(i)

(a)I can't tell whether it lies on the curve or the plane , because there would be overlap from different perspectives.

```
fig1 = plt.figure()
ax = fig1.add_subplot(111, projection='3d')
ax.scatter(x1, x2, y, label=str, color='red')
plt.title(str)
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('y')
plt.legend()
plt.show()
```

First define a plt.figure() then use add_subplot to draw the first of 1*1 image , the projection is 3D (3 axis) , the x axis is the value of x1,y axis is x2, z axis is the value of y , use plt.legend to describe the data in the plot.

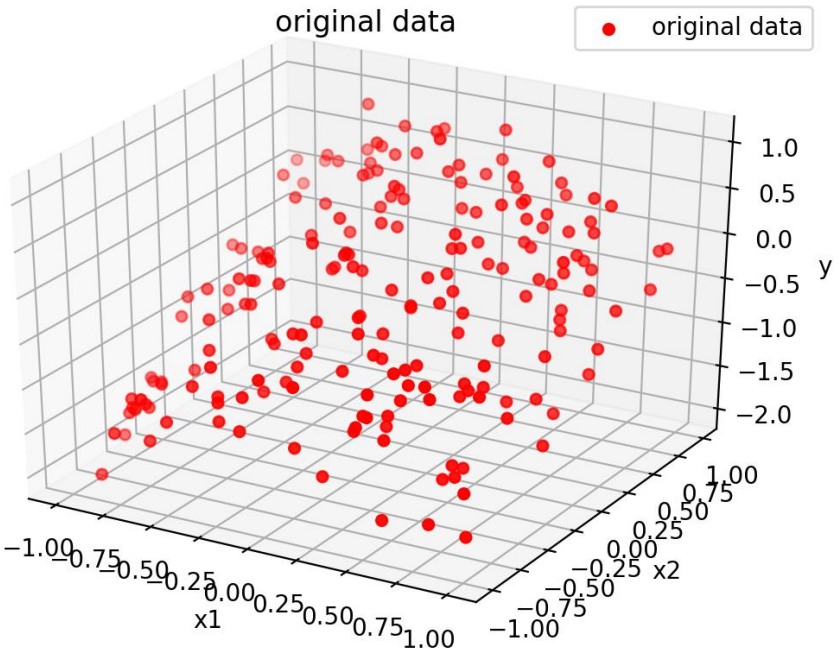

Figure 1 the original data , and the x axis is x1, y axis is x2, z axis is the value of y

(b) I use PolynomialFeatures function to make the x transform to degree 5 , then

I chose large range of values for C :

```
for inx,C in enumerate([0.001,0.01,0.1,1,10,100,1000]):
    alpha=1/C
    linlasso = Lasso(alpha=alpha,max_iter=9999).fit(x_poly, y)
    print('when C is {0},intercept is {1},coef is{2},the score is {3:.4f}\n'.format(C,
                                                                                    linlasso.intercept_, linlasso.coef_,
                                                                                    linlasso.score(x_poly, y)))
```

The value of the parameter:

When C <=1 , we can see the accuracy is 0 , the intercept and coef are all 0 , which means the model is not good , in other words , underfitting , can't capture the feature of the data.

```
when C is 0.001,intercept is -0.35252495717587945,coef is[ 0.  0. -0. -0. -0.  0.  0.  0.  0. -0. -0. -0. -0. -0.  0.  0.  0.  0.
 -0.  0.],the score is 0.0000

when C is 0.01,intercept is -0.35252495717587945,coef is[ 0.  0. -0. -0. -0.  0.  0.  0.  0. -0. -0. -0. -0. -0.  0.  0.  0.  0.
 -0.  0.],the score is 0.0000

when C is 0.1,intercept is -0.35252495717587945,coef is[ 0.  0. -0. -0. -0.  0.  0.  0.  0. -0. -0. -0. -0. -0.  0.  0.  0.  0.
 -0.  0.],the score is 0.0000

when C is 1,intercept is -0.35252495717587945,coef is[ 0.  0. -0. -0. -0.  0.  0.  0.  0. -0. -0. -0. -0. -0.  0.  0.  0.  0.
 -0.  0.],the score is 0.0000
```

When C is bigger and bigger , the accuracy gets higher and higher , and the intercept becomes bigger the coef becomes bigger .

```
when C is 10,intercept is -0.3353997476335713,coef is[-0.          0.67751823 -0.         -0.         -0.          0.
  0.          0.          0.         -0.         -0.         -0.
 -0.         -0.          0.          0.          0.          0.
 -0.          0.        ],the score is 0.6493

when C is 100,intercept is -0.0422465867385598,coef is[-0.          0.9481357  -0.87263655 -0.         -0.         -0.
  0.         -0.          0.         -0.         -0.         -0.
 -0.         -0.          0.          0.          0.          0.
 -0.          0.        ],the score is 0.9066

when C is 1000,intercept is -0.009204752632790691,coef is[-0.05563081  0.94422355 -0.76662741 -0.02463769 -0.          0.
  0.          0.          0.         -0.22284408 -0.09813672  0.
  0.08652656 -0.10312446  0.03138201  0.15268676  0.07112005 -0.
 -0.01169135  0.        ],the score is 0.9164
```

(c) Because original data x1,x2 lies in (-1,1) , so I choose to generate array of range (-1.5,1.5), and then use PolynomialFeatures to transform the Xtest.

```
Xtest =[]
grid=np.linspace(-1.5,1.5) #Return evenly spaced numbers over a specified interval.
for i in grid:
    for j in grid:
        Xtest.append([i,j])
Xtest=np.array(Xtest)
poly_test = PolynomialFeatures(degree=5, interaction_only=False,include_bias=False)
x_poly_test = poly_test.fit_transform(Xtest)
```

Then I define a figure object to plot all the models using add_subplot function. The value of z axis is the prediction of lasso when input is the Xtest transformed . And in each iteration after getting the prediction , I use plot_surface() function to plot , then label the title and each axis.

```
    y_pre_test = linlasso.predict(x_poly_test)
    y_pre_test = y_pre_test.reshape(1,-1)
    ax = fig2.add_subplot(2,4,inx+1,projection='3d')
    ax.scatter(Xtest[:,0], Xtest[:,1], y_pre_test, label="lasso regression", color='blue',s=1)
    ax.plot_surface(Xtest[:,0], Xtest[:,1], y_pre_test)
    ax.scatter(x1, x2, y, label="original data", color='red',s=1)
    plt.title("C={0}".format(C))
    ax.set_xlabel('x1')
    ax.set_ylabel('x2')
    ax.set_zlabel('y')
    plt.legend()
plt.show()
```

As we can see from Figure 2 , when C is small enough which is close to 0 , the regression model is bad , has little overlap with the original data , however when C gets bigger and bigger , the accruracy of the model increases clearly , the model is better to predict data.

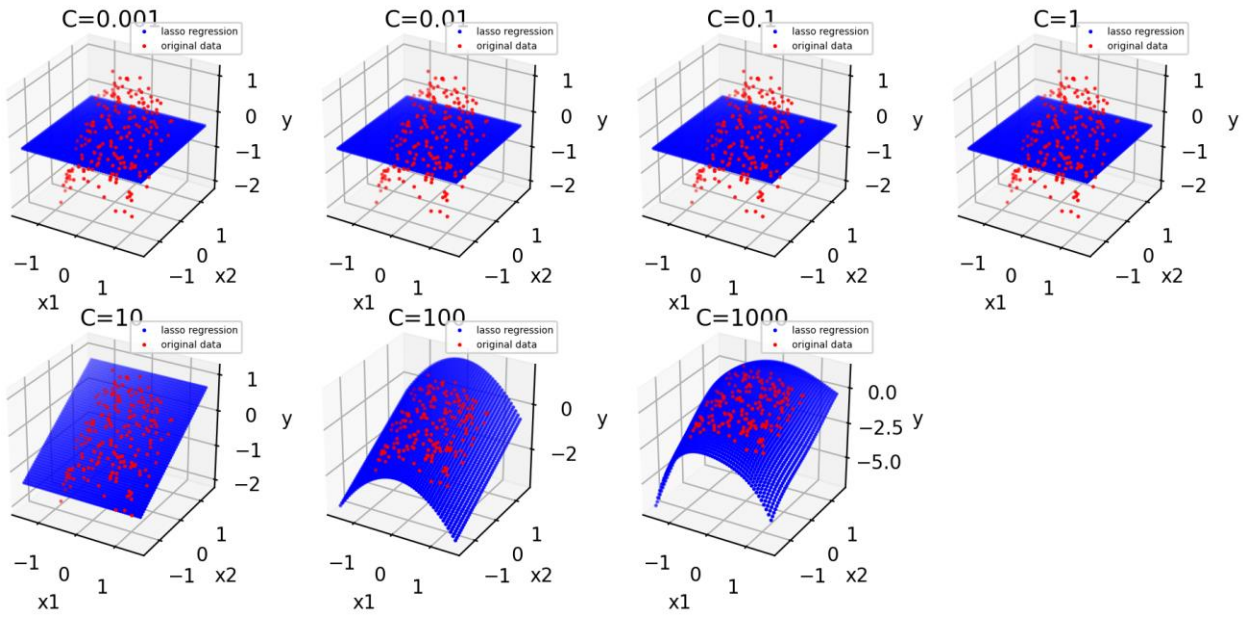

Figure 2 the original data(red points) and the lasso regression model with different value of C

(d)Underfitting: Its occurrence simply means that our model or the algorithm does

not fit the data well enough, because it cannot capture the underlying trend of the data.

Overfitting: When a model gets trained with so much of data, it starts learning from the noise and inaccurate data entries in our data set. Then the model does not categorize the data correctly, because of too many details and noise.

For lasso regression:

When C is small and close to 0, all coefficients are zero , the accuracy is 0 too , which means the model is underfitting, cannot capture the underlying trend of the data ;

And if we tune the alpha bigger and bigger , the accuracy of the training data would increase, imagine if the C is infinity , which means the model couldn't even tolerate a little error , then all the data could be considered , then the training is overfitting. .

(e) When I apply the same range of alpha of lasso's to the ridge's :

```
for inx,C in enumerate([0.001,0.01,0.1,1,10,100,1000]):
```

And all the code is the same as lasso's ,but the alpha is 1/2C:

```
alpha=1/(2*C)
```

And the parameters of the model is as follows:

As we can see when C is in the range of (0,1], **compared** with the lasso's model , the accuracy is much higher ,and the absolute value of the intercept and coef are all bigger, and with C gets bigger the accuracy is higher. The effect of C impacts much more on ridge model , which means ridge model is more sensitive

to the change of C.

However , when C is big enough , the intercept and coef change very subtle, and the accuracy won't change (maybe the changes is too little to be captured by the computer).

```
when C is 0.001,intercept is -0.3251701400970094,coef is[ 0.00250745  0.1054356  -0.03247858 -0.00353926 -0.00794104  0.00419307
  0.03582658  0.00023266  0.05894016 -0.028332   -0.00402512 -0.01187227
 -0.00297871 -0.00880803  0.00418904  0.02138583  0.00098826  0.02016806
 -0.00127182  0.03988416],the score is 0.2590

when C is 0.01,intercept is -0.2048303727575569,coef is[-0.00995212  0.42067621 -0.2103692  -0.0090184  -0.02400692  0.01127595
  0.12694302  0.00160081  0.20372652 -0.18475397 -0.0188317  -0.06939521
  0.00233107 -0.03150072  0.01622064  0.07159152  0.00869455  0.0571874
 -0.0050944   0.12231149],the score is 0.7575

when C is 0.1,intercept is -0.06730173131970185,coef is[-0.06243568  0.72635633 -0.47572268 -0.0179182  -0.00749172  0.01712442
  0.14789266  0.0256605   0.20417372 -0.38926929 -0.06617159 -0.10180507
  0.06586079 -0.0526032   0.03991527  0.08161167  0.05093718 -0.010519
 -0.0263123   0.03937915],the score is 0.9063

when C is 1,intercept is -0.02459347394544742,coef is[-0.09300399  0.86589698 -0.6685556  -0.0700203   0.05853743 -0.00678172
  0.106818    0.16186754  0.1570781  -0.32045788 -0.14001629  0.01282975
  0.18518634 -0.19142766  0.05074664  0.1651467   0.16386365 -0.14737126
 -0.24863966 -0.06022922],the score is 0.9190

when C is 10,intercept is -0.014750134649278157,coef is[-0.11182052  0.87878562 -0.8333009  -0.07631763  0.17572806 -0.08509546
  0.02405344  0.49460759  0.2284466  -0.161063   -0.1367697   0.05235555
  0.16347118 -0.36851987  0.10062882  0.3187876   0.28876269 -0.28250501
 -0.74037738 -0.14128917],the score is 0.9216

when C is 100,intercept is -0.01420775533762575,coef is[-0.11714805  0.86988028 -0.87396815 -0.06932703  0.21382827 -0.1277446
 -0.01003453  0.63058208  0.29707241 -0.1165773  -0.12663866  0.04970556
  0.13297703 -0.42084601  0.13630659  0.36925499  0.31835181 -0.31445373
 -0.92680662 -0.20588353],the score is 0.9218

when C is 1000,intercept is -0.014193803476248934,coef is[-0.11769847  0.86817723 -0.87884203 -0.06820877  0.21870028 -0.13401109
 -0.01455973  0.64848358  0.30827414 -0.11112858 -0.12516968  0.04905061
  0.12849978 -0.42745089  0.14168354  0.37564086  0.32183116 -0.31810299
 -0.95111207 -0.2163688 ],the score is 0.9218
```

Figure i the parameters of lasso model of different C

As we can see from all of the parameters , the ridge regression is similar to lasso's , when alpha(C) is getting bigger the model is more complex and accuracy would be higher.

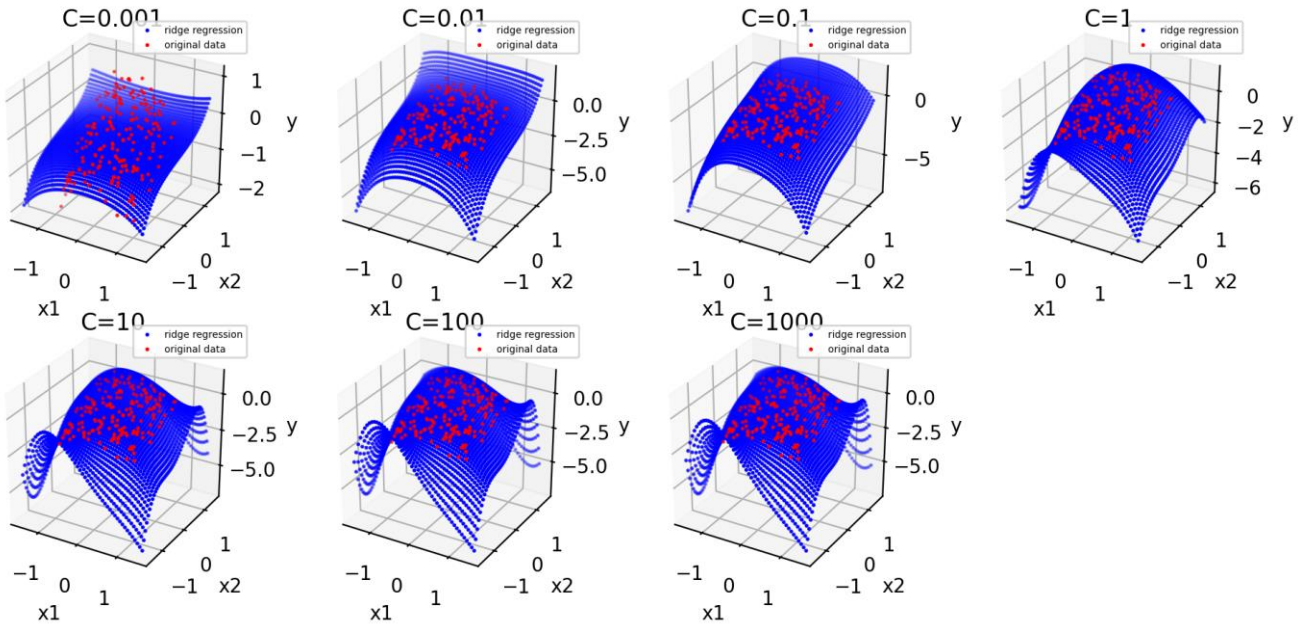

Figure 2 the original data(red points) and the ridge regression model with different value of C

(ii)

I use KFold to get mean square prediction error for each iteration , I define a mse array to store all the mean squared error , then use the np.mean() and np.var() to get the results:The mean is 0.44256087058195037, and the variance of the mse is : 0.00300698516125188

```
from sklearn.model_selection import KFold
kf=KFold(n_splits=5)
mse=[]
for train,test in kf.split(x_poly):
    linlasso = Lasso(alpha=1, max_iter=9999).fit(x_poly[train], y[train])
    ypred=linlasso.predict(x_poly)
    from sklearn.metrics import mean_squared_error
    mse.append(mean_squared_error(ypred,y))
print('the mean of the mse:{0}, the variance of the mse:{1}'.format(np.mean(mse),np.var(mse)))
```

First I define 4 arrays in case to store the mean , standard deviation , variance and the number of K perspectively.

```
mean_kfold_mse = []
standard_deviation_kfold_mse = []
variance_kfold_mse=[]
kfold=[2,5,10,25,50,100]
```

And in each iteration of kfold , and inside the each split , get the list of value of mse, and outside each split , get the variance, mean, standard deviation of each K-fold's MSE.

```
for inx,n_splits in enumerate(kfold):
    kf = KFold(n_splits=n_splits)
    mse = []
    for train, test in kf.split(x_poly):
        linlasso = Lasso(alpha=1, max_iter=9999).fit(x_poly[train], y[train])
        ypred = linlasso.predict(x_poly[test])
        from sklearn.metrics import mean_squared_error
        mse.append(mean_squared_error(ypred, y[test]))
    print('{0} of folds,the mean of the mse:{1}, the var of the mse:{2}'.format(n_splits, np.mean(mse), np.var(mse)))
    variance_kfold_mse.append(np.var(mse))
    mean_kfold_mse.append(np.mean(mse))
    standard_deviation_kfold_mse.append(np.std(mse))
```

Then I create a new figure object to draw using plot and errorbar function for each subplot . xlabel is K. And the plots are as follows.

```
fig4=plt.figure()
plt.subplot(121)
plt.plot(kfold, mean_kfold_mse,label='mean',color='red')
plt.plot(kfold, variance_kfold_mse,label='variance',color='blue')
plt.title('Mean square error and variance')
plt.xlabel('K')
plt.legend()
plt.subplot(122)
plt.errorbar(kfold, mean_kfold_mse,label='mean',color='red',yerr=standard_deviation_kfold_mse)
plt.title('Mean square error')
plt.xlabel('K')
plt.legend()
plt.show()
```

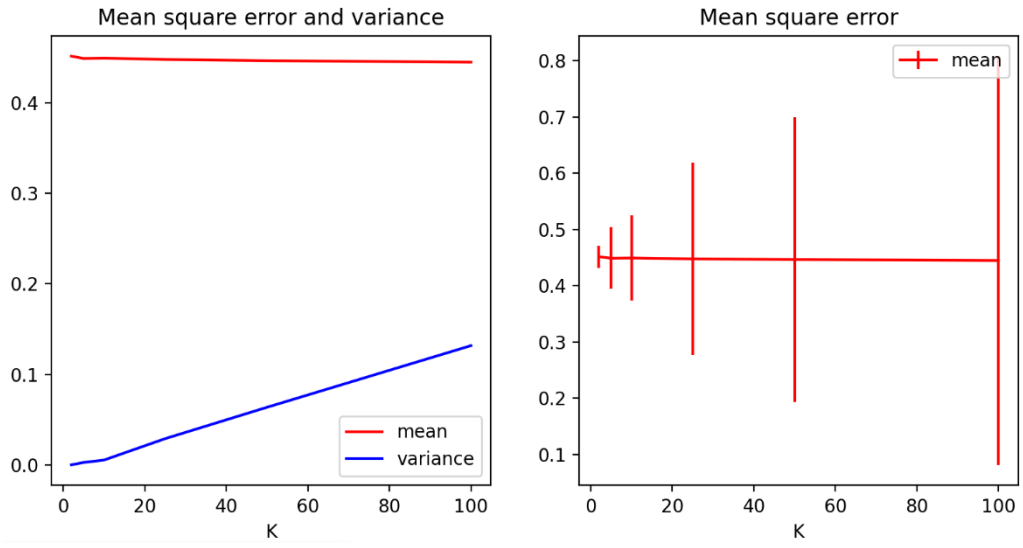

Figure 3 the MSE and variance and the MSE with standard deviation

I think the MSE cannot reflect how good is the model , so I get the accuracy of each K's model using cross_var_score, and each value of K I choose is much more close to each other than before:

```
kfold1=[2,3,4,5,6,7,8,9,10,25,50,100]
mean_score_kfold=[]
linlasso1 = Lasso(alpha=1, max_iter=9999).fit(x_poly, y)
for i in kfold1:
    mean_score_kfold.append(cross_val_score(linlasso1,x_poly,y,cv=i).mean())
```

Then I plot all the three plots just add the codes block as follows:

```
plt.subplot(133)
plt.plot(kfold1,mean_score_kfold,label='mean scores',color='red')
plt.xlabel('K')
plt.title('the scores of different K')
plt.legend()
plt.show()
```

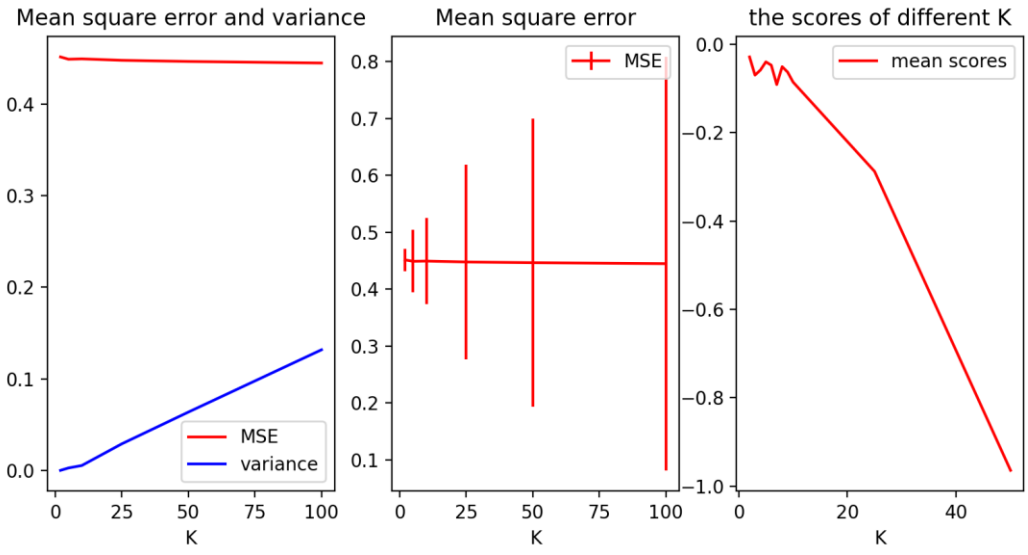

Figure 3 the MSE and variance and the MSE with standard deviation and the scores of different K

I think I would choose K=5 when C=1 , because as we can see from Figure 3 , the mean scores is the higher when K=5 the scores is the highest and the MSE is low.

(b) I choose a wide range of C:

```
c_range=[0.001,0.01,0.1,0.5,1,5,10,20,50,100,500,1000]
```

Then use 2 iterations , for each C's value and for which split the model of train and test dataset for 5 times(K=5) , then append all the mean and std of MSE . For K=10 it's the same and then draw all the two plots.

```
for inx, C in enumerate(c_range):
    kf = KFold(n_splits=5)
    mse = []
    for train, test in kf.split(x_poly):
        linlasso = Lasso(alpha=1 / C, max_iter=9999).fit(x_poly[train], y[train])
        ypred = linlasso.predict(x_poly[test])
        from sklearn.metrics import mean_squared_error
        mse.append(mean_squared_error(ypred, y[test]))
    c_mean_mse.append(np.mean(mse))
    c_std_mse.append(np.std(mse))
```

The output plot is as follows .

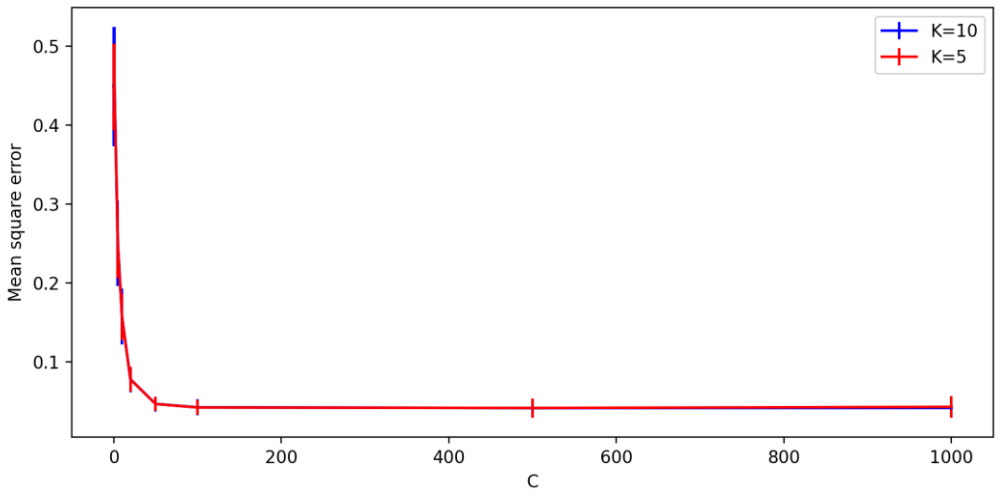

Figure 4 the MSE of different C with different K

And as C gets bigger , the MSE gets smaller , which means the model is better , when C is closer to 0 , the MSE of K=10 and K=5 are all getting bigger and when C is much larger it's hard to tell just by eyes, so I modify the range again.

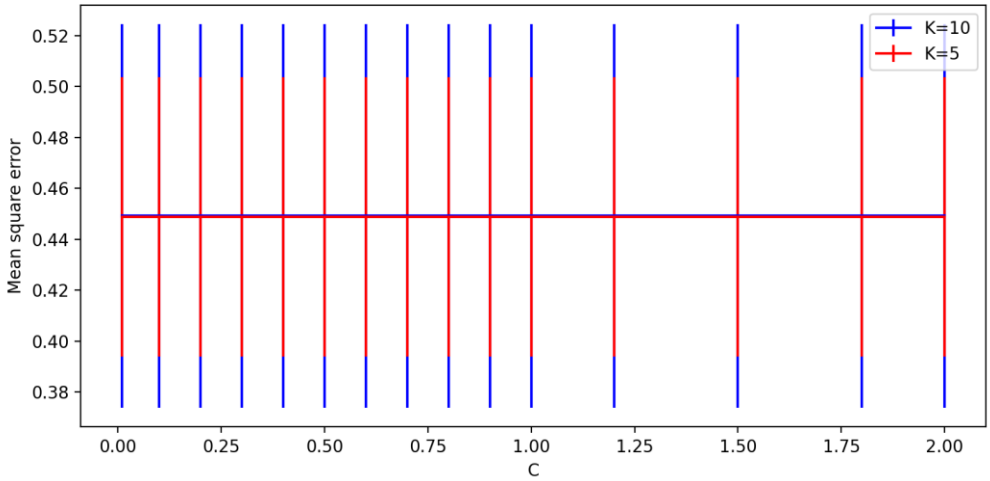

Figure 5 the MSE of different C(new range) with different K

Figure 6 Using magnifier to see the plot , we can find that when K=5 the MSE is smaller

In summary , I would choose K=5 rather than K=10 , because it has lower MSE as well as the value of MSE is distributed more concentrated.

(c)I would recommend C=100 , as we can see from Figure 4 , when C=100 the MSE is small enough even when C gets bigger it seems not changing much , and I would choose to use "simplest" model possible — smallest value of C to avoid overfitting.

(d)

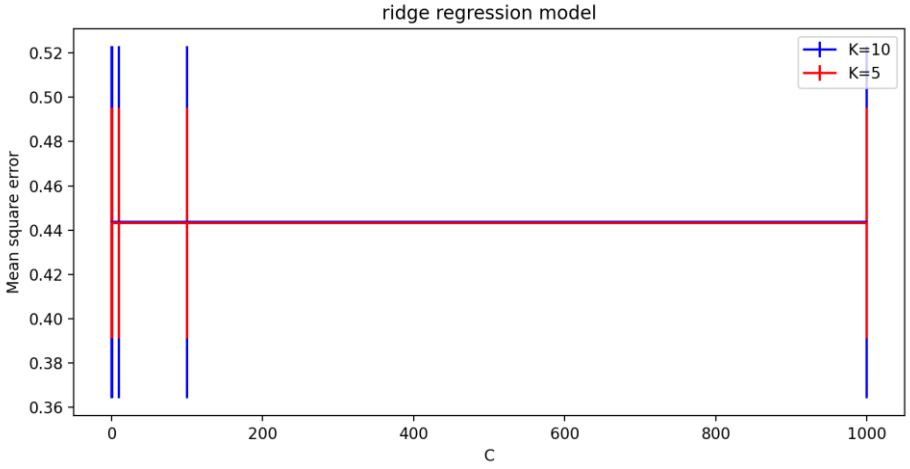

Figure 7 the MSE of different C with different K for ridge regression model

It's the same as in (b) I would choose K=5 , as we can see above , when K=5 the MSE is smaller and it's distributed more concentrated.

The first time I just choose the range of C randomly , and the result is as follows:

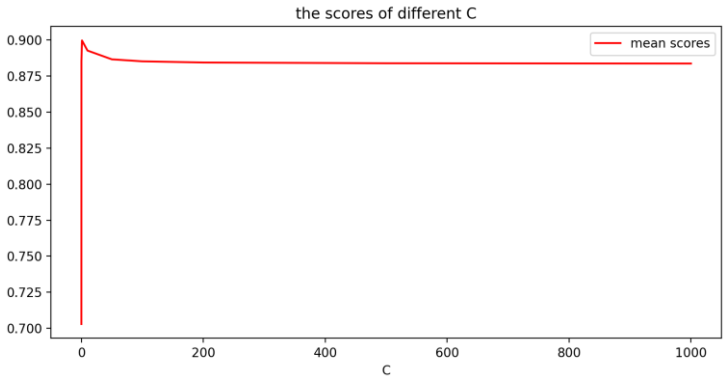

Figure 8 the scores of different C with ridge regression

So I modify the range of C again because I want to find the simplest model — smallest value of C to avoid overfitting.

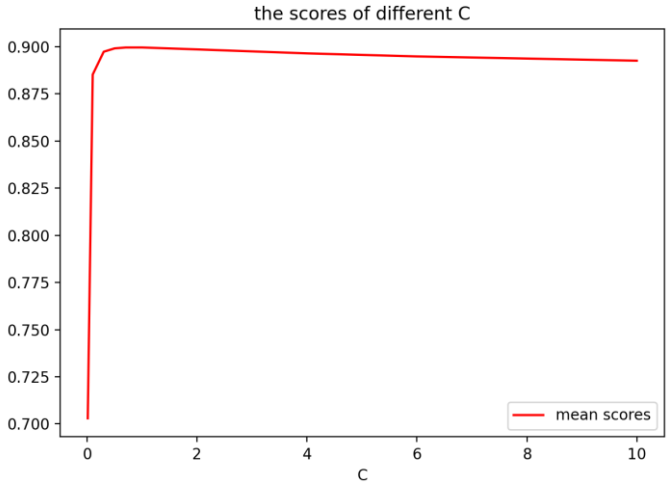

Figure 9 the scores of different C with ridge regression

I think when C is small the accuracy is high , but when it's very close to 0 the accuracy is very low , while C gets bigger the mean accuracy of the model would drop .

And from Figure 7 we would know the MSE of the model doesn't change so much as C gets bigger . So above all , I will choose C=1 .

Appendix:

```
import numpy as np
import pandas as pd
from matplotlib.ticker import MultipleLocator

from mpl_toolkits.mplot3d import Axes3D
from sklearn.linear_model import Lasso
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt
from sklearn.linear_model import Ridge
df = pd.read_csv("week3.csv")
print(df.head())
x = np.array(df.iloc[:, 0:2])
x1 = np.array(df.iloc[:, 0]) #to plot the x_axis
x2 = np.array(df.iloc[:, 1])
y = np.array(df.iloc[:, 2])
#plot original data
fig1 = plt.figure()
ax = fig1.add_subplot(111, projection='3d')
ax.scatter(x1, x2, y, label='original data', color='red')
plt.title("original data")
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('y')
plt.legend()
#plt.show()

poly = PolynomialFeatures(degree=5,
interaction_only=False,include_bias=False)
x_poly = poly.fit_transform(x)
#(c)
""" Because original data x1,x2 lies in (-1,1),
so I choose to generate array of range (-1,1), and then use
PolynomialFeatures to transform the Xtest."""
Xtest =[]
grid=np.linspace(-1.5,1.5) #Return evenly spaced numbers over a
specified interval.
for i in grid:
    for j in grid:
        Xtest.append([i,j])
Xtest=np.array(Xtest)
poly_test = PolynomialFeatures(degree=5,
interaction_only=False,include_bias=False)
```

```
x_poly_test = poly_test.fit_transform(Xtest)
fig2=plt.figure()
"""Then I define a figure object to plot all the models using
add_subplot function.
The value of z axis is the prediction of lasso when input is the
Xtest transformed ."""
#train the range of different alpha for linear lasso regression model
for inx,C in enumerate([0.001,0.01,0.1,1,10,100,1000]):
    alpha=1/C
    linlasso = Lasso(alpha=alpha,max_iter=9999).fit(x_poly, y)
    print('when C is {0},intercept is {1},coef is{2},the score is
{3:.4f}\n'.format(C,

linlasso.intercept_, linlasso.coef_,

linlasso.score(x_poly, y)))
    y_pre_test = linlasso.predict(x_poly_test)
    y_pre_test = y_pre_test.reshape(1,-1)
    ax = fig2.add_subplot(2,4,inx+1,projection='3d')
    ax.scatter(Xtest[:,0], Xtest[:,1], y_pre_test, label="lasso
regression", color='blue',s=1)
    ax.plot_surface(Xtest[:,0], Xtest[:,1], y_pre_test)
    ax.scatter(x1, x2, y, label="original data", color='red',s=1)
    plt.title("C={0}".format(C))
    ax.set_xlabel('x1')
    ax.set_ylabel('x2')
    ax.set_zlabel('y')
    plt.legend()
#plt.show()
#d
fig3=plt.figure()
for inx,C in enumerate([0.001,0.01,0.1,1,10,100,1000]):
    alpha=1/(2*C)
    rige = Ridge(alpha=alpha,max_iter=9999).fit(x_poly, y)
    print('when C is {0},intercept is {1},coef is{2},the score is
{3:.4f}\n'.format(C,

rige.intercept_,

rige.coef_,

rige.score(x_poly, y)))
    y_pre_test = rige.predict(x_poly_test)
    y_pre_test = y_pre_test.reshape(1,-1)
```

```
    ax = fig3.add_subplot(2,4,inx+1,projection='3d')
    ax.scatter(Xtest[:,0], Xtest[:,1], y_pre_test, label="ridge
regression", color='blue',s=1)
    ax.plot_surface(Xtest[:,0], Xtest[:,1], y_pre_test)
    ax.scatter(x1, x2, y, label="original data", color='red',s=1)
    plt.title("C={0}".format(C))
    ax.set_xlabel('x1')
    ax.set_ylabel('x2')
    ax.set_zlabel('y')
    plt.legend()


from sklearn.model_selection import KFold
from  sklearn.model_selection import cross_val_score
#First I define 4 arrays in case to store the mean ,
# standard deviation , variance and the number of K perspectively.
mean_kfold_mse = []
standard_deviation_kfold_mse = []
variance_kfold_mse=[]
kfold1=[2,3,4,5,6,7,8,9,10,25,50,100]
mean_score_kfold=[]
linlasso1 = Lasso(alpha=1, max_iter=9999).fit(x_poly, y)
for i in kfold1:

mean_score_kfold.append(cross_val_score(linlasso1,x_poly,y,cv=i).mean
())
kfold=[2,5,10,25,50,100]
#And in each iteration of kfold , and inside the each split , get the
list of value of mse, and outside each split
# get the variance, mean, standard deviation of each K-fold's MSE.

for inx,n_splits in enumerate(kfold):
    kf = KFold(n_splits=n_splits)
    mse = []
    for train, test in kf.split(x_poly):
        linlasso = Lasso(alpha=1, max_iter=9999).fit(x_poly[train],
y[train])
        ypred = linlasso.predict(x_poly[test])
        from sklearn.metrics import mean_squared_error
        mse.append(mean_squared_error(ypred, y[test]))
    print('{0} of folds,the mean of the mse:{1}, the var of the
mse:{2}'.format(n_splits, np.mean(mse), np.var(mse)))
    variance_kfold_mse.append(np.var(mse))
    mean_kfold_mse.append(np.mean(mse))
```

```
    standard_deviation_kfold_mse.append(np.std(mse))
#Then I create a new figure object to draw using plot and errorbar
function for each subplot .
fig4=plt.figure()
plt.subplot(131)
plt.plot(kfold, mean_kfold_mse,label='MSE',color='red')
plt.plot(kfold, variance_kfold_mse,label='variance',color='blue')
plt.title('Mean square error and variance')
plt.xlabel('K')
plt.legend()
plt.subplot(132)
plt.errorbar(kfold,
mean_kfold_mse,label='MSE',color='red',yerr=standard_deviation_kfold_
mse)
plt.title('Mean square error')
plt.xlabel('K')
plt.legend()
#plot the mean scores of each K
plt.subplot(133)
plt.plot(kfold1,mean_score_kfold,label='mean scores',color='red')
plt.xlabel('K')
plt.title('the scores of different K')
plt.legend()

fig5 = plt.figure()
c_range=[0.01,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,1.2,1.5,1.8,2]
c_mean_mse1 = []
c_std_mse1 = []
c_mean_mse2 = []
c_std_mse2 = []
#when K=5 for lasso
for inx, C in enumerate(c_range):
    kf = KFold(n_splits=5)
    mse = []
    for train, test in kf.split(x_poly):
        linlasso = Lasso(alpha=1 / C,
max_iter=9999).fit(x_poly[train], y[train])
        ypred = linlasso.predict(x_poly[test])
        from sklearn.metrics import mean_squared_error
        mse.append(mean_squared_error(ypred, y[test]))
    c_mean_mse1.append(np.mean(mse))
    c_std_mse1.append(np.std(mse))
#when K=10 for lasso
for inx, C in enumerate(c_range):
```

```
    kf = KFold(n_splits=10)
    mse = []
    for train, test in kf.split(x_poly):
        linlasso = Lasso(alpha=1 / C,
max_iter=9999).fit(x_poly[train], y[train])
        ypred = linlasso.predict(x_poly[test])
        from sklearn.metrics import mean_squared_error
        mse.append(mean_squared_error(ypred, y[test]))
    c_mean_mse2.append(np.mean(mse))
    c_std_mse2.append(np.std(mse))
plt.errorbar(c_range, c_mean_mse2, label='K=10', color='blue',
yerr=c_std_mse2)
plt.errorbar(c_range, c_mean_mse1, label='K=5', color='red',
yerr=c_std_mse1)
plt.ylabel('Mean square error')
plt.xlabel('C')
plt.legend()
#(ii)(d)
#ridge model
#when K=5 for ridge model
fig6=plt.figure()
c_range1=[0.001,0.01,0.1,1,10,100,1000]
c_mean_mse3 = []
c_std_mse3 = []
c_mean_mse4 = []
c_std_mse4 = []
#get the mean of MSE and standard deviation
for inx, C in enumerate(c_range1):
    kf = KFold(n_splits=5)
    mse = []
    for train, test in kf.split(x_poly):
        ridge = Ridge(alpha=1 / (2*C),
max_iter=9999).fit(x_poly[train], y[train])
        ypred = linlasso.predict(x_poly[test])
        from sklearn.metrics import mean_squared_error
        mse.append(mean_squared_error(ypred, y[test]))
    c_mean_mse3.append(np.mean(mse))
    c_std_mse3.append(np.std(mse))
#when K=10 for ridge model
for inx, C in enumerate(c_range1):
    kf = KFold(n_splits=10)
    mse = []
    for train, test in kf.split(x_poly):
        ridge = Ridge(alpha=1 / (2*C),
```

```
max_iter=9999).fit(x_poly[train], y[train])
        ypred = linlasso.predict(x_poly[test])
        from sklearn.metrics import mean_squared_error
        mse.append(mean_squared_error(ypred, y[test]))
    c_mean_mse4.append(np.mean(mse))
    c_std_mse4.append(np.std(mse))
#draw the MSE with different C and K
plt.errorbar(c_range1, c_mean_mse4, label='K=10', color='blue',
yerr=c_std_mse4)
plt.errorbar(c_range1, c_mean_mse3, label='K=5', color='red',
yerr=c_std_mse3)
plt.ylabel('Mean square error')
plt.title('ridge regression model')
plt.xlabel('C')
plt.legend()
#draw the mean scores of different C using ridge model
fig7=plt.figure()
mean_score_crange=[]
crange_ridge = [0.01,0.1,0.3,0.5,0.7,1,2,4,6,10]
for c in crange_ridge:
    ridge = Ridge(alpha=1/(2*c), max_iter=9999).fit(x_poly, y)
    mean_score_crange.append(cross_val_score(ridge, x_poly, y,
cv=5).mean())
plt.plot(crange_ridge,mean_score_crange,label='mean
scores',color='red')
plt.xlabel('C')
plt.title('the scores of different C')
plt.legend()

plt.show()
```