Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

CS7CS4 Machine Learning Week 6 Assignment

Boris Flesch
20300025

November 25, 2020

MSc Computer Science, Intelligent Systems

1 Downloaded Dataset

id:23-69-115

2 Questions

2.1 Part (i)

The dummy training data composed of three points can be easily created using Numpy arrays. We have to reshape the single-feature array X so that it can be used with Sklearn methods:

```
X = np.array([-1, 0, 1]).reshape(-1, 1)
y = np.array([0, 1, 0])
```

(a) kNN predictions for this data can be created using KNeighborsRegressor from Sklearn. We just have to create and give different gaussian kernel functions to the regressor depending on the value of γ (gamma) that we want to use. For predictions, a linear space can be created from -3 to 3 to go beyond the range of the training data. The following functions will allow us to plot such predictions given training data, *k* and the gaussian kernels to use:

```
def gaussian_kernel25(distances):
    weights = np.exp(-25*(distances**2))
    return weights/np.sum(weights)
# Similar functions created for gamma = 0, 1, 5, 10, etc.

def knnGaussianKernel(X, y, k, gaussianKernels):
    fig = plt.figure(num=None, figsize=(8, 5), dpi=120)
    plt.rc('font', size=12)

    if (X.size < 10):
        plt.scatter(X, y, color='black', marker='+', s=10,
        ↪  linewidth=15, label="train")
    else:
        plt.scatter(X, y, color='black', marker='+', label="train")

    Xtest = np.linspace(-3.0, 3.0, num=1000).reshape(-1, 1)

    for gaussianKernel in gaussianKernels:
        model = KNeighborsRegressor(n_neighbors=k,
        ↪  weights=gaussianKernel[1]).fit(X, y)
        ypred = model.predict(Xtest)

        plt.plot(Xtest, ypred, label="Predictions -
        ↪  gamma=%d"%gaussianKernel[0])

    plt.xlabel("X"); plt.ylabel("y")
    plt.legend(bbox_to_anchor=(1, 1), loc='upper left')
```

```
25      plt.title("kNN using Gaussian weights - k = %d" % k)
26      plt.show()
```

We can then call this function with the following code, that gives output displayed on Figure 1 below:

```
1       kernels = [
2           [0, gaussian_kernel0],
3           [1, gaussian_kernel1],
4           [5, gaussian_kernel5],
5           [10, gaussian_kernel10],
6           [25, gaussian_kernel25]
7       ]
8       knnGaussianKernel(X, y, k=3, gaussianKernels=kernels)
```

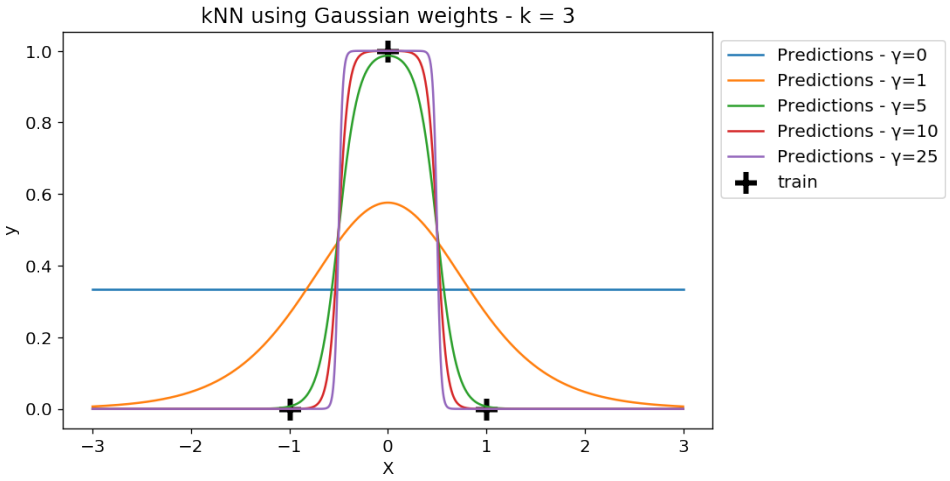

Figure 1

(b) Varying γ allow us to alter the weight (i.e. importance) that the model will give to each data point. The weight $w^{(i)} = e^{-\gamma d(x^{(i)},x)^2}$ attaches more weight to training points that are near the point we want to predict output for (i.e. x).

Having $\gamma = 0$ is equivalent to uniform weights as $e^0 = 1$. The average of our training outputs y is $\frac{0+0+1}{3} = 0.333$; and as we can see on Figure 1, it corresponds to the constant line drawn for $\gamma = 0$.

When $\gamma = 1$, we are attaching slightly more importance to data points that are near our query point x. We can take the example of the prediction for $x = 0$:

- Distances:
  - $d(x^{(1)}, x) = \sqrt{(-1-(-1))^2} = 0$
  - $d(x^{(2)}, x) = \sqrt{(0-(-1))^2} = 1$
  - $d(x^{(3)}, x) = \sqrt{(1-(-1))^2} = 2$
- Weights:
  - $w^{(1)} = e^{-1\times 0} = 1$

- $w^{(2)} = e^{-1\times1} = 0.367$
- $w^{(3)} = e^{-1\times2} = 0.135$

- Prediction:

$$\frac{\sum_{i\in N_k} w^{(i)} y^{(i)}}{\sum_{i\in N_k} w^{(i)}} = \frac{1\times0+0.367\times1+0.135\times0}{1.502} = 0.244$$

This result can be visualised with the orange curve ($\gamma = 25$) on Figure 1 and an analogous reasoning can be applied for other points.

When γ is very high (i.e. $\gamma = 25$ in this example), this will make the model give much of its attention to the nearest point from the target value. Hence for $x \in]-0.5, 0.5[$, the predicted value will be 1 as the nearest training point is $(0, 1)$.

It is also interesting to take a look outside the scope of the training data: as γ increases, the predictions rely only on the nearest data point from the query target x, thus leading to $\hat{y} = 0$ for $x \in]-\infty, -1[\cup]1, \infty[$. This happens when $\gamma \geq 5$; for $\gamma \leq 1$, predictions are still attaching more importance (i.e. weight) to the training data point $(0, 1)$, thus taking longer for the predictions curve to "reset to 0". $\gamma = 0$ is a particular case: as weights are uniform, the prediction is a flat line that will always consider data point $(0, 1)$, thus never predicting $\hat{y} = 0$.

(c) To use the sklearn KernelRidge function to train a kernalised ridge regression model, we can create a similar function to the one created in (i)(a). This function will take training data, C value and a range of gaussian kernels as parameters:

```
1  def kernalisedRidgeRegression(X, y, C, gaussianKernels,
   ↪  printCoeff=False):
2      fig = plt.figure(num=None, figsize=(8, 5), dpi=120)
3      plt.rc('font', size=12);
       ↪  plt.rcParams['figure.constrained_layout.use'] = True
4
5      if (X.size < 10):
6          plt.scatter(X, y, color='black', marker='+', s=10,
           ↪  linewidth=15, label="train")
7      else:
8          plt.scatter(X, y, color='black', marker='+', label="train")
9
10     for gaussianKernel in gaussianKernels:
11         model = KernelRidge(alpha=1.0/C, kernel='rbf',
           ↪  gamma=gaussianKernel[0]).fit(X, y)
12         Xtest = np.linspace(-3.0, 3.0, num=1000).reshape(-1, 1)
13         ypred = model.predict(Xtest)
14         plt.plot(Xtest, ypred, label="Predictions -
           ↪  gamma=%d"%gaussianKernel[0])
15         if (printCoeff):
16             print("Kernel Ridge Regression - C = %d, gamma=%d" % (C,
               ↪  gaussianKernel[0]))
17             print("gamma =", model.dual_coef_)
18             print("--")
19
```

```
    plt.xlabel("X"); plt.ylabel("y")
    plt.legend(bbox_to_anchor=(1, 1), loc='upper left')
    plt.title("Kernel Ridge Regression - C = %.1f" % (C))
    plt.show()
```

We can then call this function for $\gamma \in \{0, 1, 5, 10, 25\}$ and $C \in \{0.1, 1, 1000\}$, which gives us the following results:

```
kernalisedRidgeRegression(X, y, C=0.1, gaussianKernels=kernels,
↪ printCoeff=True)
kernalisedRidgeRegression(X, y, C=1, gaussianKernels=kernels,
↪ printCoeff=True)
kernalisedRidgeRegression(X, y, C=100, gaussianKernels=kernels,
↪ printCoeff=True)
```

- For $C = 0.1$:

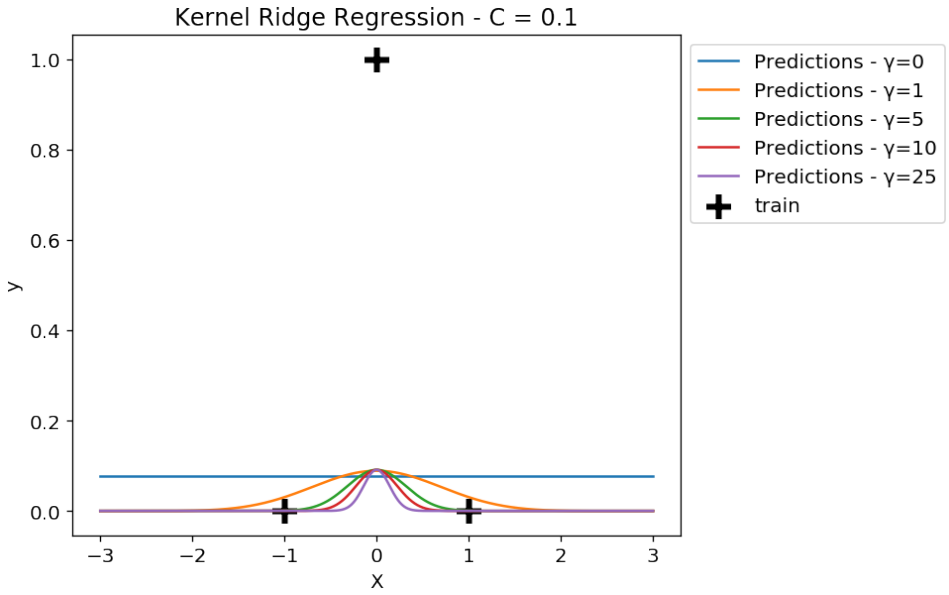

Figure 2

$$\theta_{\gamma=0} = \begin{bmatrix} -0.00769231 & 0.09230769 & -0.00769231 \end{bmatrix} \tag{1}$$

$$\theta_{\gamma=1} = \begin{bmatrix} -0.00304207 & 0.09111257 & -0.00304207 \end{bmatrix} \tag{2}$$

$$\theta_{\gamma=5} = \begin{bmatrix} -5.56855542e{-5} & 9.09091591e{-2} & -5.56855542e{-5} \end{bmatrix} \tag{3}$$

$$\theta_{\gamma=10} = \begin{bmatrix} -3.75206031e{-7} & 9.09090909e{-2} & -3.75206031e{-7} \end{bmatrix} \tag{4}$$

$$\theta_{\gamma=25} = \begin{bmatrix} -1.14776396e{-13} & 9.09090909e{-2} & -1.14776396e{-13} \end{bmatrix} \tag{5}$$

- For $C = 1$:

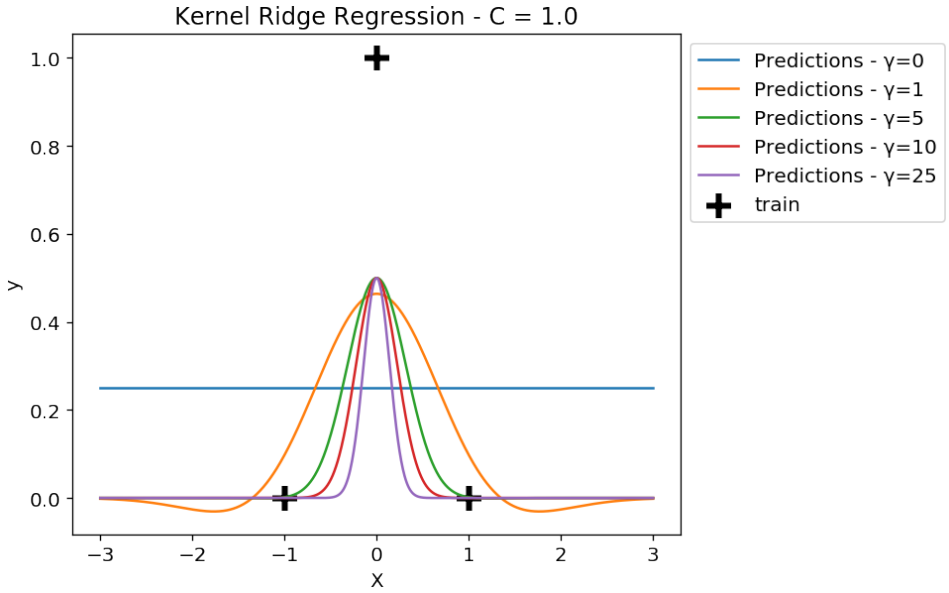

Figure 3

$$\theta_{\gamma=0} = \begin{bmatrix} -0.25 & 0.75 & -0.25 \end{bmatrix} \tag{6}$$

$$\theta_{\gamma=1} = \begin{bmatrix} -0.09768542 & 0.53593646 & -0.09768542 \end{bmatrix} \tag{7}$$

$$\theta_{\gamma=5} = \begin{bmatrix} -0.00168452 & 0.50001135 & -0.00168452 \end{bmatrix} \tag{8}$$

$$\theta_{\gamma=10} = \begin{bmatrix} -1.13499825\text{e}{-5} & 5.00000001\text{e}{-1} & -1.13499825\text{e}{-5} \end{bmatrix} \tag{9}$$

$$\theta_{\gamma=25} = \begin{bmatrix} -3.47198597\text{e}{-12} & 5.00000000\text{e}{-1} & -3.47198597\text{e}{-12} \end{bmatrix} \tag{10}$$

- For $C = 100$:

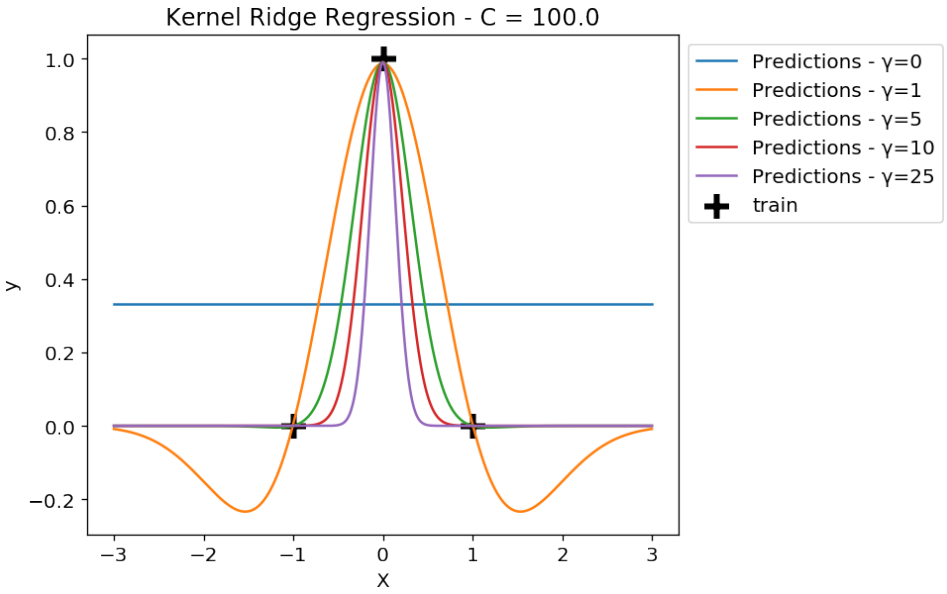

Figure 4

$$\theta_{\gamma=0} = \begin{bmatrix} -33.22259136 & 66.77740864 & -33.22259136 \end{bmatrix} \tag{11}$$

$$\theta_{\gamma=1} = \begin{bmatrix} -0.47905446 & 1.33907779 & -0.47905446 \end{bmatrix} \tag{12}$$

$$\theta_{\gamma=5} = \begin{bmatrix} -0.00660577 & 0.99018715 & -0.00660577 \end{bmatrix} \tag{13}$$

$$\theta_{\gamma=10} = \begin{bmatrix} -4.45053720e{-5} & 9.90099014e{-1} & -4.45053720e{-5} \end{bmatrix} \tag{14}$$

$$\theta_{\gamma=25} = \begin{bmatrix} -1.36142965e{-11} & 9.90099010e{-1} & -1.36142965e{-11} \end{bmatrix} \tag{15}$$

(d) The principle of a Kernalised Ridge Regression model is to use training data as features by associating a feature to each of our three training data points and then using a linear model.

Each feature i is defined by the function $y^{(i)}K(x^{(i)}, x)$ where $K(x^{(i)}, x)$ is the kernel (i.e. gaussian kernel in our case). Using these features, we can learn parameters θ by minimising a cost function: $\hat{y} = \theta_0 + \theta_1 y^{(1)}K(x^{(1)}, x) + ... + \theta_m y^{(m)}K(x^{(m)}, x)$.

Using $k = m = 3$ allows us to understand that this kernalised ridge regression model will behave as our previously used kNN model. However, this new model will add θ parameters and L2 penalty that bring more flexibility.

A L2 penalty encourages small values of parameter θ. That explains why smaller is C (i.e. greater is α as $\alpha = 1/(2C)$), flatter are the predictions. In other words, the only point which has output 1 is considered as noise by this penalty. This is why the delta between θ_1 and θ_2 (or between θ_3 and θ_2) increases along with C. For example, considering $\gamma = 1$, for $C = 0.1$, $\theta_2 - \theta_1 = 0.091 - 0.003 = 0.088$ while for $C = 100$,

$\theta_2 - \theta_1 = 1.339 - (-0.479) = 1.818$. In other terms, the importance given to data point $(0, 1)$ is ~ 20 times greater for $C = 100$ than for $C = 0.1$. Similar observations can be made for other values of γ.
Note: some values of C seem to produce negative predictions for $\gamma = 1$, which is apparently a bug with regards to discussions on Machine Learning module Blackboard's Forum.

γ has a similar behaviour as in kNN model used in (i)(a) and explained in (i)(b): when it increases, it makes the model give more importance to points which are near the point x we want to predict output for. Visually, chaging γ impacts the "width" of the curves (especially between -1 and 1) while C has more impact on their height.

2.2 Part (ii)

The downloaded training data can be read using the following code and can be visualised on Figure 5 below:

```
def readData(filepath):
    df = pd.read_csv(filepath, comment="#")
    X = df.iloc[:,0]
    y = df.iloc[:,1]
    return np.array(X).reshape(-1, 1), np.array(y)

X,y = readData("week6.csv")
```

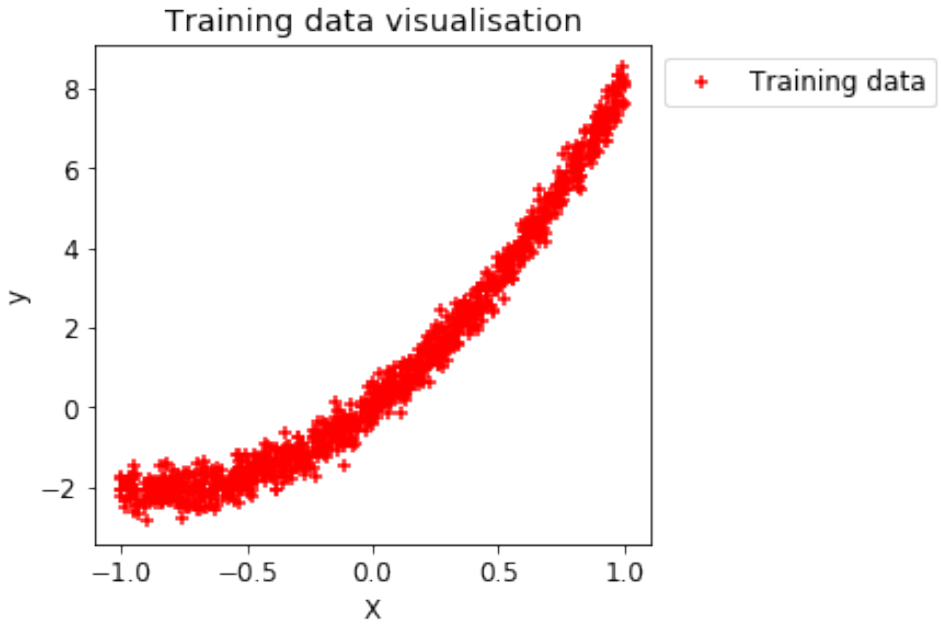

Figure 5

(a) Using the *knnGaussianKernel* method created in part (i)(a), we obtain the following predictions (Figure 6) by varying γ:

```
kernels = [
    [0, gaussian_kernel0],
    [1, gaussian_kernel1],
    [5, gaussian_kernel5],
    [10, gaussian_kernel10],
    [25, gaussian_kernel25]
]
knnGaussianKernel(X, y, k=X.size, gaussianKernels=kernels)
```

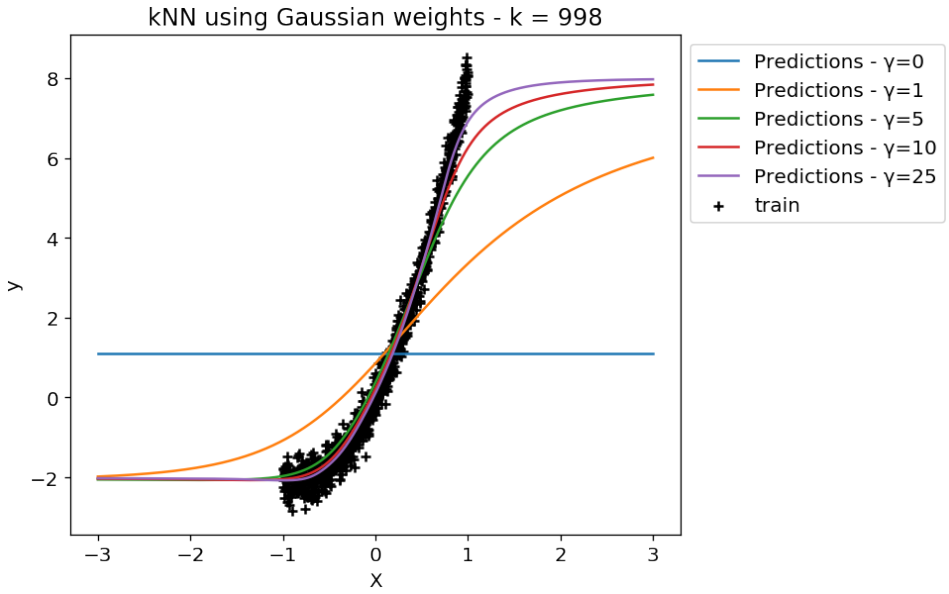

Figure 6

We can see from these predictions that the kNN adopts a similar behaviour as in (i)(a). For example, for $\gamma = 0$, a line corresponding to the training data output average is drawn (i.e. $y = 1.104$). The more γ increases, the more the predictions are "attracted" by the points which are near the query point x.

Outside the scope of the training data, the predictions flatten to the average of the closest bunch of training data points. To predict value for $x < -1$ (resp. $x > 1$), we can calculate the average of training data points which are at equal distance from the query point, that means data points having $x = -1$ (resp. $x = 1$). This is the value to which the predictions will converge beyond the range of the training data. We can use the following code to calculate it:

```
Xm1_y = []
Xp1_y = []
for i in range(X.size):
    if X[i] == -1:
        Xm1_y.append(y[i])
    elif X[i] == 1:
        Xp1_y.append(y[i])

    print("Output y average for training data points where x=-1:
    ↪ %.3f"%np.mean(Xm1_y))
    print("Output y average for training data points where x=1:
    ↪ %.3f"%np.mean(Xp1_y))
```

This code indicates us that predictions for $x < -1$ will tend to -1.985 while predictions for $x > 1$ will tend to 7.971: this is the behaviour we can observe in Figure 5 above.

Increasing γ makes the predictions converging quicker to these values, as only the nearest data points will be considered. In other words, using $\gamma = 25$ will make the model attaching all its importance only to data points having $x = 1$ sooner than $\gamma = 1$ would do. Again,

as in (i)(a), $\gamma = 0$ has a particular behaviour as it leads to uniform weights, thus never predicting other values than the average output of all training data points.

By generalising these observations, we can actually argue that the kNN model behaves like a weighted average as it associates a weight to each training data point depending on its distance (and on γ value) and outputs the average considering all weighted training data points.

(b) Before training ridge regression models for a wide range of γ values, it could be a good idea to cross-validate the L2 penalty C. To do this, we can measure the MSE (Mean Square Error) and standard deviation of our models for a wide range of C (i.e. $C \in \{0.1, 1, 10, 50, 100, 500, 1000\}$) as well as a wide range of γ to observe potential impact of γ on these measurements. This can be done using a 5-fold cross-validation (i.e. 80/20% splitting of our training dataset):

```
C_range = [0.1, 1, 10, 50, 100, 500, 1000]
kernels = [
    [1, gaussian_kernel1],
    [5, gaussian_kernel5],
    [10, gaussian_kernel10],
    [25, gaussian_kernel25]
]

for gaussianKernel in kernels:
    mean_error = []
    std_error = []
    for C in C_range:
        model = KernelRidge(alpha=1.0/C, kernel='rbf',
        ↪  gamma=gaussianKernel[0])
        scores = cross_val_score(model, X, y, cv=5,
        ↪  scoring='neg_mean_squared_error')
        mean_error.append(-np.array(scores).mean())
        std_error.append(np.array(scores).std())

    plt.errorbar(C_range, mean_error, yerr=std_error, linewidth=3,
    ↪  label="gamma = %d"%gaussianKernel[0])

plt.title("Error mean and variance for C cross-validation")
plt.gca().set(xlabel='C penalty', ylabel='Mean square error')
plt.legend();
plt.show()
```

This gives us the results on Figure 7 below. By focusing on values where changes happen (i.e. narrower range of values for C), we obtain results on Figure 8:

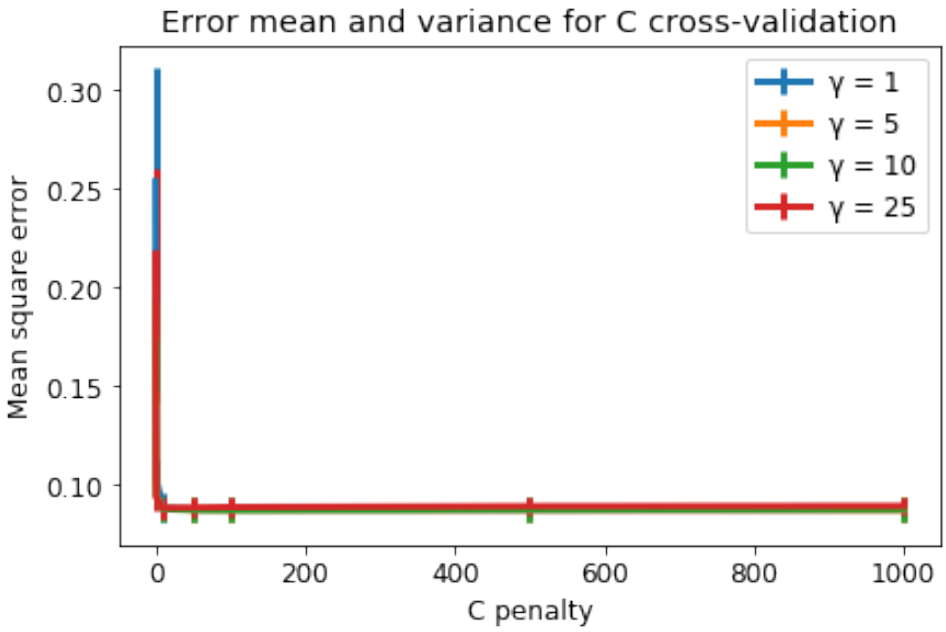

Figure 7

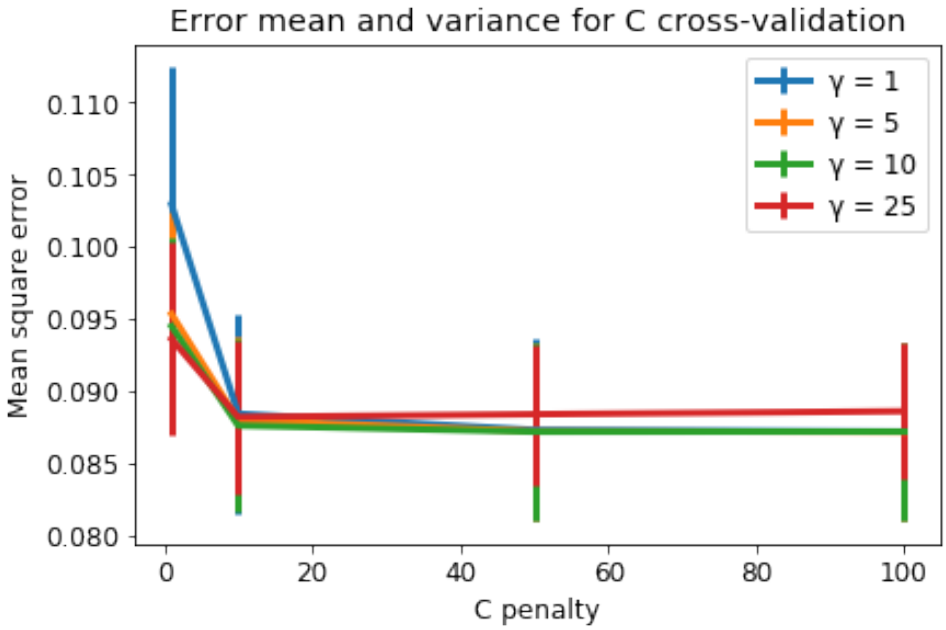

Figure 8

From this Figure 8, we can determine that the most suitable value for C to minimise MSE while avoiding over-fitting is $C = 10$.

We can now train a kernalised ridge regression model for a wide range of values γ and plot its predictions using the *kernalisedRidgeRegression* function created in part (i):

```
kernalisedRidgeRegression(X, y, C=10, gaussianKernels=kernels)
```

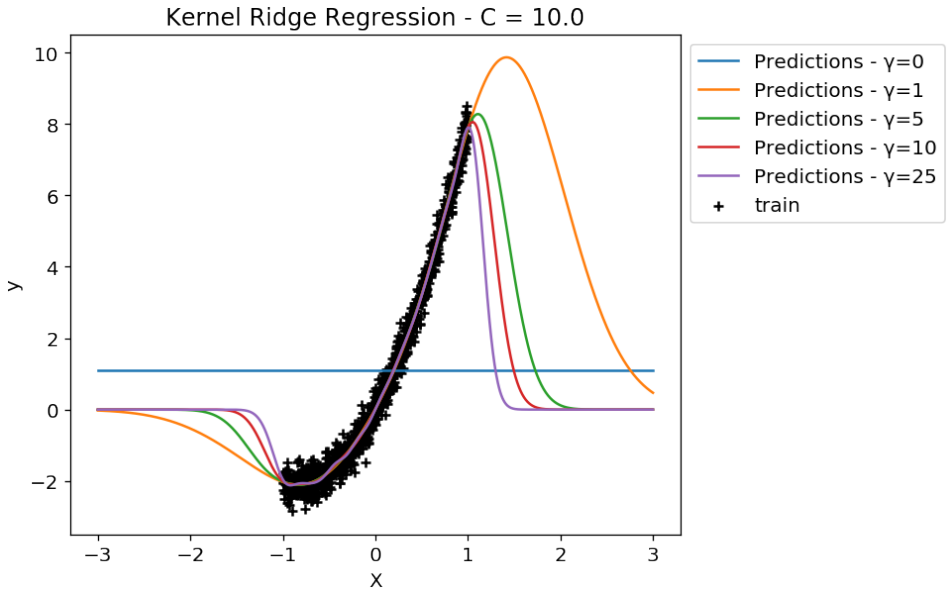

Figure 9

"Parameter γ controls how quickly $K(x^{(i)}, x)$ decreases as distance between $x^{(i)}$ and x grows." (Kernel Trick PDF, Slide 3/17). As we can see on the Figure 9 above, bigger is γ, faster the prediction curve gets back to 0. This is due to the fact that $K(x^{(i)}, x)$ decreases very quickly when γ is great. The contrary occurs when γ has a small value.

This differs from kNN model in which the nearest training data points were still considered even for values outside the scope of the training data, thus converging to some average value. In this kernalised ridge regression model, the behavior is different as the model considers that no training data point belongs to "the set of k points closest to x", thus making the model predictions converging to zero.

(c) As we did C in the previous question, we can cross-validate γ for our kNN model by comparing a wide range of values $\gamma \in \{0, 1, 5, 10, 25, 50, 100\}$:

```
std_error = []
mean_error = []
tested_kernels = []

kernels = [
    # [0, gaussian_kernel0],
    # [1, gaussian_kernel1],
    [5, gaussian_kernel5],
    [10, gaussian_kernel10],
    [25, gaussian_kernel25],
    [50, gaussian_kernel50],
    [100, gaussian_kernel100]
]

for gaussianKernel in kernels:
    kf = KFold(n_splits=5)
    temp = []
```

```
    for train, test in kf.split(X):
        model = KNeighborsRegressor(n_neighbors=X[train].size,
        ↪  weights=gaussianKernel[1])
        model.fit(X[train], y[train])
        ypred = model.predict(X[test])
        temp.append(mean_squared_error(y[test], ypred))

    mean_error.append(np.array(temp).mean())
    std_error.append(np.array(temp).std())
    tested_kernels.append(gaussianKernel[0])

plt.errorbar(tested_kernels, mean_error, yerr=std_error, linewidth=3)
plt.title("Error mean and variance for gamma cross-validation")
plt.gca().set(xlabel='Value of gamma', ylabel='Mean square error')
plt.show()
```

After excluding extreme values of γ which gives very high MSE and/or standard deviation (i.e. $\gamma \in \{0, 1\}$), we obtain the following graph:

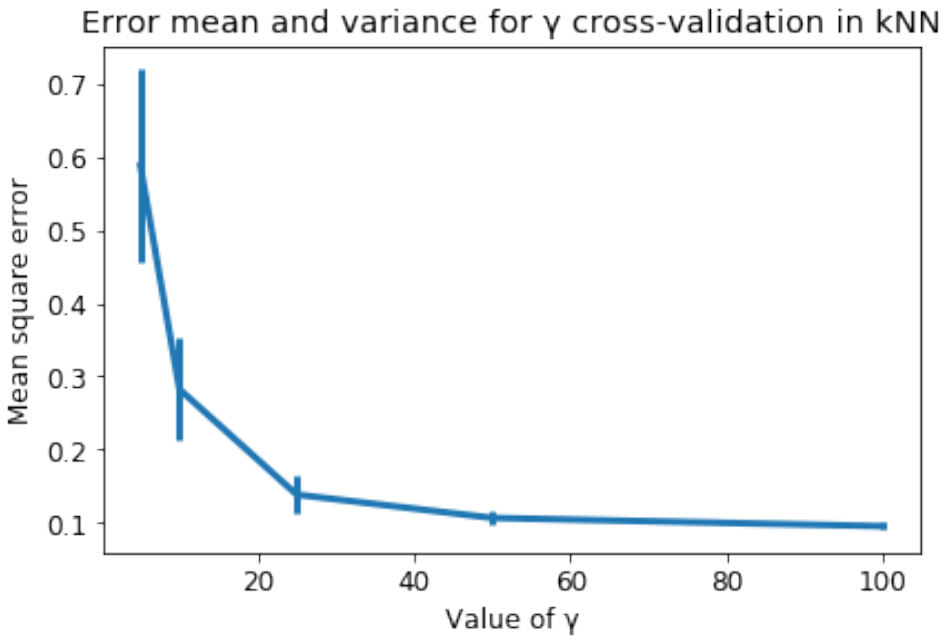

Figure 10

Based on these result, we can argue that $\gamma = 50$ is the most appropriate value as it is the smallest value that minimises MSE the most.

Regarding our kernalised ridge regression model, we already cross-validated α (i.e. $\alpha = 1/(2C)$) in the previous question. We can therefore apply a similar cross-validation for γ in this model:

```
for gaussianKernel in kernels:
    model = KernelRidge(alpha=1.0/C, kernel='rbf',
    ↪  gamma=gaussianKernel[0])
```

```
    scores = cross_val_score(model, X, y, cv=5,
    ↪  scoring='neg_mean_squared_error')
    mean_error.append(-np.array(scores).mean())
    std_error.append(np.array(scores).std())
    tested_kernels.append(gaussianKernel[0])

plt.errorbar(tested_kernels, mean_error, yerr=std_error, linewidth=3)
plt.title("Error mean and variance for gamma cross-validation")
plt.gca().set(xlabel='Value of gamma', ylabel='Mean square error')
plt.show()
```

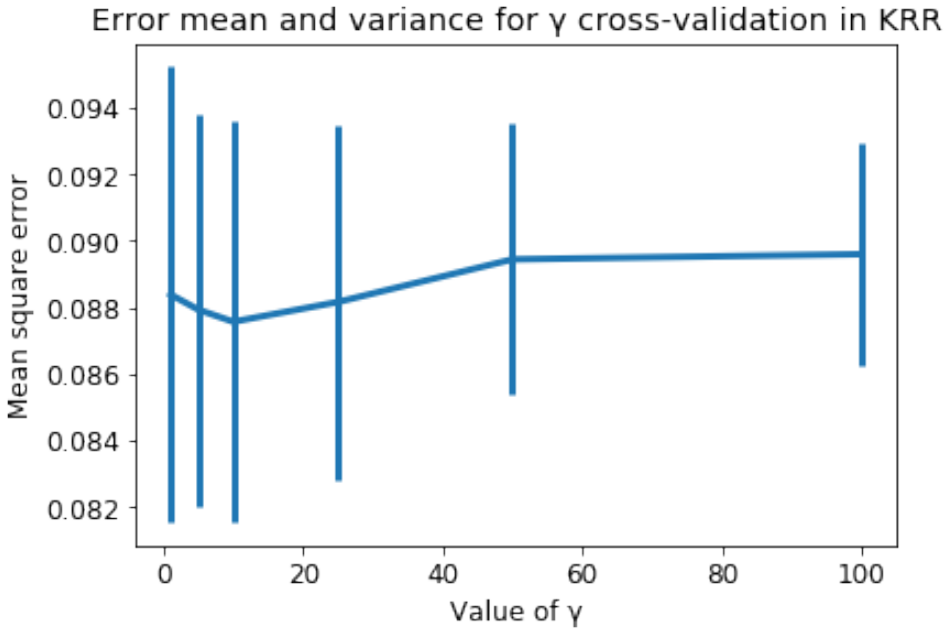

Figure 11

We can observe on the Figure 11 above that the value of γ that minimises MSE the most is $\gamma = 10$. However, the standard deviation (i.e. spreadness of predictions) is more important and $\gamma = 50$ seems to minimise it. Depending what we want to give more importance to, it would be arguable to opt either for $\gamma = 10$ or $\gamma = 50$. In this case, we will keep $\gamma = 10$ to minimise MSE.

We can now plot these two kNN and KRR models with optimised hyperparameter values:

```
knnGaussianKernel(X, y, k=X.size, gaussianKernels=[[50,
↪  gaussian_kernel50]])
kernalisedRidgeRegression(X, y, C=10, gaussianKernels=[[10,
↪  gaussian_kernel10]])
```

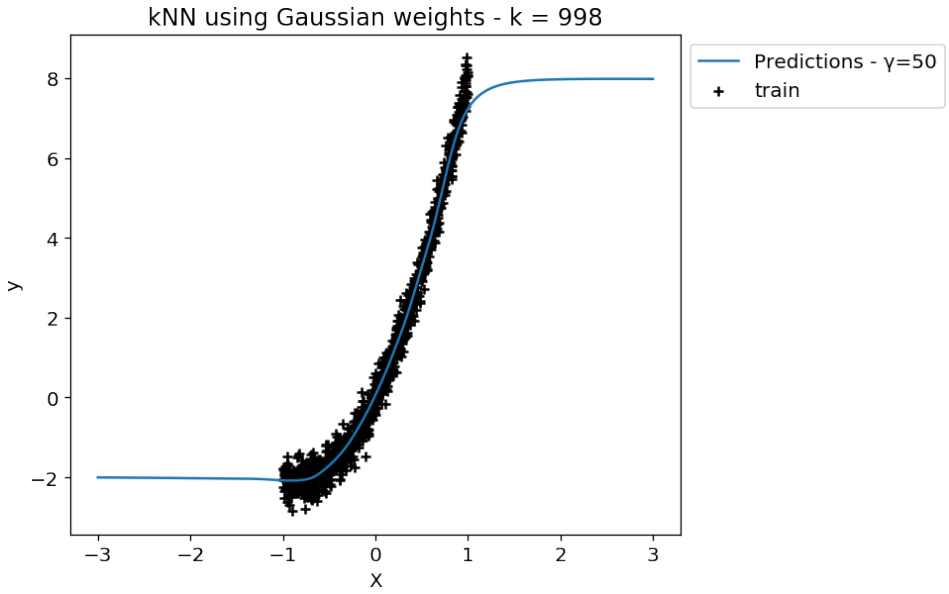

Figure 12

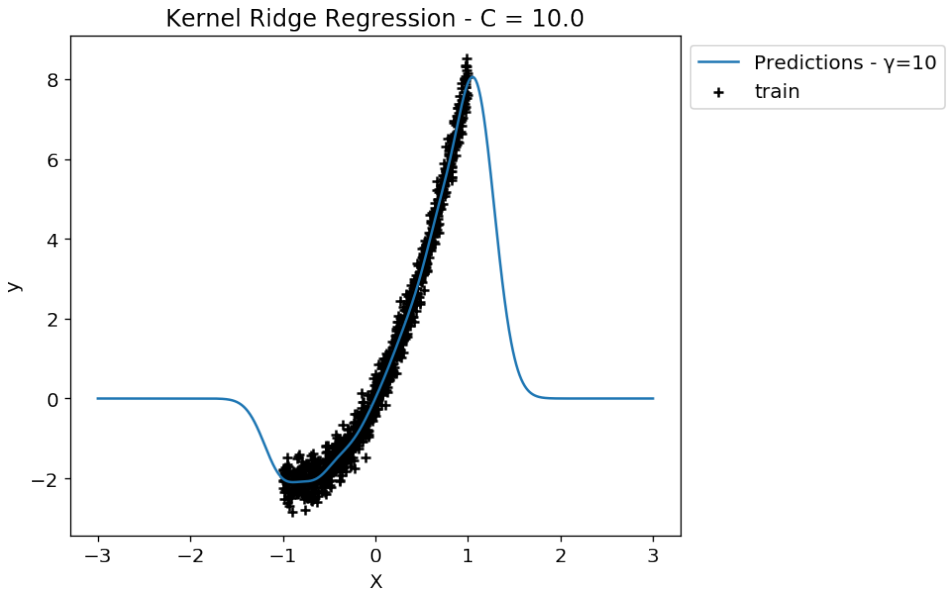

Figure 13

It seems that kNN model (Figure 12) provides more accurate predictions for $x < -1$ as the line to which it converges is in accordance with the behaviour of the training data. However, predictions for $x \approx 1$ seem more accurate in kernel ridge regression (Figure 13) and, finally, no model seems to provide accurate predictions for $x > 1$.

Actually, such models can be very good at making predictions within the scope of the training data but are not meant to go beyond this scope. We can also note that predictions of both models are very similar in the range of the training data (i.e. $x \in [-1, 1]$).

A Appendix

A.1 Python Code

```python
#!/usr/bin/env python
# coding: utf-8

# CS7CS4/CSU44061 Machine Learning
# Week 6 Assignment
# Boris Flesch (20300025)
#
# Downloaded dataset
# id:23-69-115

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

def plotData(X, y):
    plt.scatter(X, y, c='r', marker='+', label="Training data")
    plt.gca().set(title="Training data visualisation", xlabel="X",
    ↪  ylabel="y")
    plt.legend(bbox_to_anchor=(1, 1), loc='upper left')
    plt.show()

# (i)
X = np.array([-1, 0, 1]).reshape(-1, 1)
y = np.array([0, 1, 0])
plotData(X,y)

# (i)(a)
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)
```

```
44 def gaussian_kernel25(distances):
45     weights = np.exp(-25*(distances**2))
46     return weights/np.sum(weights)
47 def gaussian_kernel50(distances):
48     weights = np.exp(-50*(distances**2))
49     return weights/np.sum(weights)
50 def gaussian_kernel100(distances):
51     weights = np.exp(-100*(distances**2))
52     return weights/np.sum(weights)
53 
54 def knnGaussianKernel(X, y, k, gaussianKernels):
55     plt.figure(num=None, figsize=(8, 5), dpi=120)
56     plt.rc('font', size=12)
57 
58     if (X.size < 10):
59         plt.scatter(X, y, color='black', marker='+', s=10, linewidth=15,
        ↪  label="train")
60     else:
61         plt.scatter(X, y, color='black', marker='+', label="train")
62 
63     Xtest = np.linspace(-3.0, 3.0, num=1000).reshape(-1, 1)
64 
65     for gaussianKernel in gaussianKernels:
66         model = KNeighborsRegressor(n_neighbors=k,
        ↪  weights=gaussianKernel[1]).fit(X, y)
67         ypred = model.predict(Xtest)
68 
69         plt.plot(Xtest, ypred, label="Predictions -
        ↪  gamma=%d"%gaussianKernel[0])
70 
71     plt.xlabel("X"); plt.ylabel("y")
72     plt.legend(bbox_to_anchor=(1, 1), loc='upper left')
73     plt.title("kNN using Gaussian weights - k = %d" % k)
74     plt.show()
75 
76 kernels = [
77     [0, gaussian_kernel0],
78     [1, gaussian_kernel1],
79     [5, gaussian_kernel5],
80     [10, gaussian_kernel10],
81     [25, gaussian_kernel25]
82 ]
83 knnGaussianKernel(X, y, k=3, gaussianKernels=kernels)
84 
85 # (i)(c)
86 def kernalisedRidgeRegression(X, y, C, gaussianKernels,
   ↪  printCoeff=False):
87     plt.figure(num=None, figsize=(8, 5), dpi=120)
```

```
88      plt.rc('font', size=12);
        ↪ plt.rcParams['figure.constrained_layout.use'] = True
89
90      if (X.size < 10):
91          plt.scatter(X, y, color='black', marker='+', s=10, linewidth=15,
            ↪ label="train")
92      else:
93          plt.scatter(X, y, color='black', marker='+', label="train")
94
95      for gaussianKernel in gaussianKernels:
96          model = KernelRidge(alpha=1.0/C, kernel='rbf',
            ↪ gamma=gaussianKernel[0]).fit(X, y)
97          Xtest = np.linspace(-3.0, 3.0, num=1000).reshape(-1, 1)
98          ypred = model.predict(Xtest)
99          plt.plot(Xtest, ypred, label="Predictions -
            ↪ gamma=%d"%gaussianKernel[0])
100         if (printCoeff):
101             print("Kernel Ridge Regression - C = %d, gamma=%d" % (C,
                ↪ gaussianKernel[0]))
102             print("theta =", model.dual_coef_)
103             print("--")
104
105     plt.xlabel("X"); plt.ylabel("y")
106     plt.legend(bbox_to_anchor=(1, 1), loc='upper left')
107     plt.title("Kernel Ridge Regression - C = %.1f" % (C))
108     plt.show()
109
110 kernalisedRidgeRegression(X, y, C=0.1, gaussianKernels=kernels,
    ↪ printCoeff=True)
111 kernalisedRidgeRegression(X, y, C=1, gaussianKernels=kernels,
    ↪ printCoeff=True)
112 kernalisedRidgeRegression(X, y, C=100, gaussianKernels=kernels,
    ↪ printCoeff=True)
113
114
115 # (ii)
116 def readData(filepath):
117     df = pd.read_csv(filepath, comment="#")
118     X = df.iloc[:,0]
119     y = df.iloc[:,1]
120     return np.array(X).reshape(-1, 1), np.array(y)
121
122 X,y = readData("week6.csv")
123 plotData(X,y)
124
125
126 # (ii)(a)
127 kernels = [
```

```
    [0, gaussian_kernel0],
    [1, gaussian_kernel1],
    [5, gaussian_kernel5],
    [10, gaussian_kernel10],
    [25, gaussian_kernel25]
]
knnGaussianKernel(X, y, k=X.size, gaussianKernels=kernels)
print("Output y average: %.3f" % y.mean())

Xm1_y = []
Xp1_y = []
for i in range(X.size):
    if X[i] == -1:
        Xm1_y.append(y[i])
    elif X[i] == 1:
        Xp1_y.append(y[i])
print("Output y average for training data points where x=-1: %.3f"%np.mean(Xm1_y))
print("Output y average for training data points where x=1: %.3f"%np.mean(Xp1_y))


# (ii)(b)
C_range = [
    # 0.1,
    1,
    10,
    50,
    100,
    # 500,
    # 1000
]
kernels = [
    #[0, gaussian_kernel0],
    [1, gaussian_kernel1],
    [5, gaussian_kernel5],
    [10, gaussian_kernel10],
    [25, gaussian_kernel25]
]

for gaussianKernel in kernels:
    mean_error = []
    std_error = []
    for C in C_range:
        model = KernelRidge(alpha=1.0/C, kernel='rbf', gamma=gaussianKernel[0])
        scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')
```

```
172             mean_error.append(-np.array(scores).mean())
173             std_error.append(np.array(scores).std())
174 
175         plt.errorbar(C_range, mean_error, yerr=std_error, linewidth=3,
            ↪  label="gamma = %d"%gaussianKernel[0])
176 
177     plt.title("Error mean and variance for C cross-validation")
178     plt.gca().set(xlabel='C penalty', ylabel='Mean square error')
179     plt.legend()
180     plt.show()
181 
182     kernels = [
183         [0, gaussian_kernel0],
184         [1, gaussian_kernel1],
185         [5, gaussian_kernel5],
186         [10, gaussian_kernel10],
187         [25, gaussian_kernel25]
188     ]
189     kernalisedRidgeRegression(X, y, C=100, gaussianKernels=kernels)
190 
191     # (ii)(c)
192     std_error = []
193     mean_error = []
194     tested_kernels = []
195     kernels = [
196         # [0, gaussian_kernel0],
197         # [1, gaussian_kernel1],
198         [5, gaussian_kernel5],
199         [10, gaussian_kernel10],
200         [25, gaussian_kernel25],
201         [50, gaussian_kernel50],
202         [100, gaussian_kernel100]
203     ]
204 
205     for gaussianKernel in kernels:
206 
207         kf = KFold(n_splits=5)
208         temp = []
209         for train, test in kf.split(X):
210             model = KNeighborsRegressor(n_neighbors=X[train].size,
                ↪  weights=gaussianKernel[1])
211             model.fit(X[train], y[train])
212             ypred = model.predict(X[test])
213             temp.append(mean_squared_error(y[test], ypred))
214 
215         mean_error.append(np.array(temp).mean())
216         std_error.append(np.array(temp).std())
217         tested_kernels.append(gaussianKernel[0])
```

```
218
219 plt.errorbar(tested_kernels, mean_error, yerr=std_error, linewidth=3)
220 plt.title("Error mean and variance for gamma cross-validation in kNN")
221 plt.gca().set(xlabel='Value of gamma', ylabel='Mean square error')
222 plt.show()
223
224 C = 10
225 std_error = []
226 mean_error = []
227 tested_kernels = []
228 kernels = [
229     # [0, gaussian_kernel0],
230     [1, gaussian_kernel1],
231     [5, gaussian_kernel5],
232     [10, gaussian_kernel10],
233     [25, gaussian_kernel25],
234     [50, gaussian_kernel50],
235     [100, gaussian_kernel100],
236     # [500, gaussian_kernel500]
237 ]
238
239 for gaussianKernel in kernels:
240     model = KernelRidge(alpha=1.0/C, kernel='rbf',
        ↪ gamma=gaussianKernel[0])
241     scores = cross_val_score(model, X, y, cv=5,
        ↪ scoring='neg_mean_squared_error')
242     mean_error.append(-np.array(scores).mean())
243     std_error.append(np.array(scores).std())
244     tested_kernels.append(gaussianKernel[0])
245
246 plt.errorbar(tested_kernels, mean_error, yerr=std_error, linewidth=3)
247 plt.title("Error mean and variance for gamma cross-validation in KRR")
248 plt.gca().set(xlabel='Value of gamma', ylabel='Mean square error')
249 plt.show()
250
251 knnGaussianKernel(X, y, k=X.size, gaussianKernels=[[50,
    ↪ gaussian_kernel50]])
252 kernalisedRidgeRegression(X, y, C=10, gaussianKernels=[[10,
    ↪ gaussian_kernel10]])
```