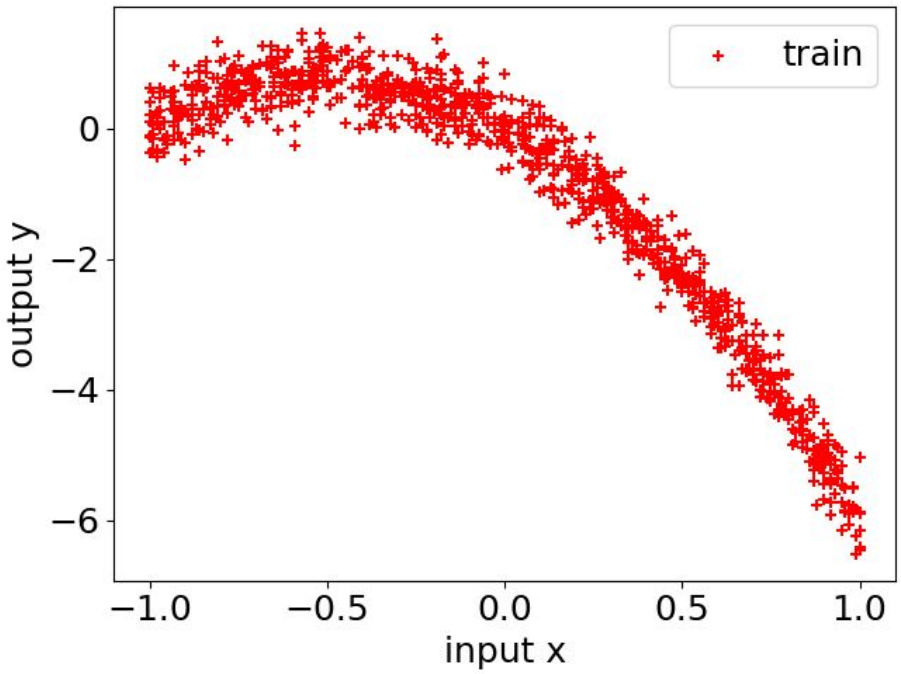
Machine Learning Assignment 6

Saul O'Driscoll (17333932)

Training Data



DATA USED = # id:2--6--6
DATA USED = # id:2--6--6
DATA USED = # id:2--6--6

NOTE:

Code snippets used are all from the original program however they may be trimmed or shortened to allow for better readability.

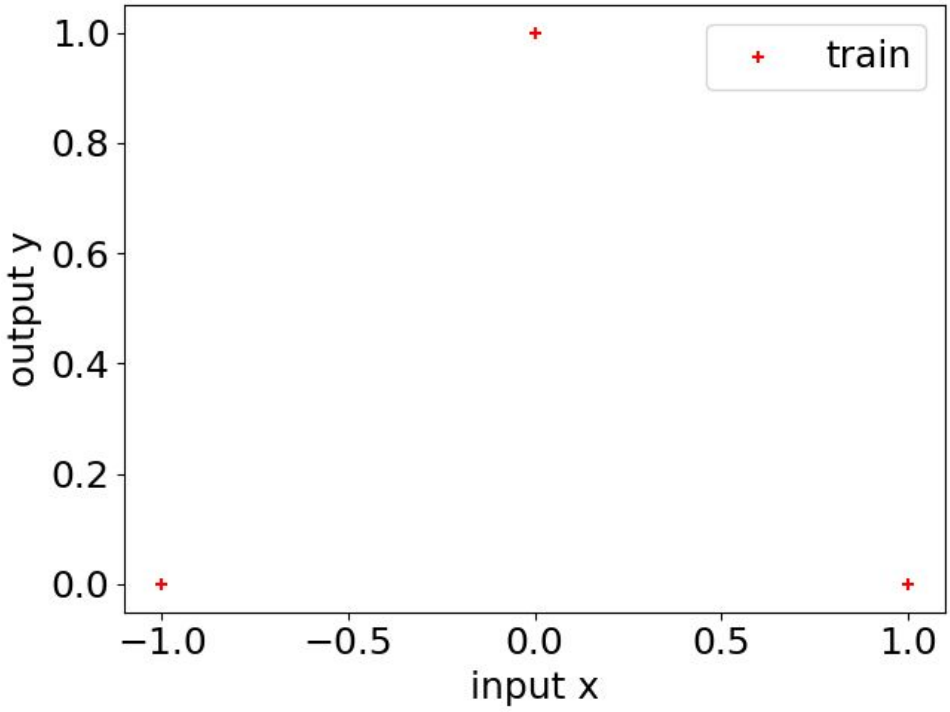
For the sake of uniformity and presentation I have often moved graphs onto the next page. This leads to a lot of blank space but higher readability.

1) The program supports both the use of dummy data as well as real data from a csv file. To change between the preloaded dummy data and the external data change the “dummy” variable to true or false depending on which you would like.

The bool to change is on line 22

- dummy = True (use dummy data)
    - Will not cross validate or evaluate accuracy
- dummy = False (use real data)
    - Will not cross validate or evaluate accuracy

This is what the simple 3 points of the dummy data look like as described in the instructions. Points are highlighted by the red pluses at the top and bottom left and right.



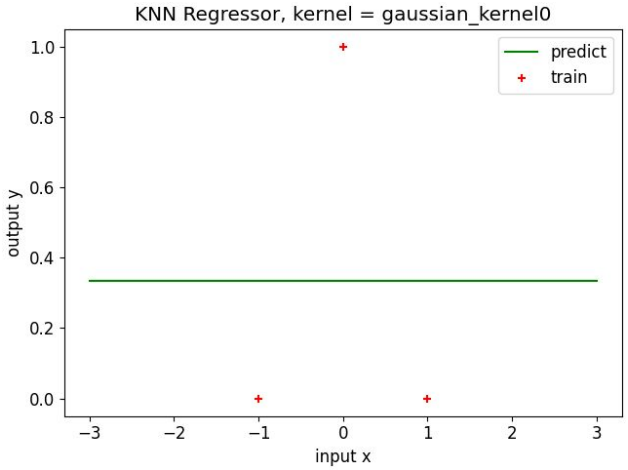
1a) Using the KNN regressor fitted to the dummy data we get the following results.

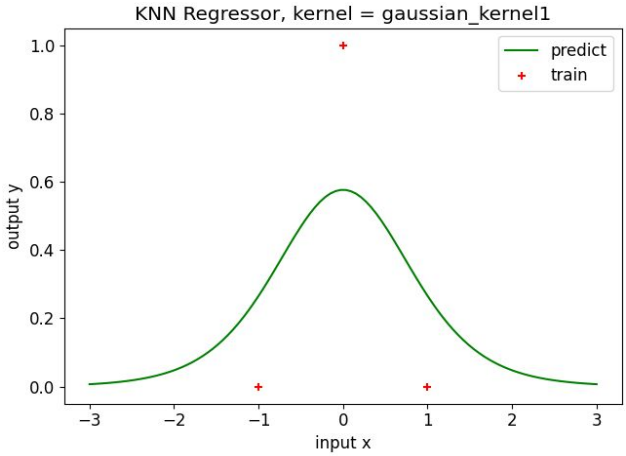
- 5 different gaussian kernels were used defined as seen in this code snippet

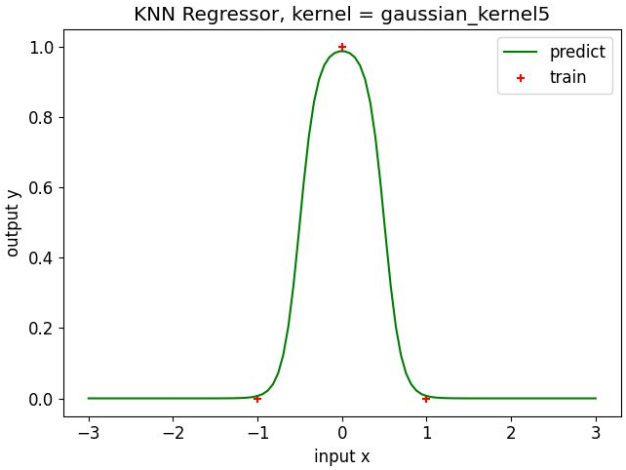
```
def gaussian_kernel0(distances):
    weights = np.exp(0*(distances**2))
    return weights/np.sum(weights)
def gaussian_kernel1(distances):
    weights = np.exp(-1*(distances**2))
    return weights/np.sum(weights)
def gaussian_kernel5(distances):
    weights = np.exp(-5*(distances**2))
    return 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)

kernel_array = [gaussian_kernel0, gaussian_kernel1, gaussian_kernel5,
gaussian_kernel10, gaussian_kernel25]

# KNN Regressor
for g in kernel_array:
    model = KNeighborsRegressor(n_neighbors=data_points,weights=g).fit(Xtrain, ytrain)

# for complete code and not the snippets please see end of file
```

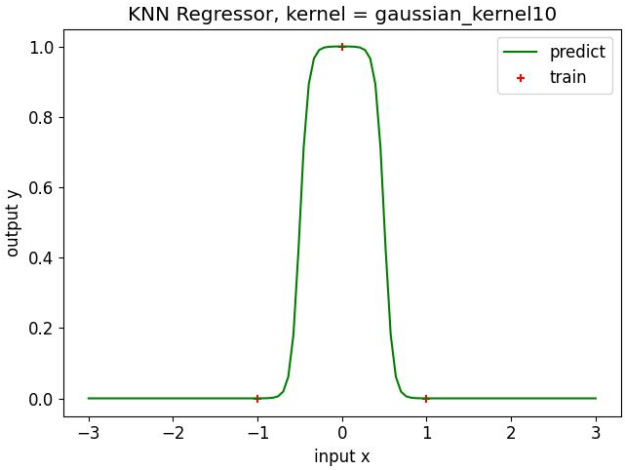
- Each kernel was used separately in the KNN regressor and 5 different curves were plotted.

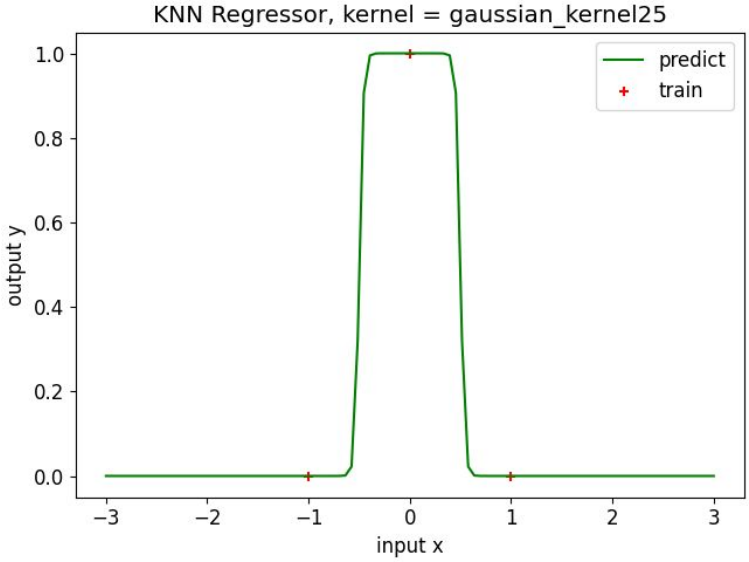
KNN Regressor plots for different kernels









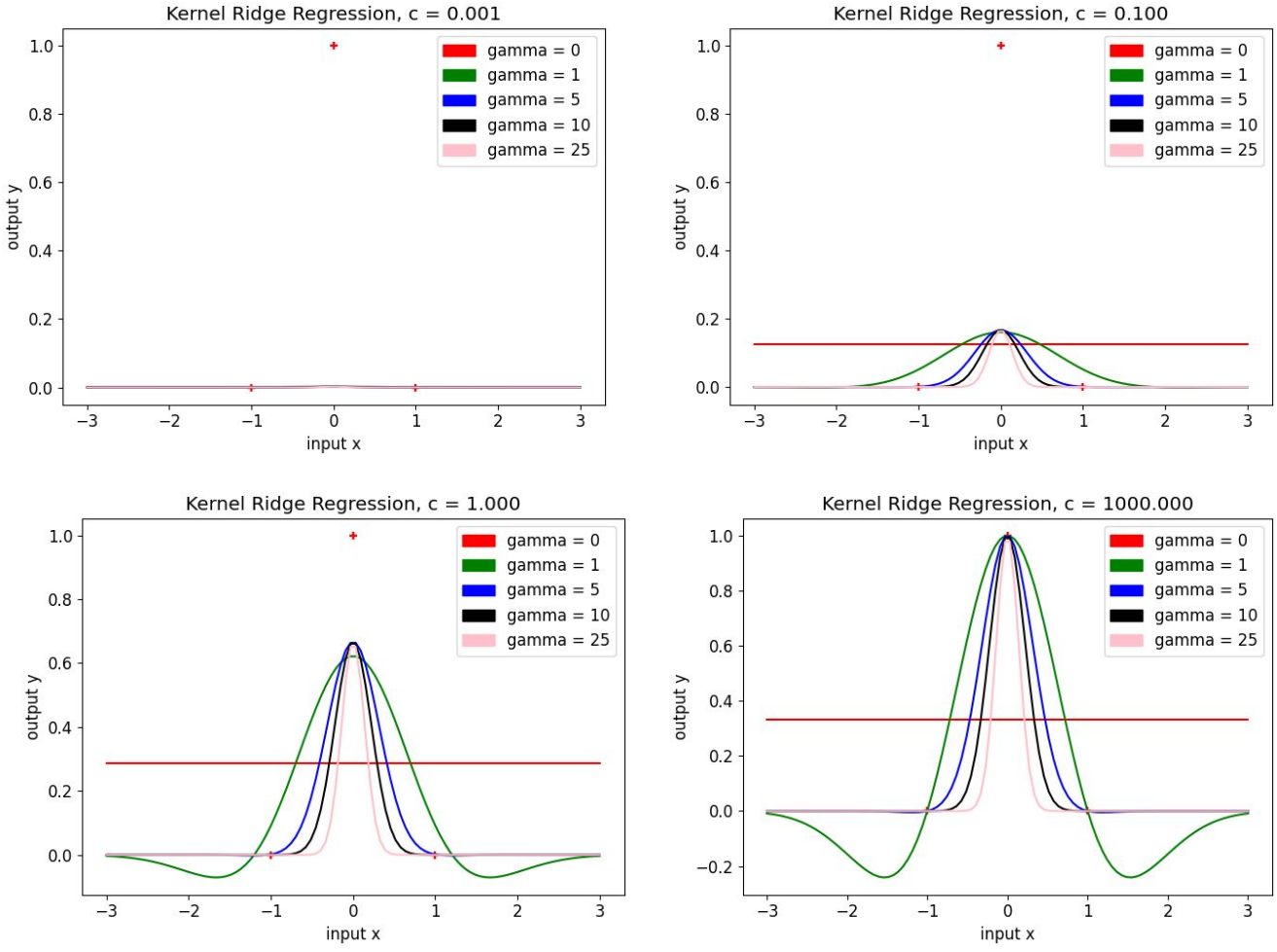


1b) In the above example we used different gaussian kernels as a means of weighting the model. The kernels can be used to smooth a model as one can see when comparing kernel5 and kernel25. Kernel25 is quite blocky and fit more stringently to the data. The cutoff point is right at -0.5 and 0.5, the median between where the dummy data features change. With the smaller kernels this transition is smoother and one can see a gradual change from -1 to +1. This highlights the importance of choosing a kernel fit for your data. If your data does not have many “in between" values then a larger kernel might be fitting resulting in a tightly fit model. A more relaxed / less overfit model may be good in other situations. This can all be evaluated using cross validation as we will see when evaluating accuracy on real data.

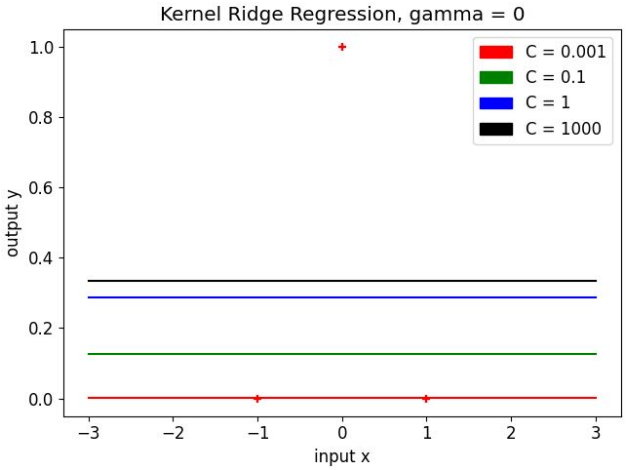
1c) Moving on to ridge regression using an ‘rbf’ kernel we can see some interesting graphs being drawn. Here we set γ (gamma) and the C parameter. We then plot these and see how they change compared to each other. I was not sure whether to plot γ (gamma) in terms of C or C in terms of gamma on the same graph. Below you will find graphs for both.

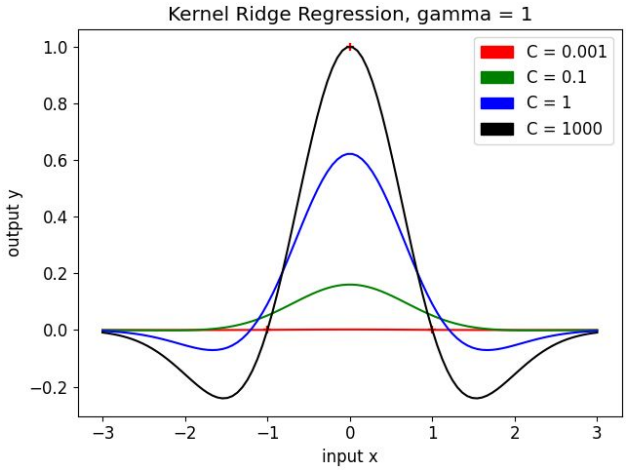
First we begin with a range of gamma values across a single C value. Then a range of C values across a single gamma value.

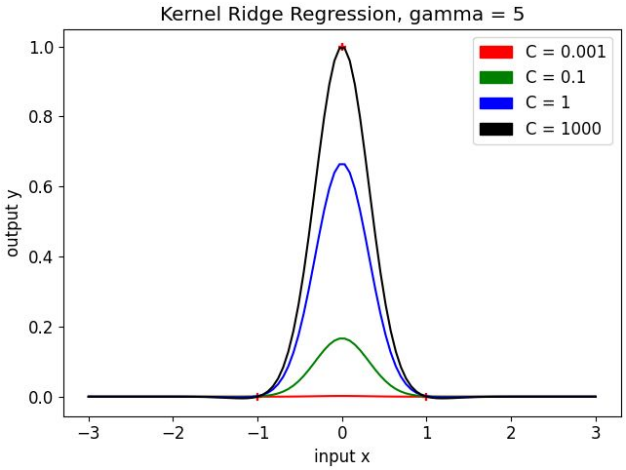
Kernel Ridge Regression (gamma values across a single C value)

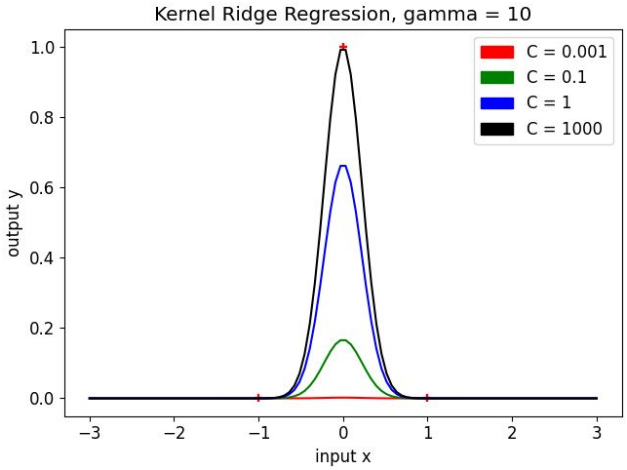


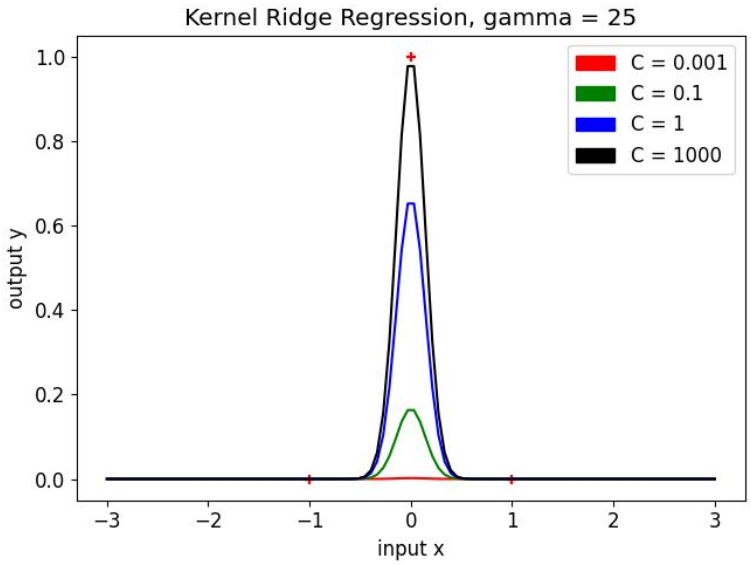
Kernel Ridge Regression (C values across a single gamma value)











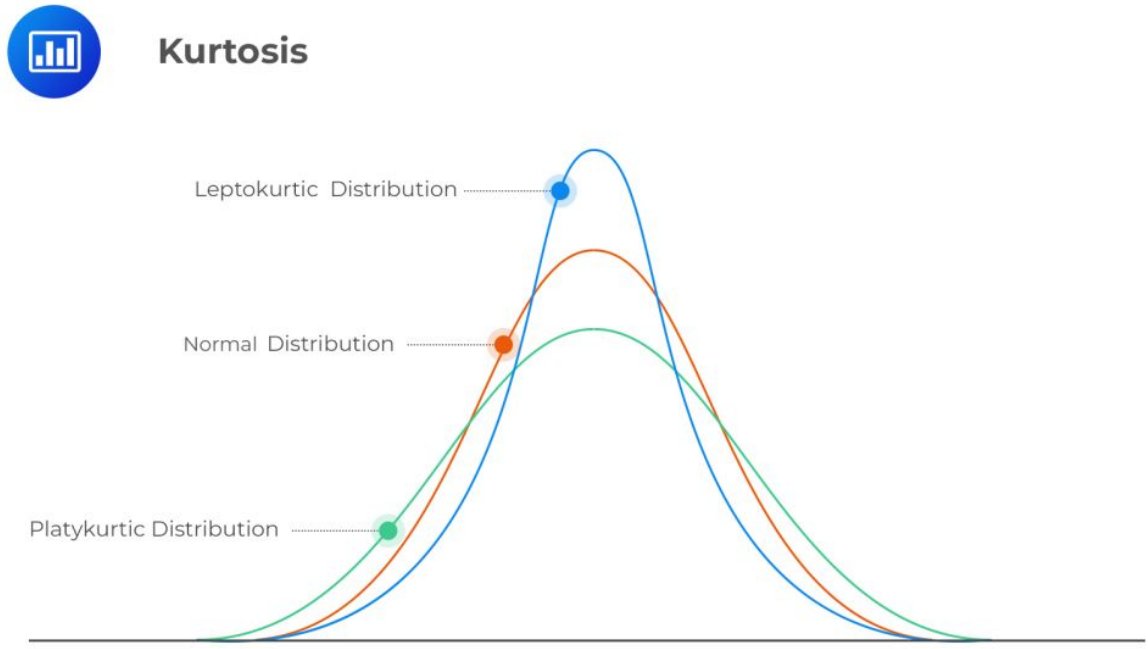
Coefficients of the different models (listed by C)

C = 1	C = 1000
C = 0.001, Gamma = 0 coefficients = [[-3.97614314e-06] [1.99602386e-03] [-3.97614314e-06]]	C = 0.1, Gamma = 0 coefficients = [[-0.025] [0.175] [-0.025]]
C = 0.001, Gamma = 1 coefficients = [[-1.46559731e-06] [1.99601014e-03] [-1.46559731e-06]]	C = 0.1, Gamma = 1 coefficients = [[-0.01026472] [0.16792539] [-0.01026472]]
C = 0.001, Gamma = 5 coefficients = [[-2.68443034e-08] [1.99600798e-03] [-2.68443034e-08]]	C = 0.1, Gamma = 5 coefficients = [[-0.00018717] [0.16666709] [-0.00018717]]
C = 0.001, Gamma = 10 coefficients = [[-1.80875494e-10] [1.99600798e-03] [-1.80875494e-10]]	C = 0.1, Gamma = 10 coefficients = [[-1.26110916e-06] [1.66666667e-01] [-1.26110916e-06]]
C = 0.001, Gamma = 25 coefficients = [[-5.53302332e-17] [1.99600798e-03] [-5.53302332e-17]]	C = 0.1, Gamma = 25 coefficients = [[-3.85776218e-13] [1.66666667e-01] [-3.85776218e-13]]

C = 1	C = 1000
C = 1, Gamma = 0 coefficients = [[-0.57142857] [1.42857143] [-0.57142857]]	C = 1000, Gamma = 0 coefficients = [[-666.55557407] [1333.44442593] [-666.55557407]]
C = 1, Gamma = 1 coefficients = [[-0.18331618] [0.75658434] [-0.18331618]]	C = 1000, Gamma = 1 coefficients = [[-0.49138748] [1.36086227] [-0.49138748]]
C = 1, Gamma = 5 coefficients = [[-0.00299476] [0.66669357] [-0.00299476]]	C = 1000, Gamma = 5 coefficients = [[-0.00673182] [0.99959092] [-0.00673182]]
C = 1, Gamma = 10 coefficients = [[-2.01777466e-05] [6.66666668e-01] [-2.01777466e-05]]	C = 1000, Gamma = 10 coefficients = [[-4.53545640e-05] [9.99500254e-01] [-4.53545640e-05]]
C = 1, Gamma = 25 coefficients = [[-6.17241950e-12] [6.66666667e-01] [-6.17241950e-12]]	C = 1000, Gamma = 25 coefficients = [[-1.38740663e-11] [9.99500250e-01] [-1.38740663e-11]]

1d) Changing gamma and C and its effects:

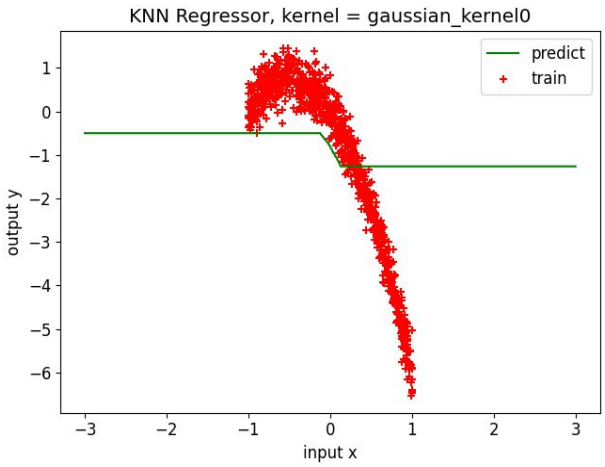
As we change gamma we can see that the kurtosis of the model changes. The models with higher C become more leptokurtic while the model becomes platykurtic for lower C values. We can see the model "reach" for that single data point between the two others. On the other hand we can see that with larger gamma values the model becomes narrower and tighter as it turns into a column rather than something more akin to the shape of a standard distribution. While these are not distributions, borrowing from the terminology, if we were to treat the models as distribution graphs we could say that as gamma increases the standard distributions falls. When looking at the coefficients we can see the slope of the individual models change as they become more/less steep under different parameterization. Lastly, changing the gaussian kernel changes the KNN model much like changing the C model affects the Ridge regression model. This is clear because they are both related to the kernel, bias/variance tolerance and fitting.

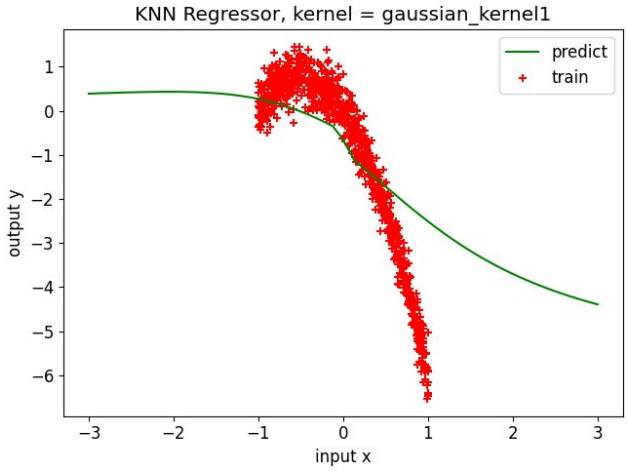


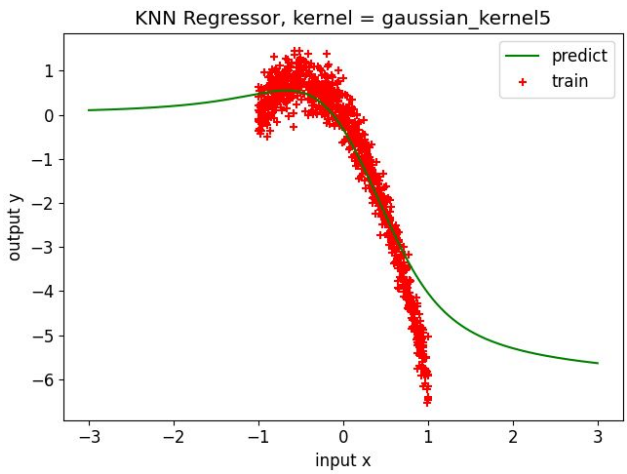
Source: https://analystprep.com/cfa-level-1-exam/quantitative-methods/kurtosis-and-skewness-types-of-distributions/

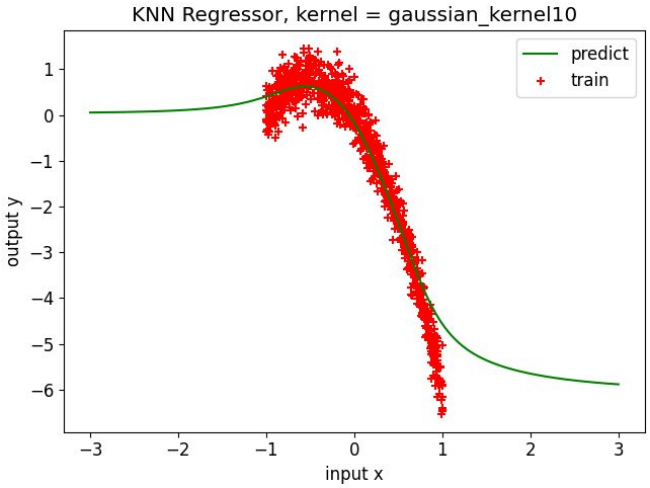
2a)

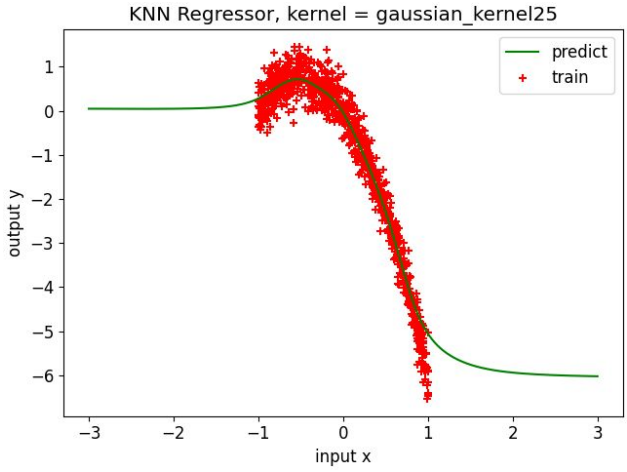
Below we have the models applied to our main data. The same kernels as in part one are used. As we can see the larger kernels begin to fit the data better and better. As we increase the kernel we are able to see the model generalise better even past the range of the input data into X values it has never seen. In the kernel5, 10 and 25 model we see it trail off where a human would also expect the data to do by instinct







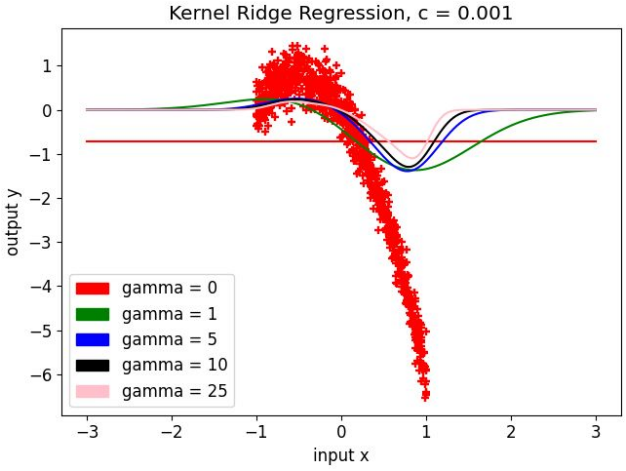


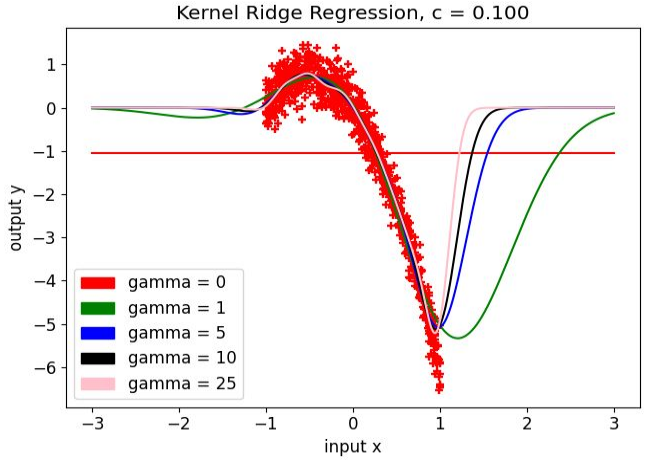


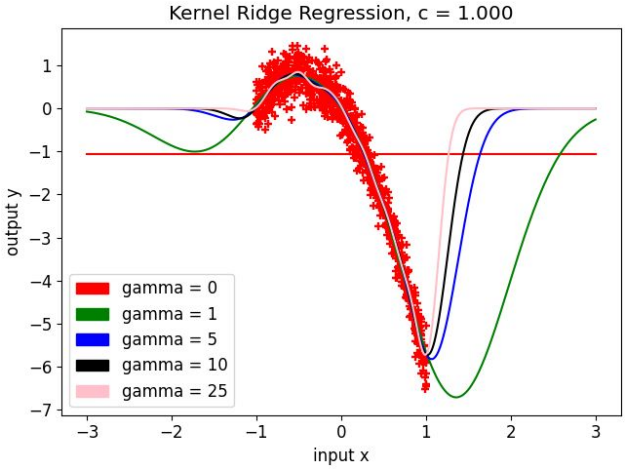
2b)

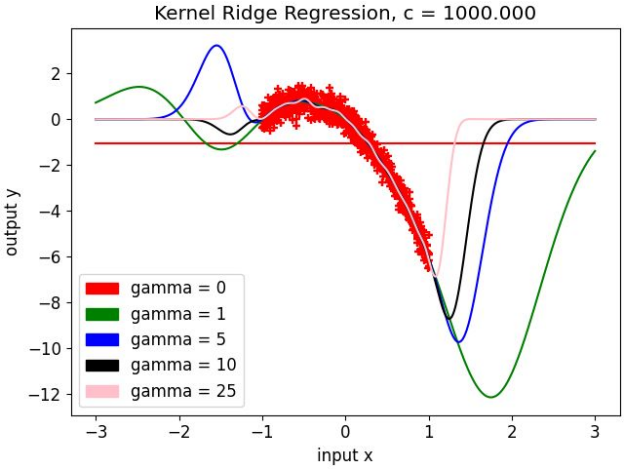
Below are the different kernel ridge regression models across different gamma values which affect the ‘rbf’ kernel, this is done across different Cs in each picture. Again we can see the model fit itself to the data as we increase C. In models with C=1000 we can see the model become a bit more erratic between -2 and -1 which shows how heavily it is influenced by that early shape that the actual data points take. We also see how that later on it seems to be quite biased to the data because all of the models in the C=1000 across all gammas seem to extended quite a bit further than the others before returning to baseline.

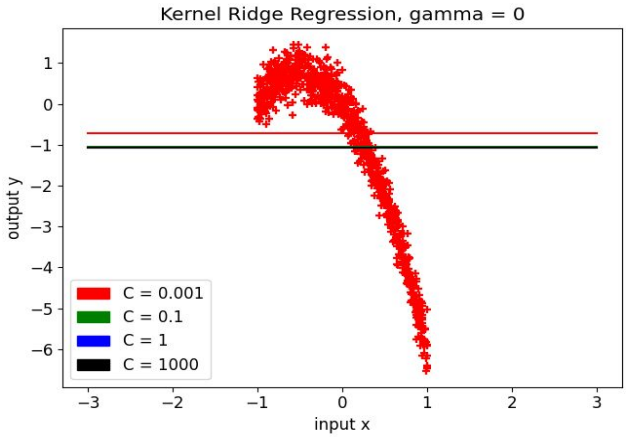
On the next page we see models that are plotted in terms of a changing C over a contiual gamma to highlight the differences there. We can see how tightly fit the models become at later gammas. This could be a sign of overfitting.

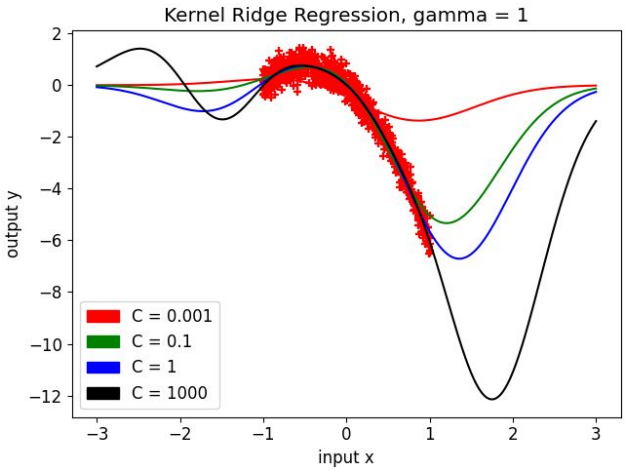


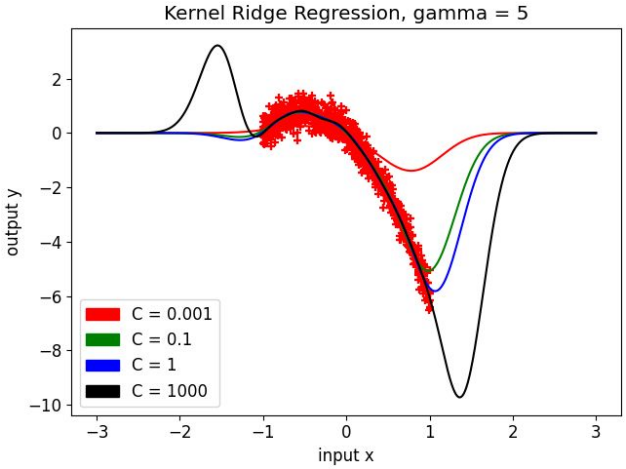


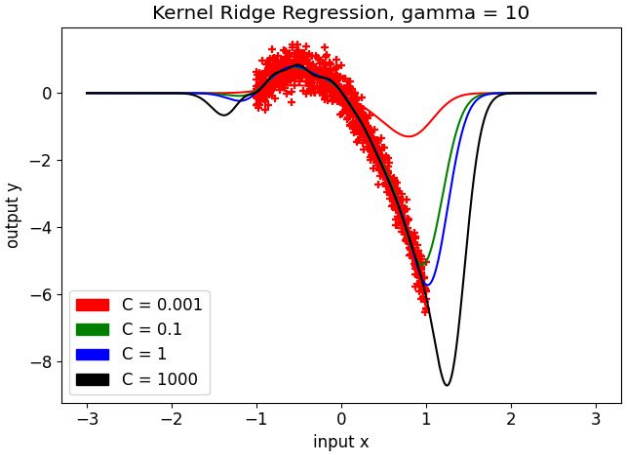


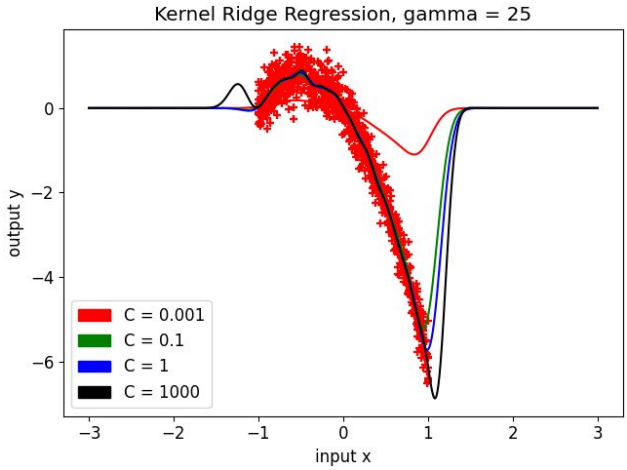



Kernel Ridge Regression, gamma = 0
output y
input x
C = 0.001
C = 0.1
C = 1
C = 1000


Kernel Ridge Regression, gamma = 1
output y
input x
C = 0.001
C = 0.1
C = 1
C = 1000


Kernel Ridge Regression, gamma = 5
output y
input x
C = 0.001
C = 0.1
C = 1
C = 1000


Kernel Ridge Regression, gamma = 10
output y
input x
C = 0.001
C = 0.1
C = 1
C = 1000

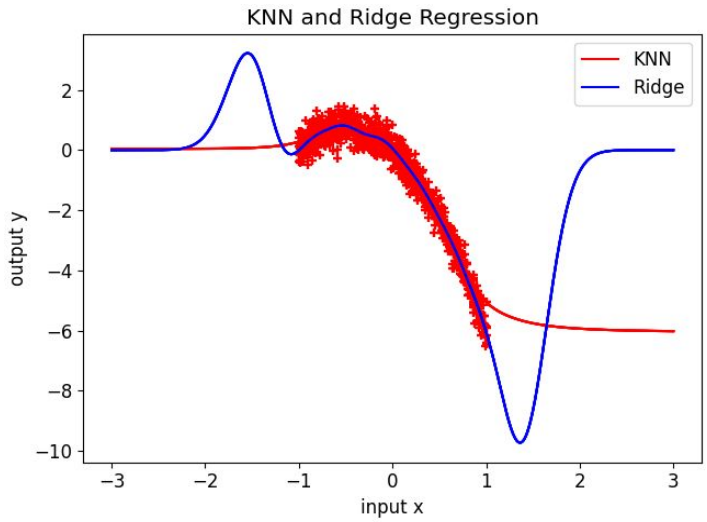

Kernel Ridge Regression, gamma = 25
output y
input x
C = 0.001
C = 0.1
C = 1
C = 1000

2 c) In this final part, cross validation was used to determine parameters that result in a *reasonably* good model.
Highlighted in green is accuracy over 90%. Highlighted in blue are the best models.
Here is the output of the code, it shows all the different accuracies for different parameters.

Kernel: gaussian_kernel0-> Accuracy: -0.01 (+/- 0.02)
Kernel: gaussian_kernel1-> Accuracy: 0.59 (+/- 0.03)
Kernel: gaussian_kernel5-> Accuracy: 0.90 (+/- 0.02)
Kernel: gaussian_kernel10-> Accuracy: 0.95 (+/- 0.02)
Kernel: gaussian_kernel25-> Accuracy: 0.97 (+/- 0.01)

Gamma: 0 C: 0.001-> Accuracy: -0.047 (+/- 0.079)
Gamma: 0 C: 0.1-> Accuracy: -0.013 (+/- 0.021)
Gamma: 0 C: 1-> Accuracy: -0.013 (+/- 0.021)
Gamma: 0 C: 1000-> Accuracy: -0.013 (+/- 0.022)
Gamma: 1 C: 0.001-> Accuracy: 0.337 (+/- 0.096)
Gamma: 1 C: 0.1-> Accuracy: 0.964 (+/- 0.011)
Gamma: 1 C: 1-> Accuracy: 0.974 (+/- 0.010)
Gamma: 1 C: 1000-> Accuracy: 0.976 (+/- 0.009)
Gamma: 5 C: 0.001-> Accuracy: 0.329 (+/- 0.109)
Gamma: 5 C: 0.1-> Accuracy: 0.970 (+/- 0.011)
Gamma: 5 C: 1-> Accuracy: 0.975 (+/- 0.009)
Gamma: 5 C: 1000-> Accuracy: 0.976 (+/- 0.009)
Gamma: 10 C: 0.001-> Accuracy: 0.270 (+/- 0.115)
Gamma: 10 C: 0.1-> Accuracy: 0.970 (+/- 0.011)
Gamma: 10 C: 1-> Accuracy: 0.975 (+/- 0.009)
Gamma: 10 C: 1000-> Accuracy: 0.976 (+/- 0.009)
Gamma: 25 C: 0.001-> Accuracy: 0.167 (+/- 0.126)
Gamma: 25 C: 0.1-> Accuracy: 0.970 (+/- 0.011)
Gamma: 25 C: 1-> Accuracy: 0.975 (+/- 0.009)
Gamma: 25 C: 1000-> Accuracy: 0.975 (+/- 0.009)

These are the best 2 models from both techniques in the same plot.
Accuracy is very similar however ridge regression seems to be able to track the last couple of data points a bit better than KNN. Ridge regression does not revert to the baseline like the KNN classifier does



CODE

```
# id:2--6--6
# Saul O'Driscoll 17333932

import random
import pandas as pd
import numpy as np
import matplotlib.patches as mpatches
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsRegressor
from sklearn.kernel_ridge import KernelRidge
from sklearn.model_selection import cross_val_score

# Reading in data from the file
df = pd.read_csv("data.csv",header=None, comment="#", names=["X1","X2"])
dataset_1_X_all = df["X1"]
dataset_1_X_labels = df["X2"]

#Opening log file
file1 = open("coef_data.txt","w+")

# change this line to switch between dummy data and the real data
dummy = False

if dummy:
   data_points = 3
   test_plane = 100
   Xtrain = [-1, 0, 1]
   ytrain = [0, 1, 0]
   Xtrain = np.array(Xtrain)
   ytrain = np.array(ytrain)
   Xtrain = Xtrain.reshape(-1, 1)
   ytrain = ytrain.reshape(-1, 1)
else:
   data_points = 874
   test_plane = 999
   Xtrain = np.array(dataset_1_X_all)
   ytrain = np.array(dataset_1_X_labels)
```

```
    Xtrain = Xtrain.reshape(-1, 1)
    ytrain = ytrain.reshape(-1, 1)

# creating test space
Xtest = np.linspace(-3,3,num=test_plane).reshape(-1, 1)

# Defining different gaussian kernels
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)

kernel_array = [gaussian_kernel0, gaussian_kernel1, gaussian_kernel5,
gaussian_kernel10, gaussian_kernel25]

# KNN Regressor
knnAccuracy_mean_array = []
knnAccuracy_std_array = []
for g in kernel_array:
    model =
KNeighborsRegressor(n_neighbors=data_points,weights=g).fit(Xtrain, ytrain)
    if(not dummy):
        scores = cross_val_score(model, Xtrain, ytrain, cv=8)
        knnAccuracy_mean_array.append(scores.mean())
        knnAccuracy_std_array.append(scores.std())
        file1.write("Kernel: " + str(g.__name__) + " ==> Accuracy: %0.2f
(+/- %0.2f) \n" % (scores.mean(), scores.std() * 2))

    ypred = model.predict(Xtest)
```

```
   plt.rc('font', size=12); plt.rcParams['figure.constrained_layout.use']
= True
   plt.scatter(Xtrain, ytrain, color='red', marker='+')
   plt.plot(Xtest, ypred, color='green')
   plt.xlabel("input x"); plt.ylabel("output y")
   plt.title("KNN Regressor, kernel = {0}".format(g.__name__))
   plt.legend(["predict","train"])

   plt.show()
   plt.clf()

if(not dummy):
   print("KNN MEAN ARRAY")
   print(knnAccuracy_mean_array)

   print("KNN STD ARRAY")
   print(knnAccuracy_std_array)


#Parameters for kernel Ridge Regression
c_array = [0.001, 0.1, 1, 1000]
gamma_array = [0, 1, 5, 10, 25]
colour_array = ['red', 'green', 'blue', 'black', 'pink']

# Writing to log file
file1.write("Kernel Ridge Regression, C iteration \n")

# kernel Ridge Regression with Cs as the outer loop

for c in c_array:
   for g, colour in zip(gamma_array, colour_array):
       # Train and predict
       clf = KernelRidge(kernel = 'rbf', alpha=1/(2*c), gamma=g)
       clf.fit(Xtrain, ytrain)
       ypred = clf.predict(Xtest)

       #Writing to log file
       # file1.write("C = {0}, Gamma = {1} \n".format(c, g))
       # file1.write("coefficients = {0} \n\n".format(clf.dual_coef_))
```

```
        #Graph printing
        plt.rc('font', size=12);
plt.rcParams['figure.constrained_layout.use'] = True
        plt.scatter(Xtrain, ytrain, color='red', marker='+')
        plt.plot(Xtest, ypred, color=colour)
        plt.xlabel("input x"); plt.ylabel("output y");
plt.legend(["predict","train"])

        #Legend / Label loop
        patches = []
        for g, colour in zip(gamma_array, colour_array):
            patches.append(mpatches.Patch(color=colour, label='gamma =
{0}'.format(g)))
        plt.title("Kernel Ridge Regression, c = {:.3f}".format(c))
        plt.legend(handles=patches)
    plt.show()
    plt.clf()

ridgeAccuracy_mean_array = []
ridgeAccuracy_std_array = []
# kernel Ridge Regression with Gammas as the outer loop
for g in gamma_array:
    for c, colour in zip(c_array, colour_array):
        # Train and predict
        clf = KernelRidge(kernel = 'rbf', alpha=1/(2*c), gamma=g)
        clf.fit(Xtrain, ytrain)

        if(not dummy):
            scores = cross_val_score(clf, Xtrain, ytrain, cv=8)
            ridgeAccuracy_mean_array.append(scores.mean())
            ridgeAccuracy_std_array.append(scores.std())
            file1.write("Gamma: " + str(g) + " C: " + str(c) + "->
Accuracy: %0.3f (+/- %0.3f) \n" % (scores.mean(), scores.std() * 2))

        ypred = clf.predict(Xtest)
        #Graph printing
        plt.rc('font', size=12);
plt.rcParams['figure.constrained_layout.use'] = True
        plt.scatter(Xtrain, ytrain, color='red', marker='+')
        plt.plot(Xtest, ypred, color=colour)
```

```
       plt.xlabel("input x"); plt.ylabel("output y");
plt.legend(["predict","train"])

       #Legend / Label loop
       patches = []
       for c, colour in zip(c_array, colour_array):
           patches.append(mpatches.Patch(color=colour, label='C =
{0}'.format(c)))
       plt.title("Kernel Ridge Regression, gamma = {0}".format(g))
       plt.legend(handles=patches)
   plt.show()
   plt.clf()

print("Ridge MEAN ARRAY")
print(ridgeAccuracy_mean_array)

print("Ridge STD ARRAY")
print(ridgeAccuracy_std_array)
```