CSU44061 Machine Learning Week 6

Student Number: 17341914 | #id: 11--88--22

(i)(a) The dummy data is defined and extracted into np arrays X and y. X is reshaped into a single column.

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

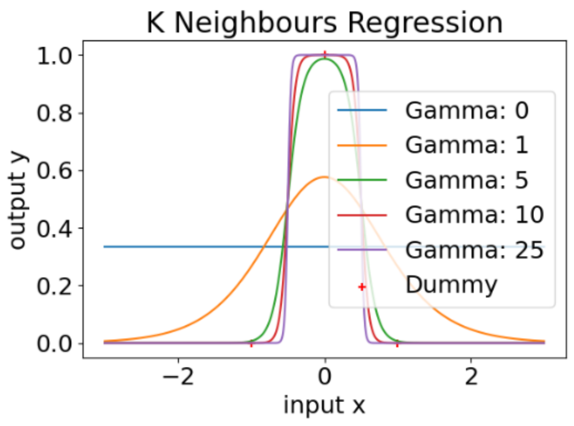
The gaussian kernel formula is given by $K(x^i, x) = e^{-\gamma d(x^i,x)^2}$ where $d(x^i, x)$ is the Euclidean distance between input x and training point x^i. Gamma (γ) controls how quickly the kernel decreases as the distance between training point x^i and input x grows. The function gaussian_kernel() describes this operation.

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

Here, gamma is a global variable that is assigned each value of gamma given in the for-loop range [0,1,5,10,25]. This function returns an array of weights that can be used within the sklearn function KNeighboursRegressor. Using this we can repeatedly plot the kNN predictions for each gamma onto one plot.

```
for g in [0,1,5,10,25]:
    gamma = g
    model = 
KNeighborsRegressor(n_neighbors=m,weights=gaussian_kernel).fit(X, y)
    Xtest = np.linspace(-3, 3, num=1000).reshape(-1, 1)
    ypred = model.predict(Xtest)
    plt.plot(Xtest, ypred, label=f"Gamma: {g}")

plt.scatter(X, y, color='red', marker='+', label="Dummy")
#... plot specifications ...
```

The resulting graph visualises the kernel decrement as gamma increases. What results are steeper slopes between training points. When for example gamma is 1, the kernel decrements slowly and thus, has a more flat and curvy line. Comparatively, when gamma is 25 the kernel decrements very quickly, resulting in a boxy line mapping to the value of the nearest kernel with little regard of points further away.



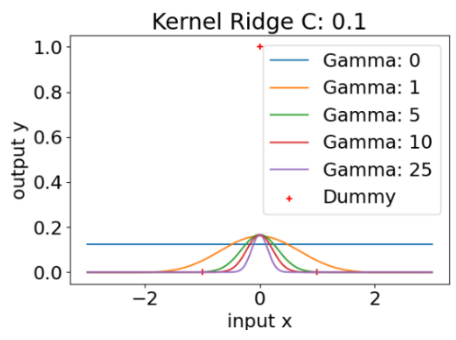
(b)A kNN model is an instance-based model that generates it's predictions by calculations based only the k nearest neighbours of the input x. Within these selected values, predictions are calculated based on collected distance and weight. Gaussian weight tells us to attach less weight to training points further away. Gamma is a hyperparameter that can fine tune how severely distances are weighted. In the gaussian kernel formula, as the distances are multiplied by negative gamma, higher gamma values correspond to higher negative weights associated to distance from the input. Hence when gamma is larger, the model will map stricter to only the training points it is closest to. Oppositely, when it is smaller, points further away will have more of a bearing on it, resulting in looser curves that describe the larger population of training points.

(c) Kernel ridge regression combines ridge regression with the kernel trick. It can be implemented in sklearn as shown:

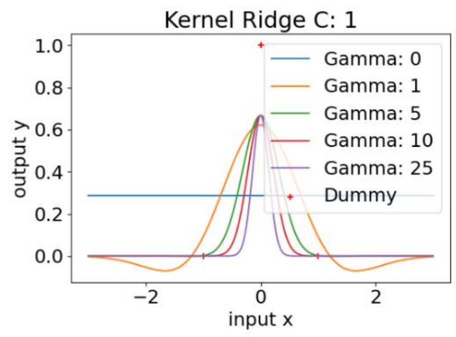
```
model = KernelRidge(alpha=1.0/(2*C), kernel='rbf', gamma=g).fit(X, y)
Xtest=np.linspace(-3,3,num=1000).reshape(-1, 1)
ypred = model.predict(Xtest)
```

Alpha defines the regularisation strength, with larger values specifying stronger regularization. Kernel ridge uses an L2 penalty which corresponds to 1/(2C). We repeat the above process for plotting with varying gamma range but enclose this in another for loop to visualise these predictions when C is [0.1,1,1000].

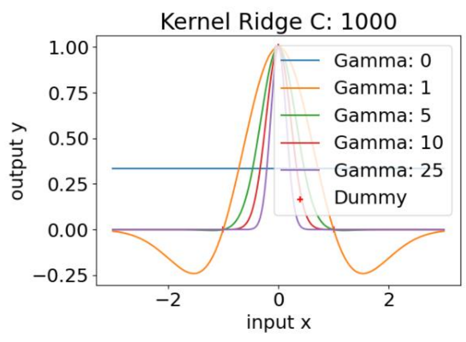
```
for C in [0.1, 1, 1000]:

    for g in [0,1,5,10,25]:
      model = KernelRidge(alpha=1.0/(2*C), kernel='rbf', gamma=g).fit(X, y)
      Xtest = np.linspace(-3, 3, num=1000).reshape(-1, 1)
      ypred = model.predict(Xtest)
      print("C:", C, " g: ", g, " dual coef:", model.dual_coef_)
      plt.plot(Xtest, ypred, label=f"Gamma: {g}")

    plt.scatter(X, y, color='red', marker='+', label="Dummy")
    #... plot specifications...
```



Gamma	Dual coefficients
0	[-0.025 0.175 -0.025]
1	[-0.01026472 0.16792539 -0.01026472]
5	[-0.00018717 0.16666709 -0.00018717]
10	[-1.26110916e-06 1.66666667e-01 -1.26110916e-06]
25	[-3.85776218e-13 1.66666667e-01 -3.85776218e-13]



Gamma	Dual coefficients
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]



Gamma	Dual coefficients
0	[-666.55557407 1333.44442593 -666.55557407]
1	[-0.49138748 1.36086227 -0.49138748]
5	[-0.00673182 0.99959092 -0.00673182]
10	[-4.53545640e-05 9.99500254e-01 -4.53545640e-05]
25	[-1.38740663e-11 9.99500250e-01 -1.38740663e-11]

The graphs and reported parameters show that increasing the value of C reduces regularisation. The gamma values within each C seem to have proportional line plots, corresponding to the regularisation imposed. Unless the value of C is sufficiently high and therefore the regularisation sufficiently low, the predictions fall short of reaching the second dummy point.

(d) As already mentioned, larger values of alpha correspond to larger regularisation. Because alpha is described 1/2C, where C is the denominator, any increase of C will reduce the size of alpha, furthermore, reducing the regularisation imposed. This model uses L2 regularisation, which adds a penalty equal to the square of the magnitude of coefficients. This means all coefficients are shrunk by the same factor.

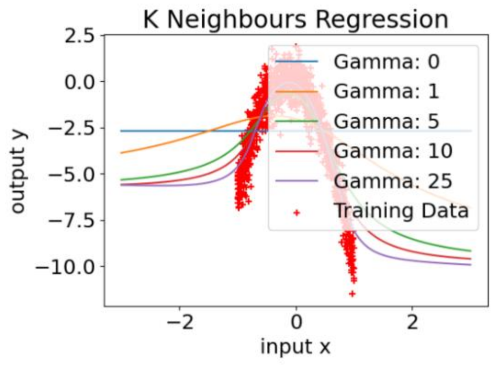
There are three dual coefficients that describe the three points in the dummy dataset. The kernelized ridge model is given by $\hat{y} = \theta_0 + w^1 K(x^1, x) + w^2 K(x^2, x)$ where parameters w^1, w^2 are learned and θ_i are the respective dual coefficients. When the weight given to the regularisation penalty is large enough, however they take the form $w^i \approx \theta_i y^i$. As these coefficients are multiplied by the gaussian kernels, param values scale the output of the kernel function.

The kernel ridge model needs more configuration to predict the data suitably. The regularisation implemented in the first to values of C stands to worsen the model and this might be because of the dataset being small and far in-between, relatively speaking. For this reason, the kernelized k-neighbours model is more suited to this input.

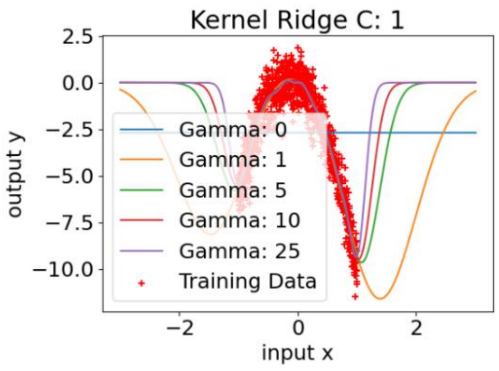
(ii)(a) The dataset is loaded in, extracted to numpy arrays X and y and reshaped appropriately.

```
df = pd.read_csv("week6.csv", comment='#')
X = np.array(df.iloc[:,0]).reshape(-1,1)
y = np.array(df.iloc[:,1])
```

From here, the implementation is identical to (i)(a), with the only difference being the data in X and y. The resulting graph acts similarly as above. When gamma is low the gaussian kernel function decreases slowly distributing the weight of distances with the k neighbours over a larger scope. This operation mimics underfitting, it does not fit the idiosyncrasies of each training data point but instead the general trend of data. When gamma is high, the gaussian kernel function decreases quickly meaning weight is placed only on the closest points in the k nearest neighbours. This operation maps the training data closely but may have undefined predictions outside the population of the dataset.



(b) Again, the implementation is identical to (i)(c). This question, however, does not need to be repeated for a range of C values, so we can leave the outer for loop out and set C to 1 instead. Repeating this execution for our new dataset we get the resulting plot to the right. The justification for gamma variations are consistent for this plot as well. Low gamma corresponds to a loose mapping of training data, high gamma corresponds to strict mapping.



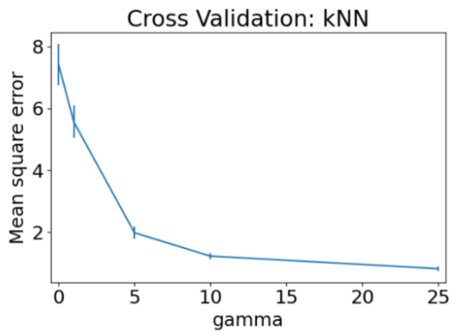
(c) **K-Nearest Neighbours Model**
Cross validation is tested on each kNN model where γ is in the range [0,1,5,10,25].

```
m_split = int(len(X)*(4/5))
mean_err = []; std_err = []
gamma_range = [0,1,5,10,25]

for g in gamma_range:
    gamma = g
    temp = []
    model = KNeighborsRegressor(n_neighbors=m_split,
    weights=gaussian_kernel).fit(X, y)

    kf = KFold(n_splits=5)
    for train, test in kf.split(X):
        model.fit(X[train], y[train])
        ypred = model.predict(X[test])
        temp.append(mean_squared_error(y[test], ypred))
    mean_err.append(np.array(temp).mean())
    std_err.append(np.array(temp).std())
```

The model specific to each γ is created outside the cross-validation loop form where it can be called. The error and standard deviation are charted for each model and from this we can plot an error bar graph that compares all model. When defining our model, it was important that we set n_neighbours equal to the population of the split training dataset. As we have opted for 5-fold validation, the split dataset to be trained is 4/5ths of the original dataset. Not doing this would result in the number of neighbours being greater than the number of samples, which would cause a runtime exception.

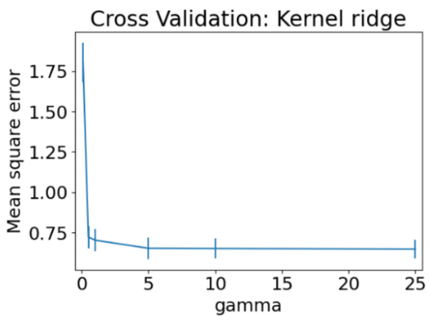
```
plt.errorbar(gamma_range, mean_err, yerr=std_err)
```

Plotting above gives predictable results. The higher gamma is the better it predicts the test split of the dataset. As we have discussed already, a higher gamma maps more strictly to training data. If we were to look back on the plot created in part (ii)(a) this relationship is reflected. When gamma is low the line plot is flat and ill representative of the training data, hence why it has worse mean squared error when tested in cross-validation.



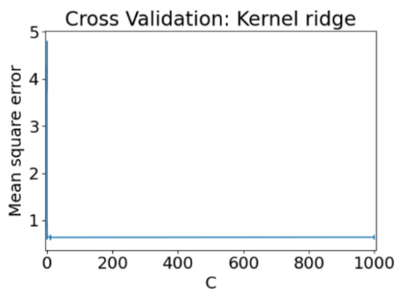
Kernel Ridge Regression
Now using cross validation to test for the optimal parameter γ and α, we use the same process. First, we will test for γ and using the optimal value for γ we will test for α.

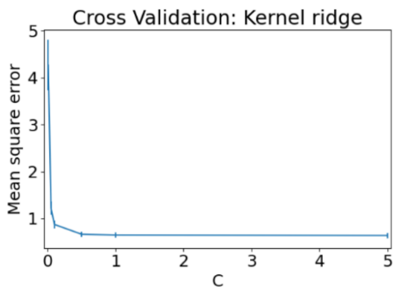
Ignoring the model outside of the training data, inspecting the plot in (ii)(b) shows why this cross-validation acts in the manner it does. Apart from when γ is 0, on visual inspection, all the models sit more or less on the same line cutting through the data. It is only outside the data set that the models disperse on their paths. Hence, we can see why once γ is above ~1 whichever one you pick will have an equal low mean square error value. We will pick gamma = 25 for using cross-validation in each α.



Testing for α with cross-validation means repeating the experiment for different values of C, which we know describes α. Hence, we repeat the above implementation for C in the range

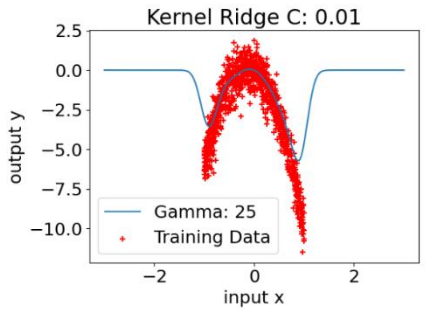
[0.01,0.1,1,10,1000]. Additionally, we test in the range [0.01,0.05,0.1,0.5,1,5] so to show a close-up of the smaller values within the larger range.

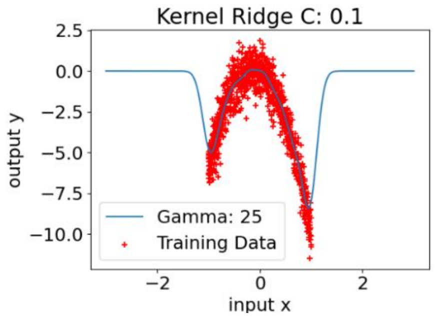




We see the familiar 'L' line plot in these tests which show that once $C > 0$, the subsequent values in the range will offer a similar outcome low error in testing. This reasoning for this error bar plot can be aided by actually plotting some of the Kernel Ridge models it describes.

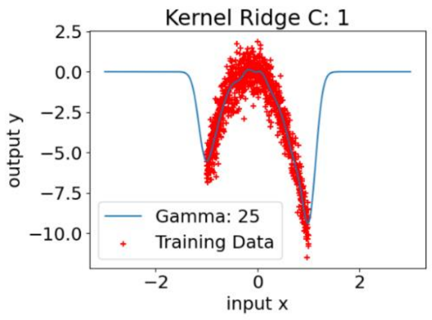
$$\alpha = \frac{1}{2 * 0.01} = 50$$

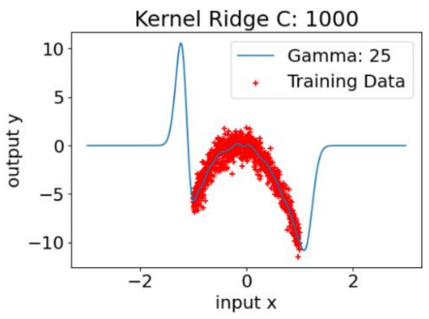




$$\alpha = \frac{1}{2 * 0.1} = 5$$

$$\alpha = \frac{1}{2 * 1} = 0.5$$





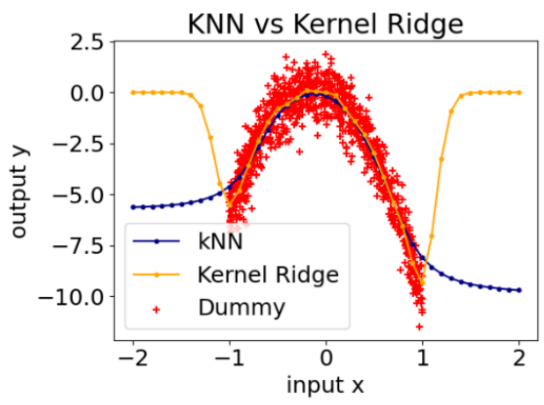
$$\alpha = \frac{1}{2 * 1000} = 0.0005$$

These graphs show that only when C is very close to 0 does the model start to fail to map to the body of the training data. When C gets larger the predictive power remains very good within the bounds of the training data but scales very poorly outside of this. We can see though how when $C = 1$ and $C = 1000$ the predictions within the dataset are virtually the same. This is what is being represented in the cross-validation graph. Next to each graph is the corresponding α value. So that we avoid using an overly complex model, we will select $C = 1$ and hence $\alpha = 0.5$ as the optimal value that describes the data when $\gamma = 25$.

Finally, we will use both models optimised from cross validation to generate predictions for arbitrarily picked points.

```
predict_values = np.linspace(-2,2,41).reshape(-1,1)

model = KNeighborsRegressor(n_neighbors=m,weights=gaussian_kernel).fit(X,y)
ypred = model.predict(predict_values)
plt.plot(predict_values, ypred, label="kNN", color="navy", marker=".")

#same for Kernel Ridge
```

The prediction points were selected with the above command and represent all values between -2 and 2 with a step of 0.1. The predictions of the input points are graphed beside. Both models perform well within the dataset, though it looks like Kernel Ridge could just about predict better near either end of the data at -1 and 1. Kernel ridge does however reset to 0 outside the dataset limit, meaning it would be a poor predictor outside of this range. KNN on the other had considers all points in the dataset even when making predictions for points at extremities (because we set n_neighbours to n_samples in our implementation). This means the model would typically beat Kernel ridge for any points outside the dataset range. For this reason, I would pick the k-Nearest Neighbours model using a gaussian weight function. This model is easy to implement but also maintains the quality of being able to be finely tuned with a powerful parameter like the discussed γ.



Appendix

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

dummy_data = [(-1,0), (0,1), (1,0)]
X, y = zip(*dummy_data)
X = np.array(X).reshape(-1,1)
y = np.array(y)

m=len(X)
gamma = 0

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

plt.rc('font', size = 18)
plt.rcParams['figure.constrained_layout.use'] = True

for g in [0,1,5,10,25]:
    gamma = g
    model = KNeighborsRegressor(n_neighbors=m,weights=gaussian_kernel).fit(
X, y)
    Xtest = np.linspace(-3, 3, num=1000).reshape(-1, 1)
    ypred = model.predict(Xtest)
    plt.plot(Xtest, ypred, label=f"Gamma: {g}")

plt.scatter(X, y, color='red', marker='+', label="Training Data")
plt.xlabel("input x"); plt.ylabel("output y")
plt.legend()
plt.title("K Neighbours Regression")
plt.show()

############################(c)##############################

for C in [0.1, 1, 1000]:
    plt.rc('font', size = 18)
    plt.rcParams['figure.constrained_layout.use'] = True

    for g in [0,1,5,10,25]:
        model = KernelRidge(alpha=1.0/(2*C), kernel='rbf', gamma=g).fit(X,
y)
        Xtest=np.linspace(-3,3,num=1000).reshape(-1, 1)
        ypred = model.predict(Xtest)
        print("g:", g, "dual coef:", model.dual_coef_)
        plt.plot(Xtest, ypred, label=f"Gamma: {g}")

    plt.scatter(X, y, color='red', marker='+', label="Training Data")
    plt.xlabel("input x")
    plt.ylabel("output y")
    plt.legend()
    plt.title(f"Kernel Ridge C: {C}")
    plt.show()
```

```
#################################(ii)(a)##############################

df = pd.read_csv("week6.csv", comment='#')
X = np.array(df.iloc[:,0]).reshape(-1,1)
y = np.array(df.iloc[:,1])
m=len(X)

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

plt.rc('font', size = 18)
plt.rcParams['figure.constrained_layout.use'] = True

for g in [0,1,5,10,25]:
    gamma = g
    model = KNeighborsRegressor(n_neighbors=m,weights=gaussian_kernel).fit(
X, y)
    Xtest = np.linspace(-3, 3, num=1000).reshape(-1, 1)
    ypred = model.predict(Xtest)
    plt.plot(Xtest, ypred, label=f"Gamma: {g}")

plt.scatter(X, y, color='red', marker='+', label="Training Data")
plt.xlabel("input x"); plt.ylabel("output y")
plt.legend()
plt.title("K Neighbours Regression")
plt.show()

#############################(ii)(b)##############################

C = 1

for g in [0,1,5,10,25]:
    model = KernelRidge(alpha=1.0/(2*C), kernel='rbf', gamma=g).fit(X, y)
    Xtest=np.linspace(-3,3,num=1000).reshape(-1, 1)
    ypred = model.predict(Xtest)
    plt.plot(Xtest, ypred, label=f"Gamma: {g}")

plt.scatter(X, y, color='red', marker='+', label="Training Data")
plt.xlabel("input x")
plt.ylabel("output y")
plt.legend()
plt.title(f"Kernel Ridge C: {C}")
plt.show()

#################################(ii)(c)###############################
#KNN Regression with Gaussian Kernel Weights

m_split = int(len(X)*(4/5))
mean_err = []; std_err = []
gamma_range = [0,1,5,10,25]

for g in gamma_range:
    gamma = g
    temp = []
    #n_neighbours can only be as big as n_samples in cross val split
    model = KNeighborsRegressor(n_neighbors=m_split, weights=gaussian_kerne
l).fit(X, y)

    kf = KFold(n_splits=5)
    for train, test in kf.split(X):
        model.fit(X[train], y[train])
        ypred = model.predict(X[test])
        temp.append(mean_squared_error(y[test], ypred))
    mean_err.append(np.array(temp).mean())
    std_err.append(np.array(temp).std())

plt.rc('font', size=18)
plt.rcParams['figure.constrained_layout.use'] = True
plt.errorbar(gamma_range, mean_err, yerr=std_err)
plt.xlabel('gamma'); plt.ylabel('Mean square error')
plt.xlim((-0.5, 25.5))
plt.title('Cross Validation: kNN')
plt.show()

#--------------Kernelised Ridge Regression (gamma)------------------------

mean_err = []; std_err = []
gamma_range = [0.1,0.5,1,5,10,25]
C = 1

for g in gamma_range:
    model = KernelRidge(alpha=1.0/C, kernel='rbf', gamma=g).fit(X, y)
    temp = []

    kf = KFold(n_splits=5)
    for train, test in kf.split(X):
        model.fit(X[train], y[train])
        ypred = model.predict(X[test])
        #print("intercept ", m.intercept_, "slope ", m.coef_,
        # " square error ", mean_squared_error(y[test], ypred))
        temp.append(mean_squared_error(y[test], ypred))
    mean_err.append(np.array(temp).mean())
    std_err.append(np.array(temp).std())

plt.rc('font', size=18)
plt.rcParams['figure.constrained_layout.use'] = True
plt.errorbar(gamma_range, mean_err, yerr=std_err)
plt.xlabel('gamma'); plt.ylabel('Mean square error')
plt.xlim((-0.5, 25.5))
plt.title('Cross Validation: Kernel ridge')
plt.show()

#---------------------Kernel Ridge (alpha)--------------------------------

mean_err = []; std_err = []
g = 25
C_range = [0.01,0.1,1,10,1000]
#C_range = [0.01,0.05,0.1,0.5,1,5]

for C in C_range:
    model = KernelRidge(alpha=1.0/C, kernel='rbf', gamma=g).fit(X, y)
    temp = []

    kf = KFold(n_splits=5)
    for train, test in kf.split(X):
        model.fit(X[train], y[train])
        ypred = model.predict(X[test])
        #print("intercept ", m.intercept_, "slope ", m.coef_,
        # " square error ", mean_squared_error(y[test], ypred))
        temp.append(mean_squared_error(y[test], ypred))
    mean_err.append(np.array(temp).mean())
    std_err.append(np.array(temp).std())

plt.rc('font', size=18)
plt.rcParams['figure.constrained_layout.use'] = True
plt.errorbar(C_range, mean_err, yerr=std_err)
plt.xlabel('C'); plt.ylabel('Mean square error')
plt.xlim((-5, 1005))
#plt.xlim((-.05, 5.05))
plt.title('Cross Validation: Kernel ridge')
plt.show()

#---------------------------------------------------

for C in [0.01, 1, 1000]:
    plt.rc('font', size = 18)
    plt.rcParams['figure.constrained_layout.use'] = True
    g = 25

    model = KernelRidge(alpha=1.0/(2*C), kernel='rbf', gamma=g).fit(X, y)
    Xtest=np.linspace(-3,3,num=1000).reshape(-1, 1)
    ypred = model.predict(Xtest)
    print("g:", g, "dual coef:", model.dual_coef_)
    plt.plot(Xtest, ypred, label=f"Gamma: {g}")

    plt.scatter(X, y, color='red', marker='+', label="Training Data")
    plt.xlabel("input x")
    plt.ylabel("output y")
    plt.legend()
    plt.title(f"Kernel Ridge C: {C}")
    plt.show()

#------------------predictions--------------------------------

gamma = 25
C = 1
predict_values = np.linspace(-2,2,41).reshape(-1,1)

plt.rc('font', size=18)
plt.rcParams['figure.constrained_layout.use'] = True
plt.scatter(X, y, color='red', marker='+', label="Dummy")

model = KNeighborsRegressor(n_neighbors=m,weights=gaussian_kernel).fit(X, y
)
ypred = model.predict(predict_values)
plt.plot(predict_values, ypred, label="kNN", color="navy", marker=".")

model2 = KernelRidge(alpha=1.0/(2*C), kernel='rbf', gamma=gamma).fit(X, y)
ypred2 = model2.predict(predict_values)
plt.plot(predict_values, ypred2, label="Kernel Ridge", color="orange", mark
er=".")

plt.xlabel("input x"); plt.ylabel("output y")
plt.legend()
plt.title("KNN vs Kernel Ridge")
plt.show()
```