

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 $k = 1, \dots, 18$ in Figure 3

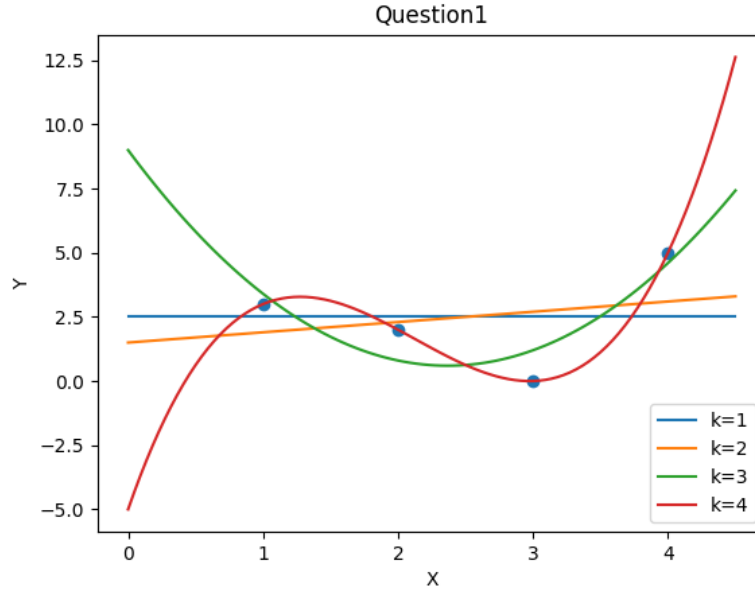


Figure 1: 1.a Linear Regression

- (c) we generate a test set T of a thousand points, and plot the log of the test error versus the polynomial dimension $k = 1, \dots, 18$ 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 $\log(\text{avg})$ of mse in Figure 5

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

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

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 $k = 1, \dots, 18$ 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 $\log(\text{avg})$ of mse in Figure 8

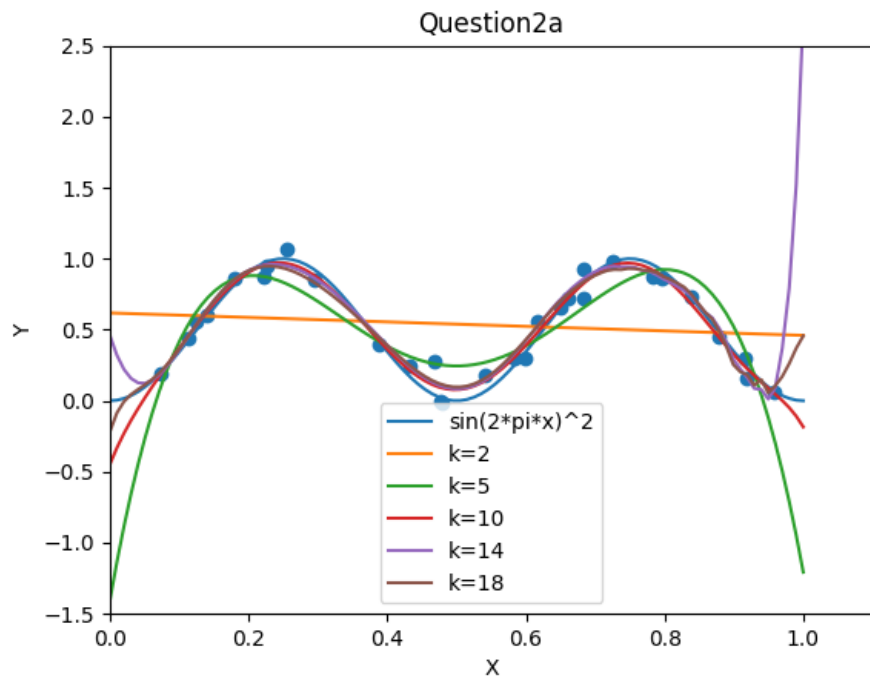


Figure 2: 2.a Question 2a

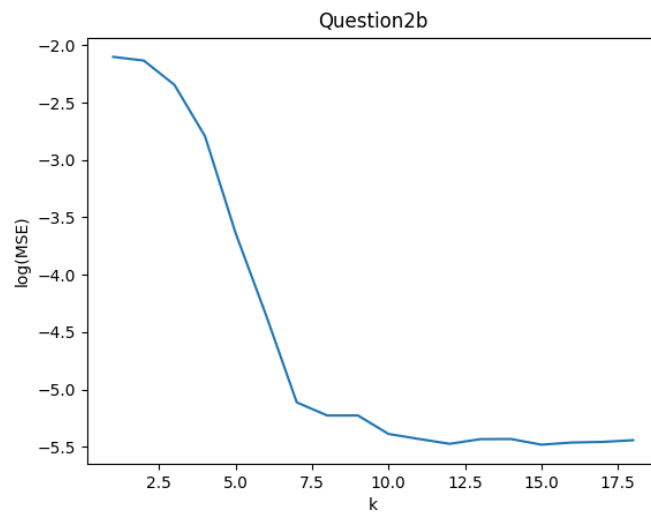


Figure 3: 2.b Question 2b

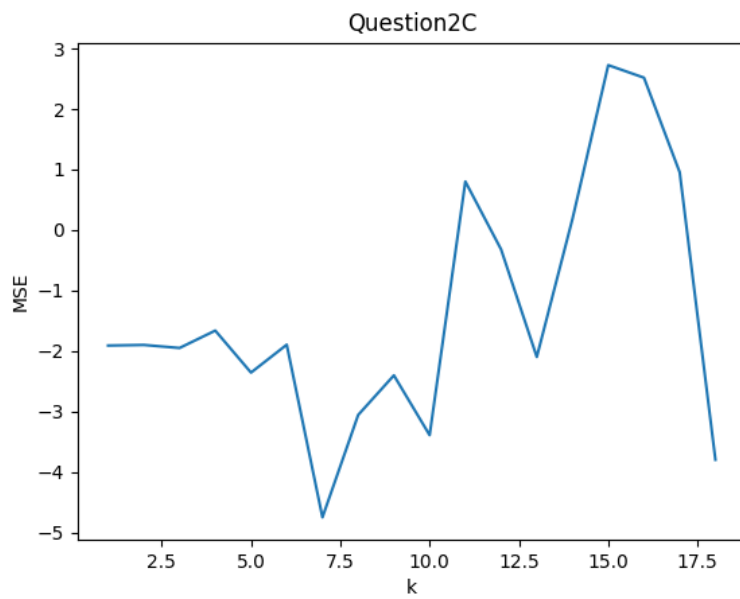


Figure 4: 2.c Question 2C

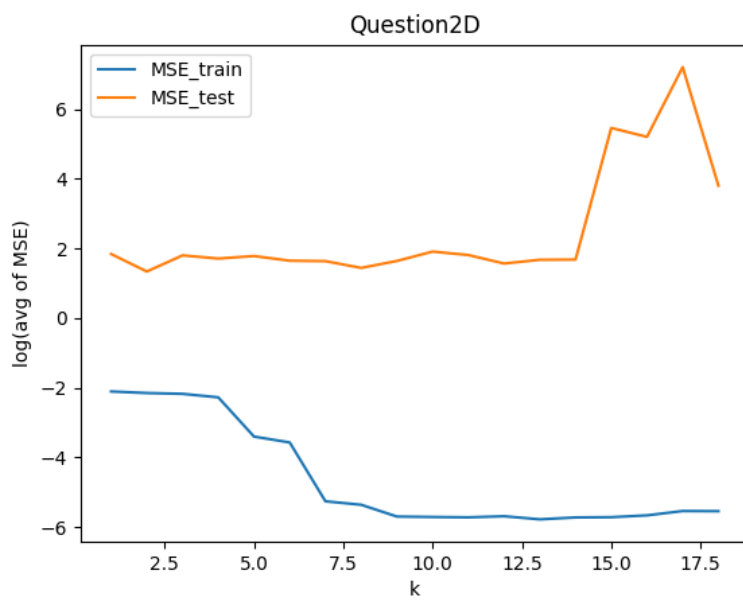


Figure 5: 2.d Question 2D

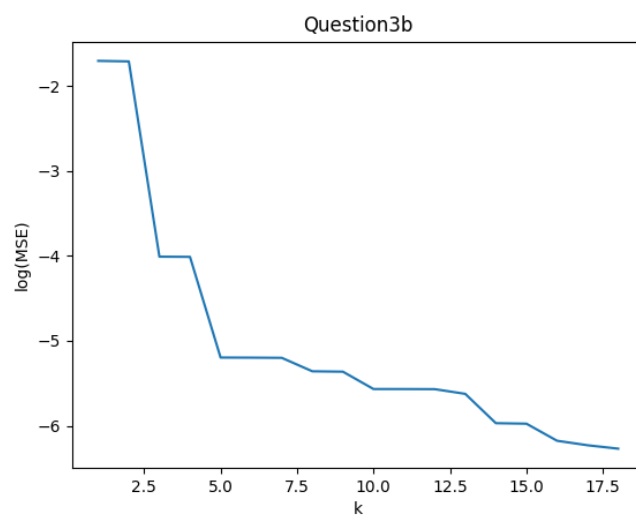


Figure 6: 3.b Question 3B

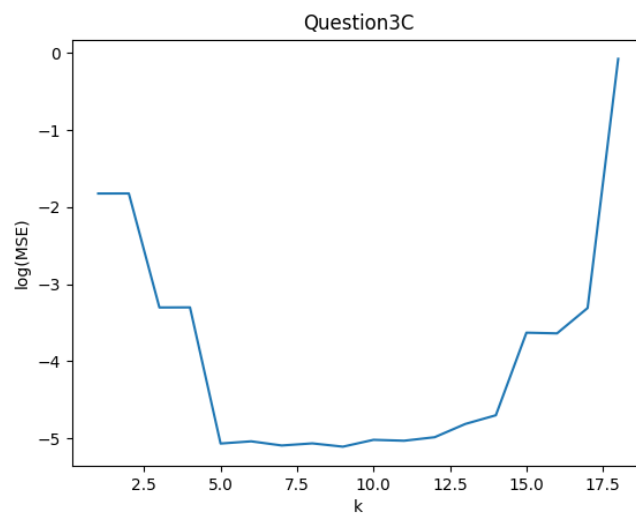


Figure 7: 3.c Question 3C

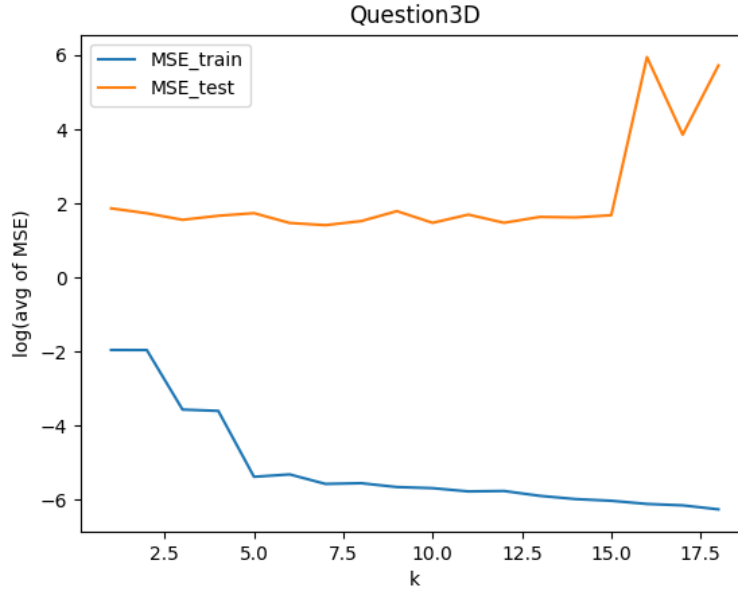


Figure 8: 3.d Question 3D

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,

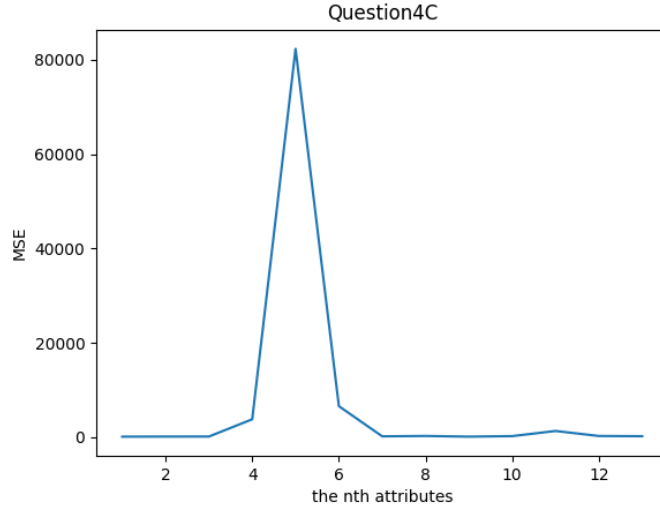


Figure 9: 4.c Question 4c

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

Question5B

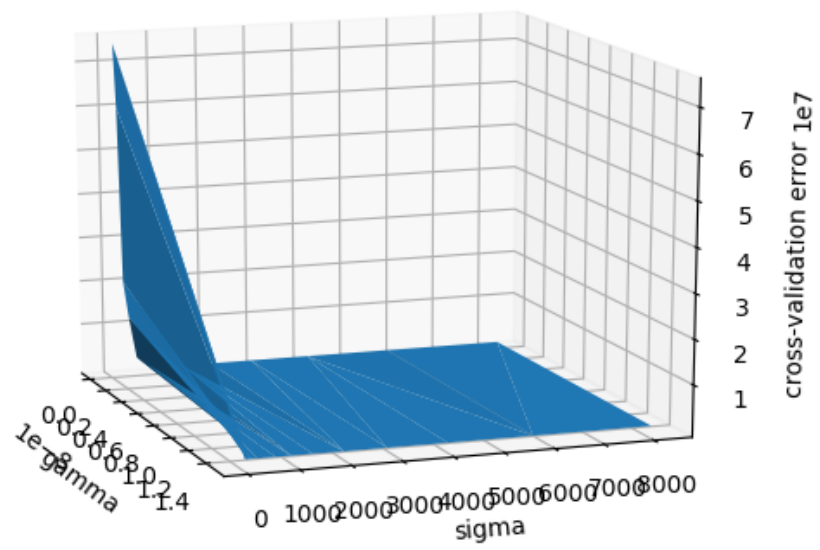


Figure 10: 5.b Question 5B

Method	MSE train	MSE test
Naive Regression	85.0215099 \pm 5.300209	84.265901 \pm 10.766989
Linear Regression(attribute 1)	3.828141 \pm 17.11996	3.128918 \pm 13.992950
Linear Regression(attribute 2)	4.115751 \pm 18.406200	3.997990 \pm 17.879556
Linear Regression(attribute 3)	3.338028 \pm 14.928118	4.951336 \pm 22.1430509
Linear Regression(attribute 4)	3.638778 \pm 16.273110	322.057991 \pm 1440.287124
Linear Regression(attribute 5)	3.732333 \pm 16.691502	3784.435049 \pm 16924.508054
Linear Regression(attribute 6)	1.773274 \pm 7.930326	502.199152 \pm 2245.902884
Linear Regression(attribute 7)	3.577382 \pm 15.998540	7.259205 \pm 32.464154
Linear Regression(attribute 8)	3.795431 \pm 16.973683	19.826130 \pm 88.665151
Linear Regression(attribute 9)	3.472839 \pm 15.53101	4.182220 \pm 18.703456
Linear Regression(attribute 10)	3.269255 \pm 14.620552	9.158062 \pm 40.956102
Linear Regression(attribute 11)	3.411703 \pm 15.257602	73.125048 \pm 327.025158
Linear Regression(attribute 12)	3.739120 \pm 16.721856	11.381800 \pm 50.900960
Linear Regression(attribute 13)	2.142587 \pm 9.581942	8.911042 \pm 39.851392
Linear Regression(all attributes)	23.296191 \pm 1.442286	21.810731 \pm 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 $L_c : [k] \times [k] \rightarrow [0, \infty)$ 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:

$$\begin{aligned} \text{Estimator: } \hat{y} &= \text{*arg min}_y E([y \neq \hat{y}]c_y) \\ L(y, \hat{y}) &= I(y \neq \hat{y}) = 0 \quad \text{if } y = \hat{y} \\ L(y, \hat{y}) &= I(y \neq \hat{y}) = c_y \quad \text{if } y \neq \hat{y} \end{aligned}$$

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

$$\begin{aligned} \text{So that: } E(L(y, \hat{y})|x) &= \sum_{y \neq \hat{y}} \rho(y|x) = 1 - \rho(\hat{y}|x) \\ \hat{y} &= \text{*arg min}_y E(L(y, \hat{y})|x) = \text{*arg max}_y \rho(y|x) \end{aligned}$$

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

(b) For this subquestion $Y \subset \mathbb{R}$. 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

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

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

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

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 $x, z \in \mathbb{R}^n$.

(a) For what values of $c \in \mathbb{R}$ 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 $x, z \in \mathbb{R}^n$, 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 $c \in \mathbb{R}$

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

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

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

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 re-

gression 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 preform 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 reasoning.

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

```
1      import matplotlib
2      matplotlib.use('TkAgg')
3      import numpy as np
4      import pandas as pd
5      import matplotlib.pyplot as plt
6      import math
7
8      #generate the feature map
9      #use input data x and the dimension of polynomial bases k
10     def feature_map(x,k):
11         X=[]
12         for i in range(k+1):
13             X.append(x**i)
14         X=np.mat(X).T
15         X=X.astype(np.float32)
16         return X
17
18     #generate the weight w
19     #use the feature map X=feature_map(x,k) and the goal number
20     y
21     def fit(x,y,k):
22         X=np.around(feature_map(x,k).astype('float64'),decimals
23                     = 7) #feature map
24         Y = np.array(y).reshape((len(y), 1))
25         XT = X.transpose()
26         w = np.dot(np.dot(np.linalg.inv(np.dot(XT, X)), XT), Y)
27         return w
28
29     #predict function
30     #use the feature x and weight w and the dimension of
31     polynomial bases k
32     def predict(x,w,k):
33         X=feature_map(x,k)
34         #print(X)
35         y=X@w
36         return y
37
38     #calculate MSE
39     #MSE=SSE/n
40     #input is y_true and y_predict
41     def cal_mse(y_t,y_p):
42         result=[]
43         if(y_t.size!=y_p.size):
44             print("Input error")
45         for i in range(len(y_t)):
46             mse1=pow((y_t[i]-y_p[i]),2)
47             result.append(mse1)
48         mse=sum(result)/len(y_t)
49         return mse
50
51     #Question 1
52     def Q1():
53         #data
54         data = np.array([(1, 3), (2, 2), (3, 0), (4, 5)])
55         x = data[:, 0]
```

```

53     y = data[:, 1]

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

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

67     #plot a picture to illustrate the linear function
68     xn=np.linspace(0,4.5,100)
69     Y_t1=predict(xn,w1,0)
70     Y_t2=predict(xn,w2,1)
71     Y_t3=predict(xn,w3,2)
72     Y_t4=predict(xn,w4,3)
73     plt.figure(1)
74     plt.scatter(x,y)
75     l1,=plt.plot(xn,Y_t1)
76     l2,=plt.plot(xn,Y_t2)
77     l3,=plt.plot(xn,Y_t3)
78     l4,=plt.plot(xn,Y_t4)
79     plt.legend(handles=[l1, l2,l3,l4 ], labels=['k=1', 'k=2',
        'k=3', 'k=4'], loc='best')
80     plt.xlabel('X')
81     plt.ylabel('Y')
82     plt.title("Question1")
83     #predict Y with different weight w and different
        polynomial bases
84     Y_t1=predict(x,w1,0)
85     Y_t2=predict(x,w2,1)
86     Y_t3=predict(x,w3,2)
87     Y_t4=predict(x,w4,3)

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

```

B Question 2

```

2     import matplotlib
3     matplotlib.use('TkAgg')
4     import numpy as np
5     import pandas as pd
6     import matplotlib.pyplot as plt

```

```

6      import math

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

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

28     #predict function
29     #use the feature x and weight w and the dimension of
30     polynomial bases k
31     def predict(x,w,k):
32         X=feature_map(x,k)
33         #print(X)
34         y=X@w
35         return y

36     #calculate MSE
37     #MSE=SSE/n
38     #input is y_true and y_predict
39     def cal_mse(y_t,y_p):
40         result=[]
41         if(y_t.size!=y_p.size):
42             print("Input error")
43         for i in range(len(y_t)):
44             mse1=pow((y_t[i]-y_p[i]),2)
45             result.append(mse1)
46         mse=sum(result)/len(y_t)
47         return mse

48     #Question2(a)
49     def Question2A():
50         #y1=sin(2*pi*x)^2+noise
51         np.random.seed(50)
52         noise=np.random.normal(0,0.07,30)
53         x1=np.random.uniform(0,1,30)
54         y1=np.square(np.sin(2*math.pi*x1))
55         i=0
56         while(i<y1.size):
57             y1[i]=y1[i]+noise[i]
58             i=i+1

```

```

60 #y_t=sin(2*pi*x)^2
    xn=np.linspace(0,1,100)
62 y_t=np.square(np.sin(2*math.pi*xn))
    #weight w when k=2,k=5,k=10,k=14,k=18
64 w1=fit(x1,y1,1)
    w2=fit(x1,y1,4)
66 w3=fit(x1,y1,9)
    w4=fit(x1,y1,13)
68 w5=fit(x1,y1,17)
    plt.figure(1)
    plt.scatter(x1,y1)
70 xn=np.linspace(0,1,100)
    Y_t1=predict(xn,w1,1)
72 Y_t2=predict(xn,w2,4)
    Y_t3=predict(xn,w3,9)
74 Y_t4=predict(xn,w4,13)
    Y_t5=predict(xn,w5,17)
76
78 l1,=plt.plot(xn,y_t)
    l2,=plt.plot(xn,Y_t1)
80 l3,=plt.plot(xn,Y_t2)
    l4,=plt.plot(xn,Y_t3)
82 l5,=plt.plot(xn,Y_t4)
    l6,=plt.plot(xn,Y_t5)
84 plt.xlabel('X')
    plt.ylabel("Y")
86 plt.title("Question2a")
    plt.legend(handles=[l1, l2, l3, l4,l5,l6],
88               labels=['sin(2*pi*x)^2', 'k=2', 'k=5',
                        'k=10', 'k=14', 'k=18'], loc='best')
90 plt.xlim(0,1.1)
    plt.ylim(-1.5,2.5)
92 plt.show()

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

102    i=0
    while(i<y1.size):
104        y1[i]=y1[i]+noise[i]
        i=i+1
106    #create a mse for MSE
    mse=[]
108    for i in range(18):
        w=fit(x1,y1,i)
110        Y_t=predict(x1,w,i)
        mse.append(math.log(cal_mse(y1,Y_t)))
112
114    plt.figure(1)
    plt.title("Question2b")
    plt.xlabel("k")
116    plt.ylabel("log(MSE)")

```



```

118     x_mse=np.linspace(1,18,18)
119     plt.plot(x_mse,mse)
120     plt.show()
121
122 #Question2C
123 def Question2C():
124     #when k=1,...,18
125     #train data generate the w
126     def train(k):
127         np.random.seed(30)
128         s = np.random.normal(0, 0.07, 30)
129
130         x1 = np.random.uniform(0, 1, 30)
131         x1.sort()
132         y1 = np.square(np.sin(2 * math.pi * x1))
133
134         i = 0
135         while (i < y1.size):
136             y1[i] = y1[i] + s[i]
137             i = i + 1
138
139         w = fit(x1, y1, k)
140         return w
141
142     #use the w from train to compute the y_predict
143     #compute the MSE in different k
144     def test(k):
145         w = train(k)
146         np.random.seed(30)
147         s = np.random.normal(0, 0.07, 1000)
148
149         x1 = np.random.uniform(0, 1, 1000)
150         y1 = np.square(np.sin(2 * math.pi * x1))
151         i = 0
152         while (i < y1.size):
153             y1[i] = y1[i] + s[i]
154             i = i + 1
155
156         y_t = predict(x1, w, k)
157         y_t = np.array(y_t)
158         mse = cal_mse(y_t, y1)
159         mse_log = math.log(mse)
160         return mse_log
161
162     #create list for mse
163     res_mse = []
164     for i in range(18):
165         res_mse.append(test(i))
166
167     plt.figure(1)
168     x = np.linspace(1, 18, 18)
169     plt.plot(x, res_mse)
170     plt.xlabel("k")
171     plt.ylabel("MSE")
172     plt.title("Question2C")
173     plt.show()

```

```

174 def Question2D():
175     #create two list for MSE_train and MSE_test
176     MSE_train=[]
177     MSE_test=[]
178     #loop for k from 1 to 18
179     for k in range(18):
180         print("current K is "+str(k) )
181         mse_train=0
182         mse_test=0
183         #run train and test 100 times
184         #compute each time mse
185         for m in range(100):
186             #train data
187             s = np.random.normal(0, 0.07, 30)
188
189             x1 = np.random.uniform(0, 1, 30)
190             y1 = np.square(np.sin(2 * math.pi * x1))
191             i = 0
192             while (i < y1.size):
193                 y1[i] = y1[i] + s[i]
194                 i = i + 1
195
196             w=fit(x1,y1,k)
197             Y_t1=predict(x1,w,k)
198             #train finish!
199             #compute mse of train
200             mse_train+=cal_mse(y1,Y_t1)
201
202             #test data
203             s2=np.random.normal(0,0.07,1000)
204             x2=np.random.uniform(0,1,1000)
205             y2=np.square(np.sin(2*math.pi*x2))
206             n=0
207             while(n<y2.size):
208                 y2[i]=y2[i]+s2[i]
209                 n=n+1
210             #"start test!"
211             Y_t=predict(x2,w,k)
212             mse_test=mse_test+cal_mse(y2,Y_t)
213
214             #compute the log(average of MSE)
215             Mse_train=math.log(mse_train/100)
216             Mse_test=math.log(mse_test/100)
217             MSE_test.append(Mse_test)
218             MSE_train.append(Mse_train)
219
220         plt.figure(1)
221         x=np.linspace(1,18,18)
222         l1,=plt.plot(x,np.array(MSE_train))
223         l2,=plt.plot(x,np.array(MSE_test))
224         plt.title("Question2D")
225         plt.xlabel("k")
226         plt.ylabel("log(avg of MSE)")
227         plt.legend(handles=[l1, l2,], labels=['MSE_train', '
228             MSE_test'], loc='best')
229         plt.show()

```

C Question 3

```
1      import matplotlib
2      matplotlib.use('TkAgg')
3      import numpy as np
4      import pandas as pd
5      import matplotlib.pyplot as plt
6      import math
7
8      #generate the feature map
9      #use input data x and the dimension of sin(n*pi*x) bases k
10     def feature_map2(x,k):
11         X=[]
12         for i in range(k+1):
13             X.append(np.sin((i+1)*np.pi*x))
14         X=np.mat(X).T
15         X=X.astype(np.float32)
16         return X
17
18     #generate the weight w
19     #use the feature map X=feature_map(x,k) and the goal number
20     y
21     def fit(x,y,k):
22         X=np.around(feature_map2(x,k).astype('float64'),
23                     decimals = 7) #feature map
24         Y = np.array(y).reshape((len(y), 1))
25         XT = X.transpose()
26         w = np.dot(np.dot(np.linalg.inv(np.dot(XT, X)), XT), Y)
27         return w
28
29     #predict function
30     #use the feature x and weight w and the dimension of
31     polynomial bases k
32     def predict(x,w,k):
33         X=feature_map2(x,k)
34         #print(X)
35         y=X@w
36         return y
37
38     #calculate MSE
39     #MSE=SSE/n
40     #input is y_true and y_predict
41     def cal_mse(y_t,y_p):
42         result=[]
43         if(y_t.size!=y_p.size):
44             print("Input error")
45         for i in range(len(y_t)):
46             mse1=pow((y_t[i]-y_p[i]),2)
47             result.append(mse1)
48         mse=sum(result)/len(y_t)
49         return mse
50
51     #Question3b
52     def Question3b():
53         np.random.seed(20)
54         #y1=sin(2*pi*x)^2+noise
55         noise=np.random.normal(0,0.07,30)
```

```

53 x1=np.random.uniform(0,1,30)
54 y1=np.square(np.sin(2*math.pi*x1))
55
56 i=0
57 while(i<y1.size):
58     y1[i]=y1[i]+noise[i]
59     i=i+1
60 #create a mse for MSE
61 mse=[]
62 for i in range(18):
63     w=fit(x1,y1,i)
64     Y_t=predict(x1,w,i)
65     mse.append(math.log(cal_mse(y1,Y_t)))
66
67 plt.figure(1)
68 plt.title("Question3b")
69 plt.xlabel("k")
70 plt.ylabel("log(MSE)")
71 x_mse=np.linspace(1,18,18)
72 plt.plot(x_mse,mse)
73 plt.show()
74
75 #Question3c
76 def Question3c():
77     # when k=1,...,18
78     # train data generate the w
79     def train(k):
80         np.random.seed(30)
81         s = np.random.normal(0, 0.07, 30)
82
83         x1 = np.random.uniform(0, 1, 30)
84         x1.sort()
85         y1 = np.square(np.sin(2 * math.pi * x1))
86
87         i = 0
88         while (i < y1.size):
89             y1[i] = y1[i] + s[i]
90             i = i + 1
91
92         w = fit(x1, y1, k)
93         return w
94
95     # use the w from train to compute the y_predict
96     # compute the MSE in different k
97     def test(k):
98         w = train(k)
99         np.random.seed(30)
100        s = np.random.normal(0, 0.07, 1000)
101
102        x1 = np.random.uniform(0, 1, 1000)
103        y1 = np.square(np.sin(2 * math.pi * x1))
104        i = 0
105        while (i < y1.size):
106            y1[i] = y1[i] + s[i]
107            i = i + 1
108
109        y_t = predict(x1, w, k)

```

```

111         y_t = np.array(y_t)
112         mse = cal_mse(y_t, y1)
113         mse_log = math.log(mse)
114         return mse_log
115
116     # create list for mse
117     res_mse = []
118     for i in range(18):
119         res_mse.append(test(i))
120
121     plt.figure(1)
122     x = np.linspace(1, 18, 18)
123     plt.plot(x, res_mse)
124     plt.xlabel("k")
125     plt.ylabel("log(MSE)")
126     plt.title("Question3C")
127     plt.show()
128
129 def Question3d():
130     #create two list for MSE_train and MSE_test
131     MSE_train=[]
132     MSE_test=[]
133     #loop for k from 1 to 18
134     for k in range(18):
135         print("current K is "+str(k) )
136         mse_train=0
137         mse_test=0
138         #run train and test 100 times
139         #compute each time mse
140         for m in range(100):
141             #train data
142             s = np.random.normal(0, 0.07, 30)
143
144             x1 = np.random.uniform(0, 1, 30)
145             y1 = np.square(np.sin(2 * math.pi * x1))
146             i = 0
147             while (i < y1.size):
148                 y1[i] = y1[i] + s[i]
149                 i = i + 1
150
151             w=fit(x1,y1,k)
152             Y_t1=predict(x1,w,k)
153             #train finish!
154             #compute mse of train
155             mse_train+=cal_mse(y1, Y_t1)
156
157             #test data
158             s2=np.random.normal(0,0.07,1000)
159             x2=np.random.uniform(0,1,1000)
160             y2=np.square(np.sin(2*math.pi*x2))
161             n=0
162             while(n<y2.size):
163                 y2[i]=y2[i]+s2[i]
164                 n=n+1
165             #start test!
166             Y_t=predict(x2,w,k)
167             mse_test=mse_test+cal_mse(y2, Y_t)

```

```

167         #compute the log(average of MSE)
169         Mse_train=math.log(mse_train/100)
171         Mse_test=math.log(mse_test/100)
173         MSE_test.append(Mse_test)
175         MSE_train.append(Mse_train)
177     plt.figure(1)
179     x=np.linspace(1,18,18)
181     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

```

1     import matplotlib
2     matplotlib.use('TkAgg')
3     import numpy as np
4     import matplotlib.pyplot as plt
5     import scipy.io as sio
6
7     #load the data
8     input=sio.loadmat('boston.mat')
9     data=input['boston']
10
11    #generate the feature map
12    #use input data x and the dimension of polynomial bases k
13    def feature_map(x,k):
14        X=[]
15        for i in range(k+1):
16            X.append(x**i)
17        X=np.mat(X).T
18        X=X.astype(np.float32)
19        return X
20
21    #use for linear regression for many attributes
22    def fit2(x,y):
23        X=x
24        Y=y.reshape((len(y),1))
25        XT=x.transpose()
26        w = np.dot(np.dot(np.linalg.inv(np.dot(XT, X)), XT), Y)
27        return w
28
29    #generate the weight w
30    #use the feature map X=feature_map(x,k) and the goal number
31    y
32    def fit(x,y,k):
33        X=np.around(feature_map(x,k).astype('float64'), decimals
34                    = 7) #feature map

```

```

33     Y = np.array(y).reshape((len(y), 1))
34     XT = X.transpose()
35     w = np.dot(np.dot(np.linalg.inv(np.dot(XT, X)), XT), Y)
36     return w
37
38     #predict function
39     #use the feature x and weight w and the dimension of
40     #polynomial bases k
41     def predict(x,w,k):
42         X=feature_map(x,k)
43         #print(X)
44         y=X@w
45         return y
46
47     #predict function in many attributes
48     def predict2(x,w):
49         y=np.dot(x,w)
50         return y
51
52     #calculate MSE
53     #MSE=SSE/n
54     #input is y_true and y_predict
55     def cal_mse(y_t,y_p):
56         result=[]
57         if (y_t.size!=y_p.size):
58             print("Input error")
59         for i in range(len(y_t)):
60             mse1=pow((y_t[i]-y_p[i]),2)
61             result.append(mse1)
62         mse=sum(result)/len(y_t)
63         return mse
64
65     #Calclute MSE for train set and test set at the same time
66     def Calculate_MSE(y_train,y_test,y_mean):
67         result_train=[]
68         result_test=[]
69         for i in range(len(y_train)):
70             mse1=pow((y_train[i]-y_mean),2)
71             result_train.append(mse1)
72         train_mse=sum(result_train)/len(y_train)
73         for i in range(len(y_test)):
74             mse2=pow((y_test[i]-y_mean),2)
75             result_test.append(mse2)
76
77         test_mse=sum(result_test)/len(y_test)
78         return train_mse,test_mse
79
80     #for Question A
81     def QuestionA():
82         MSE=[0,0]
83         for loop in range(20):
84             #split data
85             number1=round(data.shape[0]*2/3)
86             np.random.shuffle(data)
87             training, test = data[:number1,:], data[number1:,:]
88
89             y_train=training[:,13]

```

```

89         y_test=test[:,13]
90         y_mean=y_train.mean()
91         mse_train, mse_test=Calculate_MSE(y_train, y_test,
92                                           y_mean)
93         MSE[0]+=mse_train
94         MSE[1]+=mse_test
95     MSE[0]=MSE[0]/20
96     MSE[1]=MSE[1]/20
97
98     print("MSE of training set is " +str(MSE[0]))
99     print("MSE of test set is " +str(MSE[1]))
100
101 #QuestionC
102 def QuestionC():
103     mse=[0.0]*13
104     mse=np.array(mse)
105     for i in range(len(mse)):
106         print("feature column now "+str(i))
107         for loop in range(20):
108             #split data
109             number1=round(data.shape[0]*2/3)
110             np.random.shuffle(data)
111             training, test = data[:number1,:], data[number1:,:]
112             y_train=training[:,13]
113             x_train=training[:,i]
114             y_test=test[:,13]
115             x_test=test[:,0]
116             #train
117             w=fit(x_train, y_train, 1)
118             #predict and test
119             y_predict1=predict(x_test, w, 1)
120             mse1=cal_mse(y_t=y_test, y_p=y_predict1)
121             mse[i]+=mse1
122         mse[i]=mse[i]/20
123     #figure
124     plt.figure(1)
125     x=np.linspace(1,13,13)
126     plt.xlabel("the nth attributes")
127     plt.ylabel("MSE")
128     plt.title("Question4C")
129     plt.show()
130
131 #QuestionD
132 def QuestionD(data):
133     mse=0
134     for loop in range(20):
135         print(loop)
136         number1 = round(data.shape[0] * 2 / 3)
137         number2 = data.shape[0] - number1
138         np.random.shuffle(data)
139         training, test = data[:number1, :], data[number1:, :]
140         y_train = training[:, 13]
141         x_train = training[:, :13]
142         x_test=test[:,13]
143         y_test=test[:,13]

```



```

143         ones1=np.ones((number1,1))
144         ones2=np.ones((number2,1))
145         x_train=np.hstack((x_train,ones1))
146         x_test=np.hstack((x_test,ones2))
147         w = fit2(x_train,y_train)
148         y=predict2(x_test,w)
149         mse1=cal_mse(y_test,y)
150         mse+=mse1
151         print("mse is "+str(mse/20))

153 #Question for 5D
154 #repeate exercise 4a over 20 random (2/3,1/3) splits of
155 #your data
156 def Question5D_1():
157     MSE_test=[0.0]*20
158     MSE_train=[0.0]*20
159     MSE_train=np.array(MSE_train)
160     MSE_test=np.array(MSE_test)
161     for loop in range(20):
162         #split data
163         number1=round(data.shape[0]*2/3)
164         np.random.shuffle(data)
165         training, test = data[:number1,:], data[number1:,:]
166         y_train=training[:,13]
167         y_test=test[:,13]
168         y_mean=y_train.mean()
169         #calculte the MSE
170         mse_train,mse_test=Calculate_MSE(y_train,y_test,
171                                         y_mean)
172         MSE_train[loop]=mse_train
173         MSE_test[loop]=mse_test

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

178     #print the results
179     print("The average MSE of training set is "+str(
180           Mse_test))
181     print("The average MSE of test set is "+str(Mse_train))
182     print("The standard deviarions of train error is " +
183           str(STD_train))
184     print("The standard deviarions of test error is " + str(
185           STD_test))

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

```

```

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

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

219
    result[i,0]=Mse_train
221     result[i,1]=Mse_test
    result[i,2]=STD_train
223     result[i,3]=STD_test

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

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

```

```

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

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

271     print("The average MSE of training set is " +str(
        Mse_test))
273     print("The average MSE of test set is " +str(Mse_train))
        print("The standard deviations of train error is " +
        str(STD_train))
275     print("The standard deviations of test error is " + str
        (STD_test))

```

E Question 5

```

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

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

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

```

```

15         return distance

17 #compute the kernel function
18 #return kernel function k
19 def kernel(x1,x2,sigma):
20     m1=x1.shape[0]
21     m2=x2.shape[0]
22     k=[[0.0]*m2]*m1
23     k=np.array(k)
24     for i in range(m1):
25         for j in range(m2):
26             k[i][j] = np.exp(-pow(distance(x1[i], x2[j]),
27                                     2) / (2 * pow(sigma, 2)))
28     return k

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

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

43
44 #calcalute MSE
45 def cal_mse(y_true,y_train):
46     MSE=np.sum(np.power((y_train.reshape(-1,1) - y_true),
47                         2))/len(y_train)
48     return MSE

49
50 #plot a 3D picture for sigma and gamma
51 def plot_3d(result):
52     fig = plt.figure()
53     ax = fig.gca(projection='3d')
54     X=result[:,0]
55     Y=result[:,1]
56     Z=result[:,2]
57     ax.plot_trisurf(X,Y,Z)
58     ax.set_xlabel('gamma')
59     ax.set_ylabel('sigma')
60     ax.set_zlabel('cross-validation error')
61     plt.title("Question5B")
62     plt.show()

63
64 #split data for 5-fold cross validation
65 #k from 0 to 4 to get different data set
66 def split_data(k):
67     number1 = round(data.shape[0] * 1 / 5)
68     number2=round(data.shape[0] * 2 / 5)
69     number3=round(data.shape[0] * 3 / 5)

```

```

71     number4=round(data.shape[0] * 4 / 5)
72     y_train=0
73     x_train=0
74     y_test=0
75     x_test=0
76     if (k==0):
77         test, training = data[:number1, :], data[number1:,
78             :]
79         y_train = training[:, 13]
80         x_train = training[:, :13]
81         y_test = test[:, 13]
82         x_test = test[:, :13]
83     elif (k==1):
84         test= data[number1:number2, :]
85         training=np.vstack((data[:number1,:], data[number2
86             :, :]))
87         y_train = training[:, 13]
88         x_train = training[:, :13]
89         y_test = test[:, 13]
90         x_test = test[:, :13]
91     elif (k==2):
92         test= data[number2:number3, :]
93         training=np.vstack((data[:number2,:], data[number3
94             :, :]))
95         y_train = training[:, 13]
96         x_train = training[:, :13]
97         y_test = test[:, 13]
98         x_test = test[:, :13]
99     elif (k==3):
100         test= data[number3:number4, :]
101         training=np.vstack((data[:number3,:], data[number4
102             :, :]))
103         y_train = training[:, 13]
104         x_train = training[:, :13]
105         y_test = test[:, 13]
106         x_test = test[:, :13]
107     elif (k==4):
108         test, training= data[number4:, :], data[:number4, :]
109         y_train = training[:, 13]
110         x_train = training[:, :13]
111         y_test = test[:, 13]
112         x_test = test[:, :13]
113     return y_train, x_train, y_test, x_test
114
115 #QuestionA
116 def QuestionA():
117     #gamma
118     gamma = [pow(2, -40), pow(2, -39), pow(2, -38), pow(2,
119         -37),
120         pow(2, -36), pow(2, -35), pow(2, -34), pow(2,
121             -33),
122         pow(2, -32), pow(2, -31), pow(2, -30), pow(2,
123             -29),
124         pow(2, -28), pow(2, -27), pow(2, -26)]
125
126     #sigma

```

```

119 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,
121             11),
            pow(2, 11.5), pow(2, 12), pow(2, 12.5), pow(2,
              13)]
length = len(gamma) * len(sigma)
123 res = [[0.0] * 3] * length #array to store the results
res = np.array(res)
125 for m in range(len(gamma)):
    print("gamma now is " + str(gamma[m]))
127     for n in range(len(sigma)):
        print("sigma now is " + str(sigma[n]))
129         mse = 0
        for i in range(5):
131             # print("loop is "+str(i))
            y_train, x_train, y_test, x_test =
                split_data(i)
133             K = kernel(x_train, x_train, sigma[n])
            alpha = a_star(x_train, y_train, K, gamma[m
                ])
135             y = predict(x_train=x_train, x_test=x_test,
                a=alpha, sigma=sigma[n])
            mse += cal_mse(y, y_test)
137             print("mse now is " + str(mse))
        mse_res = mse / 5
139         print("avarage mse is " + str(mse_res))
        print("m" + str(m))
141         print("n" + str(n))
        number = 13 * m + n
143         print("now number is " + str(number))
        res[number][0] = gamma[m]
145         res[number][1] = sigma[n]
        res[number][2] = mse_res
147         print(res[number])
    np.savetxt("Q5_result.txt", res)
149    result = res[res[:, 2].argsort()]
    print(result)
151    print("the best gamma is " + str(result[0][0]) +
        "the best sigam is " + str(result[0][1]) +
153        "the mse now is " + str(result[0][2]))

155 #QuestionB
def QuestionB():
157     res=np.loadtxt("Q5_result.txt")
    plot_3d(res)
159
161 #QuestionC
def QuestionC():
    number=round(data.shape[0] * 2 / 3)
163     training, test = data[:number, :], data[number:, :]
    gamma=pow(2,-26)
165     sigma=pow(2,13)
    y_train = training[:, 13]
167     x_train = training[:, :13]
    y_test = test[:, 13]
169     x_test = test[:, :13]

```

```

171     K = kernel(x_train, x_train, sigma)
172     alpha = a_star(x_train, y_train, K, gamma)
173     y1 = predict(x_train=x_train, x_test=x_test, a=alpha,
174                 sigma=sigma)
175     y2 = predict(x_train=x_train, x_test=x_train, a=alpha,
176                 sigma=sigma)
177     mse_test= cal_mse(y1, y_test)
178     mse_train=cal_mse(y2, y_train)
179     print("MSE of test set "+str(mse_test))
180     print("MSE of training set "+str(mse_train))
181
182 #repeat for Question5c over 20 random(2/3,1/3) splits of
183 your data
184 def QuestionD_4():
185     MSE_test=[0.0]*20
186     MSE_train=[0.0]*20
187     MSE_train=np.array(MSE_train)
188     MSE_test=np.array(MSE_test)
189     for i in range(20):
190         number=round(data.shape[0] * 2 / 3)
191         np.random.shuffle(data)
192         training, test = data[:number, :], data[number:, :]
193         gamma=pow(2, -26)
194         sigma=pow(2, 13)
195         y_train = training[:, 13]
196         x_train = training[:, :13]
197         y_test = test[:, 13]
198         x_test = test[:, :13]
199         K = kernel(x_train, x_train, sigma)
200         alpha = a_star(x_train, y_train, K, gamma)
201         y1 = predict(x_train=x_train, x_test=x_test, a=
202                     alpha, sigma=sigma)
203         y2 = predict(x_train=x_train, x_test=x_train, a=alpha
204                     , sigma=sigma)
205         mse_test= cal_mse(y1, y_test)
206         mse_train=cal_mse(y2, y_train)
207         MSE_test[i]=mse_test
208         MSE_train[i]=mse_train
209     Mse_train = MSE_train.mean()
210     Mse_test = MSE_test.mean()
211     STD_train = np.std(MSE_train, ddof=1)
212     STD_test = np.std(MSE_test, ddof=1)
213
214     print("The average MSE of training set is " + str(
215         Mse_test))
216     print("The average MSE of test set is " + str(Mse_train
217         ))
218     print("The standard deviarions of train error is " +
219         str(STD_train))
220     print("The standard deviarions of test error is " + str
221         (STD_test))

```