, y)) \geq 0 \iff c \geq 0$

In conclusion, $c \geq 0$ *and* $c \in \Re$

(b) Suppose we use K_c as a kernel function with linear regression (least squares). Explain how c influences the solution.

$K(x, z) = \varphi(x)^T \varphi(z)$

$(x_i, z_i) \to (x_i, z_i + \frac{c}{nx_i})$

Thus, $z_i = kx_i + b$ before transformation to high-dimensional. After transformation, $z_i = kx_i + b + \frac{c}{n} x_i^{-1}$.

If we regard $x_i^{-1} = x_1,\ x_i = x_2$, then $z_i = \frac{c}{n} x_1 + kx_2 + b$

So, it is a multiple linear regression.

Thus, c transform the original linear regression function to multiple linear regression function. Besides, by recalculating $\frac{\partial SSE}{\partial b} = 0$ and $\frac{\partial SSE}{\partial k} = 0$, we can get new k and new b that minimise sum squares of error.

8. Suppose we prefrom linear regression with a Gaussian kernel $K_\beta(x,t) = exp(-\beta\|x-t\|^2)$ to train a classifier for two-class data (i.e., $y \in \{-1,1\}$). This classifier depends on the parameter β selected for the kernel. How should one choose β so that the learned linear classifier simulates a $1-NEAREST\ NEIGHBOR\ CLASSIFIER$? Give an argument supporting your resoning.

We consider a dataset $T = \{(x_1,y_1),(x_2,y_2),...,(x_N,y_N)\}$ and $y \in \{-1,+1\}$, we aim to build a model for linear regression, we use the dual representation, the model is $f(x) = sign(\sum_{i=1}^{N} \alpha_j y_j x_i \cdot x + b)$
As we know, in dual representation, the train set represented with inner dot formal, we got a matrix $G = [x_i \cdot x_j]_{N*N}$ we consider the Gaussian kernel
$K_\beta(x,t) = exp(-\beta\|x_i - x_j\|^2) = G = [x_i \cdot x_j]_{N*N}$ then we got $\beta = \frac{log x_i \cdot x_j}{\|x_i - x_j\|^2}$ So we find the relationship between inner dot of two vectors and the distance of two vectors

A Question 1

```
import matplotlib
matplotlib.use('TkAgg')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math

#generate the feature map
#use input data x and the dimension of polynomial bases k
def feature_map(x,k):
    X=[]
    for i in range(k+1):
        X.append(x**i)
    X=np.mat(X).T
    X=X.astype(np.float32)
    return X

#generate the weight w
#use the feature map X=feature_map(x,k) and the goal number
     y
def fit(x,y,k):
    X=np.around(feature_map(x,k).astype('float64'),decimals
         = 7) #feature map
    Y = np.array(y).reshape((len(y), 1))
    XT = X.transpose()
    w = np.dot(np.dot(np.linalg.inv(np.dot(XT, X)), XT), Y)
    return w

#predict function
#use the feature x and weight w and the dimension of
    polynomial bases k
def predict(x,w,k):
    X=feature_map(x,k)
    #print(X)
    y=X@w
    return y

#calculate MSE
#MSE=SSE/n
#input is y_true and y_predict
def cal_mse(y_t,y_p):
    result=[]
    if(y_t.size!=y_p.size):
        print("Input error")
    for i in range(len(y_t)):
        mse1=pow((y_t[i]-y_p[i]),2)
        result.append(mse1)
    mse=sum(result)/len(y_t)
    return mse

#Question 1
def Q1():
    #data
    data = np.array([(1, 3), (2, 2), (3, 0), (4, 5)])
    x = data[:, 0]
```

```
        y = data[:, 1]

        #wn is the weight of linear regression
        w1 = fit(x, y, 0)    #k=1
        w2 = fit(x, y, 1)    #k=2
        w3=fit(x, y, 2)      #k=3
        w4=fit(x, y, 3)      #k=4

        #generate the equation of different linear function
        print("k=1: ("+str(w1[0])+")")
        print("k=2: ("+str(w2[1])+")*x"+"+("+str(w2[0])+")")
        print("k=3: ("+str(w3[2])+")*x^2+("+str(w3[1])+")*x+("+str(w3[0])+")")
        print("k=4: ("+str(w4[3])+")*x^3+("+str(w4[2])+")*x^2+("+str(w4[1])+")*x+("+str(w4[0])+")")

        #plot a picture to illustrate the linear function
        xn=np.linspace(0,4.5,100)
        Y_t1=predict(xn,w1,0)
        Y_t2=predict(xn,w2,1)
        Y_t3=predict(xn,w3,2)
        Y_t4=predict(xn,w4,3)
        plt.figure(1)
        plt.scatter(x,y)
        l1,=plt.plot(xn,Y_t1)
        l2,=plt.plot(xn,Y_t2)
        l3,=plt.plot(xn,Y_t3)
        l4,=plt.plot(xn,Y_t4)
        plt.legend(handles=[l1, l2,l3,l4 ], labels=['k=1', 'k=2','k=3','k=4'], loc='best')
        plt.xlabel('X')
        plt.ylabel('Y')
        plt.title("Question1")
        #predict Y with different weight w and different polynomial bases
        Y_t1=predict(x,w1,0)
        Y_t2=predict(x,w2,1)
        Y_t3=predict(x,w3,2)
        Y_t4=predict(x,w4,3)

        print("k=1,the mse="+str(cal_mse(y,Y_t1)))
        print("k=2,the mse="+str(cal_mse(y,Y_t2)))
        print("k=3,the mse="+str(cal_mse(y,Y_t3)))
        print("k=4,the mse="+str(cal_mse(y,Y_t4)))
        plt.show()
```

B Question 2

```
import matplotlib
matplotlib.use('TkAgg')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
import math

#generate the feature map
#use input data x and the dimension of polynomial bases k
def feature_map(x,k):
    X=[]
    for i in range(k+1):
        X.append(x**i)
    X=np.mat(X).T
    X=X.astype(np.float32)
    return X

#generate the weight w
#use the feature map X=feature_map(x,k) and the goal number y
def fit(x,y,k):
    X=np.around(feature_map(x,k).astype('float64'),decimals = 7) #feature map
    Y = np.array(y).reshape((len(y), 1))
    XT = X.transpose()
    w = np.dot(np.dot(np.linalg.inv(np.dot(XT, X)), XT), Y)
    return w

#predict function
#use the feature x and weight w and the dimension of polynomial bases k
def predict(x,w,k):
    X=feature_map(x,k)
    #print(X)
    y=X@w
    return y

#calculate MSE
#MSE=SSE/n
#input is y_true and y_predict
def cal_mse(y_t,y_p):
    result=[]
    if(y_t.size!=y_p.size):
        print("Input error")
    for i in range(len(y_t)):
        mse1=pow((y_t[i]-y_p[i]),2)
        result.append(mse1)
    mse=sum(result)/len(y_t)
    return mse

#Question2(a)
def Question2A():
    #y1=sin(2*pi*x)^2+noise
    np.random.seed(50)
    noise=np.random.normal(0,0.07,30)
    x1=np.random.uniform(0,1,30)
    y1=np.square(np.sin(2*math.pi*x1))
    i=0
    while(i<y1.size):
        y1[i]=y1[i]+noise[i]
        i=i+1
```

```
        #y_t=sin(2*pi*x)^2
        xn=np.linspace(0,1,100)
        y_t=np.square(np.sin(2*math.pi*xn))
        #weight w when k=2,k=5,k=10,k=14,k=18
        w1=fit(x1,y1,1)
        w2=fit(x1,y1,4)
        w3=fit(x1,y1,9)
        w4=fit(x1,y1,13)
        w5=fit(x1,y1,17)
        plt.figure(1)
        plt.scatter(x1,y1)
        xn=np.linspace(0,1,100)
        Y_t1=predict(xn,w1,1)
        Y_t2=predict(xn,w2,4)
        Y_t3=predict(xn,w3,9)
        Y_t4=predict(xn,w4,13)
        Y_t5=predict(xn,w5,17)

        l1,=plt.plot(xn,y_t)
        l2,=plt.plot(xn,Y_t1)
        l3,=plt.plot(xn,Y_t2)
        l4,=plt.plot(xn,Y_t3)
        l5,=plt.plot(xn,Y_t4)
        l6,=plt.plot(xn,Y_t5)
        plt.xlabel('X')
        plt.ylabel("Y")
        plt.title("Question2a")
        plt.legend(handles=[l1, l2, l3, l4,l5,l6],
                   labels=['sin(2*pi*x)^2', 'k=2', 'k=5',
                           'k=10','k=14','k=18'], loc='best')
        plt.xlim(0,1.1)
        plt.ylim(-1.5,2.5)
        plt.show()

    #Question2B
    def Question2B():
        np.random.seed(20)
        #y1=sin(2*pi*x)^2+noise
        noise=np.random.normal(0,0.07,30)
        x1=np.random.uniform(0,1,30)
        y1=np.square(np.sin(2*math.pi*x1))

        i=0
        while(i<y1.size):
            y1[i]=y1[i]+noise[i]
            i=i+1
        #create a mse for MSE
        mse=[]
        for i in range(18):
            w=fit(x1,y1,i)
            Y_t=predict(x1,w,i)
            mse.append(math.log(cal_mse(y1,Y_t)))

        plt.figure(1)
        plt.title("Question2b")
        plt.xlabel("k")
        plt.ylabel("log(MSE)")
```

```
        x_mse=np.linspace(1,18,18)
        plt.plot(x_mse,mse)
        plt.show()

    #Question2C
    def Question2C():
        #when k=1,...,18
        #train data generate the w
        def train(k):
            np.random.seed(30)
            s = np.random.normal(0, 0.07, 30)

            x1 = np.random.uniform(0, 1, 30)
            x1.sort()
            y1 = np.square(np.sin(2 * math.pi * x1))

            i = 0
            while (i < y1.size):
                y1[i] = y1[i] + s[i]
                i = i + 1

            w = fit(x1, y1, k)
            return w

        #use the w from train to compute the y_predict
        #compute the MSE in different k
        def test(k):
            w = train(k)
            np.random.seed(30)
            s = np.random.normal(0, 0.07, 1000)

            x1 = np.random.uniform(0, 1, 1000)
            y1 = np.square(np.sin(2 * math.pi * x1))
            i = 0
            while (i < y1.size):
                y1[i] = y1[i] + s[i]
                i = i + 1

            y_t = predict(x1, w, k)
            y_t = np.array(y_t)
            mse = cal_mse(y_t, y1)
            mse_log = math.log(mse)
            return mse_log

        #create list for mse
        res_mse = []
        for i in range(18):
            res_mse.append(test(i))

        plt.figure(1)
        x = np.linspace(1, 18, 18)
        plt.plot(x, res_mse)
        plt.xlabel("k")
        plt.ylabel("MSE")
        plt.title("Question2C")
        plt.show()
```

```
def Question2D():
    #create two list for MSE_train and MSE_test
    MSE_train=[]
    MSE_test=[]
    #loop for k from 1 to 18
    for k in range(18):
        print("current K is "+str(k) )
        mse_train=0
        mse_test=0
        #run train and test 100 times
        #compute each time mse
        for m in range(100):
            #train data
            s = np.random.normal(0, 0.07, 30)

            x1 = np.random.uniform(0, 1, 30)
            y1 = np.square(np.sin(2 * math.pi * x1))
            i = 0
            while (i < y1.size):
                y1[i] = y1[i] + s[i]
                i = i + 1

            w=fit(x1,y1,k)
            Y_t1=predict(x1,w,k)
            #train finish!
            #compute mse of train
            mse_train+=cal_mse(y1,Y_t1)

            #test data
            s2=np.random.normal(0,0.07,1000)
            x2=np.random.uniform(0,1,1000)
            y2=np.square(np.sin(2*math.pi*x2))
            n=0
            while(n<y2.size):
                y2[i]=y2[i]+s2[i]
                n=n+1
            #"start test!"
            Y_t=predict(x2,w,k)
            mse_test=mse_test+cal_mse(y2,Y_t)

        #compute the log(average of MSE)
        Mse_train=math.log(mse_train/100)
        Mse_test=math.log(mse_test/100)
        MSE_test.append(Mse_test)
        MSE_train.append(Mse_train)
    plt.figure(1)
    x=np.linspace(1,18,18)
    l1,=plt.plot(x,np.array(MSE_train))
    l2,=plt.plot(x,np.array(MSE_test))
    plt.title("Question2D")
    plt.xlabel("k")
    plt.ylabel("log(avg of MSE)")
    plt.legend(handles=[l1, l2,], labels=['MSE_train', '
        MSE_test'], loc='best')
    plt.show()
```

C Question 3

```
import matplotlib
matplotlib.use('TkAgg')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math

#generate the feature map
#use input data x and the dimension of sin(n*pi*x) bases k
def feature_map2(x,k):
    X=[]
    for i in range(k+1):
        X.append(np.sin((i+1)*np.pi*x))
    X=np.mat(X).T
    X=X.astype(np.float32)
    return X

#generate the weight w
#use the feature map X=feature_map(x,k) and the goal number y
def fit(x,y,k):
    X=np.around(feature_map2(x,k).astype('float64'), decimals = 7) #feature map
    Y = np.array(y).reshape((len(y), 1))
    XT = X.transpose()
    w = np.dot(np.dot(np.linalg.inv(np.dot(XT, X)), XT), Y)
    return w

#predict function
#use the feature x and weight w and the dimension of polynomial bases k
def predict(x,w,k):
    X=feature_map2(x,k)
    #print(X)
    y=X@w
    return y

#calculate MSE
#MSE=SSE/n
#input is y_true and y_predict
def cal_mse(y_t,y_p):
    result=[]
    if(y_t.size!=y_p.size):
        print("Input error")
    for i in range(len(y_t)):
        mse1=pow((y_t[i]-y_p[i]),2)
        result.append(mse1)
    mse=sum(result)/len(y_t)
    return mse

#Question3b
def Question3b():
    np.random.seed(20)
    #y1=sin(2*pi*x)^2+noise
    noise=np.random.normal(0,0.07,30)
```

```
    x1=np.random.uniform(0,1,30)
    y1=np.square(np.sin(2*math.pi*x1))

    i=0
    while(i<y1.size):
        y1[i]=y1[i]+noise[i]
        i=i+1
    #create a mse for MSE
    mse=[]
    for i in range(18):
        w=fit(x1,y1,i)
        Y_t=predict(x1,w,i)
        mse.append(math.log(cal_mse(y1,Y_t)))

    plt.figure(1)
    plt.title("Question3b")
    plt.xlabel("k")
    plt.ylabel("log(MSE)")
    x_mse=np.linspace(1,18,18)
    plt.plot(x_mse,mse)
    plt.show()

#Question3c
def Question3c():
    # when k=1,...,18
    # train data generate the w
    def train(k):
        np.random.seed(30)
        s = np.random.normal(0, 0.07, 30)

        x1 = np.random.uniform(0, 1, 30)
        x1.sort()
        y1 = np.square(np.sin(2 * math.pi * x1))

        i = 0
        while (i < y1.size):
            y1[i] = y1[i] + s[i]
            i = i + 1

        w = fit(x1, y1, k)
        return w

    # use the w from train to compute the y_predict
    # compute the MSE in different k
    def test(k):
        w = train(k)
        np.random.seed(30)
        s = np.random.normal(0, 0.07, 1000)

        x1 = np.random.uniform(0, 1, 1000)
        y1 = np.square(np.sin(2 * math.pi * x1))
        i = 0
        while (i < y1.size):
            y1[i] = y1[i] + s[i]
            i = i + 1

        y_t = predict(x1, w, k)
```

```
            y_t = np.array(y_t)
            mse = cal_mse(y_t, y1)
            mse_log = math.log(mse)
            return mse_log

        # create list for mse
        res_mse = []
        for i in range(18):
            res_mse.append(test(i))

        plt.figure(1)
        x = np.linspace(1, 18, 18)
        plt.plot(x, res_mse)
        plt.xlabel("k")
        plt.ylabel("log(MSE)")
        plt.title("Question3C")
        plt.show()

    def Question3d():
        #create two list for MSE_train and MSE_test
        MSE_train=[]
        MSE_test=[]
        #loop for k from 1 to 18
        for k in range(18):
            print("current K is "+str(k) )
            mse_train=0
            mse_test=0
            #run train and test 100 times
            #compute each time mse
            for m in range(100):
                #train data
                s = np.random.normal(0, 0.07, 30)

                x1 = np.random.uniform(0, 1, 30)
                y1 = np.square(np.sin(2 * math.pi * x1))
                i = 0
                while (i < y1.size):
                    y1[i] = y1[i] + s[i]
                    i = i + 1

                w=fit(x1,y1,k)
                Y_t1=predict(x1,w,k)
                #train finish!
                #compute mse of train
                mse_train+=cal_mse(y1,Y_t1)

                #test data
                s2=np.random.normal(0,0.07,1000)
                x2=np.random.uniform(0,1,1000)
                y2=np.square(np.sin(2*math.pi*x2))
                n=0
                while(n<y2.size):
                    y2[i]=y2[i]+s2[i]
                    n=n+1
                #"start test!"
                Y_t=predict(x2,w,k)
                mse_test=mse_test+cal_mse(y2,Y_t)
```

```
            #compute the log(average of MSE)
            Mse_train=math.log(mse_train/100)
            Mse_test=math.log(mse_test/100)
            MSE_test.append(Mse_test)
            MSE_train.append(Mse_train)
        plt.figure(1)
        x=np.linspace(1,18,18)
        l1,=plt.plot(x,np.array(MSE_train))
        l2,=plt.plot(x,np.array(MSE_test))
        plt.title("Question3D")
        plt.xlabel("k")
        plt.ylabel("log(avg of MSE)")
        plt.legend(handles=[l1, l2,], labels=['MSE_train', 'MSE_test'], loc='best')
        plt.show()
```

D Question 4

```
import matplotlib
matplotlib.use('TkAgg')
import numpy as np
import matplotlib.pyplot as plt
import scipy.io as sio

#load the data
input=sio.loadmat('boston.mat')
data=input['boston']

#generate the feature map
#use input data x and the dimension of polynomial bases k
def feature_map(x,k):
    X=[]
    for i in range(k+1):
        X.append(x**i)
    X=np.mat(X).T
    X=X.astype(np.float32)
    return X

#use for linear regression for many attributes
def fit2(x,y):
    X=x
    Y=y.reshape((len(y),1))
    XT=x.transpose()
    w = np.dot(np.dot(np.linalg.inv(np.dot(XT, X)), XT), Y)
    return w

#generate the weight w
#use the feature map X=feature_map(x,k) and the goal number y
def fit(x,y,k):
    X=np.around(feature_map(x,k).astype('float64'),decimals = 7) #feature map
```

```
        Y = np.array(y).reshape((len(y), 1))
        XT = X.transpose()
        w = np.dot(np.dot(np.linalg.inv(np.dot(XT, X)), XT), Y)
        return w

    #predict function
    #use the feature x and weight w and the dimension of
        polynomial bases k
    def predict(x,w,k):
        X=feature_map(x,k)
        #print(X)
        y=X@w
        return y

    #predict function in many attributes
    def predict2(x,w):
        y=np.dot(x,w)
        return y

    #calculate MSE
    #MSE=SSE/n
    #input is y_true and y_predict
    def cal_mse(y_t,y_p):
        result=[]
        if(y_t.size!=y_p.size):
            print("Input error")
        for i in range(len(y_t)):
            mse1=pow((y_t[i]-y_p[i]),2)
            result.append(mse1)
        mse=sum(result)/len(y_t)
        return mse

    #Calculte MSE for train set and test set at the same time
    def Calculate_MSE(y_train,y_test,y_mean):
        result_train=[]
        result_test=[]
        for i in range(len(y_train)):
            mse1=pow((y_train[i]-y_mean),2)
            result_train.append(mse1)
        train_mse=sum(result_train)/len(y_train)
        for i in range(len(y_test)):
            mse2=pow((y_test[i]-y_mean),2)
            result_test.append(mse2)

        test_mse=sum(result_test)/len(y_test)
        return train_mse,test_mse

    #for Question A
    def QuestionA():
        MSE=[0,0]
        for loop in range(20):
            #split data
            number1=round(data.shape[0]*2/3)
            np.random.shuffle(data)
            training, test = data[:number1,:], data[number1:,:]

            y_train=training[:,13]
```

```
        y_test=test[:,13]
        y_mean=y_train.mean()
        mse_train,mse_test=Calculate_MSE(y_train,y_test,
            y_mean)
        MSE[0]+=mse_train
        MSE[1]+=mse_test
    MSE[0]=MSE[0]/20
    MSE[1]=MSE[1]/20

    print("MSE of training set is " +str(MSE[0]))
    print("MSE of test set is "+str(MSE[1]))

#QuestionC
def QuestionC():
    mse=[0.0]*13
    mse=np.array(mse)
    for i in range(len(mse)):
        print("feature column now "+str(i))
        for loop in range(20):
            #split data
            number1=round(data.shape[0]*2/3)
            np.random.shuffle(data)
            training, test = data[:number1,:], data[number1
                :,:]
            y_train=training[:,13]
            x_train=training[:,i]
            y_test=test[:,13]
            x_test=test[:,0]
            #train
            w=fit(x_train,y_train,1)
            #predict and test
            y_predict1=predict(x_test,w,1)
            mse1=cal_mse(y_t=y_test,y_p=y_predict1)
            mse[i]+=mse1
        mse[i]=mse[i]/20
    #figure
    plt.figure(1)
    x=np.linspace(1,13,13)
    plt.xlabel("the nth attributes")
    plt.ylabel("MSE")
    plt.title("Question4C")
    plt.show()

#QuestionD
def QuestionD(data):
    mse=0
    for loop in range(20):
        print(loop)
        number1 = round(data.shape[0] * 2 / 3)
        number2 = data.shape[0] - number1
        np.random.shuffle(data)
        training, test = data[:number1, :], data[number1:,
            :]
        y_train = training[:, 13]
        x_train = training[:, :13]
        x_test=test[:,:13]
        y_test=test[:,13]
```

```
        ones1=np.ones((number1,1))
        ones2=np.ones((number2,1))
        x_train=np.hstack((x_train,ones1))
        x_test=np.hstack((x_test,ones2))
        w = fit2(x_train,y_train)
        y=predict2(x_test,w)
        mse1=cal_mse(y_test,y)
        mse+=mse1
    print("mse is "+str(mse/20))

#Question for 5D
#repeate exercise 4a over 20 random (2/3,1/3) splits of your data
def Question5D_1():
    MSE_test=[0.0]*20
    MSE_train=[0.0]*20
    MSE_train=np.array(MSE_train)
    MSE_test=np.array(MSE_test)
    for loop in range(20):
        #split data
        number1=round(data.shape[0]*2/3)
        np.random.shuffle(data)
        training, test = data[:number1,:], data[number1:,:]
        y_train=training[:,13]
        y_test=test[:,13]
        y_mean=y_train.mean()
        #calculte the MSE
        mse_train,mse_test=Calculate_MSE(y_train,y_test,y_mean)
        MSE_train[loop]=mse_train
        MSE_test[loop]=mse_test

    Mse_train=MSE_train.mean()
    Mse_test=MSE_test.mean()
    STD_train=np.std(MSE_train,ddof=1)
    STD_test=np.std(MSE_test,ddof=1)

    #print the results
    print("The average MSE of training set is " +str(Mse_test))
    print("The average MSE of test set is "+str(Mse_train))
    print("The standard deviarions of train error is " + str(STD_train))
    print("The standard deviarions of test error is " + str(STD_test))

#Question for 5D
#repeate exercise 4c over 20 random (2/3,1/3) splits of your data
def Question5D_2():
    result=[[0.0]*4]*13
    result=np.array(result)
    for i in range(13): #i is the nth attribute
        print("feature column now "+str(i))
        MSE_test = [0.0] * 20
        MSE_train = [0.0] * 20
        MSE_train = np.array(MSE_train)
```

```
            MSE_test = np.array(MSE_test)
            for loop in range(20):
                #split data
                number1=round(data.shape[0]*2/3)
                np.random.shuffle(data)
                training, test = data[:number1,:], data[number1:,:]
                y_train=training[:,13]
                x_train=training[:,i]
                y_test=test[:,13]
                x_test=test[:,0]
                #train
                w=fit(x_train,y_train,1)
                #predict and test
                y_predict1=predict(x_test,w,1)
                y_predict2=predict(x_train,w,1)
                #calculate MSE
                mse_test=cal_mse(y_t=y_test,y_p=y_predict1)
                mse_train=cal_mse(y_train,y_predict2)
                MSE_test[i] = mse_test
                MSE_train[i] = mse_train

            Mse_train = MSE_train.mean()
            Mse_test = MSE_test.mean()
            STD_train = np.std(MSE_train, ddof=1)
            STD_test = np.std(MSE_test, ddof=1)

            result[i,0]=Mse_train
            result[i,1]=Mse_test
            result[i,2]=STD_train
            result[i,3]=STD_test

        #print the results
        for m in range(13):
            print("With Attribute "+str(m+1))
            print("The average MSE of training set is " + str(result[m,0]))
            print("The average MSE of test set is " + str(result[m,1]))
            print("The standard deviarions of train error is " + str(result[m,2]))
            print("The standard deviarions of test error is " + str(result[m,3]))

    #Question for 5D
    #repeate exercise 4d over 20 random (2/3,1/3) splits of your data
    def Question5D_3():
        #create two list for store the MSE_train and MSE_test
        MSE_test = [0.0] * 20
        MSE_train = [0.0] * 20
        MSE_train = np.array(MSE_train)
        MSE_test = np.array(MSE_test)
        for loop in range(20):
            #print(loop)
            #split data
            number1 = round(data.shape[0] * 2 / 3)
```

```
            number2 = data.shape[0] - number1
            np.random.shuffle(data)     #shuffle data so that each time data set is different
            training, test = data[:number1, :], data[number1:, :]
            y_train = training[:, 13]
            x_train = training[:, :13]
            x_test=test[:,:13]
            y_test=test[:,13]
            ones1=np.ones((number1,1))
            ones2=np.ones((number2,1))
            x_train=np.hstack((x_train,ones1))
            x_test=np.hstack((x_test,ones2))
            #train
            w = fit2(x_train,y_train)
            #predict &test
            y1=predict2(x_test,w)
            y2=predict2(x_train,w)
            #calculate the MSE
            mse_test=cal_mse(y_test,y1)
            mse_train=cal_mse(y_train,y2)
            MSE_test[loop] = mse_test
            MSE_train[loop] = mse_train

        Mse_train = MSE_train.mean()
        Mse_test = MSE_test.mean()
        STD_train = np.std(MSE_train, ddof=1)
        STD_test = np.std(MSE_test, ddof=1)

        print("The average MSE of training set is " +str(Mse_test))
        print("The average MSE of test set is "+str(Mse_train))
        print("The standard deviarions of train error is " + str(STD_train))
        print("The standard deviarions of test error is " + str(STD_test))
```

E Question 5

```
import scipy.io as sio
import matplotlib
matplotlib.use('TkAgg')
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

#input data
input=sio.loadmat('boston.mat')
data=input['boston']

#compute the distance between two vectors
def distance(x1,x2):
    distance=np.linalg.norm(x1-x2)
```

```
            return distance

    #compute the kernel function
    #return kernel function k
    def kernel(x1,x2,sigma):
        m1=x1.shape[0]
        m2=x2.shape[0]
        k=[[0.0]*m2]*m1
        k=np.array(k)
        for i in range(m1):
            for j in range(m2):
                k[i][j] = np.exp(-pow(distance(x1[i], x2[j]), 2) / (2 * pow(sigma, 2)))
        return k

    #compute a_star
    def a_star(x,y,K,gamma):
        m,n=x.shape
        I=np.eye(m,dtype=float)
        temp=gamma*m*I
        a=np.linalg.inv(K+temp)@y
        return a

    #predict with x_train and a
    def predict(x_train,x_test,a,sigma):
        K=kernel(x_train,x_test,sigma)
        a=np.array([a])
        y=a@K
        return y

    #calcalute MSE
    def cal_mse(y_true,y_train):
         MSE=np.sum(np.power((y_train.reshape(-1,1) - y_true),2))/len(y_train)
         return MSE


    #plot a 3D picture for sigma and gamma
    def plot_3d(result):
        fig = plt.figure()
        ax = fig.gca(projection='3d')
        X=result[:,0]
        Y=result[:,1]
        Z=result[:,2]
        ax.plot_trisurf(X,Y,Z)
        ax.set_xlabel('gamma')
        ax.set_ylabel('sigma')
        ax.set_zlabel('cross-validation error')
        plt.title("Question5B")
        plt.show()

    #split data for 5-fold cross validation
    #k from 0 to 4 to get different data set
    def split_data(k):
        number1 = round(data.shape[0] * 1 / 5)
        number2=round(data.shape[0] * 2 / 5)
        number3=round(data.shape[0] * 3 / 5)
```

```
    number4=round(data.shape[0] * 4 / 5)
    y_train=0
    x_train=0
    y_test=0
    x_test=0
    if(k==0):
        test, training = data[:number1, :], data[number1:, :]
        y_train = training[:, 13]
        x_train = training[:, :13]
        y_test = test[:, 13]
        x_test = test[:, :13]
    elif(k==1):
        test= data[number1:number2, :]
        training=np.vstack((data[:number1,:],data[number2:,:]))
        y_train = training[:, 13]
        x_train = training[:, :13]
        y_test = test[:, 13]
        x_test = test[:, :13]
    elif(k==2):
        test= data[number2:number3, :]
        training=np.vstack((data[:number2,:],data[number3:,:]))
        y_train = training[:, 13]
        x_train = training[:, :13]
        y_test = test[:, 13]
        x_test = test[:, :13]
    elif(k==3):
        test= data[number3:number4, :]
        training=np.vstack((data[:number3,:],data[number4:,:]))
        y_train = training[:, 13]
        x_train = training[:, :13]
        y_test = test[:, 13]
        x_test = test[:, :13]
    elif(k==4):
        test,training= data[number4:, :],data[:number4,:]
        y_train = training[:, 13]
        x_train = training[:, :13]
        y_test = test[:, 13]
        x_test = test[:, :13]
    return y_train,x_train,y_test,x_test

#QuestionA
def QuestionA():
    #gamma
    gamma = [pow(2, -40), pow(2, -39), pow(2, -38), pow(2, -37),
             pow(2, -36), pow(2, -35), pow(2, -34), pow(2, -33),
             pow(2, -32), pow(2, -31), pow(2, -30), pow(2, -29),
             pow(2, -28), pow(2, -27), pow(2, -26)]

    #sigma
```

```
        sigma = [pow(2, 7), pow(2, 7.5), pow(2, 8), pow(2, 8.5), pow(2, 9),
                 pow(2, 9.5), pow(2, 10), pow(2, 10.5), pow(2, 11),
                 pow(2, 11.5), pow(2, 12), pow(2, 12.5), pow(2, 13)]
        length = len(gamma) * len(sigma)
        res = [[0.0] * 3] * length  #array to store the results
        res = np.array(res)
        for m in range(len(gamma)):
            print("gamma now is " + str(gamma[m]))
            for n in range(len(sigma)):
                print("sigma now is " + str(sigma[n]))
                mse = 0
                for i in range(5):
                    # print("loop is "+str(i))
                    y_train, x_train, y_test, x_test = split_data(i)
                    K = kernel(x_train, x_train, sigma[n])
                    alpha = a_star(x_train, y_train, K, gamma[m])
                    y = predict(x_train=x_train, x_test=x_test, a=alpha, sigma=sigma[n])
                    mse += cal_mse(y, y_test)
                    print("mse now is " + str(mse))
                mse_res = mse / 5
                print("avarage mse is " + str(mse_res))
                print("m" + str(m))
                print("n" + str(n))
                number = 13 * m + n
                print("now number is " + str(number))
                res[number][0] = gamma[m]
                res[number][1] = sigma[n]
                res[number][2] = mse_res
                print(res[number])
        np.savetxt("Q5_result.txt", res)
        result = res[res[:, 2].argsort()]
        print(result)
        print("the best gamma is " + str(result[0][0]) +
              "the best sigam is " + str(result[0][1]) +
              "the mse now is " + str(result[0][2]))

    #QuestionB
    def QuestionB():
        res=np.loadtxt("Q5_result.txt")
        plot_3d(res)

    #QuestionC
    def QuestionC():
        number=round(data.shape[0] * 2 / 3)
        training, test = data[:number, :], data[number:, :]
        gamma=pow(2,-26)
        sigma=pow(2,13)
        y_train = training[:, 13]
        x_train = training[:, :13]
        y_test = test[:, 13]
        x_test = test[:, :13]
```

```
        K = kernel(x_train, x_train, sigma)
        alpha = a_star(x_train, y_train, K, gamma)
        y1 = predict(x_train=x_train, x_test=x_test, a=alpha, sigma=sigma)
        y2 = predict(x_train=x_train,x_test=x_train,a=alpha,sigma=sigma)
        mse_test= cal_mse(y1, y_test)
        mse_train=cal_mse(y2,y_train)
        print("MSE of test set "+str(mse_test))
        print("MSE of training set "+str(mse_train))


    #repeat for Question5c over 20 random(2/3,1/3) splits of your data
    def QuestionD_4():
        MSE_test=[0.0]*20
        MSE_train=[0.0]*20
        MSE_train=np.array(MSE_train)
        MSE_test=np.array(MSE_test)
        for i in range(20):
            number=round(data.shape[0] * 2 / 3)
            np.random.shuffle(data)
            training, test = data[:number, :], data[number:, :]
            gamma=pow(2,-26)
            sigma=pow(2,13)
            y_train = training[:, 13]
            x_train = training[:, :13]
            y_test = test[:, 13]
            x_test = test[:, :13]
            K = kernel(x_train, x_train, sigma)
            alpha = a_star(x_train, y_train, K, gamma)
            y1 = predict(x_train=x_train, x_test=x_test, a=alpha, sigma=sigma)
            y2 = predict(x_train=x_train,x_test=x_train,a=alpha,sigma=sigma)
            mse_test= cal_mse(y1, y_test)
            mse_train=cal_mse(y2, y_train)
            MSE_test[i]=mse_test
            MSE_train[i]=mse_train
        Mse_train = MSE_train.mean()
        Mse_test = MSE_test.mean()
        STD_train = np.std(MSE_train, ddof=1)
        STD_test = np.std(MSE_test, ddof=1)

        print("The average MSE of training set is " + str(Mse_test))
        print("The average MSE of test set is " + str(Mse_train))
        print("The standard deviarions of train error is " + str(STD_train))
        print("The standard deviarions of test error is " + str(STD_test))
```