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

CS7CS4/CSU44061 - Machine Learning
Assignment 5:

Student Name:
Hugh Jordan

Student Number:
16321743

Assignment 5

(i) Small Dataset

(a) The dummy training data was visualised using Matplotlib's scatter function. This is shown below in Figure 1. It is evident from Figure 1 that there is no clear relationship between the input feature and the target value. This is because there are only three data points in the dataset.

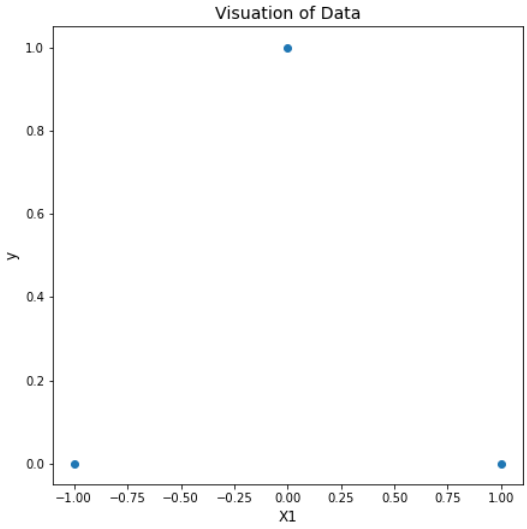

Figure 1. Visualisation of Data

Sklearn's KNeighborsRegressor function was used to create kNN predictions for this data. The parameter k was assigned a value equal to the number of training data points, where k is the number of neighbours used to classify the input feature. Furthermore, a Gaussian kernel was used to weigh neighbours based on their distance from the input feature. The predictions for a range of values for the parameter γ are shown in Figure 2. It is evident from Figure 2 that as the parameter γ increases, the kernel decreases more quickly with distance. This means that the predictions tend to be less smooth and snap to the nearest training data point.

(b) The Gaussian kernel weighs neighbours based on their distance from the input feature. The kernel assigned more weight to training data points that are closer to the input feature and less weight to training data points that are further away from the input feature. It is evident from Figure 2 that as the parameter γ increases, the kernel decreases more rapidly with distance. This means that the predictions tend to be less smooth and snap to the nearest training data point. When γ = 25, the predictions are close to 1.0 for all input feature values within the range (-0.5, 0.5) because the predictions snap to the nearest data point which is [1.0, 1.0].

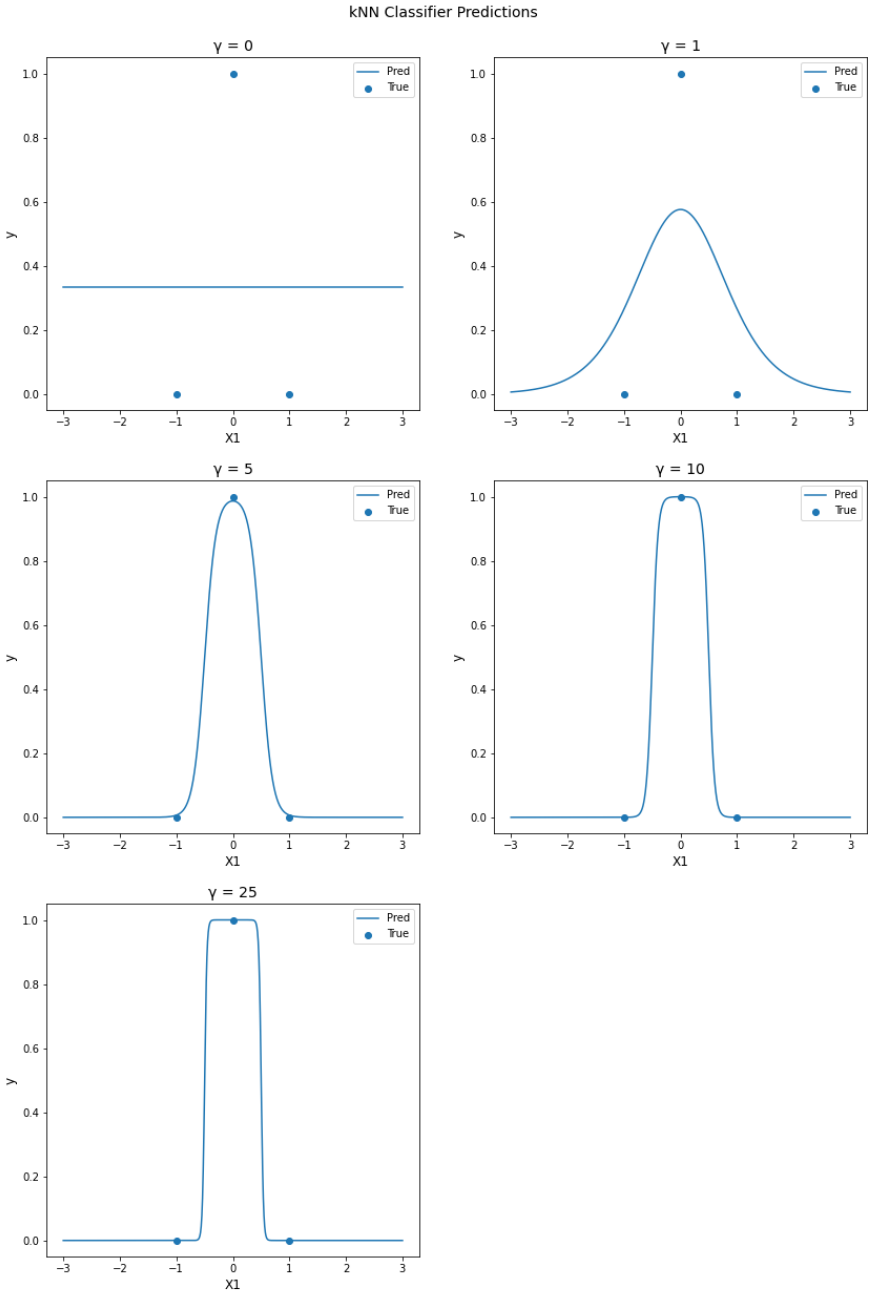

Figure 2. kNN Predictions for range of parameter γ values.

(c) Sklearn's KernelRidge function was used to create kernelised ridge regression predictions for the data. The parameter C was assigned a value of 1.0 initially, where C is the penalty assigned to the cost function. Furthermore, a Gaussian kernel was used to weigh neighbours based on their distance from the input feature. The predictions for a range parameter γ values are shown in Figure 3, and the corresponding parameter values are shown in Table 1. The relationship between the kernelised ridge regression predictions and the parameter values can be described by the equation below.

$$\hat{y} = \theta_0 + w_1 K\left(x^{(1)}, x\right) + w_2 K\left(x^{(2)}, x\right) + w_3 K\left(x^{(3)}, x\right), \text{where } w_i \approx \theta_i y^{(i)}$$

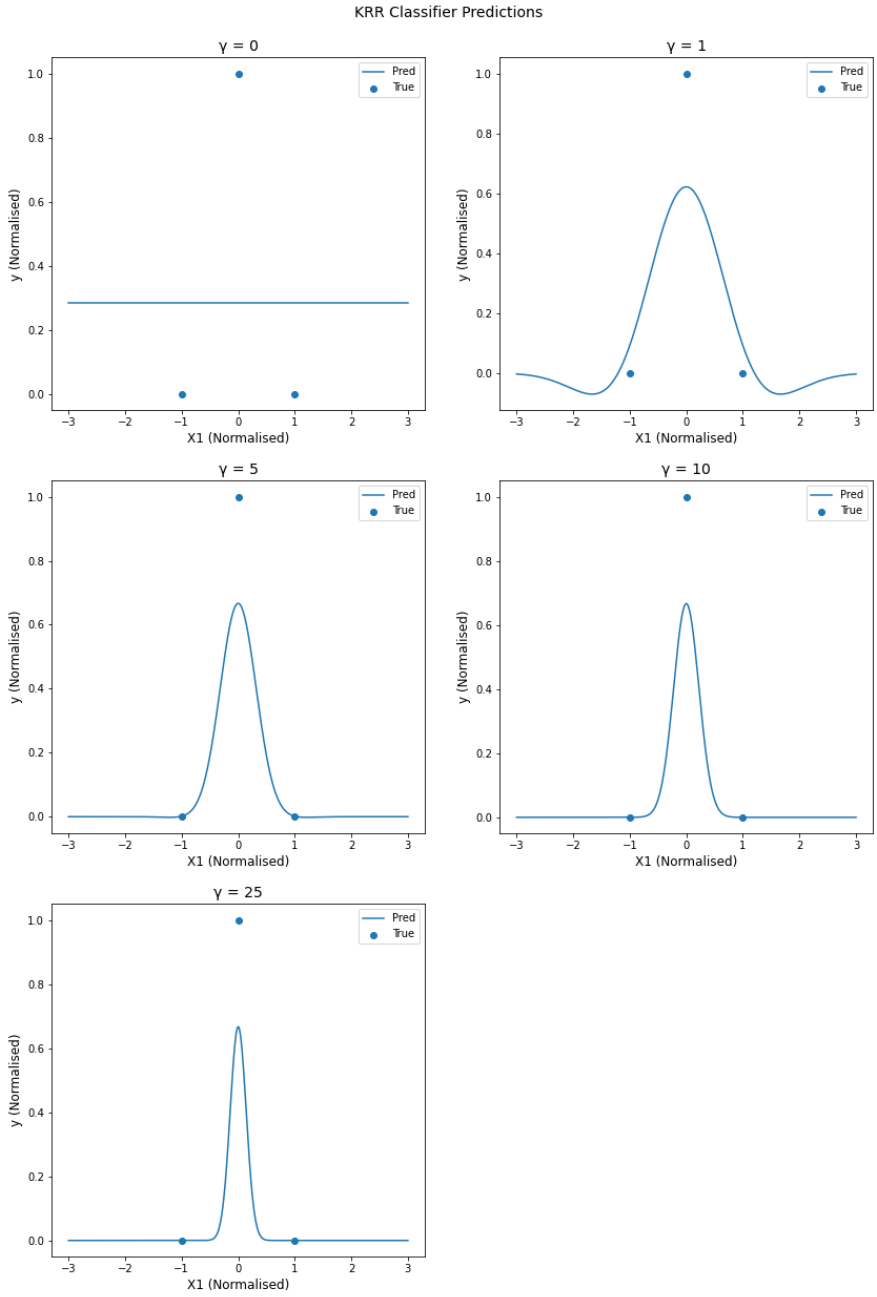

Figure 3. KRR Predictions for range of parameter γ values.

γ	θ_1	θ_2	θ_3
0	-0.57142857	1.42857143	-0.57142857
1	-0.18331618	0.75658434	-0.18331618
5	-0.00299476	0.66669357	-0.00299476
10	-2.01777466e-05	6.66666668e-01	-2.01777466e-05
25	-6.17241950e-12	6.66666667e-01	-6.17241950e-12

Table 1. KRR Parameter Values for range of parameter γ values.

The parameter γ was then assigned a value of 5, and the predictions for a range of parameter C values were generated. The predictions are shown in Figure 4, and the corresponding parameter values are shown in Table 2.

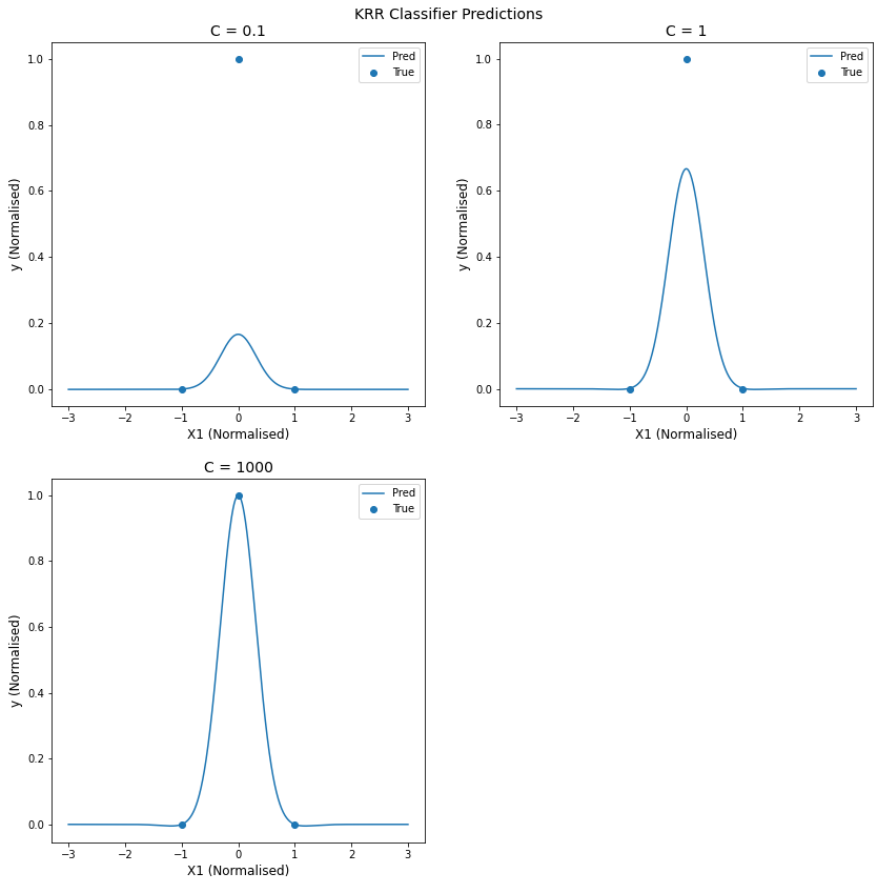

Figure 4. KRR Predictions for range of parameter C values.

C	θ_1	θ_2	θ_3
0.1	-0.00018717	0.16666709	-0.00018717
1	-0.00299476	0.66669357	-0.00299476
1000	-0.00673182	0.99959092	-0.00673182

Table 2. KRR Parameter Values for range of parameter C values.

(d) It is evident from Figure 3 that as the parameter γ increases, the predictions tend to be less smooth and snap to the nearest training data point. This is because the kernel decreases more rapidly as the parameter γ increases.

It is also evident from this Table 1 that as the parameter γ increases, the parameters θ_1 and θ_3 approach 0.0 whereas the parameter θ_2 approaches 0.66. This means that as the parameter γ increases, the contribution the first and third data points have on the predictions becomes negligible, and only the second data point contributes to the predictions. This is also visible in Figure 3. It is evident from Figure 3 that the maximum value of the predictions is 0.66. This is

because the weight assigned to the parameter θ_3 is equal to 0.66 and only the second data point has an effect on the predictions.

It is evident from Figure 4 that as the parameter C increases, the weight assigned to given to each of the data points increases. This is because kernelised ridge regression uses L2 regularisation. L2 regularisation adds the sum of the square of the parameter w_i as a penalty to the loss function to reduce overfitting This type of regularisation forces weights to be small. However, it does not force them to be zero. When C = 0.1, the weight assigned to the second data point is small in size, however, the weight is not zero.

It is also evident from Table 2 that as parameter C increases, the weight assigned to each of the parameters increases. This means that as the parameter C increases, the contribution each data point has on the predictions increases. However, the parameters θ_1 and θ_3 were very small when compared to the parameter θ_2. This means that the contribution the first and third data points have on the predictions is much lower when compared to the contribution the second data point has on the predictions. This is visible in Figure 4. It is evident from Figure 4, that when C = 1000, the maximum value of the predictions is almost 1.0. This is because the weight assigned to the parameter θ_3 is equal to 0.9995, whereas the weight assigned to the other two parameters -0.0067.

(ii) Large Dataset

(a) The downloaded data (# id:19-152-38) was visualised using Matplotlib's scatter function. This is shown below in Figure 5. It is evident from Figure 5 that there is a clear relationship between the input feature and the target value.

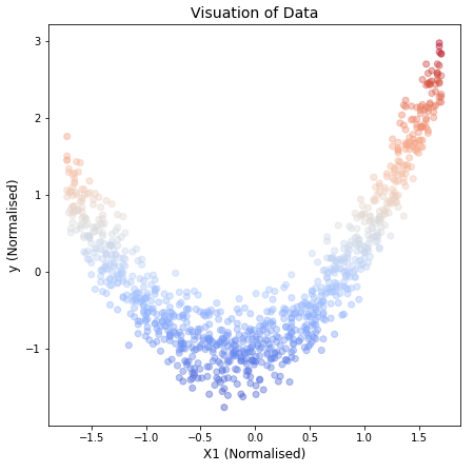

Figure 5. Visualisation of Data

Sklearn's KNeighborsRegressor function was used to create kNN predictions for this data. The parameter k was assigned a value equal to the number of training data points, and a Gaussian kernel was used to weigh neighbours based on their distance from the input feature. The predictions for a range of values for the parameter γ are shown in Figure 6. It is evident from Figure 6 that when γ = 0,

the model underfits the data. However, as the parameter γ increases, the model begins to fit the data, up until the point where γ = 10. The model overfits the data after this point. When γ = 25, the model starts fitting the noise in the data. The predictions are less smooth, and they start snapping to the nearest training data point. Finally, as the parameter γ increases, the predictions outside of the training set range start snapping to the nearest data point in the training set, and they plateau.

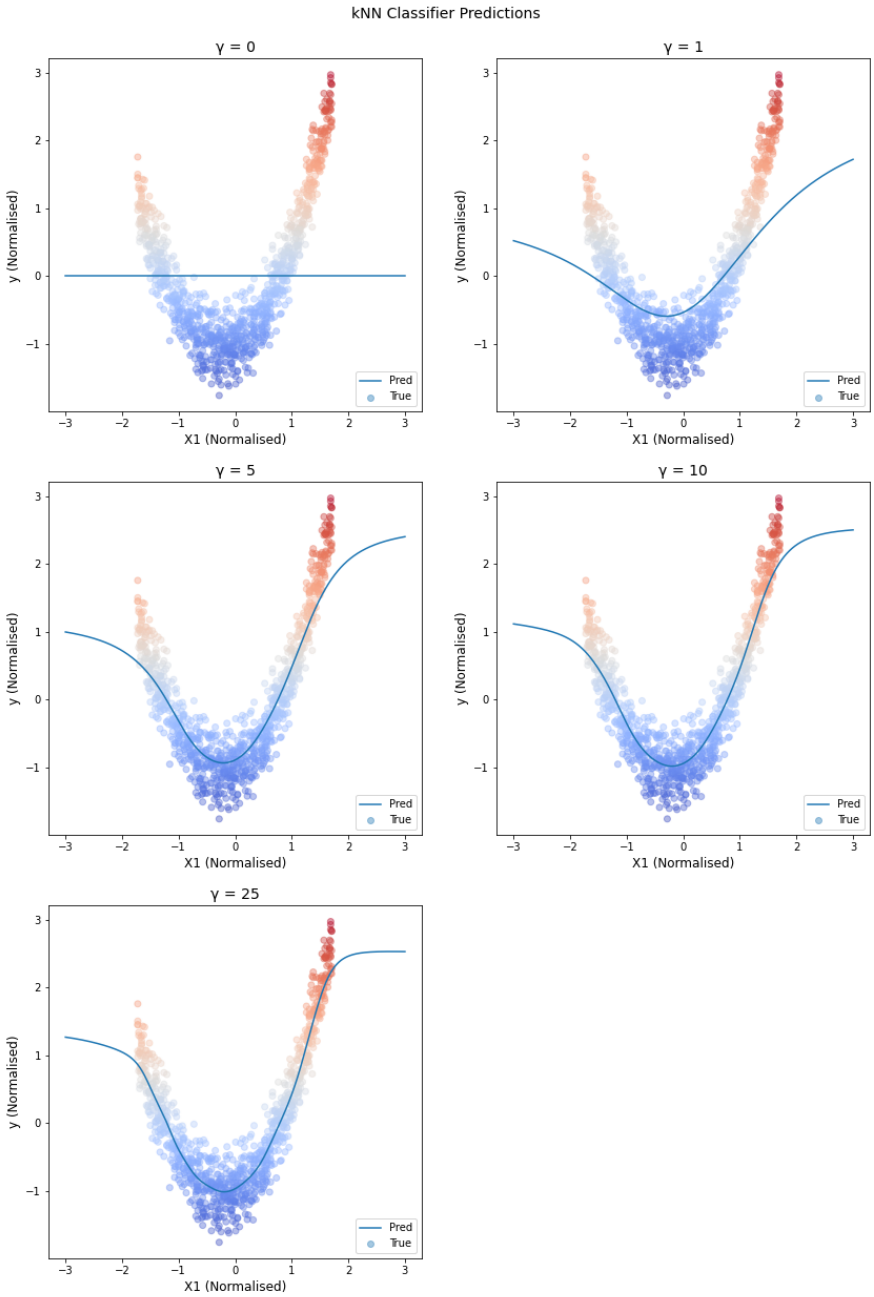

Figure 6. kNN Predictions for range of parameter γ values.

(b) Sklearn's KernelRidge function was used to create kernelised ridge regression predictions for the data. The parameter C was assigned a value of 1 initially, and a Gaussian kernel was used to weigh neighbours based on their distance from the input feature. It is evident from Figure 6 that when $\gamma = 0$, the model underfits the data. However, as the parameter γ increases, the model begins to fit the data, up until the point where $\gamma = 1$. The model begins to overfit the data after this point. When $\gamma = 5$, the model starts fitting the noise in the data. This effect is worse when $\gamma = 25$, and the predictions are less smooth and snap to the nearest training data point. This is because the kernel decreases more rapidly as the parameter γ increases.

Finally, the predictions begin decreasing toward the mean of the training set outside of the training set range. The predictions decrease very gradually when $\gamma = 1$. However, the predictions decrease much more rapidly as the parameter γ increases. When $\gamma = 25$, the predictions are all close to the mean of the training set for all input feature values outside of the training set's range.

The parameter γ was then assigned a value of 1 and predictions for a range of values for the parameter C were generated. This value for the parameter was chosen γ using 5-Fold cross-validation in (c). The predictions are shown in Figure 8. It is evident from Figure 8 that when $C = 0.1$, the model slightly underfits the data. However, as the parameter C increases, the model begins to fit the data, up until a point after $C = 1$. The model begins to overfits the data after this point and when $C = 1000$, the model starts fitting the noise in the data. Finally, as the parameter C increases, the predictions outside of the training set's range increase until a certain point. After this point, the predictions would begin decreasing again toward the mean of the training set. This was observed in Figure 7.

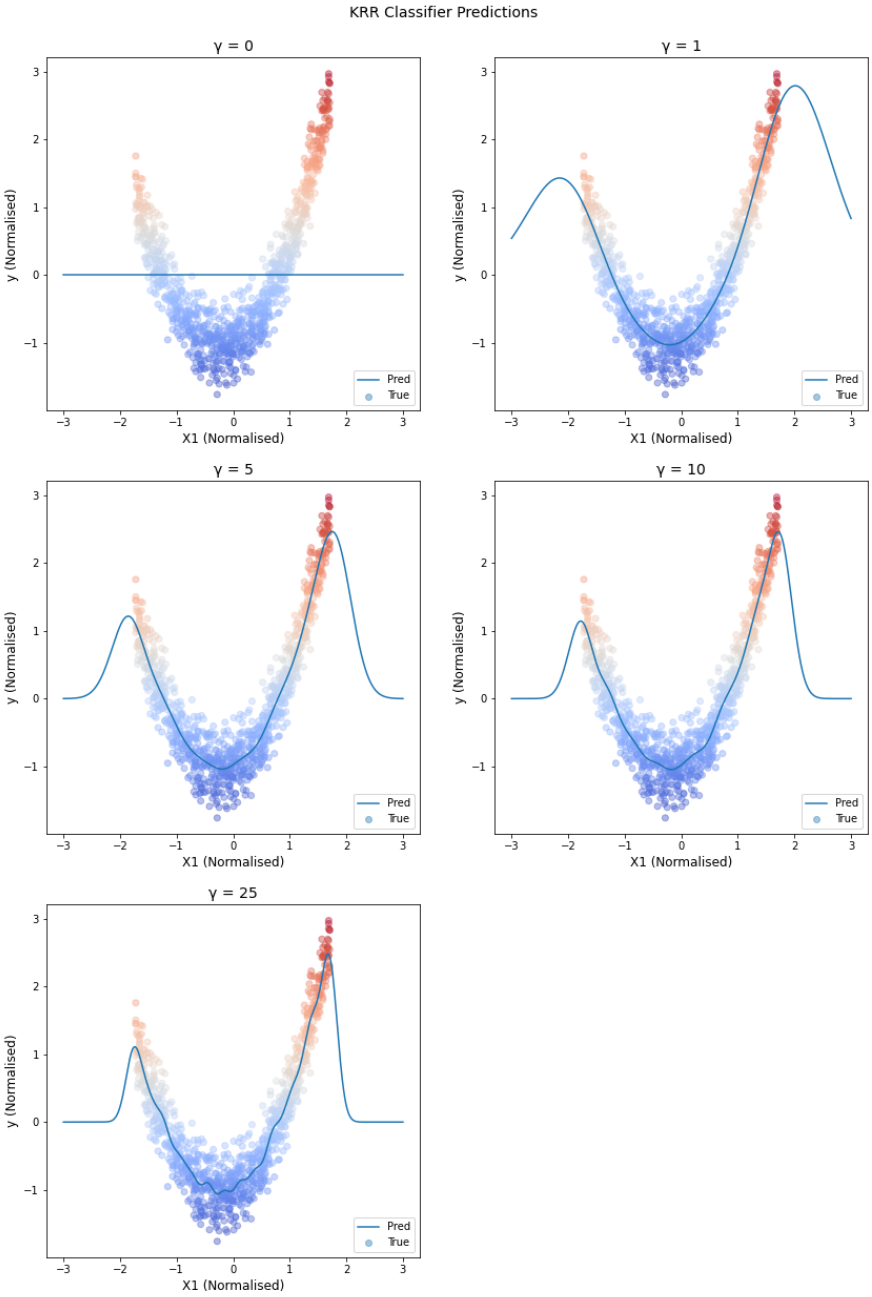

Figure 7. KRR Predictions for range of parameter γ values.

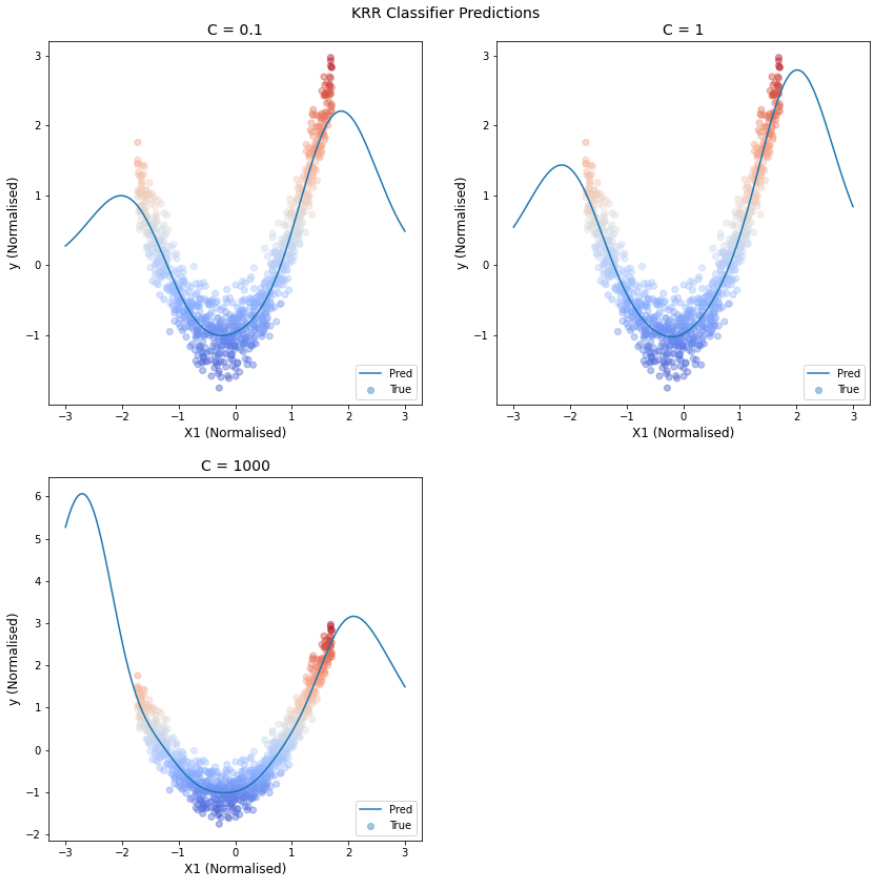

Figure 8. KRR Predictions for range of parameter C values.

(c) 5-Fold cross-validation was used to select the value of parameter γ for the kNN classifier. The segment of code used to implement this is shown in Figure 9.1. It is evident from Figure 3.1 that the program loops through a range of γ values and calculates the mean square error of each classifier. The program calculates these metrics using K-Folds cross-validator function. This class provides the train/test indices for the training and test sets. The program then calculates the mean and standard deviation of the mean square error.

```
kf = KFold(n_splits=5)
mse = []; mse_std = []
parameters = [0, 1, 5, 10, 25]
for itr, parameter_knn in enumerate(parameters):
    error = []
    knn = KNeighborsRegressor(weights=gaussian_kernel)
    for train, test in kf.split(X):
        knn.set_params(n_neighbors=len(y[test]))
        knn.fit(X[train], y[train].ravel())
        y_knn = knn.predict(X[test])
        error.append(mean_squared_error(y[test], y_knn))
    mse.append(np.array(error).mean())
    mse_std.append(np.array(error).std())
```

Figure 9.1. 5-Fold Cross-Validation Implementation

The cross-validation is shown in Figure 9.2. It is evident from Figure 9.2 that the performance of the model improves as parameter γ increases. The performance of the model improves rapidly until the point where γ = 10, and then the

performance improves more gradually. Therefore, a value of 10 was chosen as the optimal value. This is because the lowest value of parameter γ that maximises the performance of the classifier should be chosen to reduce overfitting.

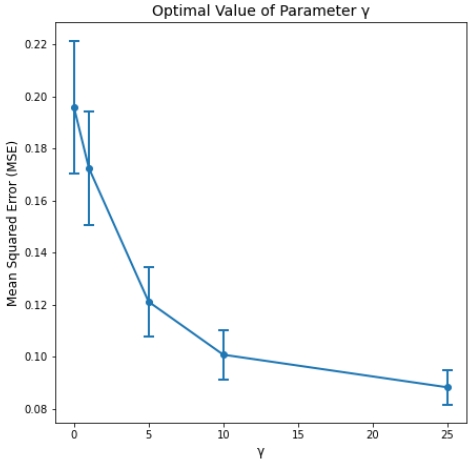

Figure 9.2. Cross-Validation of Parameter γ for kNN classifier

Finally, the optimised kNN classifier was used to generate predictions. These predictions are shown in Figure 10. It is evident from Figure 10 that the classifier was able to capture the relationship between the training data and the target values. However, the predictions plateaued outside of the training data's range.

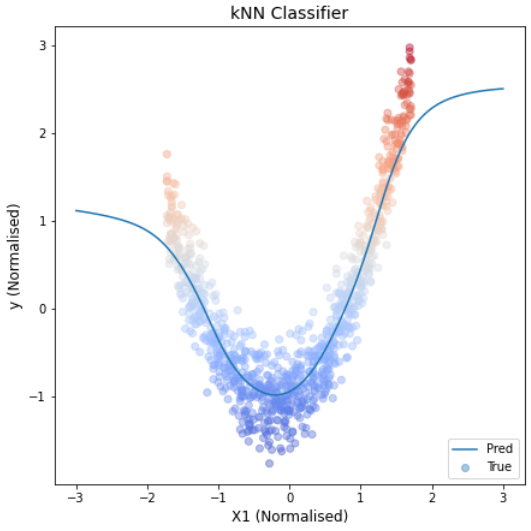

Figure 10. Optimised kNN Classifier

5-Fold cross-validation was then used to select the value of parameter γ for the KRR classifier. This is shown in Figure 11. It is evident from Figure 11 that the performance of the model improves as parameter γ increases, up until the point where γ = 1. The performance of the model begins to plateaued after this point. Therefore, a value of 1 was chosen as the optimal value. This is because the lowest value of parameter γ that maximises the performance of the classifier should be chosen to reduce overfitting.

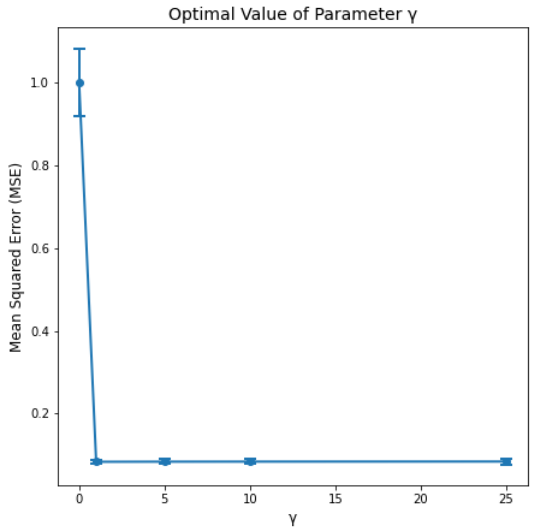

Figure 11. Cross-Validation of Parameter γ for kNN classifier

5-Fold cross-validation was then used to select the value of parameter C for the KRR classifier. This is shown in Figure 12. It is evident from Figure 12 that the performance of the model improves as parameter C increases, up until the point where C = 5. The performance of the model begins to plateaued after this point. Therefore, a value of 5 was chosen as the optimal value. This is because the lowest value of parameter C that maximises the performance of the classifier should be chosen to reduce overfitting.

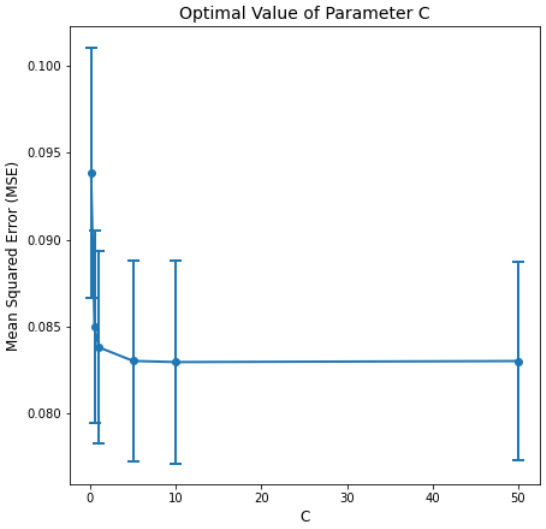

Figure 12. Cross-Validation of Parameter C for kNN classifier

Finally, the optimised KRR classifier was used to generate predictions. These predictions are shown in Figure 13. It is evident from Figure 13 that the classifier was able to capture the relationship between the training data and the target values. The predictions increase up until a certain point outside out of the training data's range before they begin decreasing.

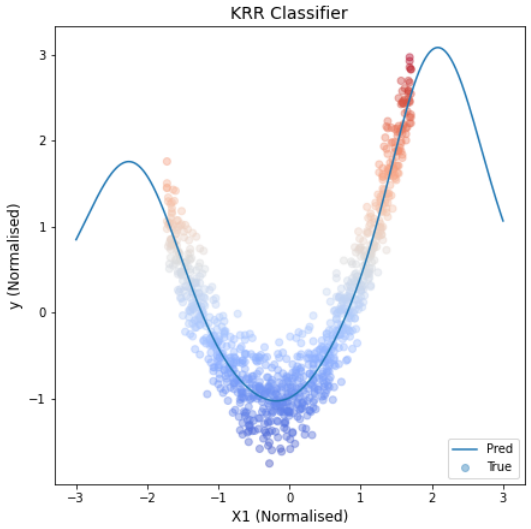

Figure 13. Optimised KRR Classifier

The kNN and KRR classifiers were evaluated by calculating the mean square error of the classifiers using 5-Fold cross-validation. The two classifiers were also evaluated against a baseline classifier that always predicts the mean of the training data. This baseline classifier was created using the DummyRegressor function. This evaluation is shown in Table 3. It is evident from Table 3 that both the kNN and KRR classifiers performed significantly better than the baseline classifier because the mean square error and standard deviation are both much lower. This is because the two classifiers were able to capture the relationship between the training data and the target values. However, the mean square error and standard deviation of the KRR classifier is slightly lower than that of the kNN classifier. This is because the KRR classifier was able to capture the relationship slightly better than the kNN model. Furthermore, the KRR model generates reasonable predictions outside of the training data's range, whereas the kNN model predicted an almost constant number. This is shown in Figure 11 and 13.

Model	Mean Square Error (MSE)	Standard Deviation (STD)
kNN	0.100757	0.009417
KRR	0.083027	0.005799
Baseline	1.001041	0.081032

Table 3. Mean Square Error of Classifiers

Appendix

```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsRegressor
from sklearn.kernel_ridge import KernelRidge
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import scale
from sklearn.model_selection import ShuffleSplit
from sklearn.dummy import DummyRegressor

def gaussian_kernel(distances):
    weights = np.exp(-parameter_knn*(distances**2))
    return weights/np.sum(weights)

data = np.array([[-1,0],[0,1],[1,0]])
X = data[:, 0]
X = X.reshape(-1, 1)
y = data[:, 1]
y = y.reshape(-1, 1)
xx = np.linspace(-3.0, 3.0, num=1000)
xx = xx.reshape(-1,1)

fig = plt.figure(figsize=(7, 7))
ax = fig.add_subplot(1, 1, 1)
ax.set_title('Visuation of Data', fontsize=14)
ax.set_xlabel('X1', fontsize=12)
ax.set_ylabel('y', fontsize=12)
ax.scatter(X, y, marker='o')
plt.savefig('small_dataset_visualisation')
plt.show()

parameters = [0, 1, 5, 10, 25]
fig = plt.figure(figsize=(14, 21))
fig.suptitle('kNN Classifier Predictions', fontsize=14, y=0.9125)
for itr, parameter_knn in enumerate(parameters, 1):
    knn = KNeighborsRegressor(n_neighbors=3, weights=gaussian_kernel)
    knn.fit(X, y.ravel())
    y_knn = knn.predict(xx)
    ax = fig.add_subplot(3, 2, itr)
    ax.set_title('γ = ' + str(parameter_knn), fontsize=14)
    ax.set_xlabel('X1', fontsize=12)
    ax.set_ylabel('y', fontsize=12)
    ax.scatter(X, y, marker='o')
    ax.plot(xx, y_knn)
```

```
    ax.legend(['Pred', 'True'], loc='upper right')
plt.savefig('knn_dataset1_paramaters')
plt.show()

models = []
parameters = [0, 1, 5, 10, 25]
fig = plt.figure(figsize=(14, 21))
fig.suptitle('KRR Classifier Predictions', fontsize=14, y=0.9125)
for itr, parameter_krr in enumerate(parameters, 1):
    krr = KernelRidge(alpha=1.0/(2 * 1.0), kernel='rbf', gamma=parameter_krr)
    krr.fit(X, y.ravel())
    models.append(krr)
    y_krr = krr.predict(xx)
    ax = fig.add_subplot(3, 2, itr)
    ax.set_title('γ = ' + str(parameter_krr), fontsize=14)
    ax.set_xlabel('X1 (Normalised)', fontsize=12)
    ax.set_ylabel('y (Normalised)', fontsize=12)
    ax.scatter(X, y, marker='o')
    ax.plot(xx, y_krr)
    ax.legend(['Pred', 'True'], loc='upper right')
plt.savefig('krr_dataset1_paramaters')
plt.show()

print('KRR Parameter Table:')
print('%-13s %-12s' % ('Parameter(γ)', 'Parameter(θ)'))
for itr, model in enumerate(models):
    print('%-13s %-12a' % (parameters[itr], model.dual_coef_))

parameter_krr = 5;

models = []
penalties = [0.1, 1, 1000]
fig = plt.figure(figsize=(14, 14))
fig.suptitle('KRR Classifier Predictions', fontsize=14, y=0.9125)
for itr, penalty in enumerate(penalties, 1):
    krr = KernelRidge(alpha=1.0/(2*penalty), kernel='rbf', gamma=parameter_krr)
    krr.fit(X, y.ravel())
    models.append(krr)
    y_krr = krr.predict(xx)
    ax = fig.add_subplot(2, 2, itr)
    ax.set_title('C = ' + str(penalty), fontsize=14)
    ax.set_xlabel('X1 (Normalised)', fontsize=12)
    ax.set_ylabel('y (Normalised)', fontsize=12)
    ax.scatter(X, y, marker='o')
    ax.plot(xx, y_krr)
    ax.legend(['Pred', 'True'], loc='upper right')
plt.savefig('krr_dataset1_penalties')
```

```
plt.show()

print('KRR Penalty Parameter Table:')
print('%-11s %-12s' % ('Penalty(C)', 'Parameter(θ)'))
for itr, model in enumerate(models):
    print('%-11s %-12a' % (penalties[itr], model.dual_coef_))



df = pd.read_csv('week5.csv',comment='#')
X = np.array(df.iloc[:, 0])
X = X.reshape(-1, 1)
X = scale(X)
y = np.array(df.iloc[:, 1])
y = y.reshape(-1, 1)
y = scale(y)

fig = plt.figure(figsize=(7, 7))
ax = fig.add_subplot(1, 1, 1)
ax.set_title('Visuation of Data', fontsize=14)
ax.set_xlabel('X1 (Normalised)', fontsize=12)
ax.set_ylabel('y (Normalised)', fontsize=12)
ax.scatter(X, y, c=y, marker='o', cmap='coolwarm', alpha=0.4)
plt.savefig('large_dataset_visualisation')
plt.show()

models = []
k = len(y)
parameters = [0, 1, 5, 10, 25, 50]
fig = plt.figure(figsize=(14, 21))
fig.suptitle('kNN Classifier Predictions', fontsize=14, y=0.9125)
for itr, parameter_knn in enumerate(parameters, 1):
    knn = KNeighborsRegressor(n_neighbors=k, weights=gaussian_kernel)
    knn.fit(X, y.ravel())
    models.append(knn)
    y_knn = knn.predict(xx)
    ax = fig.add_subplot(3, 2, itr)
    ax.set_title('γ = ' + str(parameter_knn), fontsize=14)
    ax.set_xlabel('X1 (Normalised)', fontsize=12)
    ax.set_ylabel('y (Normalised)', fontsize=12)
    ax.scatter(X, y, c=y, marker='o', cmap='coolwarm', alpha=0.4)
    ax.plot(xx, y_knn)
    ax.legend(['Pred', 'True'], loc='lower right')
plt.savefig('knn_dataset2_paramaters')
plt.show()

kf = KFold(n_splits=5)
```

```
mse = []; mse_std = []
parameters = [0, 1, 5, 10, 25, 50]
for itr, parameter_knn in enumerate(parameters):
   error = []
   knn = KNeighborsRegressor(weights=gaussian_kernel)
   for train, test in kf.split(X):
      knn.set_params(n_neighbors=len(y[test]))
      knn.fit(X[train], y[train].ravel())
      y_knn = knn.predict(X[test])
      error.append(mean_squared_error(y[test], y_knn))
   mse.append(np.array(error).mean())
   mse_std.append(np.array(error).std())

fig = plt.figure(figsize=(7, 7))
ax = fig.add_subplot(1, 1, 1)
ax.set_title('Optimal Value of Parameter γ', fontsize=14)
ax.set_xlabel('γ', fontsize=12)
ax.set_ylabel('Mean Squared Error (MSE)', fontsize=12)
ax.errorbar(parameters, mse, yerr=mse_std, linewidth=2.0, capsize=5.0, elinewidth=2.0,
markeredgewidth=2.0)
ax.scatter(parameters, mse, marker='o')
plt.savefig('knn_parameter_cross_validation')
plt.show()

parameter_knn = 10
print('Optimal Value: %d\n' % (parameter_knn))

knn = KNeighborsRegressor(n_neighbors=k, weights=gaussian_kernel)
knn.fit(X, y.ravel())
y_knn = knn.predict(xx)
fig = plt.figure(figsize=(7, 7))
ax = fig.add_subplot(1, 1, 1)
ax.set_title('kNN Classifier', fontsize=14)
ax.set_xlabel('X1 (Normalised)', fontsize=12)
ax.set_ylabel('y (Normalised)', fontsize=12)
ax.scatter(X, y, c=y, marker='o', cmap='coolwarm', alpha=0.4)
ax.plot(xx, y_knn)
ax.legend(['Pred', 'True'], loc='lower right')
plt.savefig('knn_classifier')
plt.show()

models = []
parameters = [0, 1, 5, 10, 25]
fig = plt.figure(figsize=(14, 21))
fig.suptitle('KRR Classifier Predictions', fontsize=14, y=0.9125)
for itr, parameter_krr in enumerate(parameters, 1):
   krr = KernelRidge(alpha=1.0/(2 * 1.0), kernel='rbf', gamma=parameter_krr)
```

```
    krr.fit(X, y.ravel())
    models.append(krr)
    y_krr = krr.predict(xx)
    ax = fig.add_subplot(3, 2, itr)
    ax.set_title('γ = ' + str(parameter_krr), fontsize=14)
    ax.set_xlabel('X1 (Normalised)', fontsize=12)
    ax.set_ylabel('y (Normalised)', fontsize=12)
    ax.scatter(X, y, c=y, marker='o', cmap='coolwarm', alpha=0.4)
    ax.plot(xx, y_krr)
    ax.legend(['Pred', 'True'], loc='lower right')
plt.savefig('krr_dataset2_paramaters')
plt.show()

kf = KFold(n_splits=5)
mse = []; mse_std = []
for model in models:
    error = []
    for train, test in kf.split(X):
        model.fit(X[train], y[train].ravel())
        y_pred = model.predict(X[test])
        error.append(mean_squared_error(y[test], y_pred))
    mse.append(np.array(error).mean())
    mse_std.append(np.array(error).std())

fig = plt.figure(figsize=(7, 7))
ax = fig.add_subplot(1, 1, 1)
ax.set_title('Optimal Value of Parameter γ', fontsize=14)
ax.set_xlabel('γ', fontsize=12)
ax.set_ylabel('Mean Squared Error (MSE)', fontsize=12)
ax.errorbar(parameters, mse, yerr=mse_std, linewidth=2.0, capsize=5.0, elinewidth=2.0,
markeredgewidth=2.0)
ax.scatter(parameters, mse, marker='o')
plt.savefig('krr_parameter_cross_validation')
plt.show()

parameter_krr = 1
print('Optimal Value: %d\n' % (parameter_krr))

models = []
penalties = [0.1, 1, 1000]
fig = plt.figure(figsize=(14, 14))
fig.suptitle('KRR Classifier Predictions', fontsize=14, y=0.9125)
for itr, penalty in enumerate(penalties, 1):
    krr = KernelRidge(alpha=1.0/(2*penalty), kernel='rbf', gamma=parameter_krr)
    krr.fit(X, y.ravel())
    models.append(krr)
    y_krr = krr.predict(xx)
```

```
    ax = fig.add_subplot(2, 2, itr)
    ax.set_title('C = ' + str(penalty), fontsize=14)
    ax.set_xlabel('X1 (Normalised)', fontsize=12)
    ax.set_ylabel('y (Normalised)', fontsize=12)
    ax.scatter(X, y, c=y, marker='o', cmap='coolwarm', alpha=0.4)
    ax.plot(xx, y_krr)
    ax.legend(['Pred', 'True'], loc='lower right')
plt.savefig('krr_dataset2_penalties')
plt.show()

models = []
penalties = [0.1, 0.5, 1, 5, 10, 50]
for penalty in penalties:
    krr = KernelRidge(alpha=1.0/(2*penalty), kernel='rbf', gamma=parameter_krr)
    krr.fit(X, y.ravel())
    models.append(krr)

kf = KFold(n_splits=5)
mse = []; mse_std = []
for model in models:
    error = []
    for train, test in kf.split(X):
        model.fit(X[train], y[train].ravel())
        y_pred = model.predict(X[test])
        error.append(mean_squared_error(y[test], y_pred))
    mse.append(np.array(error).mean())
    mse_std.append(np.array(error).std())

fig = plt.figure(figsize=(7, 7))
ax = fig.add_subplot(1, 1, 1)
ax.set_title('Optimal Value of Parameter C', fontsize=14)
ax.set_xlabel('C', fontsize=12)
ax.set_ylabel('Mean Squared Error (MSE)', fontsize=12)
ax.errorbar(penalties, mse, yerr=mse_std, linewidth=2.0, capsize=5.0, elinewidth=2.0,
markeredgewidth=2.0)
ax.scatter(penalties, mse, marker='o')
plt.savefig('krr_penalty_cross_validation')
plt.show()

penalty_krr = 5
print('Optimal Value: %d\n' % (penalty_krr))

krr = KernelRidge(alpha=1.0/(2*penalty_krr), kernel='rbf', gamma=parameter_krr)
krr.fit(X, y.ravel())
y_krr = krr.predict(xx)
fig = plt.figure(figsize=(7, 7))
ax = fig.add_subplot(1, 1, 1)
```

```
ax.set_title('KRR Classifier', fontsize=14)
ax.set_xlabel('X1 (Normalised)', fontsize=12)
ax.set_ylabel('y (Normalised)', fontsize=12)
ax.scatter(X, y, c=y, marker='o', cmap='coolwarm', alpha=0.4)
ax.plot(xx, y_krr)
ax.legend(['Pred', 'True'], loc='lower right')
plt.savefig('krr_classifier')
plt.show()

knn = KNeighborsRegressor(n_neighbors=len(y[test]), weights=gaussian_kernel)
krr = KernelRidge(alpha=1.0/(2*penalty_krr), kernel='rbf', gamma=parameter_krr)
dummy = DummyRegressor(strategy='mean')

kf = KFold(n_splits=5)
mse = []; mse_std = []
models = [knn, krr, dummy]
for itr, model in enumerate(models):
   error = []
   for train, test in kf.split(X):
      model.fit(X[train], y[train].ravel())
      y_pred = model.predict(X[test])
      error.append(mean_squared_error(y[test], y_pred))
   mse.append(np.array(error).mean())
   mse_std.append(np.array(error).std())

print('Evaluation:')
print('%-12s %-10s %-10s' % ('Classifier', 'MSE', 'STD'))
labels = ['kNN', 'KRR', 'Dummy']
for itr in range(len(mse)):
   print('%-12s %-10f %-10f' % (labels[itr], mse[itr], mse_std[itr]))
```