(i)

(a)First plot the dummy training data, just 3 points there is one input feature and when it equals -1 or 1 the output is 0 and when it equals 0 the output is 1:

```
x_dummy_training_data=np.array([-1,0,1]).reshape(-1,1)
y_dummy_training_data=np.array([0,1,0]).reshape(-1,1)
```

Then I generate the Xtest use grid of feature values range from -3 to 3:

```
Xtest =[]
grid=np.linspace(-3,3)
for x in grid:
    Xtest.append(x)
Xtest=np.array(Xtest).reshape(-1,1)
```

And I define a `gaussian_kernel_funcfile.py` to get the gaussian_kernel0 function with different gamma values.(N is the value of gamma, to write shortly just use N as format)

```
def gaussian_kernelN(distances):
    weights=np.exp(-N*(distances**2))
    return weights/np.sum(weights)
```

Then train the KNeighborsRegressor model(**i** is the sequence of the model:1,2,3,4,5 of different gamma value:0,1,5,10,25) with the data (`x_dummy_training_data, y_dummy_training_data`), then get the predictions of Xtest.

```
modeli=KNeighborsRegressor(n_neighbors=3,weights=gaussian_kernelN).fit(x_dummy_training
_data, y_dummy_training_data)
ypredi = modeli.predict(Xtest)
```

Then plot the different predictions:

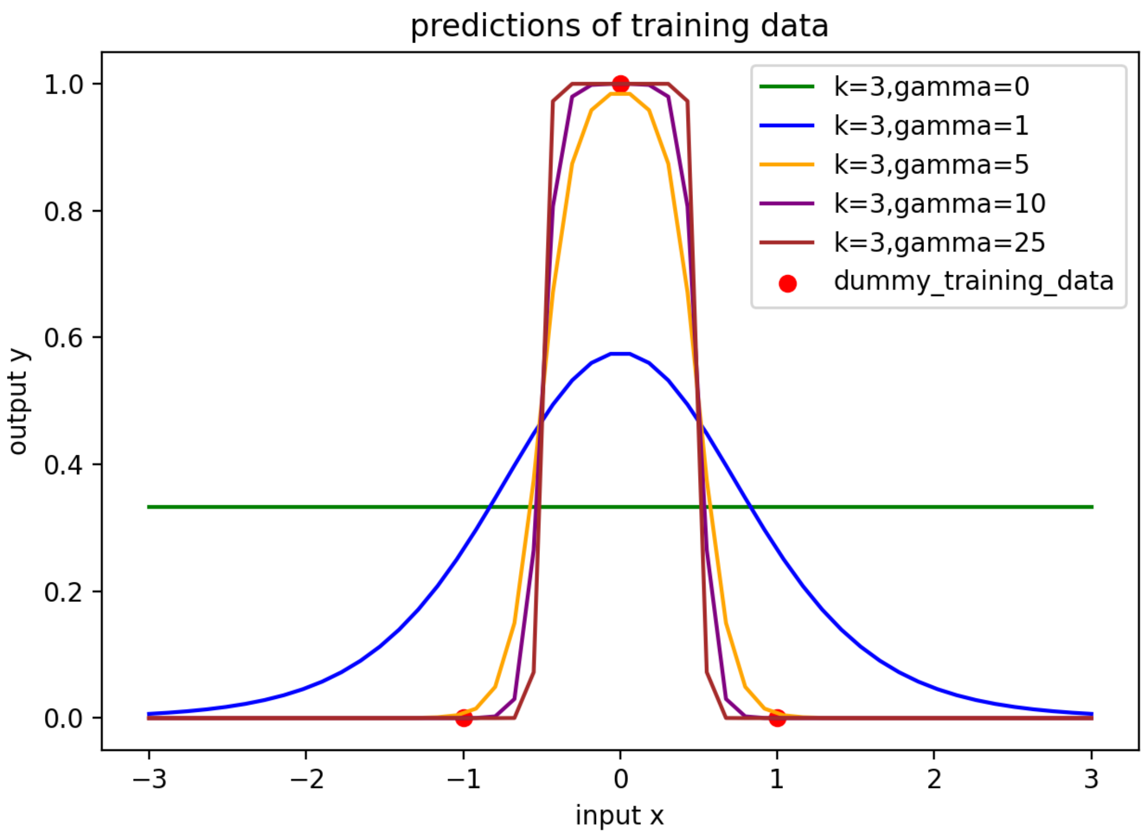


Figure ai.1 the predictions of Xtest

As we can see from Figure ai.1 the dummy_training_data is 3 points , and all the predictions are symmetrical, when gamma=0 it’s a straight line paralleling the x axis, and when gamma gets bigger the curve gets shaper, the speed of getting close to 1 is higher.

(b) The gamma parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’. As γ increases the kernel decreases more quickly with distance. This makes the predictions tend to be less smooth and to just snap to the nearest training point.
If gamma has a high value, means that each training example only has a close reach so the plot has the curve and gets sharper as gamma gets bigger.
If gamma is small, then that means every point has a far reach, the model is too constrained and cannot capture the complexity or “shape” of the data. when gamma=0 it’s a straight line paralleling the x axis, which means all the 3 training data have the same influence on predictions.

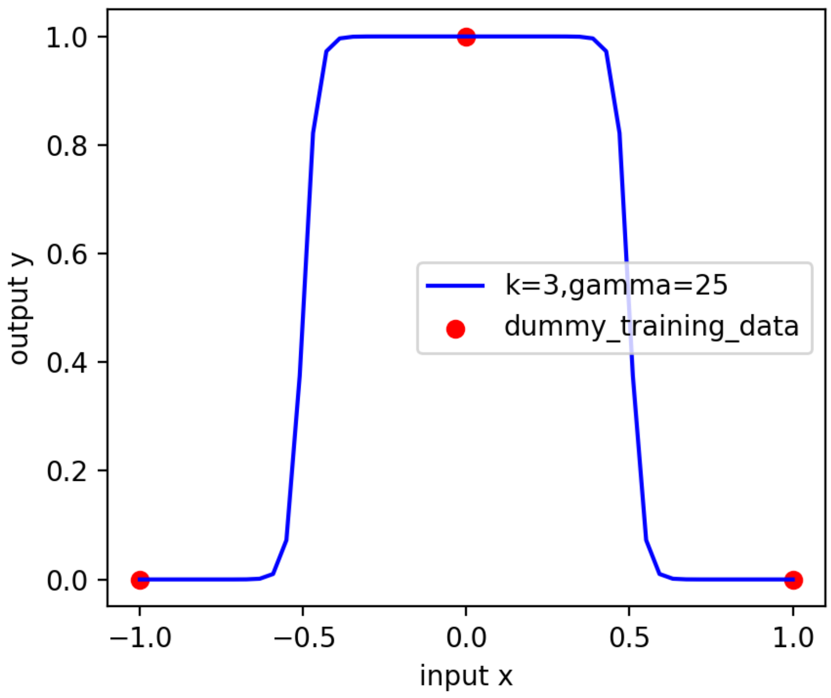


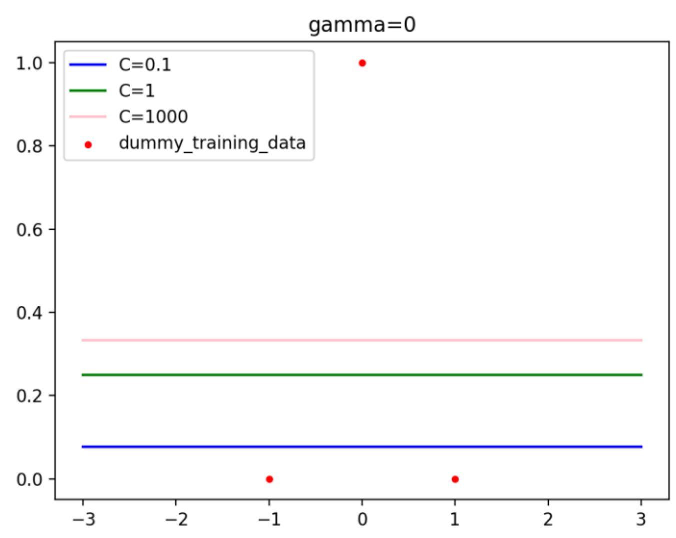
Figure bi.1 the predictions of the model(k=3,gamma=25)when the range of input x is (-1,1)

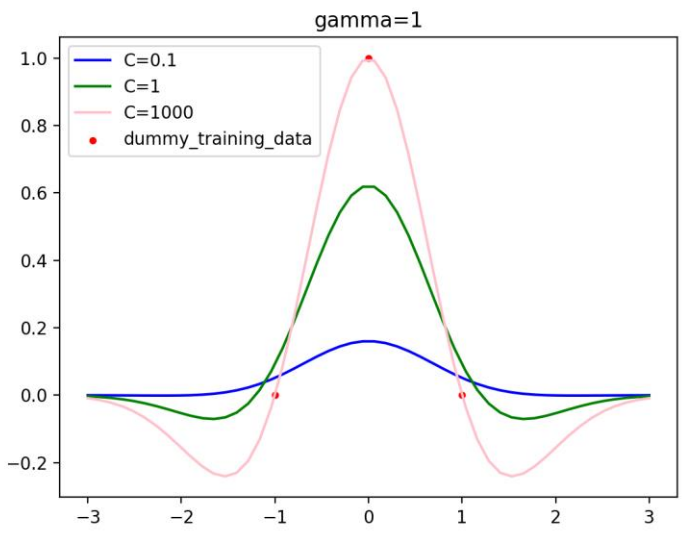
The model we train only with 3 data: (-1,0),(0,1),(1,0), when gamma=25 the value is big and which means the weight of each training data is big, so the influence of the data (0,1) would be much bigger when X is in the range(-0.5,0.5), because the X in this range means closer to (0,1) so influence much more with this data’s value which equals 1.
(c)First I define the 3 ranges as follows,
the blue line is for C=0.1,green line C=1,and pink line means C=1000:

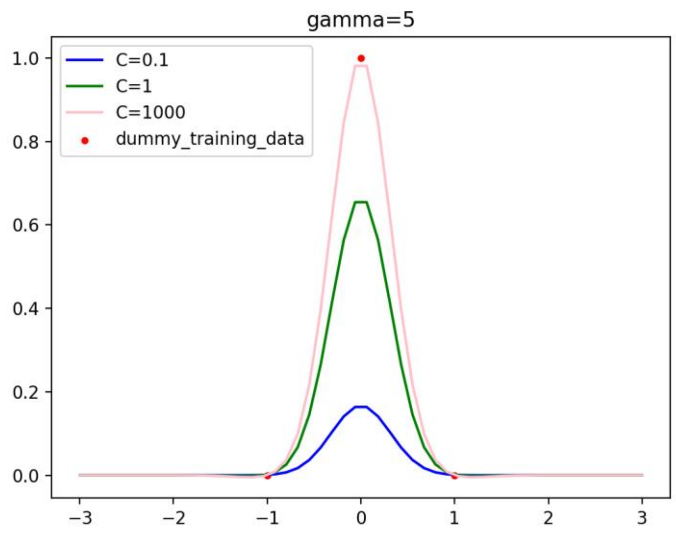
```
c_range=[0.1,1,1000]
gamma_range=[0,1,5,10,25]
color_range=['blue','green','pink']
```

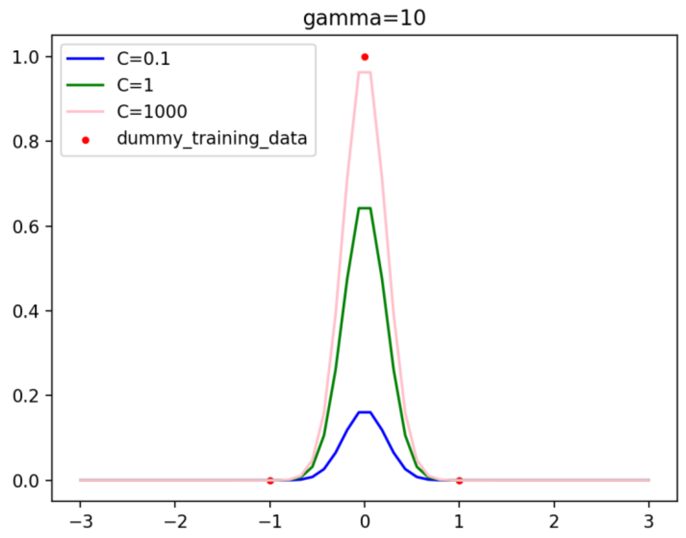
Then I use 2 iterations to plot, in gamma iteration I define a new figure to plot with different C value.In c_range iteration I train the KernelRidge with alpha=1/2C and gaussian kernel.

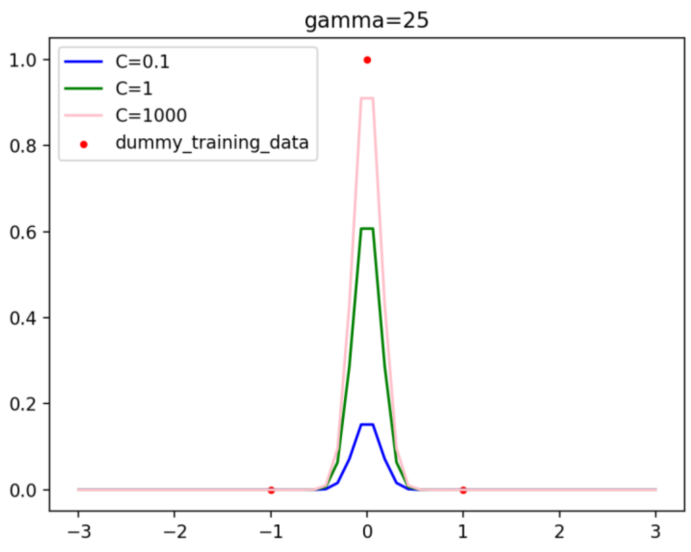
```
for inx1,gamma in enumerate(gamma_range):
    fig3 = plt.figure()
    plt.scatter(x_dummy_training_data, y_dummy_training_data, color='red',
                label="dummy_training_data",s=10)  # plot the dummy training data
    for inx2,C in enumerate(c_range):
        model = KernelRidge(alpha=1.0/(2*C), kernel='rbf',
gamma=gamma).fit(x_dummy_training_data, y_dummy_training_data)
        ypred_kernelridge=model.predict(Xtest)
        plt.plot(Xtest,ypred_kernelridge,color=color_range[inx2],label='C={}'.format(C))
        plt.legend(loc='upper left', fontsize=10)
        plt.title('gamma={}'.format(gamma))
    plt.show()
```











PLOT DESCRIPTION:
When gamma=0 no matter what's the value of C the model is a straight line paralleling the x axis and with C gets bigger the line is nearer the data (0,1). And as gamma gets bigger the curve gets sharper, as C gets bigger, the closer to the data (0,1).
And the parameters of RidgeKernel model are as follows, and the value of theta1,theta2,theta3 is the value of dual_coef_ respectively.

γ	C	KernelRidge model	parameters values	γ	C	KernelRidge model	parameters values	KernelRidge model	γ	C	parameters values
0	0.1	(1+θ1 f1(x)+θ 2f2(x)+ θ3f3(x))	[-0.025] [0.175] [-0.025]	1	0.1	(1+θ1 f1(x)+θ 2f2(x)+ θ3f3(x))	[-0.01026472] [0.16792539] [-0.01026472]	(1+θ1 f1(x)+θ 2f2(x)+ θ3f3(x))	10	0.1	[-1.26110916e-06] [1.66666667e-01] [-1.26110916e-06]
0	1	(1+θ1 f1(x)+θ 2f2(x)+ θ3f3(x))	[-0.57142857] [1.42857143] [-0.57142857]	1	1	(1+θ1 f1(x)+θ 2f2(x)+ θ3f3(x))	[-0.18331618] [0.75658434] [-0.18331618]	(1+θ1 f1(x)+θ 2f2(x)+ θ3f3(x))	10	1	[-2.01777466e-05] [6.66666668e-01] [-2.01777466e-05]
0	1000	(1+θ1 f1(x)+θ 2f2(x)+ θ3f3(x))	[-666.55557407] [1333.44442593] [-666.55557407]	1	1000	(1+θ1 f1(x)+θ 2f2(x)+ θ3f3(x))	[-0.49138748] [1.36086227] [-0.49138748]	(1+θ1 f1(x)+θ 2f2(x)+ θ3f3(x))	10	1000	[-4.53545640e-05] [9.99500254e-01] [-4.53545640e-05]
5	0.1	(1+θ1 f1(x)+θ 2f2(x)+ θ3f3(x))	[-0.00018717] [0.16666709] [-0.00018717]	25	0.1	(1+θ1 f1(x)+θ 2f2(x)+ θ3f3(x))	[-3.85776218e-13] [1.66666667e-01] [-3.85776218e-13]				
5	1	(1+θ1 f1(x)+θ 2f2(x)+ θ3f3(x))	[-0.00299476] [0.66669357] [-0.00299476]	25	1	(1+θ1 f1(x)+θ 2f2(x)+ θ3f3(x))	[-6.17241950e-12] [6.66666667e-01] [-6.17241950e-12]				
5	1000	(1+θ1 f1(x)+θ 2f2(x)+ θ3f3(x))	[-0.00673182] [0.99959092] [-0.00673182]	25	1000	(1+θ1 f1(x)+θ 2f2(x)+ θ3f3(x))	[-1.38740663e-11] [9.99500250e-01] [-1.38740663e-11]				

As C gets bigger, the dual_coef gets bigger, and as gamma gets bigger the dual_coef get smaller.

(d)

The gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. As γ increases the kernel decreases more quickly with distance. This makes the predictions tend to be less smooth and to just snap to the nearest training point. The behavior of the model is very sensitive to the gamma parameter. If gamma is too large, the radius of the area of influence of the support vectors only includes the support vector itself and no amount of regularization with C will be able to prevent overfitting. When gamma is very small, the model is too constrained and cannot capture the complexity or "shape" of the data. The region of influence of any selected support vector would include the whole training set. The resulting model will behave similarly to a linear model with a set of hyperplanes that separate the centers of high density of any pair of two classes.

Finally one can also observe that for some intermediate values of gamma we get equally performing models when C becomes very large: it is not necessary to regularize by enforcing a larger margin.

If gamma has a high value, means that each training example only has a close reach so the plot has the curve and gets sharper as gamma gets bigger.

If gamma is small, then that means every point has a far reach, the model is too constrained and cannot capture the complexity or "shape" of the data. when gamma=0 it's a straight line paralleling the x axis, which means all the 3 training data have the same influence on predictions.

Because the KernelRidge model performs L2 regularization, penalty equivalent to square of the magnitude of coefficients(the values of θi), so the C parameter trades off correct regression of training examples against maximization of the decision function's margin. A high C value can lead to over-fitting, whereas a low C value can lead to under-fitting.

$\alpha \sum_{i=1}^{n} \theta_i^2$ is added to the cost function This forces the learning algorithm to not only fit the data but also keep the model weights as small as possible. The hyperparameter C controls how much you want to regularize the model. If C is very large all weights(the values of θi) gets large which may lead to over-fitting. If C is close to 0, then all weights(the values of θi) end up very close to zero and the result is a flat line going through the data's mean. The cost function is :

$$J(\theta) = \text{MSE}(\theta) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$$

For larger values of C of the same value of gamma, the dual_coef_ gets bigger, the θi would get larger, which means less error could be tolerate. A lower C will encourage a larger margin, therefore a simpler decision function, at the cost of training accuracy. In other word C behaves as a regularization parameter. So as C gets smaller the θi tends to 0 so the plot would be flatter. And the behaviour of the kernalised ridge regression predictions with that of the kNN predictions is quite similar as gamma gets larger, the plot gets sharper, when data is in (-0.5,0.5) the predictions are close to 1 in a higher speed.
(ii)(a)
First read the data and reshape it.
```
df = pd.read_csv("week6.csv")
x = np.array(df.iloc[:, 0]).reshape(-1,1)#read the x value as array
y = np.array(df.iloc[:, 1]).reshape(-1,1)#read the y value
```
Then I define a weights range(I use a larger range of gamma:0,1,5,10,25,50,100,200) to use this iteration to plot all the predictions, plot the original training data in the plot too. In each iteration train the KNN model with K=999 and predict with the Xtest to get the ypred_knn then plot as y value in different colors (Figure aii.1)
```
plt.scatter(x, y, color='red', label="original training data",s=3)
for inx3,weights in enumerate(weights_range):
    model_knn = KNeighborsRegressor(n_neighbors=999, weights=weights).fit(x, y)
    ypred_knn = model_knn.predict(Xtest)
    plt.plot(Xtest, ypred_knn,
color=color_range[inx3],label='gamma={}'.format(gamma_range[inx3]))
    plt.xlabel('input x');plt.ylabel('output y');
    plt.title('KNN model predictions when k=999 with different gamma')
    plt.legend(fontsize=5)
plt.show()
```
(Describe how the predictions change as you vary γ, and explain why):
The gamma parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’. As γ increases the kernel decreases more quickly with distance. This makes the predictions tend to be less smooth and to just snap to the nearest training point.
If gamma has a higher value, means that each training example only has a closer reach so the plot has the curve and gets sharper and more fit the original data as gamma gets bigger.
If gamma is small, then that means every point has a far reach, the model is too constrained and cannot capture the complexity or “shape” of the data, when gamma=0 it’s a straight line paralleling the x axis and gets more flat and less fit the data as gamma is smaller.
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.
When gamma is bigger, this may lead to overfitting since the data is shaped like a convex surface but the prediction is not convex near the edges, the prediction is much steeper than the training data near the boundary. If a model has been trained too well on training data, it will be unable to generalize. It will make inaccurate predictions when given new data, making the model useless even though it is able to make accurate predictions for the training data.
When gamma is small would lead underfitting is when the model is too simple to fit the data. When gamma=0 the prediction is a plane(just straight line paralleling X axis) cannot model

quadratic data, it is not specific enough to capture relevant information. It is unable to make accurate predictions for training or new data, this would be unable to generalize either.

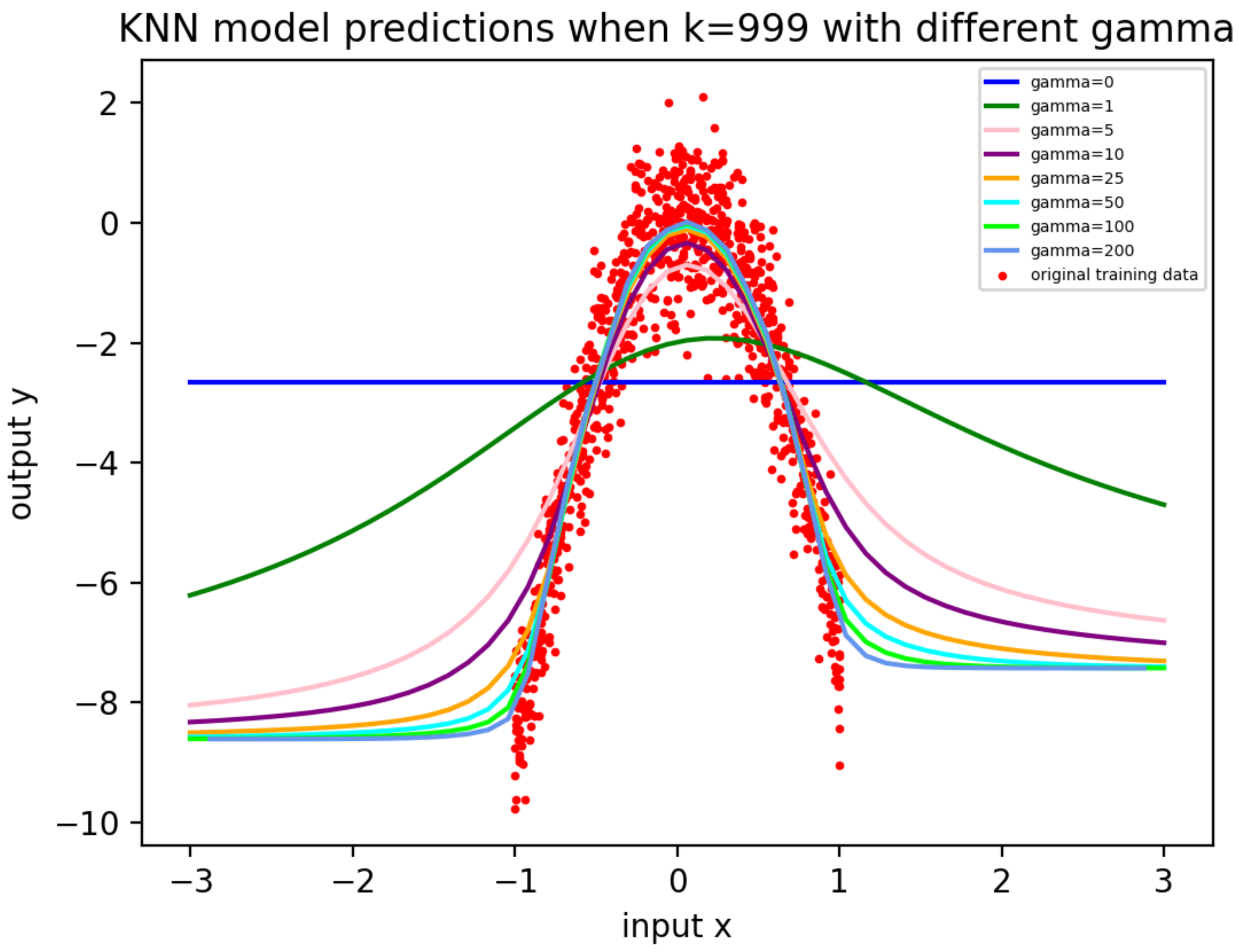
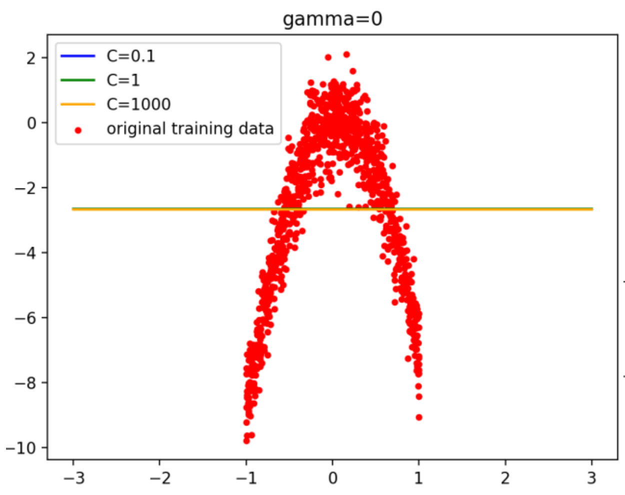
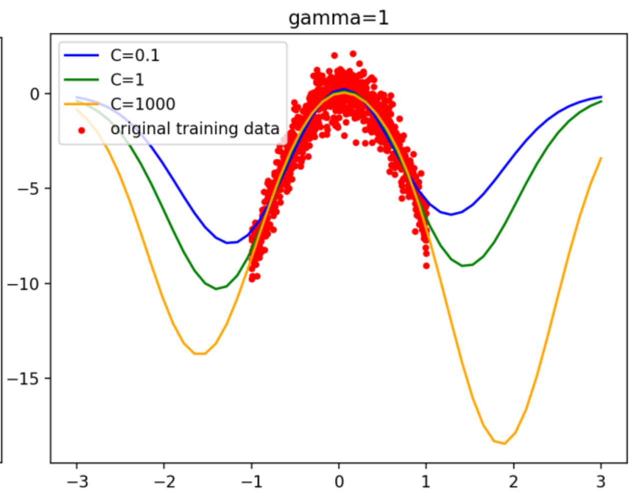


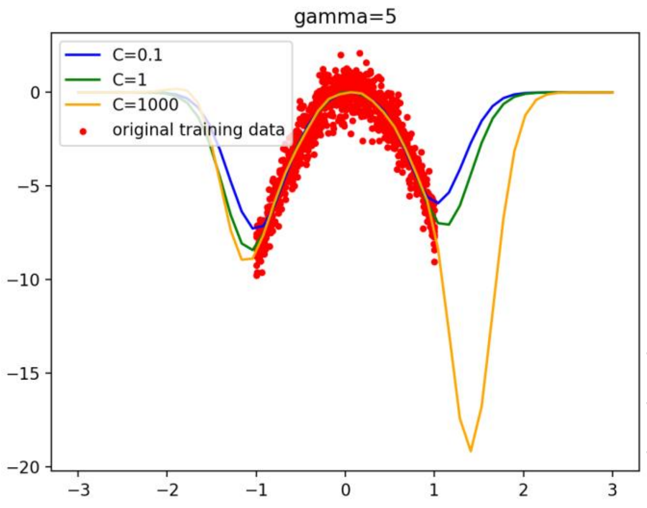
Figure aii.1 KNN model predictions when k=999 with different gamma

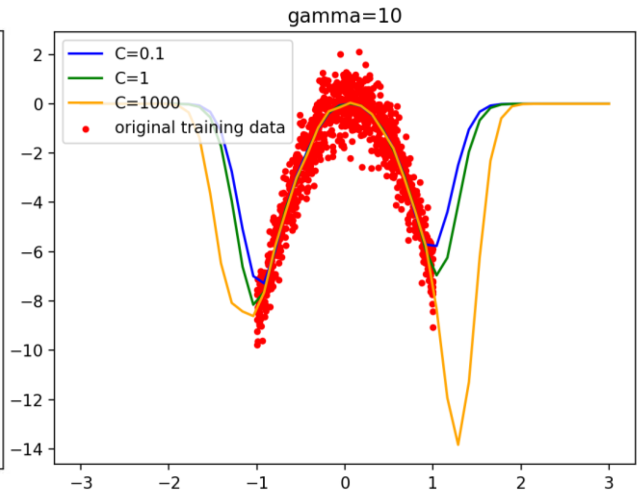
(b)First train the KernelRidge model with the original data, then use 2 iteration to plot for each gamma(I use a larger range of gamma:0,1,5,10,25,50,100,200) with different value of C:

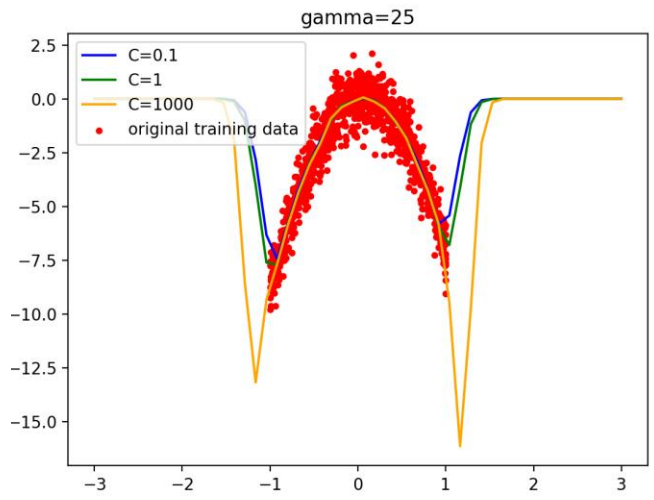
```
for inx4,gamma in enumerate(gamma_range1):
    fig5 = plt.figure() #for each gamma has one plot with different values of C
    plt.scatter(x, y, color='red',
                label="original training data",s=10)  # plot the original training data
    for inx4,C in enumerate(c_range):
        model = KernelRidge(alpha=1.0/(2*C), kernel='rbf', gamma=gamma).fit(x, y)
        ypred_kernelridge=model.predict(Xtest)
        print("KernelRidge model when
gammma={0},C={1},dual_coef={2}".format(gamma,C,model.dual_coef_))
        plt.plot(Xtest,ypred_kernelridge,color=color_range[inx4],label='C={}'.format(C))
        plt.legend(loc='upper left', fontsize=10)
        plt.title('gamma={}'.format(gamma))
    plt.show()
```

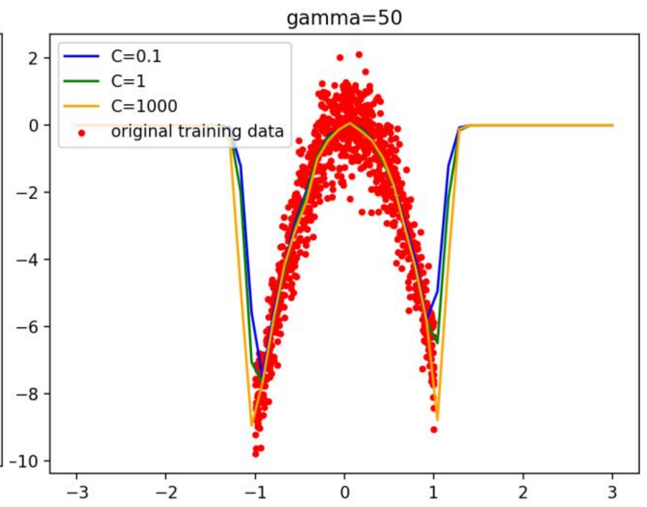

gamma=0
C=0.1
C=1
C=1000
original training data

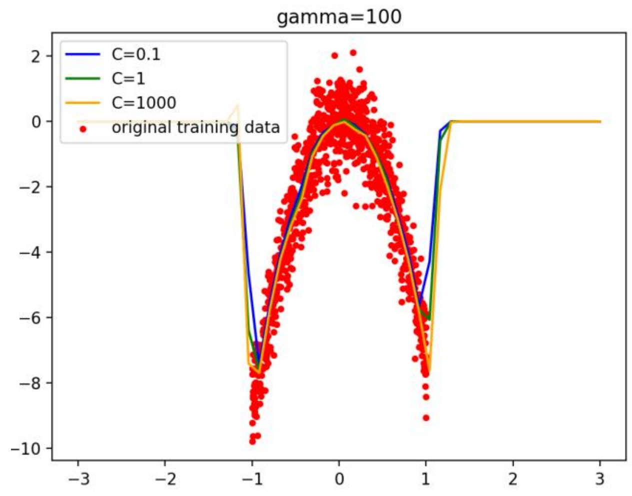

gamma=1
C=0.1
C=1
C=1000
original training data

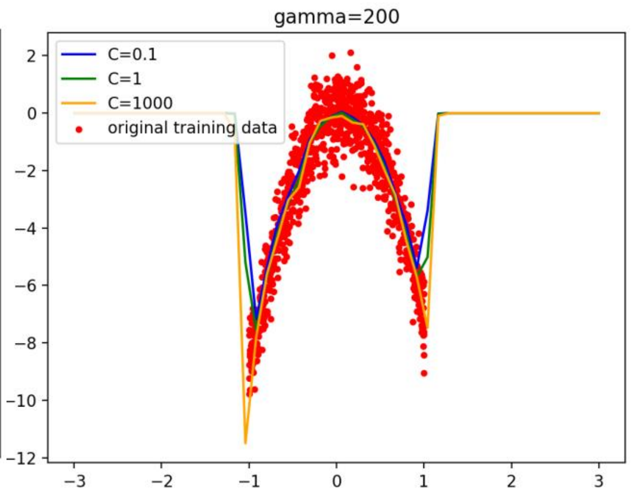

gamma=5
C=0.1
C=1
C=1000
original training data


gamma=10
C=0.1
C=1
C=1000
original training data


gamma=25
C=0.1
C=1
C=1000
original training data


gamma=50
C=0.1
C=1
C=1000
original training data


gamma=100
C=0.1
C=1
C=1000
original training data


gamma=200
C=0.1
C=1
C=1000
original training data

The gamma parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’. As γ increases the kernel decreases more quickly with distance. This makes the predictions tend to be less smooth and to just snap to the nearest training point. The behavior of the model is very sensitive to the gamma parameter.

Finally one can also observe that for some intermediate values of gamma we get equally performing models when C becomes very large: it is not necessary to regularize by enforcing a larger margin.

When gamma is bigger, the data is shaped like a convex surface and the prediction is much steeper than the training data near the boundary.
If gamma has a high value, means that each training example only has a close reach so the plot has the curve and gets sharper as gamma gets bigger.
If gamma is small, then that means every point has a far reach, the model is too constrained and cannot capture the complexity or “shape” of the data, and when gamma=0 the value of dual_coef_ is 0 and it’s a straight line paralleling the x axis.
(d)
First I use cross-validation to get the MSE and score of different models of different hyperparameter γ for the kNN model. Because in kf.split the number of x[train] is 799, so I set the K=799.

```
for inx5,weights in enumerate(weights_range1):
    kf = KFold(n_splits=5)
    model_knn1 = KNeighborsRegressor(n_neighbors=799, weights=weights)
    mse = []
    for train, test in kf.split(x):
        model_knn1.fit(x[train], y[train])
        y_pre_knn1 = model_knn1.predict(x[test])
        from sklearn.metrics import mean_squared_error
        mse.append(mean_squared_error(y[test],y_pre_knn1))
    mean_error_knn.append(np.array(mse).mean())
    std_error_knn.append(np.array(mse).std())
    mean_score_knn.append(cross_val_score(model_knn1, x, y, cv=5).mean())
```

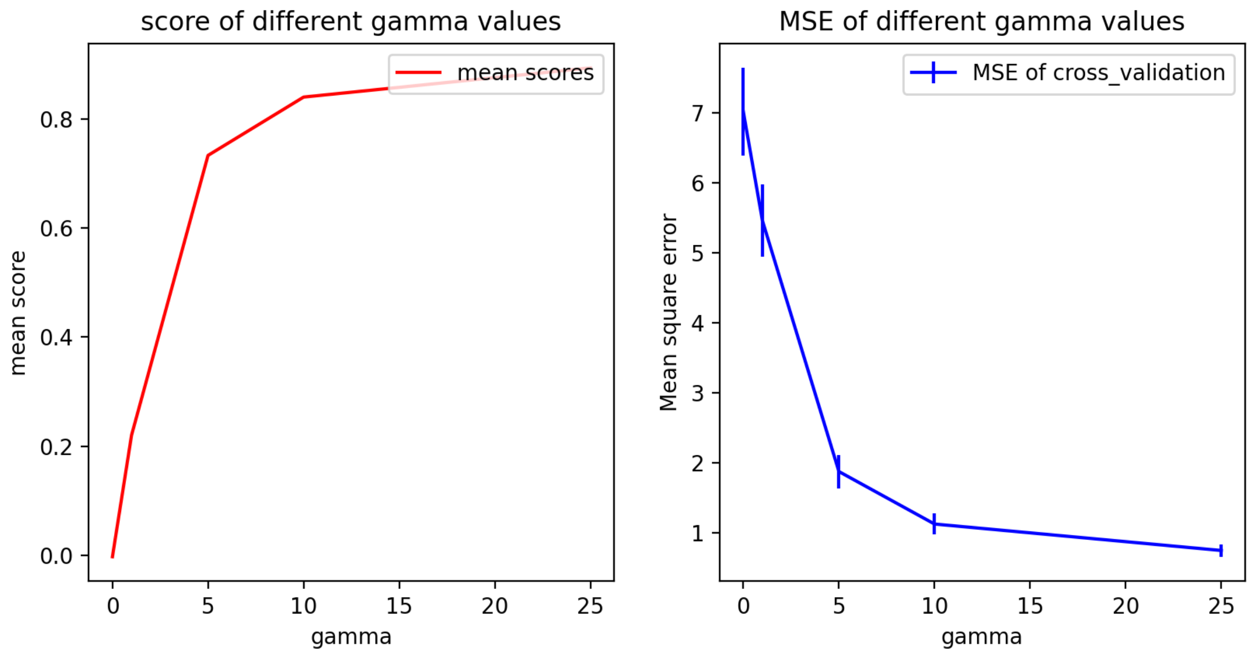


Figure dii.1 when gamma_range:[0,1,5,10,25]

As gamma gets bigger the mean score gets bigger and the MSE gets lower, so I decide to choose a larger range of gamma values to see in detail.

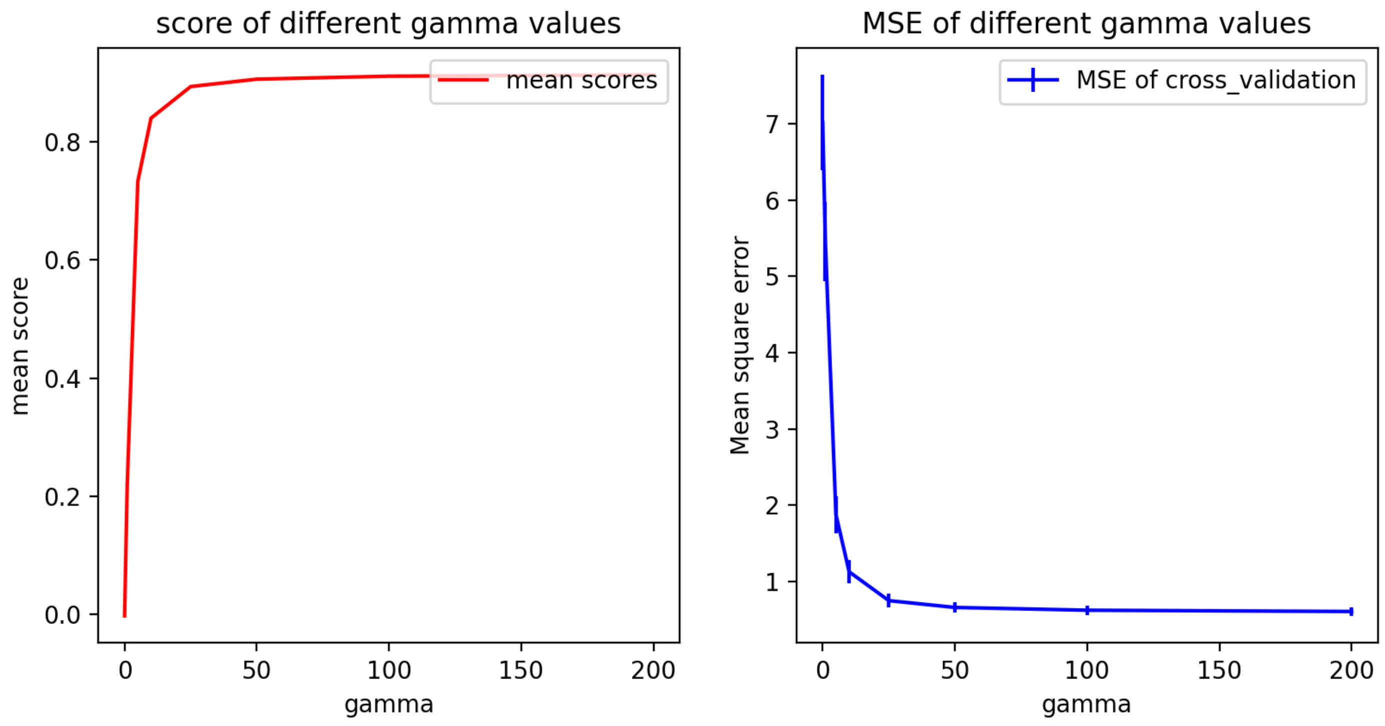


Figure dii.2 the mean score and MSE of KNN model(gamma range:[0,1,5,10,25,50,100,200])

When gamma >=50 the mean score and MSE doesn't change much as gamma varies, to avoid over-fitting I decide to choose simplest model with high score and low MSE, that's gamma=50.Then I train the final model and plot it. As we can see Figure dii.3 the prediction fits really well with the original training data.

```
 model_knn_final =
KNeighborsRegressor(n_neighbors=999,weights=gaussian_kernel50).fit(x,y)
ypred_knn_final=model_knn_final.predict(Xtest)
```

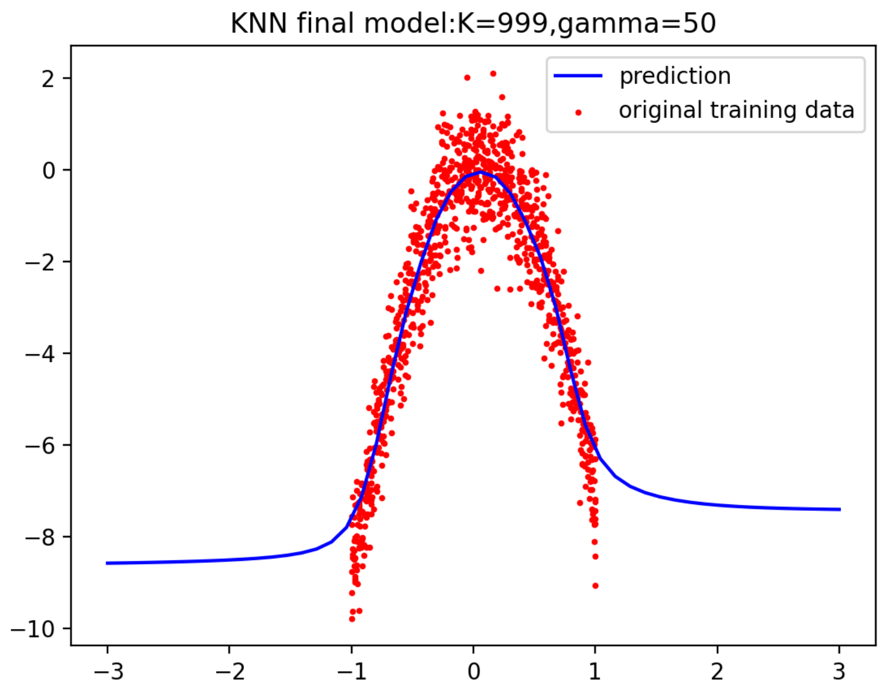


Figure dii.3 KNN final model when gamma=50

As we can see from (i)(b) when C is large the curve of the prediction is too sharper and there is the possibility to be over-fitting, so I decided to choose C=0.1 in beginning, that is alpha=1/(2*0.1)=5, to find the best gamma.

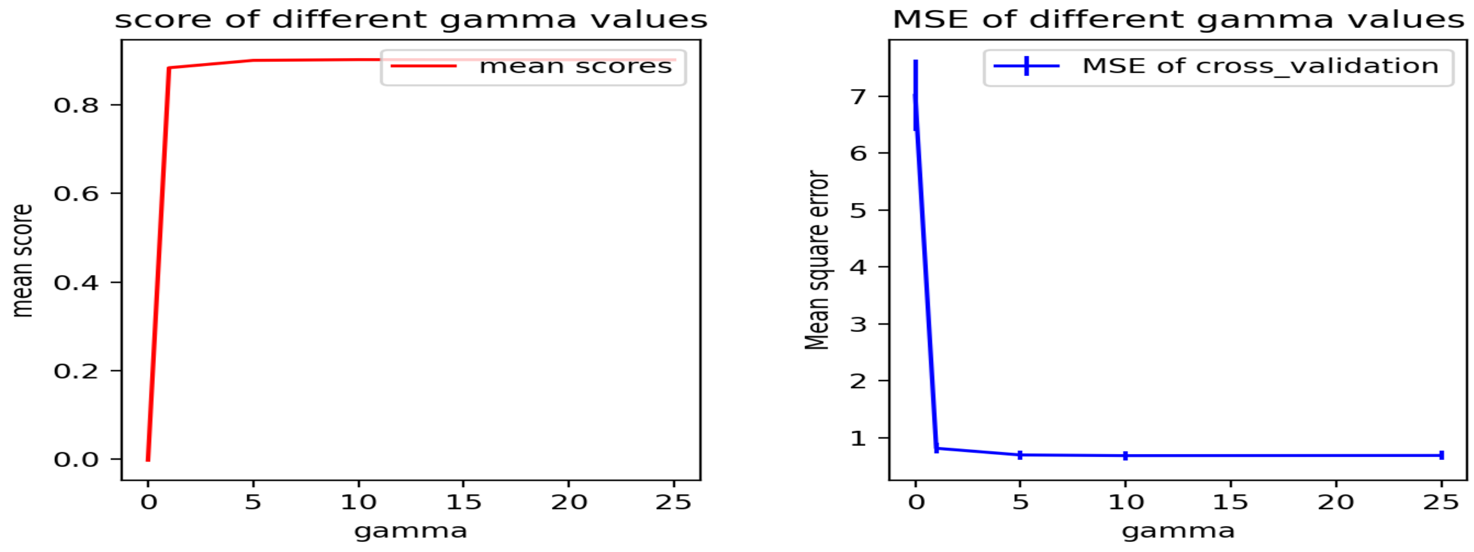


Figure dii.4 the mean score and MSE of KernalRidge of different gamma

As we can see from figure dii.4 when gamma gets bigger the mean score gets bigger however when gamma>=5 the mean score doesn’t change much, and the MSE gets smaller and when gamma>=5 it doesn’t change much either, to avoid over-fitting I decide to choose the simplest model, that is gamma=5.

Then it’s turn for alpha: `alpha_range=[0.01,0.05,0.1,0.5,1,2,3,4,5]`

```
for alpha in alpha_range:
    kf = KFold(n_splits=5)
    kr_model1 = KernelRidge(alpha=alpha, kernel='rbf', gamma=5)
    mse = []
    for train, test in kf.split(x):
        kr_model1.fit(x[train], y[train])
        y_pre_kr = kr_model1.predict(x[test])
        from sklearn.metrics import mean_squared_error
        mse.append(mean_squared_error(y[test],y_pre_kr))
    mean_error_kr1.append(np.array(mse).mean())
    std_error_kr1.append(np.array(mse).std())
    mean_score_kr1.append(cross_val_score(kr_model1, x, y, cv=5).mean())
```

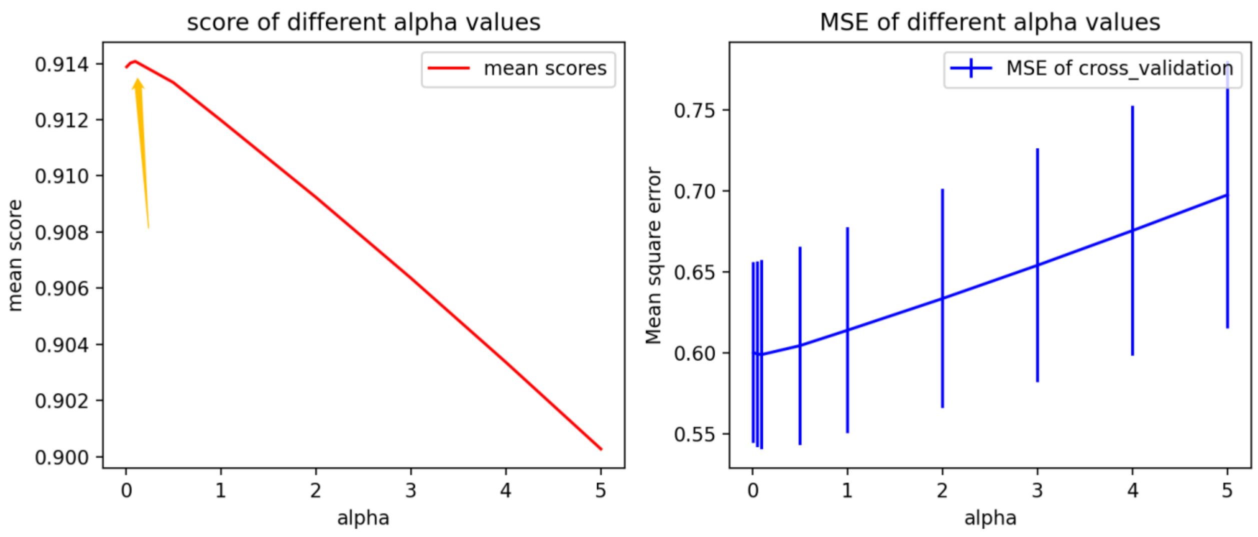


Figure dii.5 the mean score and MSE of KernalRidge of different alpha

To see more clear of the plot as I point in yellow arrow, I modify the alpha range to a small values:

```
alpha_range=[0.01,0.03,0.05,0.07,0.1,0.15,0.2]
```

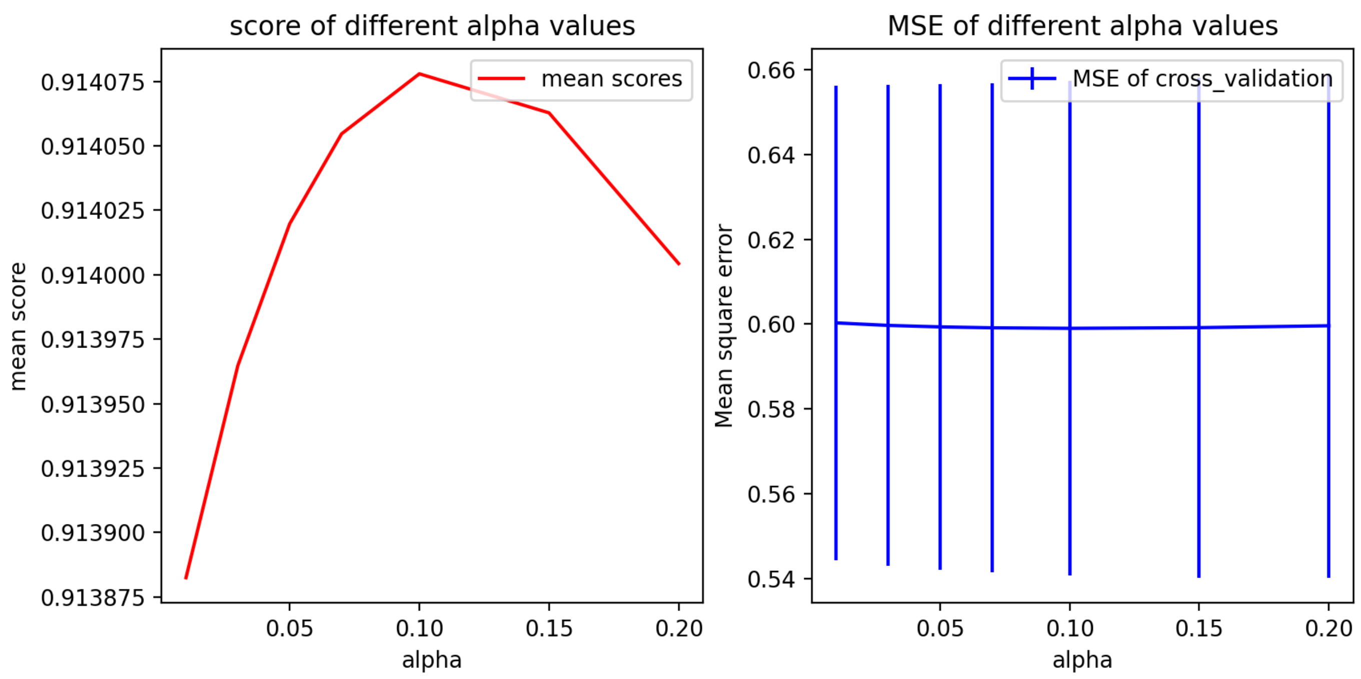


Figure dii.6 the mean score and MSE of KernalRidge of different alpha

As we can see from Figure dii.6 when alpha gets bigger the mean score increases when alpha<=0.1 and after that then decreases, and when alpha is small the MSE doesn’t change much (but as alpha gets bigger the MSE gets bigger clearly in Figure dii.5), when alpha is too small there is the chance of being over-fitting, so just choose the simplest model: alpha=0.1.
Then train the new model and plot it:

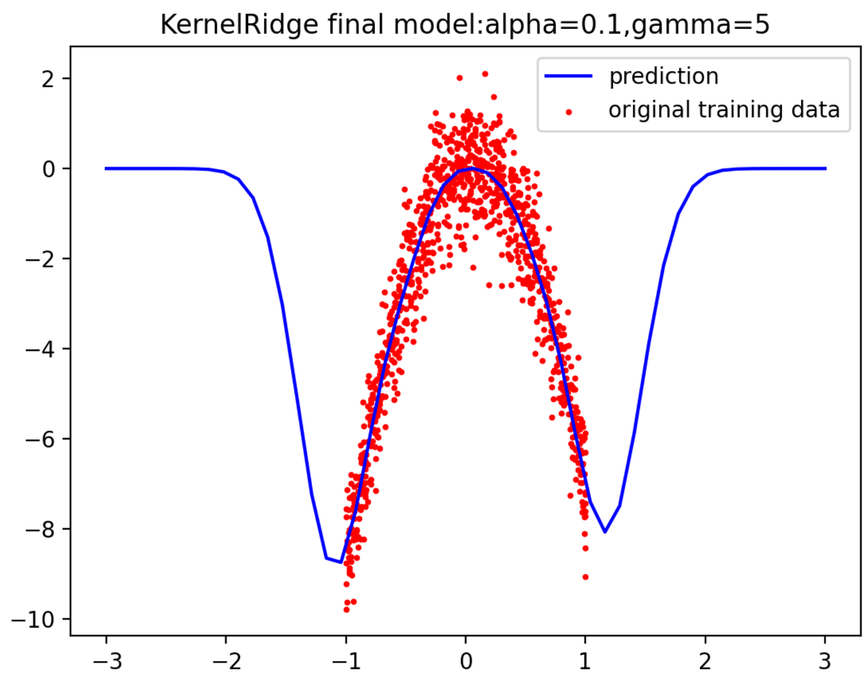


Figure dii.7 KernelRidge final model when alpha=0.1,gamma=5

To see more clearly I plot the 2 predictions together:

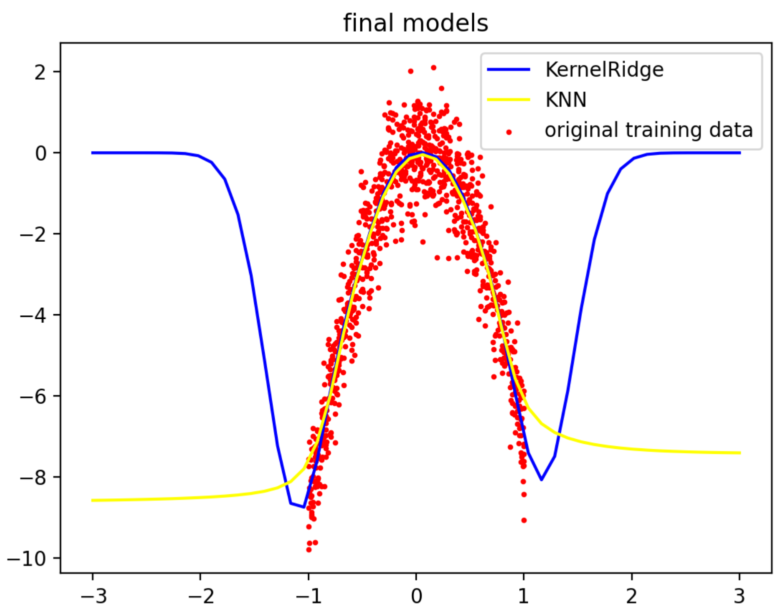


Figure dii.8 final predictions

The 2 predictions all fit the training data well and in range(-1,1) the two models overlap which means they predict the same, however the KernelRidge model has more curves, and the data is shaped like a convex surface and the prediction is much steeper than the training data near the boundary, which means KernelRidge model is over-fitting, suits training data well but is unable to generalize. The KernelRidge model is more sensitive to the parameters than KNN.

Appendix:

1.gaussian_kernel_funcfile.py:

```
import numpy as np

def gaussian_kernel0(distances):
    weights=np.exp(-0*(distances**2))
    return weights/np.sum(weights)
def gaussian_kernel1(distances):
    weights=np.exp(-1*(distances**2))
    return weights/np.sum(weights)
def gaussian_kernel5(distances):
    weights=np.exp(-5*(distances**2))
    return weights/np.sum(weights)
def gaussian_kernel10(distances):
    weights=np.exp(-10*(distances**2))
    return weights/np.sum(weights)
def gaussian_kernel25(distances):
    weights=np.exp(-25*(distances**2))
    return weights/np.sum(weights)
def gaussian_kernel50(distances):
    weights=np.exp(-50*(distances**2))
    return weights/np.sum(weights)
def gaussian_kernel100(distances):
    weights=np.exp(-100*(distances**2))
    return weights/np.sum(weights)
def gaussian_kernel200(distances):
    weights=np.exp(-200*(distances**2))
    return weights/np.sum(weights)
```

1.week6.py:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold, cross_val_score
from sklearn.preprocessing import PolynomialFeatures

from gaussian_kernel_funcfile import *
#(i)(a)
x_dummy_training_data=np.array([-1,0,1]).reshape(-1,1)
y_dummy_training_data=np.array([0,1,0]).reshape(-1,1)
fig1=plt.figure()
from sklearn.neighbors import KNeighborsRegressor
# KNeighborsRegressor model :k = 3  γ=0, 1, 5, 10, 25
# generate predictions on a grid of feature values that range from -3 to 3
Xtest =[]
grid=np.linspace(-3,3)
for x in grid:
    Xtest.append(x)
Xtest=np.array(Xtest).reshape(-1,1)
model1 =
KNeighborsRegressor(n_neighbors=3,weights=gaussian_kernel0).fit(x_dummy_training_data,
y_dummy_training_data)
ypred1 = model1.predict(Xtest)
model2 =
KNeighborsRegressor(n_neighbors=3,weights=gaussian_kernel1).fit(x_dummy_training_data,
y_dummy_training_data)
ypred2 = model2.predict(Xtest)
model3 =
KNeighborsRegressor(n_neighbors=3,weights=gaussian_kernel5).fit(x_dummy_training_data,
y_dummy_training_data)
ypred3 = model3.predict(Xtest)
model4 =
KNeighborsRegressor(n_neighbors=3,weights=gaussian_kernel10).fit(x_dummy_training_data,
y_dummy_training_data)
ypred4 = model4.predict(Xtest)
model5 =
KNeighborsRegressor(n_neighbors=3,weights=gaussian_kernel25).fit(x_dummy_training_data,
y_dummy_training_data)
ypred5 = model5.predict(Xtest)
plt.scatter(x_dummy_training_data, y_dummy_training_data,
color='red',label="dummy_training_data")#plot the dummy training data
plt.plot(Xtest,ypred1,color='green',label="k=3,gamma=0")
plt.plot(Xtest, ypred2, color='blue',label="k=3,gamma=1")
plt.plot(Xtest, ypred3, color='orange',label="k=3,gamma=5")
plt.plot(Xtest, ypred4, color='purple',label="k=3,gamma=10")
plt.plot(Xtest, ypred5, color='brown',label="k=3,gamma=25")
plt.xlabel('input x')
plt.ylabel('output y')
plt.title("predictions of training data")
plt.legend()
plt.show()
#(b)
fig2=plt.figure()
Xtest_1 =[]
grid=np.linspace(-1,1)
for x in grid:
    Xtest_1.append(x)
Xtest_1=np.array(Xtest_1).reshape(-1,1)
ypred_1=model5.predict(Xtest_1)
plt.scatter(x_dummy_training_data, y_dummy_training_data,
color='red',label="dummy_training_data")#plot the dummy training data
plt.plot(Xtest_1, ypred_1, color='blue',label="k=3,gamma=25")
plt.xlabel('input x')
plt.ylabel('output y')
plt.legend()
plt.show()
#(c)
from sklearn.kernel_ridge import KernelRidge
c_range=[0.1,1,1000]
gamma_range=[0,1,5,10,25]
color_range=['blue','green','orange','lime','cyan','pink','purple','cornflowerblue']
"""
for inx1,gamma in enumerate(gamma_range):
    fig3 = plt.figure() #for each gamma has one plot with different values of C
    plt.scatter(x_dummy_training_data, y_dummy_training_data, color='red',
              label="dummy_training_data",s=10)  # plot the dummy training data
    for inx2,C in enumerate(c_range):
        model = KernelRidge(alpha=1.0/(2*C), kernel='rbf',
gamma=gamma).fit(x_dummy_training_data, y_dummy_training_data)
        ypred_kernelridge=model.predict(Xtest)
        print("KernelRidge model when
gammma={0},C={1},dual_coef={2}".format(gamma,C,model.dual_coef_))
        plt.plot(Xtest,ypred_kernelridge,color=color_range[inx2],label='C={}'.format(C))
        plt.legend(loc='upper left', fontsize=10)
        plt.title('gamma={}'.format(gamma))
    plt.show()
"""
#(ii)(a)
gamma_range1=[0,1,5,10,25,50,100,200]
df = pd.read_csv("week6.csv")
x = np.array(df.iloc[:, 0]).reshape(-1,1)#read the x value as array
y = np.array(df.iloc[:, 1]).reshape(-1,1)#read the y value
# kNN model with Gaussian kernel weights: k=999 gamma=0,1,5,10,25
weights_range=[gaussian_kernel0,gaussian_kernel1,gaussian_kernel5,gaussian_kernel10,gau
ssian_kernel25,gaussian_kernel50,gaussian_kernel100,gaussian_kernel200]
fig4=plt.figure()
plt.scatter(x, y, color='red', label="original training data",s=3)  # plot the CSV
training data
for inx3,weights in enumerate(weights_range):
    model_knn = KNeighborsRegressor(n_neighbors=999, weights=weights).fit(x, y)
    ypred_knn = model_knn.predict(Xtest)
    plt.plot(Xtest, ypred_knn,
color=color_range[inx3],label='gamma={}'.format(gamma_range1[inx3]))
    plt.xlabel('input x')
    plt.ylabel('output y')
    plt.title('KNN model predictions when k=999 with different gamma')
    plt.legend(fontsize=5)
plt.show()
#(ii)(b)
"""
for inx4,gamma in enumerate(gamma_range1):
    fig5 = plt.figure() #for each gamma has one plot with different values of C
    plt.scatter(x, y, color='red',
                label="original training data",s=10)  # plot the original training data
    for inx4,C in enumerate(c_range):
        model = KernelRidge(alpha=1.0/(2*C), kernel='rbf', gamma=gamma).fit(x, y)
        ypred_kernelridge=model.predict(Xtest)
        print("KernelRidge model when
gammma={0},C={1},dual_coef={2}".format(gamma,C,model.dual_coef_))
        plt.plot(Xtest,ypred_kernelridge,color=color_range[inx4],label='C={}'.format(C))
        plt.legend(loc='upper left', fontsize=10)
        plt.title('gamma={}'.format(gamma))
    plt.show()"""
#(ii)(c)
""" Use cross-validation to choose a reasonable value for hyperparameter γ for the
kNN model. Now use cross-validation to choose γ and α hyperparameter for the
kernalised ridge regression model. Generate predictions for both models using
these "optimised" hyperparameter values. """
fig6=plt.figure()
mean_error_knn=[]
std_error_knn=[]
mean_score_knn=[]
weights_range1=[gaussian_kernel0,gaussian_kernel1,gaussian_kernel5,gaussian_kernel10,ga
ussian_kernel25]
for inx5,weights in enumerate(weights_range):
    kf = KFold(n_splits=5)
    model_knn1 = KNeighborsRegressor(n_neighbors=799, weights=weights)
    mse = []
    for train, test in kf.split(x):
        model_knn1.fit(x[train], y[train])
        y_pre_knn1 = model_knn1.predict(x[test])
        from sklearn.metrics import mean_squared_error
        mse.append(mean_squared_error(y[test],y_pre_knn1))
    mean_error_knn.append(np.array(mse).mean())
    std_error_knn.append(np.array(mse).std())
    mean_score_knn.append(cross_val_score(model_knn1, x, y, cv=5).mean())
plt.subplot(1,2,1)
plt.plot(gamma_range1,mean_score_knn,label='mean scores',color='red')
plt.ylabel('mean score')
plt.xlabel('gamma')
plt.title("score of different gamma values")
plt.legend(loc='upper right', fontsize=10)
plt.subplot(1,2,2)
plt.errorbar(gamma_range1, mean_error_knn, label="MSE of cross_validation",
color='blue', yerr=std_error_knn)
plt.ylabel('Mean square error')
plt.xlabel('gamma')
plt.title("MSE of different gamma values")
plt.legend(loc='upper right', fontsize=10)
plt.show()
fig7=plt.figure()
model_knn_final = KNeighborsRegressor(n_neighbors=999,
weights=gaussian_kernel50).fit(x,y)
ypred_knn_final=model_knn_final.predict(Xtest)
plt.scatter(x, y, color='red', label="original training data",s=3)  # plot the CSV
training data
plt.plot(Xtest,ypred_knn_final,color='blue',label='prediction')
plt.title("KNN final model:K=999,gamma=50")
plt.legend()
#alpha=5 to find gamma for kernalised ridge regression model
mean_error_kr=[]
std_error_kr=[]
mean_score_kr=[]
for inx5,gamma in enumerate(gamma_range):
    kf = KFold(n_splits=5)
    kr_model = KernelRidge(alpha=5, kernel='rbf', gamma=gamma)
    mse = []
    for train, test in kf.split(x):
        kr_model.fit(x[train], y[train])
        y_pre_kf = kr_model.predict(x[test])
        from sklearn.metrics import mean_squared_error
        mse.append(mean_squared_error(y[test],y_pre_kf))
    mean_error_kr.append(np.array(mse).mean())
    std_error_kr.append(np.array(mse).std())
    mean_score_kr.append(cross_val_score(kr_model, x, y, cv=5).mean())
plt.subplot(1,2,1)
plt.plot(gamma_range,mean_score_kr,label='mean scores',color='red')
plt.ylabel('mean score')
plt.xlabel('gamma')
plt.title("score of different gamma values")
plt.legend(loc='upper right', fontsize=10)
plt.subplot(1,2,2)
plt.errorbar(gamma_range, mean_error_kr, label="MSE of cross_validation", color='blue',
yerr=std_error_kr)
plt.ylabel('Mean square error')
plt.xlabel('gamma')
plt.title("MSE of different gamma values")
plt.legend(loc='upper right', fontsize=10)
#gamma=5 to find the best alpha
fig8=plt.figure()
mean_error_kr1=[]
std_error_kr1=[]
mean_score_kr1=[]
alpha_range=[0.01,0.03,0.05,0.07,0.1,0.15,0.2]
for alpha in alpha_range:
    kf = KFold(n_splits=5)
    kr_model1 = KernelRidge(alpha=alpha, kernel='rbf', gamma=5)
    mse = []
    for train, test in kf.split(x):
        kr_model1.fit(x[train], y[train])
        y_pre_kr = kr_model1.predict(x[test])
        from sklearn.metrics import mean_squared_error
        mse.append(mean_squared_error(y[test],y_pre_kr))
    mean_error_kr1.append(np.array(mse).mean())
    std_error_kr1.append(np.array(mse).std())
    mean_score_kr1.append(cross_val_score(kr_model1, x, y, cv=5).mean())
plt.subplot(1,2,1)
plt.plot(alpha_range,mean_score_kr1,label='mean scores',color='red')
plt.ylabel('mean score')
plt.xlabel('alpha')
plt.title("score of different alpha values")
plt.legend(loc='upper right', fontsize=10)
plt.subplot(1,2,2)
plt.errorbar(alpha_range, mean_error_kr1, label="MSE of cross_validation",
color='blue', yerr=std_error_kr1)
plt.ylabel('Mean square error')
plt.xlabel('alpha')
plt.title("MSE of different alpha values")
plt.legend(loc='upper right', fontsize=10)
plt.show()
#plot final KR model gamma=5 alpha=0.1
fig9=plt.figure()
kr_model_final = KernelRidge(alpha=0.1, kernel='rbf', gamma=5).fit(x,y)
kr_pre_final=kr_model_final.predict(Xtest)
plt.scatter(x, y, color='red', label="original training data",s=3)  # plot the CSV
training data
plt.plot(Xtest,kr_pre_final,color='blue',label='KernelRidge')
plt.plot(Xtest,ypred_knn_final,color='yellow',label='KNN')
plt.title("final models")
plt.legend()
plt.show()
```