Assignment 6

Student number: 20324412

Module: CS7CS4/CSU44061

Part i

Part a. Kernelised knn

The training data was as follows; (−1, 0), (0, 1), (1, 0). The data was fit with kernelised k-nearest neighbour regression models. These were implemented for varying the value of the parameter γ in the Gaussian kernel in the range [0, 1, 5, 10, 25]. The Gaussian kernel is of the following form;

$$K\left(x^{(i)}, x\right) = e^{-\gamma d\left(x^{(i)},\ x\right)^2}$$

The resultant plot of the model predictions for varying gamma along with the training data is shown in Figure 1 which was implemented using the function *plot_kernelised_knn_regression*.

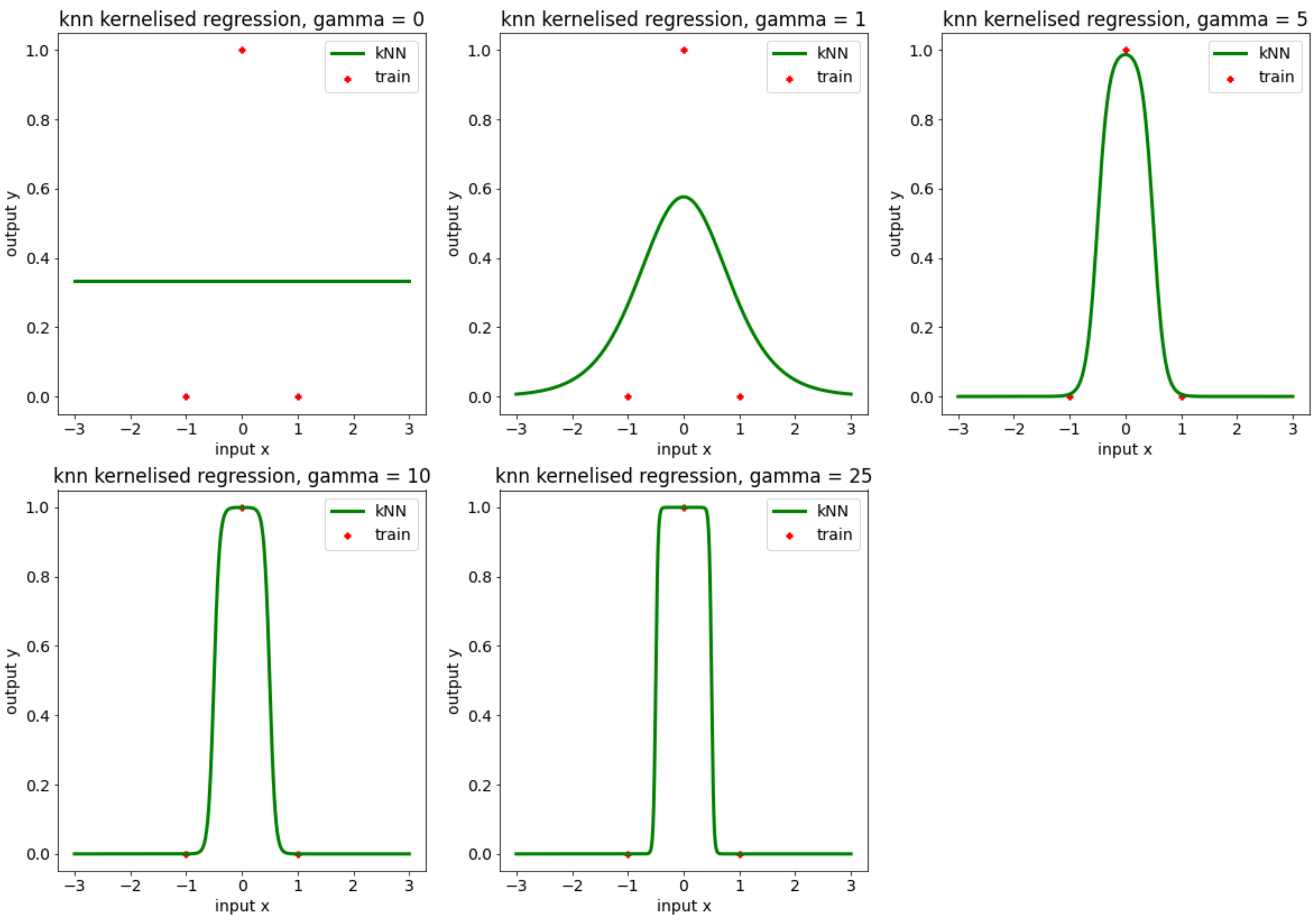


Figure 1. Kernelised knn model for varying gamma [0, 1, 5, 10, 25].

Function plot_kernelised_knn_regression

```
def plot_kernelised_knn_regression(Xtrain, ytrain, k, range_preds,
range_gamma, plot_dims):

    'Plot knn predictions for varying parameter gamma in kernelised knn'

    #Setup

    fig = plt.figure(figsize=(20, 20))

    count = 0

    #Generate test data

    Xtest=np.linspace(range_preds[0], range_preds[1], num=1000).reshape(-1,
1)

    for gammaX in range_gamma:

        #Set up gaussian kernel function

        def gaussian_kernel(distances):

            weights = np.exp(-gammaX*(distances**2))

            return weights/np.sum(weights)

        model = KNeighborsRegressor(n_neighbors= k,
weights=gaussian_kernel).fit(Xtrain, ytrain)

        #Predictions

        ypred = model.predict(Xtest)

        #Plot

        count +=1

        plt.subplot(plot_dims[0], plot_dims[1], count)

        plt.scatter(Xtrain, ytrain, color= 'red', marker='+', linewidth =
3)

        plt.plot(Xtest, ypred, 'g-',linewidth = 3)

        plt.xlabel('input x'); plt.ylabel('output y')

        plt.legend(['kNN','train'])

        plt.title('knn kernelised regression, gamma = {}'.format(gammaX))

        plt.show()
```

Part b. knn kernelised predictions for varying gamma

The knn model generates it's predictions by taking the average of the k nearest neighbours to a given training data point. For regression, the weighted mean for aggregating the k neighbour points N_k is as follows;

$$\hat{y} = \frac{\sum_{i=1}^{m} K(x,x_i) y_i}{\sum_{i=1}^{m} K(x,x_i)} \qquad \text{(Equation 1)}$$

The weighting function used, a Gaussian kernel, decays with distance, i.e;

$$K\left(x^{(i)}, x\right) = e^{-\gamma d(x^{(i)},\ x)^2}$$

As the weight parameter is increased, more significance is placed on the neighbours nearest the data point, i.e those were the distance is at a minimum. This is seen as gamma is increased from 0 through to 25. When $\gamma = 0$, the kernel is equal to one and so the output is simply the average of the training points, i.e $\frac{1}{3}$ in this case, as shown in the top left figure. For $\gamma = 1,\ \gamma = 5$ the distribution of the predictions is Gaussian.

$\gamma = 25$

For $\gamma = 25$, the weighting function decays exponentially as the distance of the training points is increased. For datapoints close to zero, i.e within the range [-0.5, 0.5], the distance to the training point x_2 $(x = 0, y = 1)$ is less than that of the distance to x_1 $(x = -1, y = 0)$ or x_3 $(x = 1, y = 0)$. Thus in this range the weighting function corresponding to x_2 has significant weight, while the weights corresponding to x_2, x_3 go to zero. As a result the predictions in this range are fit to the training point x_2, i.e y = 1.

Part c. Kernelised Ridge Regression

The data was then fit with kernelised ridge regression models. These were implemented for varying the value of the parameter γ in the Gaussian kernel in the range [0, 1, 5, 10, 25] and for the penalty term C in the range; [0.1, 1, 1000]. The resultant plot of the model predictions for varying gamma along with the training data is shown in Figure 2 which was implemented using the function *plot_kernel_Ridge.*

***Figure 2a. Kernalised Ridge Regression*, C = 0.1, $\gamma = [0, 1, 5, 10, 25]$**

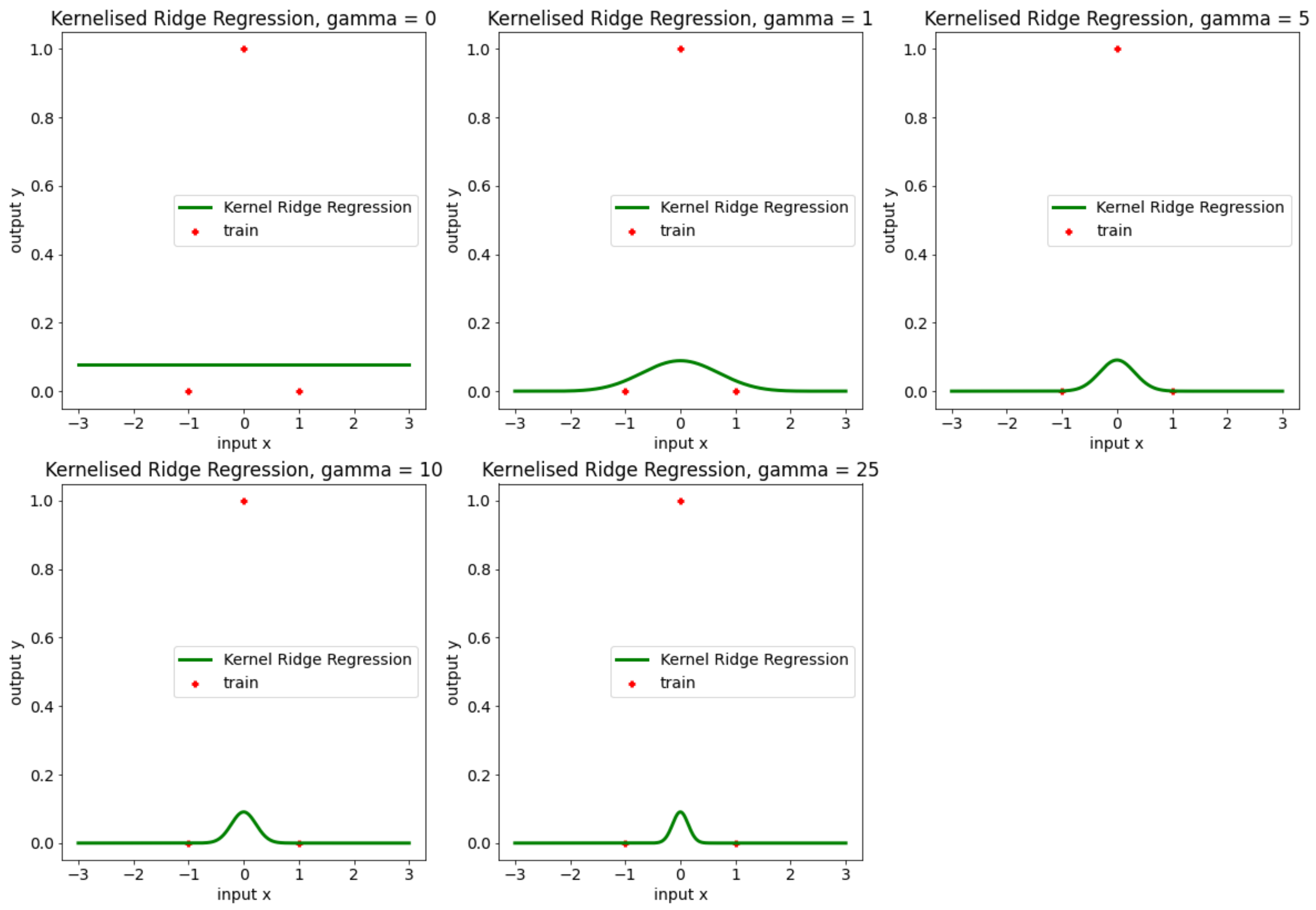


Figure 2b. Kernelised Ridge Regression, C = 1, $\gamma = [0, 1, 5, 10, 25]$

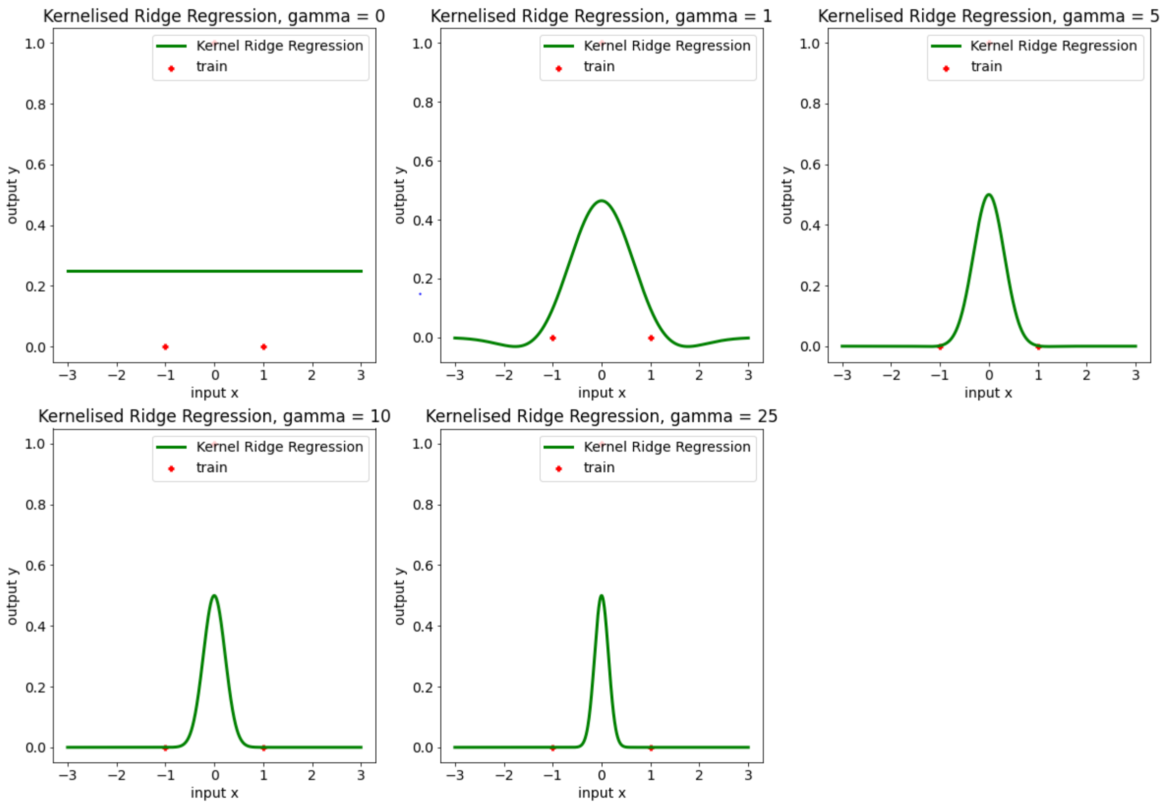
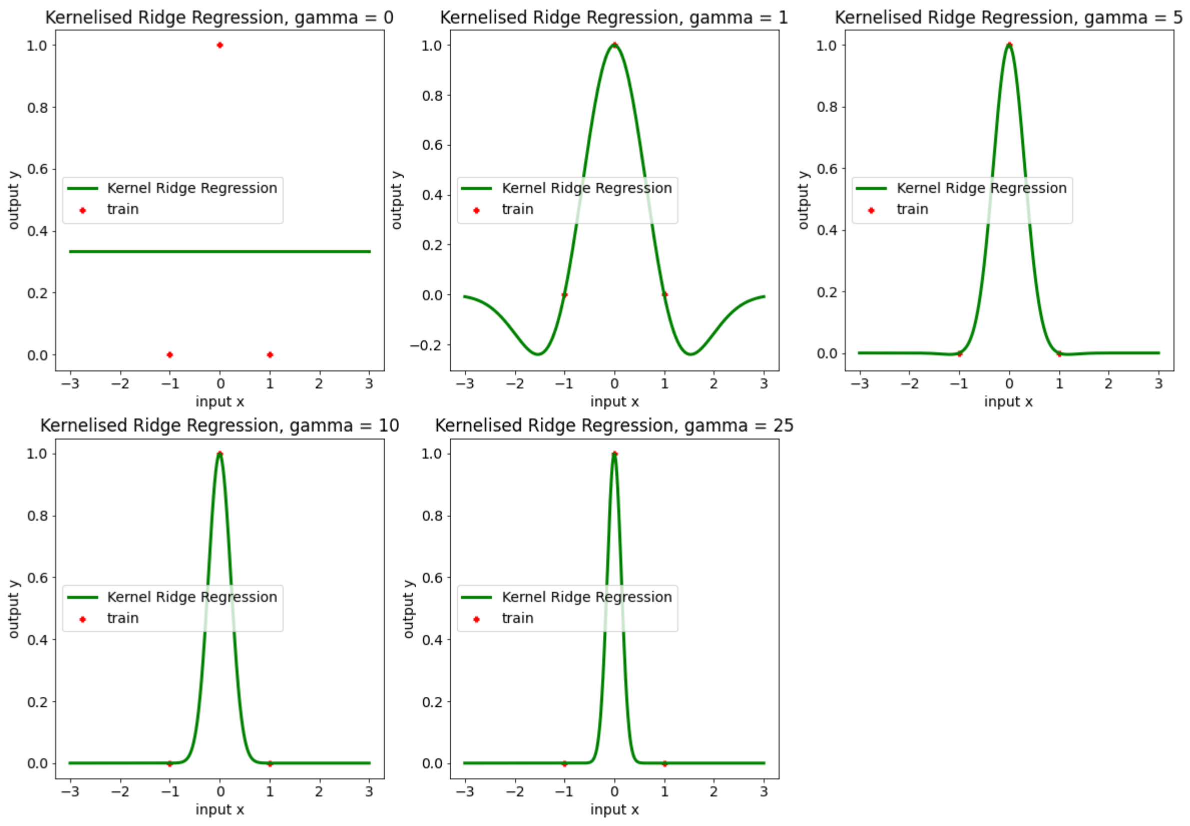


Figure 2c. Kernelised Ridge Regression, C = 1000, $\gamma = [0, 1, 5, 10, 25]$



Function plot_kernel_Ridge

```
def plot_kernel_Ridge(Xtrain, ytrain, C, range_preds, range_gamma, 
plot_dims):

    'Plot kernel ridge regression predictions for varying the parameter 
gamma in the kernel and return coefficients in a dataframe'

    #Setup

    fig = plt.figure(figsize=(20, 20))

    results = []

    count = 0

    #Generate test data

    Xtest=np.linspace(range_preds[0], range_preds[1], num=1000).reshape(-1, 
1)

    #Test range of gamma values

    for gammaX in range_gamma:

        count += 1

        #Model

        model = KernelRidge(alpha=1.0/C, kernel= 'rbf', 
gamma=gammaX).fit(Xtrain, ytrain)

        #Predictions

        ypred = model.predict(Xtest)

        #Coefficients

        d = {

        'gamma' : gammaX,

        'coefficients' :  np.around(model.dual_coef_, decimals = 3),

        }

        results.append(d)

        #Plot

        plt.subplot(plot_dims[0], plot_dims[1], count)



        plt.scatter(Xtrain, ytrain, color= 'red', marker='+', linewidth = 
3)

        plt.plot(Xtest, ypred, 'g-',linewidth = 3)
        plt.xlabel('input x'); plt.ylabel('output y')

        plt.legend(['Kernel Ridge Regression','train'], loc = 'center
left')

        plt.title('Kernelised Ridge Regression, gamma = {}'.format(gammaX))

        plt.show()


        #Return results

        df_coeffs = pd.DataFrame(results)

    return df_coeffs
```

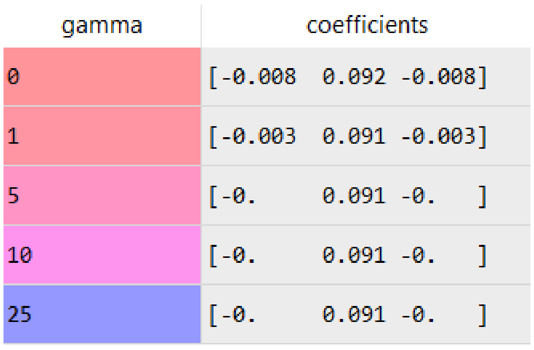
#Apply Function

```
range_gamma = [0, 1, 5, 10, 25]

C = 0.1

legend_loc = 'center right'

df_coeffs = plot_kernel_ridge(Xtrain, ytrain, C, range_preds, range_gamma,
plot_dims, legend_loc)
```

The resultant model coefficients, which the function returns, are shown in Table 1a-1c. The model coefficients correspond to $\alpha_1, \theta_2, \theta_3$ and are a representation of the weight vectors in kernel space as discussed in part d.

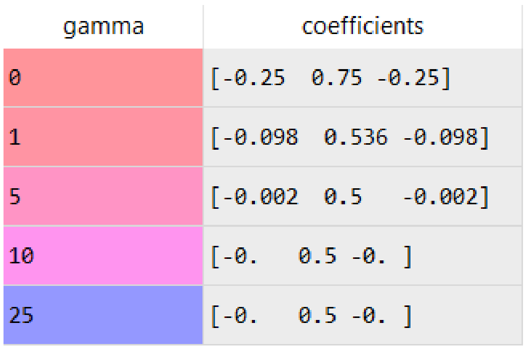
Table 1. ***Kernelised Ridge Regression*** **- Model Coefficients** $(\alpha_1, \alpha_2, \alpha_3)$

Table 1a. C = 0.01.



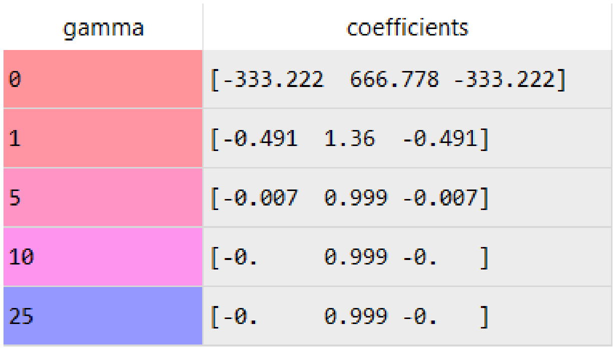
gamma	coefficients
0	[-0.008 0.092 -0.008]
1	[-0.003 0.091 -0.003]
5	[-0. 0.091 -0.]
10	[-0. 0.091 -0.]
25	[-0. 0.091 -0.]

Table 1b. C = 1.



gamma	coefficients
0	[-0.25 0.75 -0.25]
1	[-0.098 0.536 -0.098]
5	[-0.002 0.5 -0.002]
10	[-0. 0.5 -0.]
25	[-0. 0.5 -0.]

Table 1c. C = 1000



gamma	coefficients
0	[-333.222 666.778 -333.222]
1	[-0.491 1.36 -0.491]
5	[-0.007 0.999 -0.007]
10	[-0. 0.999 -0.]
25	[-0. 0.999 -0.]

Part d. Discuss - Kernelised Ridge Regression Results

In Kernelised Ridge Regression, the model predictions are of the form;

$$\hat{y} = \sum_{j=1}^{m} \alpha_j y^{(i)} K(x, x^{(j)}) \qquad \text{(Equation 2)}$$

The coefficients are a representation of the weight vectors in kernel space. In this instance with merely three training points, the model predictions are as follows;

$$\hat{y} = \alpha_0 + \alpha_1 * K(x_1, x) * y_1 + \alpha_2 * K(x_2, x) * y_2 + \alpha_3 * K(x_3, x) * y_3$$

Given that y_1, y_3 are equal to zero, this becomes;

$$\hat{y} = \alpha_0 + \alpha_2 * K(x_2, x) * y_2$$

Therefore the predictions vary depending on the value of α_2.

As C is increased, the amplitude of the coefficient α_2 is increased, which is reflected both in the plots and the coefficients in the tables. For example, for C = 0.01, $\gamma = 10$, the coefficient $\alpha_2 = 0.091$, whereas for C = 1000, and again $\gamma = 10$, the coefficient $\alpha_2 = 0.999$; the coefficient has dramatically increased. This is because in ridge regression, the cost function includes an L_2 penalty. The hyper parameter C allows the influence of the penalty term to be controlled. The smaller the value of C, the greater the impact the penalty has within the cost function which results in the model parameters shrinking towards zero. This is seen for all values C = 0.1 as in Figure 2a and Table 1a. The model coefficients are very small, almost negligible. For C = 1, the model coefficients have started to increase. For C = 1000, significantly less importance is placed on the penalty term and the coefficients of the model are considerably larger, as seen in Figure 2c and Table 1c.

The Kernel used in this instance is a Gaussian Kernel of the form;

$$K\left(x^{(i)}, x\right) = e^{-\gamma d\left(x^{(i)},\ x\right)^2}$$

Thus the predictions are Gaussian shaped. This Kernel or weighting function decays with distance. Therefore as the weight parameter is increased, more significance is placed on the training points nearest the data point, i.e those were the distance is at a minimum. This is seen as γ is increased from 0 through to 25. For smaller values of γ the weight is spread across all three training points, as seen in the table results. For example, in Table 1b, C = 1, $\gamma = 1$, the model coefficients are $(\alpha_1 = -0.098, \alpha_2 = 0.536, \alpha_3 = -0.098)$, whereas for C = 1, $\gamma = 25$, the model coefficients are $(\alpha_1 = 0, \alpha_2 = 0.5, \alpha_3 = 0)$

Comparing the predictions of the kernelised ridge regression to those of the kNN predictions, there are some similarities and some differences. A Gaussian kernel is used in both cases and so the weighting decays with distance. Thus for smaller values of γ the weigh is spread across all three training points whereas for greater values, more significance is placed on the nearest training points.

However in the knn model, the predictions are a weighted average of the nearest training points which is particularly evident for $\gamma = 0$ and $\gamma = 25$ in Figure 1. This is not the case in the Kernelised Ridge Regression, whose predictions are determined by equation 2, and not merely the weighted average of it's nearest neighbours. It's predictions maintain a Gaussian shape across all values of γ.

Furthermore in knn the weights are normalised (see equation 1) but in Ridge Regression they are not (Equation 2). I.e in knn the prediction involves division by $\sum_{i=1}^{m} K(x, x_i)$. Thus for points far away from the training data, where $K\left(x^{(i)}, x\right)$ is close to 0 for all x_i, the denominator $\sum_{i=1}^{m} K(x, x_i)$ is also close to 0 and so compensates. As a result the predictions do not generally shrink to zero. This is not the case in Kernelised ridge regression where there is no weighted division.

Part 2: Dataset

Part a Kernelised knn

The data was fit with kernelised k-nearest neighbour regression models, again using the function *plot_kernelised_knn_regression*. The resultant plot of the model predictions for varying gamma along with the training data is shown in Figure 3. In the knn model, the output is the average of the neighbours response. Since the number of neighbours k was set equal to the number of training points, this equates to the average of all training points. Thus for $\gamma = 0$, where the weighted distance has no effect, the prediction is constant for all input values, including those for features outside the training data. As γ is increased, more significance is placed on the training points nearest the data point, i.e those were the distance is at a minimum. This results in a narrowing of the Gaussian curve, comparable to a reduction in it's standard deviation as seen in Figure 3.
In knn, since the weights are normalised, as described in part 1, the model predictions do not shrink to zero even for data points far from the training data. This is seen in Figure 3 for data x >1 and x < -1, the predictions do not shrink to zero due to the normalisation.

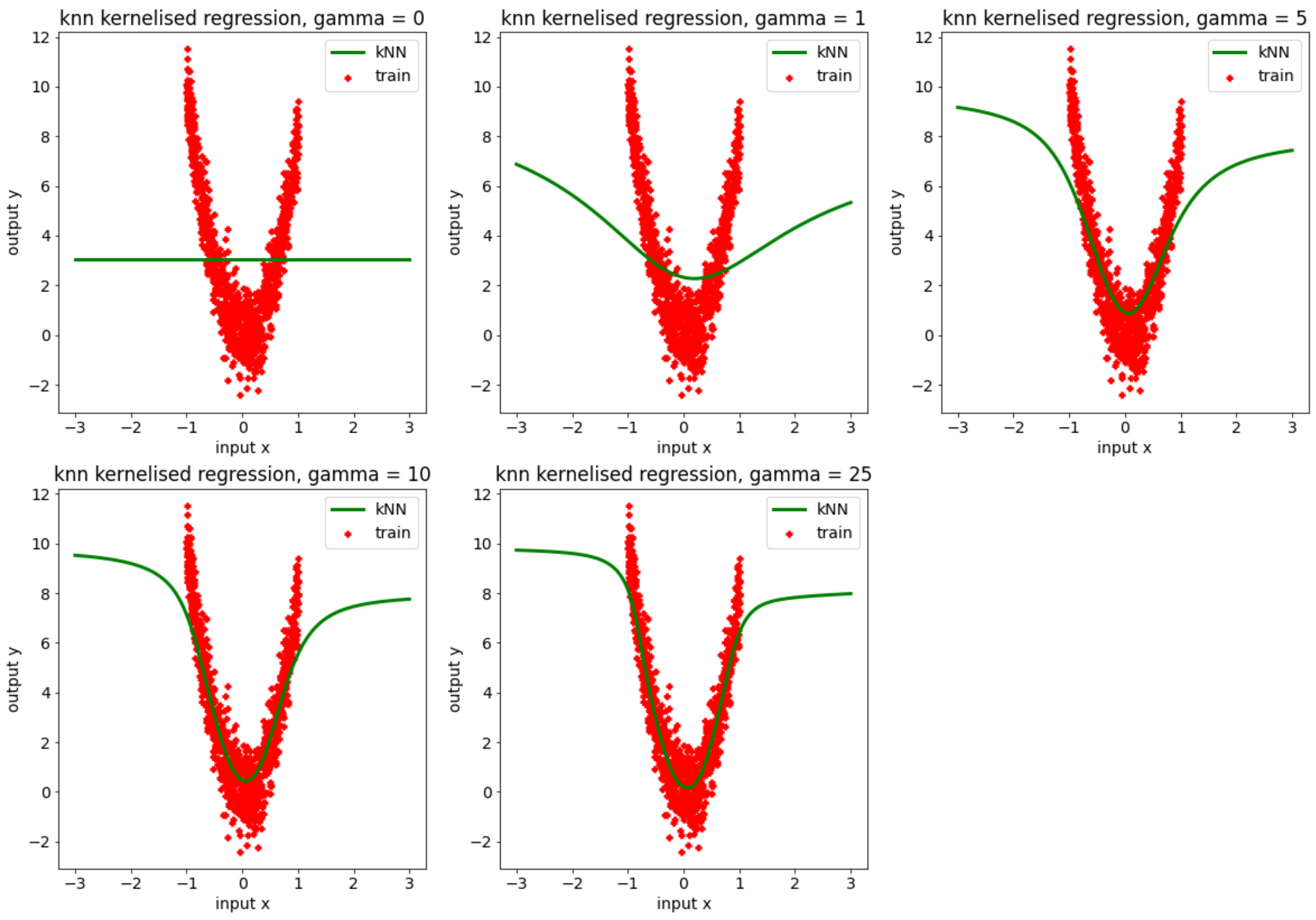


Figure 3. knn kernelised regression for gamma in the range [0, 1, 5, 10, 25]

Part b Kernelised Ridge Regression

The data was fit with kernelised ridge regression models, again using the function *plot_kernel_ridge*. The resultant plot of the model predictions for varying gamma along with the training data is shown in Figure 4. As γ is increased, more significance is placed on the training points nearest the data point, i.e those were the distance is at a minimum. Again, for $\gamma = 0$, where the weighted distance has no effect as the Kernal evaluates to 1, the prediction is constant for all input values, including those for features outside the training data. Since there is no normalisation in Kernelised ridge regression, the predictions go to zero for data far from the range of the training data as $K(x, x_i)$ is close to 0 for all x_i. This becomes more prominent as γ is increased as the amplitude of $K(x, x_i)$ decays with distance, seen when $\gamma = 1$ and $\gamma = 25$ are compared below.

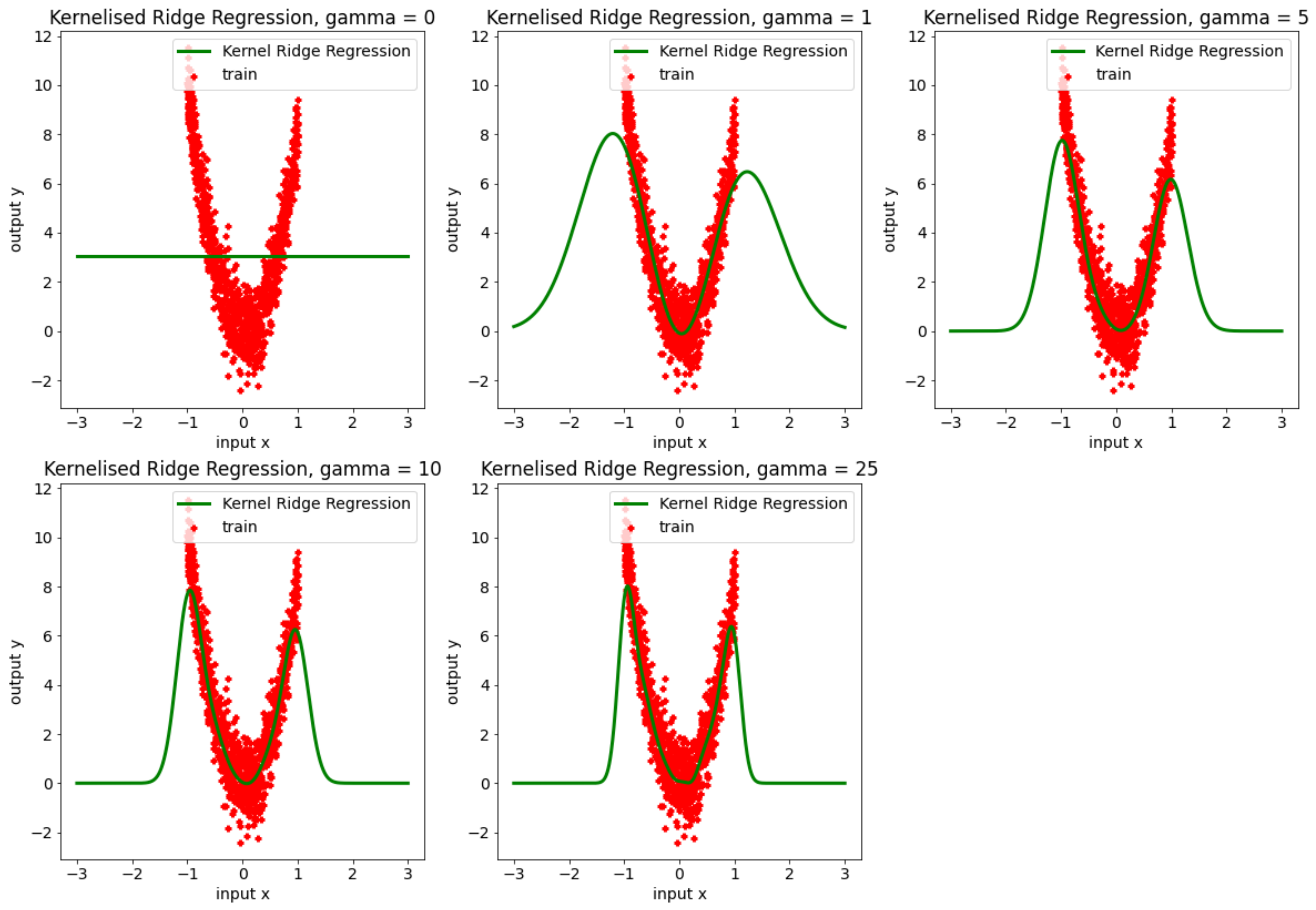


Figure 4. knn kernelised regression for gamma in the range [0, 1, 5, 10, 25]

Part c. Cross Validation Choice of Hyper parameters

Optimal γ – Kernelised knn

To select the optimal value of gamma in Kernelised knn, 5 fold cross validation was implemented. This was implemented using the function *choose_gamma_knn* as below. The resultant plot is shown in Figure 5. The mean square error (MSE) reduces significantly as gamma is increased from 0, 1 up to 50. It appears to have reduced sufficiently at a gamma value of 25. There is minimal reduction in the MSE as gamma is increased and thus 25 was chosen as the optimal value. Larger values of gamma could result in the model overfitting to the training data.

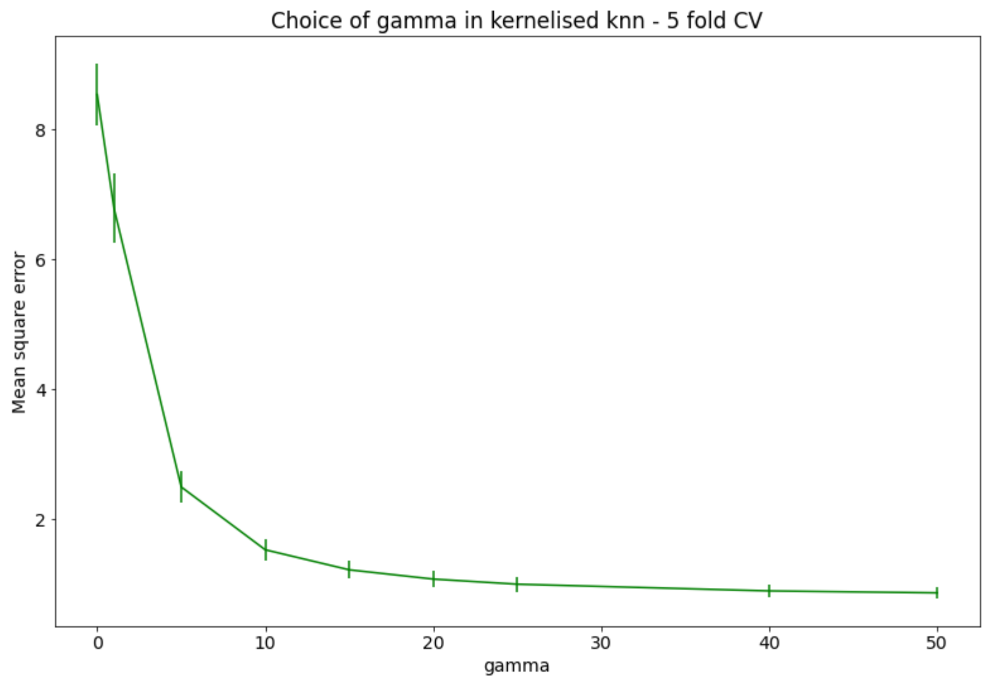


Figure 5. Kernelised knn – Choice of gamma using 5 fold CV for gamma in the range [0, 50]

Function- choose_gamma_knn

```
def choose_gamma_knn(X, y, range_gamma, plot_color):

    '''Implement 5 fold cv to determine optimal gamma'''

    #Param setup

    kf = KFold(n_splits = 5)

    mean_error=[]; std_error=[];

    for gammaX in range_gamma:

        #Params

        mse_temp = []
        #Set up gaussian kernel function
        def gaussian_kernel(distances):
            weights = np.exp(-gammaX*(distances**2))
            return weights/np.sum(weights)


        for train, test in kf.split(X):
            #Model
            model = KNeighborsRegressor(n_neighbors= len(train),
weights=gaussian_kernel)
            model.fit(X[train], y[train])
            ypred = model.predict(X[test])
            mse = mean_squared_error(y[test], ypred)
            mse_temp.append(mse)


        #Get mean & variance
        mean_error.append(np.array(mse_temp).mean())
        std_error.append(np.array(mse_temp).std())
    #Plot
    fig = plt.figure(figsize=(15,12))
    plt.errorbar(range_gamma, mean_error, yerr=std_error, color =
plot_color)
    plt.xlabel('gamma')
    plt.ylabel('Mean square error')
    plt.title('Choice of gamma in kernelised knn - 5 fold CV')
    plt.show()
```

Optimal γ and α – Kernelised Ridge Regression

The optimal value of γ was determined in the range [0, 1, 5, 10, 15, 20, 30] using 5 fold cross validation. This was implemented for four variations of the penalty term alpha; [0.01, 1, 10, 1000]. The resultant plot is shown in Figure 6, obtained using the function *choose_gamma_alpha_ridge* as below. In all cases the MSE drops dramatically as gamma is increased from zero and reaches a minimum at γ equal to 5. The optimal value of gamma was thus chosen as 5.

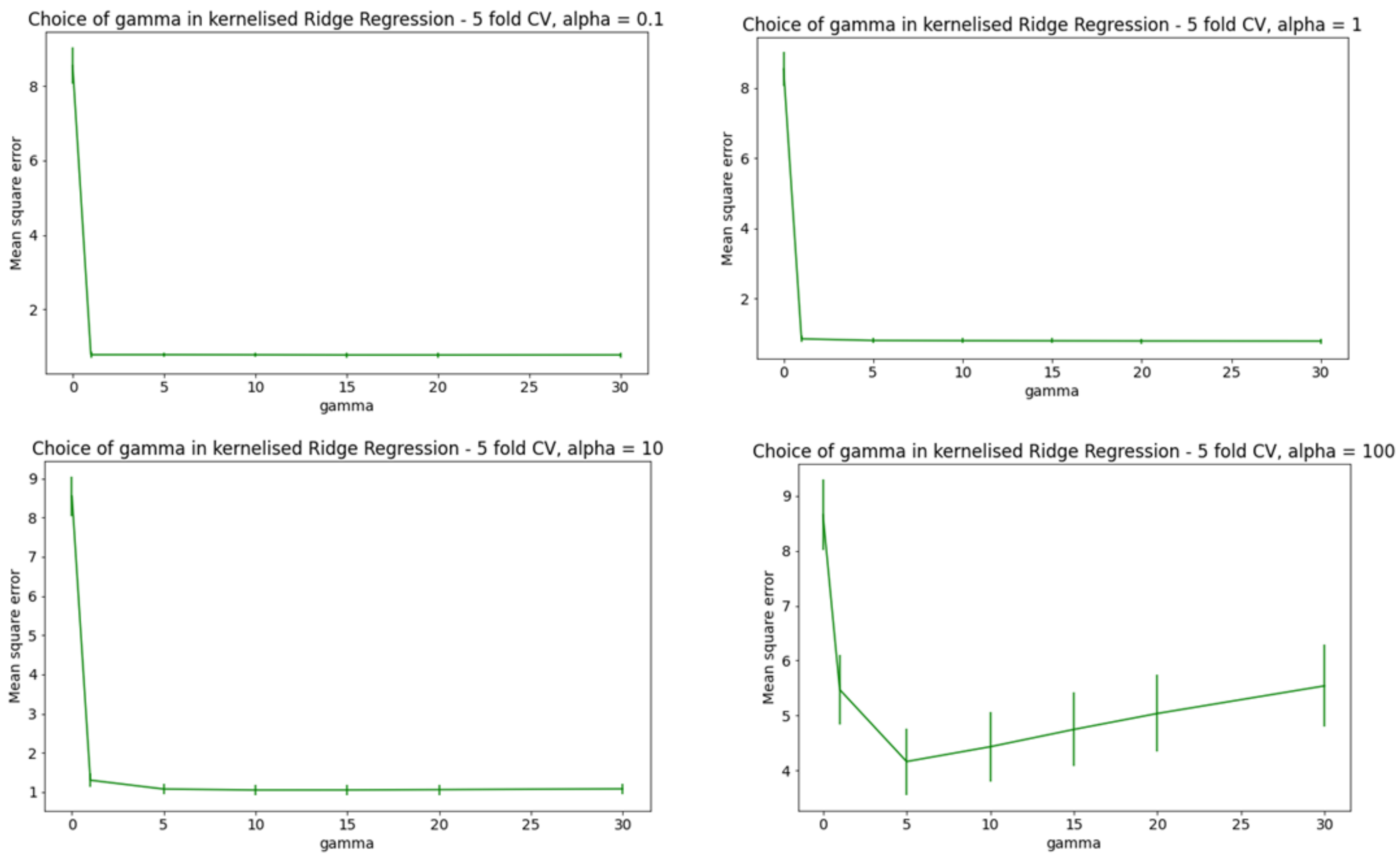


Figure 6. Kernelised Ridge Regression – Choice of gamma using 5 fold CV for gamma in the range [0, 30] and for α = 0.01, 1, 10, 1000.

Optimal α

The optimal value of the penalty term α was then determined. The penalty term α can be written as $\alpha = \frac{1}{2C}$ and thus the optimal value was determined by plotting the MSE vs C. This was implemented using the function *choose_alpha_ridge*. As shown in Figure 7 the MSE drops dramatically at a value of C = 1, which was thus chosen as the optimal. This equates to a value of 0.5 for alpha.

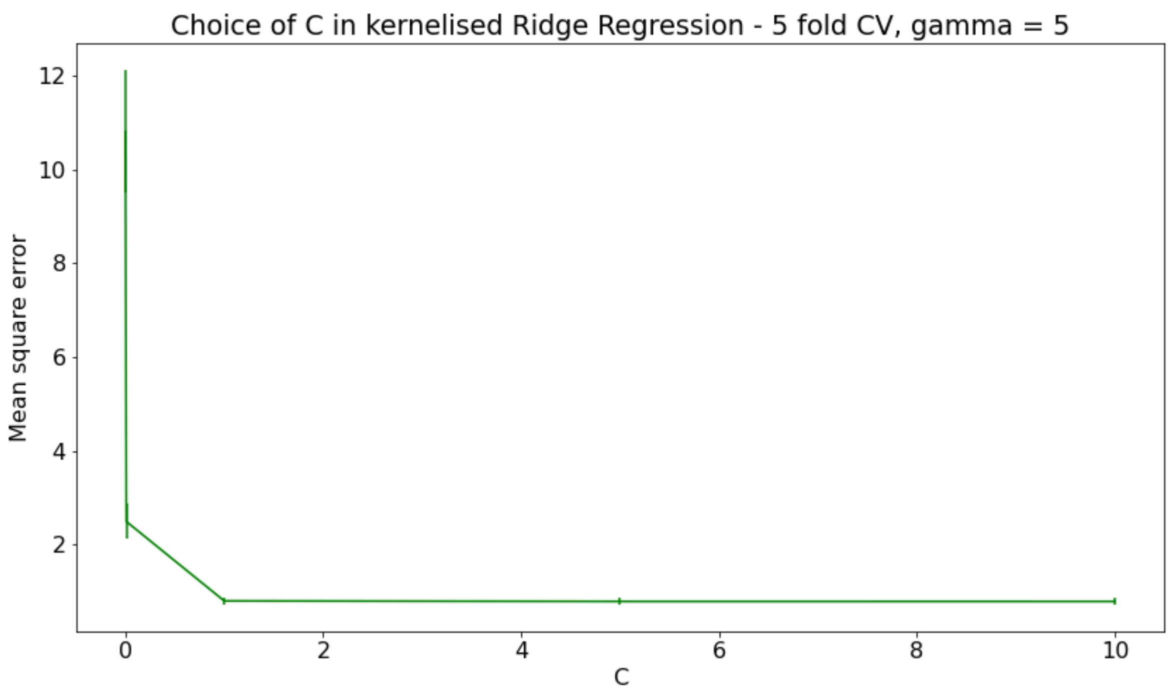


Figure 7. Kernelised Ridge Regression – Choice of penalty term using 5 fold CV for C in the range [0,10]

Predictions

Optimal knn Kernelised regression model

The optimal knn Kernelised model, with $\gamma = 25$ was fit to the data. Specifically it was trained on the training data and fit to the test data as implemented using the code below. The predictions on the test data are plotted against the actual data in Figure 8.

```
#Data

Xtest=np.linspace(range_preds[0], range_preds[1], num=1000).reshape(-1, 1)

#Gaussian kernel

gammaX = 25

def gaussian_kernel(distances):

    weights = np.exp(-gammaX*(distances**2))

    return weights/np.sum(weights)

#Model

model = KNeighborsRegressor(n_neighbors= len(X), weights=gaussian_kernel)

model.fit(X, y)

#Predictions

predictions = model.predict(Xtest)
```

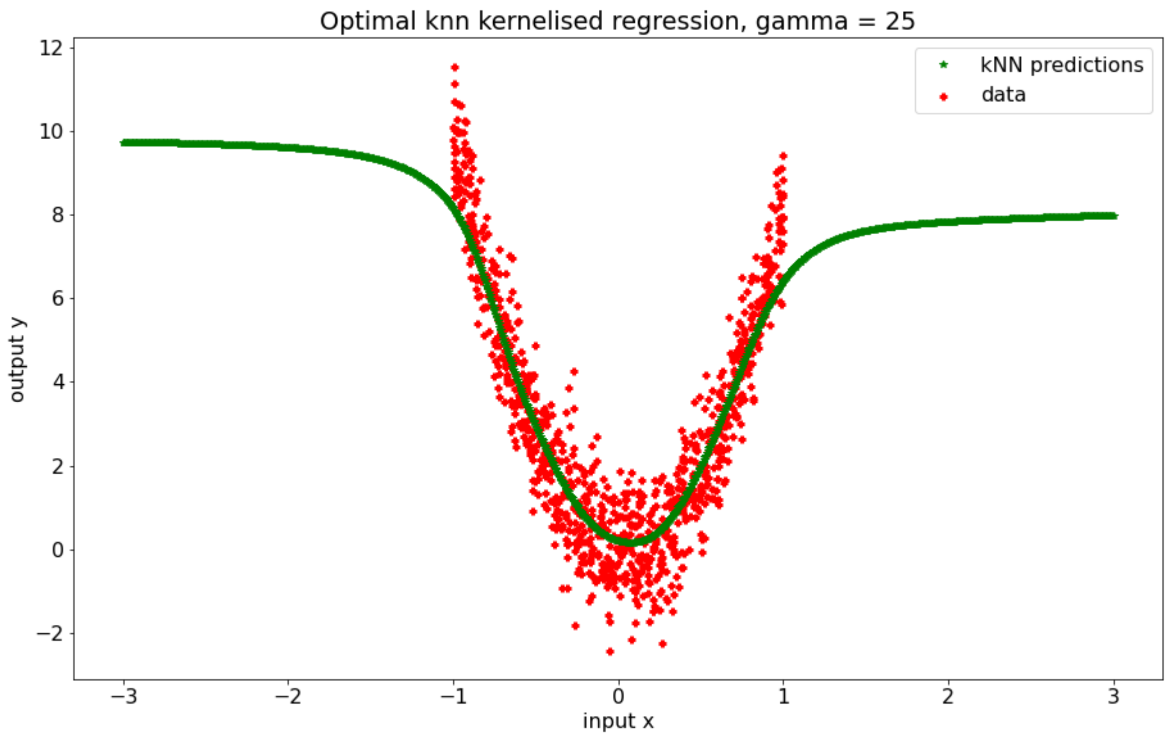


Figure 8. Optimal knn Kernelised regression model ($\gamma = 25$) *- Plot of model predictions on the test data.*

Optimal Kernelised Ridge regression model

The optimal knn Kernelised model, with $\gamma = 25$ was fit to the data. Specifically it was trained on the training set and fit to the test set as implemented using the code below. The predictions on the test data are plotted against the actual data in Figure 9.

```
#Data

Xtest=np.linspace(range_preds[0], range_preds[1], num=1000).reshape(-1, 1)

#Model

C = 1

gammaX = 5

model = KernelRidge(alpha= 1.0/(2*C), kernel= 'rbf', gamma=gammaX)

model.fit(X, y)

#Predictions

predictions = model.predict(Xtest)
```

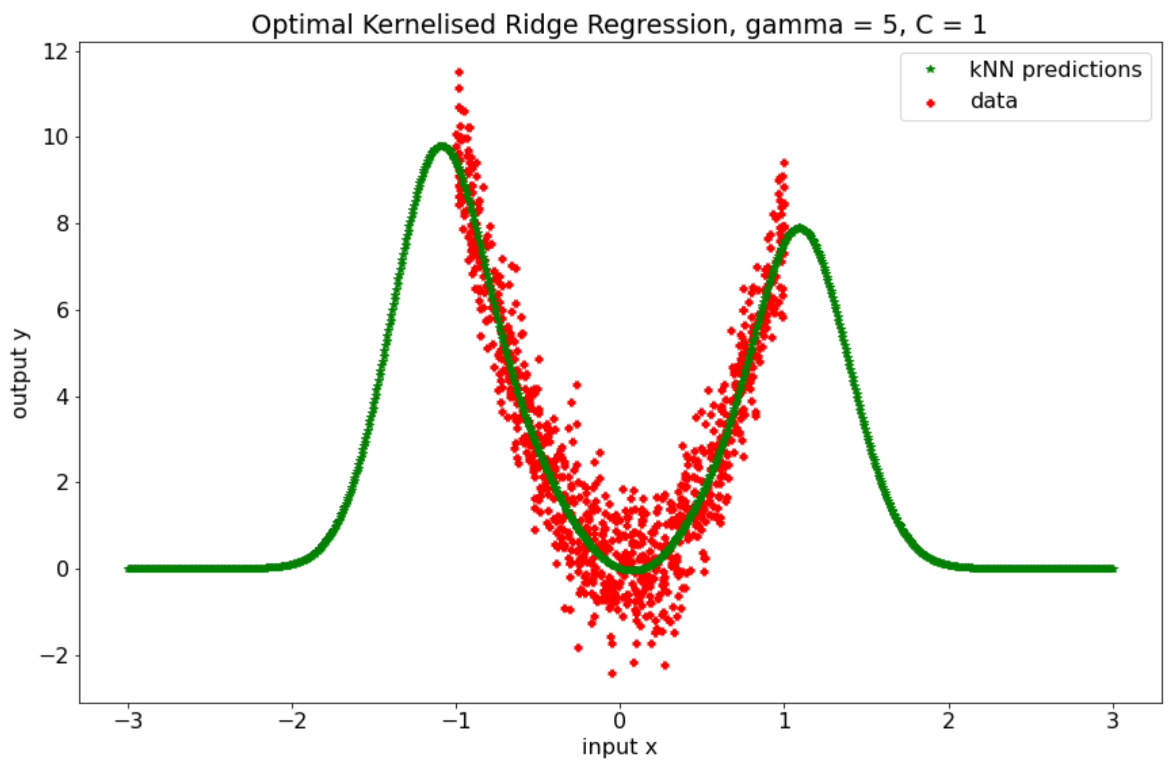


Figure 9. Optimal Kernelised Ridge regression model ($\gamma = 5,\ C = 1$) *- Plot of model predictions on the test data.*

Comparison of predictions

Both optimised models provide quite a good fit to the test data that is within the range of the training data. The predicted y value in both cases is largely centred on the actual y value for data in the range [1, -1]. However, for data outside the range of the training data, there is a significant difference in predictions. In the case of the knn, the weights are normalised as in equation 1, i.e;

The division by $\sum_{i=1}^{m} K(x, x_i)$ makes a significant difference to the predictions. For points far from the training data, where $K\left(x^{(i)}, x\right)$ is close to 0 for all x_i, the denominator $\sum_{i=1}^{m} K(x, x_i)$ is also close to 0 and so compensates. As a result the predictions do not shrink to zero. This is the case in Figure 8 where the predictions on the test data outside the training data range converge towards a value of 9.8. This is not the case in Kernelised ridge regression where there is no normalisation, i.e;

$$\hat{y} = \sum_{j=1}^{m} \alpha_j y^{(i)} K(x, x^{(j)})$$

Thus as the distance from the training data is increased, and $K\left(x^{(i)}, x\right)$ tends to 0 for all x_i, the predictions fall to zero. This is the case in Figure 9 where the predictions converge towards zero as the distance from the training data range [-1,1] is increased.

Appendix - Code

```
# -*- coding: utf-8 -*-
"""
Created on Mon Nov 16 10:46:11 2020
@author: Hannah Craddock
"""
#Imports

import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsRegressor
from sklearn.kernel_ridge import KernelRidge
import matplotlib.pylab as plt
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

plt.rc('font', size=16);
%matplotlib qt



#**********************
#Part 1: Dummy
#Data
Xtrain = np.array([-1, 0, 1]).reshape(-1, 1)
ytrain = [0,1,0]

#**************************************
#Part a
```

```
#Function plot for varying gamma

def plot_knn_regressor(Xtrain, ytrain, k, range_preds, range_gamma,
plot_dims):

    'Plot knn predictions for varying parameter gamma in kernelised
knn'

    #Setup

    fig = plt.figure(figsize=(20, 20))

    count = 0

    #Generate test data

    Xtest=np.linspace(range_preds[0], range_preds[1],
num=1000).reshape(-1, 1)

    for gammaX in range_gamma:


        #Set up gaussian kernel function

        def gaussian_kernel(distances):

            weights = np.exp(-gammaX*(distances**2))

            return weights/np.sum(weights)


        #Model

        model = KNeighborsRegressor(n_neighbors= k,
weights=gaussian_kernel).fit(Xtrain, ytrain)

        #Predictions

        ypred = model.predict(Xtest)

        #Plot

        count +=1

        plt.subplot(plot_dims[0], plot_dims[1], count)


        plt.scatter(Xtrain, ytrain, color= 'red', marker='+',
linewidth = 3)

        plt.plot(Xtest, ypred, 'g-',linewidth = 3)

        plt.xlabel('input x'); plt.ylabel('output y')
        plt.legend(['kNN','train'])

        plt.title('knn kernelised regression, gamma =
{}'.format(gammaX))

        plt.show()


#Apply function

k = 3

range_preds = [-3, 3]

range_gamma = [0, 1, 5, 10, 25]

plot_dims = [2, 3]

#Apply

plot_knn_regressor(Xtrain, ytrain, k, range_preds, range_gamma,
plot_dims)


#**************************************

#Part c - Kernelised Ridge Regression


def plot_kernel_ridge(Xtrain, ytrain, C, range_preds, range_gamma,
plot_dims, legend_loc):

    'Plot kernel ridge regression predictions for varying the
parameter gamma in the kernel and return coefficients in a
dataframe'


    #Setup

    fig = plt.figure(figsize=(20, 20))

    results = []

    count = 0


    #Generate test data

    Xtest=np.linspace(range_preds[0], range_preds[1],
num=1000).reshape(-1, 1)
```

```
    #Test range of gamma values

    for gammaX in range_gamma:

        count += 1


        #Model

        model = KernelRidge(alpha=1.0/2*C, kernel= 'rbf',
gamma=gammaX).fit(Xtrain, ytrain)


        #Predictions

        ypred = model.predict(Xtest)


        #Coefficients

        d = {

        'gamma' : gammaX,

        'coefficients' :  np.around(model.dual_coef_, decimals = 3),

        }


        results.append(d)


        #Plot

        plt.subplot(plot_dims[0], plot_dims[1], count)


        plt.scatter(Xtrain, ytrain, color= 'red', marker='+',
linewidth = 3)

        plt.plot(Xtest, ypred, 'g-',linewidth = 3)

        plt.xlabel('input x'); plt.ylabel('output y')

        plt.legend(['Kernel Ridge Regression','train'], loc =
legend_loc)
```

```
        plt.title('Kernelised Ridge Regression, gamma =
{}'.format(gammaX))

        plt.show()


        #Return results

        df_coeffs = pd.DataFrame(results)


    return df_coeffs


#To do

range_gamma = [0, 1, 5, 10, 25]

C = 0.1

legend_loc = 'center right'

df_coeffs = plot_kernel_ridge(Xtrain, ytrain, C, range_preds,
range_gamma, plot_dims, legend_loc)



#*******************************************************************

#Part 2: Dataset

data = pd.read_csv('data_week6.txt')

data.reset_index(inplace=True)

X = np.array(data.iloc[:,0]).reshape(-1, 1) #

y = np.array(data.iloc[:,1]).reshape(-1, 1)


#Inspect

max(X) #1

min(X) #-1


#Part a - kernelised knn
```

```
#Apply function

k = len(X)

range_preds = [-3, 3] #M

range_gamma = [0, 1, 5, 10, 25]

plot_dims = [2, 3]

#Apply

plot_knn_regressor(X, y, k, range_preds, range_gamma, plot_dims)


#Part b - kernelised ridge

C = 0.1

legend_loc = 'upper right'

df_coeffs = plot_kernel_ridge(X, y, C, range_preds, range_gamma,
plot_dims, legend_loc)


#******************************************************************
#Part c Cross validation


def choose_gamma_knn(X, y, range_gamma, plot_color):

    '''Implement 5 fold cv to determine optimal gamma'''


    #Param setup

    kf = KFold(n_splits = 5)

    mean_error=[]; std_error=[];


    for gammaX in range_gamma:

        #Params

        mse_temp = []


        #Set up gaussian kernel function
        def gaussian_kernel(distances):
            weights = np.exp(-gammaX*(distances**2))
            return weights/np.sum(weights)

        for train, test in kf.split(X):
            #Model
            model = KNeighborsRegressor(n_neighbors= len(train),
weights=gaussian_kernel)
            model.fit(X[train], y[train])
            ypred = model.predict(X[test])
            mse = mean_squared_error(y[test], ypred)
            mse_temp.append(mse)

        #Get mean & variance
        mean_error.append(np.array(mse_temp).mean())
        std_error.append(np.array(mse_temp).std())

    #Plot
    fig = plt.figure(figsize=(15,12))
    plt.errorbar(range_gamma, mean_error, yerr=std_error, color =
plot_color)
    plt.xlabel('gamma')
    plt.ylabel('Mean square error')
    plt.title('Choice of gamma in kernelised knn - 5 fold CV')
    plt.show()

#Apply
k = len(X)
range_gamma = [0, 1, 5, 10, 15, 20, 25, 40, 50]
```

```
plot_color = 'green'

choose_gamma_knn(X, y, range_gamma, plot_color)


#*********************

#Apply optimal model - optimal gamma

#Data

Xtest=np.linspace(range_preds[0], range_preds[1],
num=1000).reshape(-1, 1)


#Gaussian kernel

gammaX = 25

def gaussian_kernel(distances):

    weights = np.exp(-gammaX*(distances**2))

    return weights/np.sum(weights)


#Model

model = KNeighborsRegressor(n_neighbors= len(X),
weights=gaussian_kernel)

model.fit(X, y)


#Predictions

predictions = model.predict(Xtest)


#Plot

fig = plt.figure(figsize=(15,12))

plt.scatter(X, y, color= 'red', marker='+', linewidth = 3)

plt.plot(Xtest, predictions, 'g*',linewidth = 3)

plt.xlabel('input x'); plt.ylabel('output y')

plt.legend(['kNN predictions','data'])
plt.title('Optimal knn kernelised regression, gamma =
{}'.format(gammaX))

plt.show()


#******************************************************************

#Part c Cross validation - kernelised Ridge Regression


def choose_gamma_ridge(X, y, range_gamma, alphaX, plot_color):

    '''Implement 5 fold cv to determine optimal gamma'''


    #Param setup

    kf = KFold(n_splits = 5)

    mean_error=[]; std_error=[];


    for gammaX in range_gamma:

        #Params

        mse_temp = []

        #Model

        model = KernelRidge(alpha= alphaX, kernel= 'rbf',
gamma=gammaX)


        #5 fold CV

        for train, test in kf.split(X):

            #Model

            model.fit(X[train], y[train])

            ypred = model.predict(X[test])

            mse = mean_squared_error(y[test], ypred)

            mse_temp.append(mse)
        #Get mean & variance

        mean_error.append(np.array(mse_temp).mean())

        std_error.append(np.array(mse_temp).std())


    #Plot

    fig = plt.figure(figsize=(15,12))

    plt.errorbar(range_gamma, mean_error, yerr=std_error, color =
plot_color)

    plt.xlabel('gamma')

    plt.ylabel('Mean square error')

    plt.title('Choice of gamma in kernelised Ridge Regression - 5
fold CV, alpha = {}'.format(alphaX))

    plt.show()


#Apply

alphaX = 1

range_gamma = [ 0, 1, 5, 10, 15, 20, 30]

choose_gamma_alpha_ridge(X, y, range_gamma, alphaX, plot_color)



#***************************

def choose_alpha_ridge(X, y, range_C, gammaX, plot_color):

    '''Implement 5 fold cv to determine optimal gamma'''


    #Param setup

    kf = KFold(n_splits = 5)

    mean_error=[]; std_error=[];


    for C in range_C:
        #Params

        mse_temp = []

        #Model

        model = KernelRidge(alpha= 1.0/(2*C), kernel= 'rbf',
gamma=gammaX)


        #5 fold CV

        for train, test in kf.split(X):

            #Model

            model.fit(X[train], y[train])

            ypred = model.predict(X[test])

            mse = mean_squared_error(y[test], ypred)

            mse_temp.append(mse)


        #Get mean & variance

        mean_error.append(np.array(mse_temp).mean())

        std_error.append(np.array(mse_temp).std())


    #Plot

    fig = plt.figure(figsize=(15,12))

    plt.errorbar(range_C, mean_error, yerr=std_error, color =
plot_color)

    plt.xlabel('C')

    plt.ylabel('Mean square error')

    plt.title('Choice of C in kernelised Ridge Regression - 5 fold
CV, gamma = {}'.format(gammaX))

    plt.show()


#Apply

gammaX = 5
```

```
range_C = [0.001, 0.01, 1, 5, 10]# 100]

choose_alpha_ridge(X, y, range_C, gammaX, plot_color)


#*****************

#Optimal model

#Data

#Xtrain, Xtest, ytrain, ytest = train_test_split(X, y,
test_size=0.33, random_state=42)

Xtest=np.linspace(range_preds[0], range_preds[1],
num=1000).reshape(-1, 1)


#Model

C = 1

gammaX = 5

model = KernelRidge(alpha= 1.0/(2*C), kernel= 'rbf', gamma=gammaX)

model.fit(X, y)

#Params

model.dual_coef_

#Predictions

predictions = model.predict(Xtest)


#Plot

fig = plt.figure(figsize=(15,12))

plt.scatter(X, y, color= 'red', marker='+', linewidth = 3)

plt.plot(Xtest, predictions, 'g*',linewidth = 3)

plt.xlabel('input x'); plt.ylabel('output y')

plt.legend(['kNN predictions','data'])

plt.title('Optimal Kernelised Ridge Regression, gamma = {}, C =
{}'.format(gammaX, C))

plt.show()
```