Machine Learning - Week 3 Assignment

Mark Lysaght - 19334391

i) a)

Figure 1(a) shows the data given by the dataset id: 25-25-25. The data is shown on a 3D scatter plot where the X1 feature is on the x-axis, the X2 feature is on the y-axis and the y target value is on the z-axis. I gave two different angles of the visualisation so that we can see more clearly in Figure 1(b) that the data appears to lie on a curve moreso than on a plane.

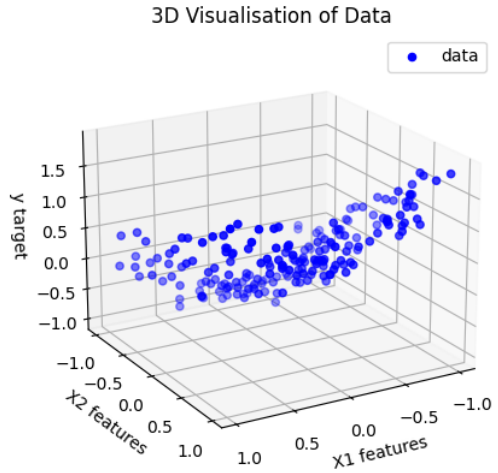


Figure 1(a). Visualisation of data

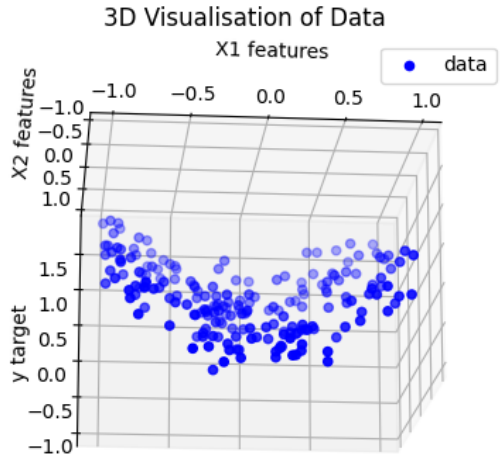


Figure 1(b) Different angle of (a)

b)

Using the sklearn PolynomialFeatures function, extra polynomial features were added to the training data. Lasso regression was applied to the new model. The new equation for the model exists in the form:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \ldots + \theta_{20} x_1 x_2^4 + \theta_{21} x_2^5$$

Below is the cost function for Lasso regression.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{1}{C}\sum_{j=1}^{n}|\theta_j|.$$

Lasso regression performs L1 regularisation. L1 regularisation shrinks the less important features' coefficients to zero and so we can see which feature(s) weigh more importance as we vary the penalty term C. Lasso regression models were trained for values of C = [1,10,100,1000]. The results were as follows.

In each of the following models, the intercept corresponds to θ_0 and the coefficients correspond to each of θ_{1-21}.

C: 1
Intercept: 0.394793
Coefficients: [0. -0. 0. 0. -0. 0. -0. 0. -0. 0. 0. -0. 0. -0. 0. -0. 0. -0. 0. -0. 0.]
As we can see, having a small value for C gives the penalty more impact to the cost function and all parameter coefficients are equal to 0 as a result.

C: 10
Intercept: 0.1944788
Coefficients: [0. -0. 0.82287 0.40625 -0. 0. -0. 0. -0. 0. 0. -0. 0. -0. 0. -0. 0. -0. 0. -0. 0. -0. 0.]
As C starts to increase, there is more leniency applied to the penalty and we can see

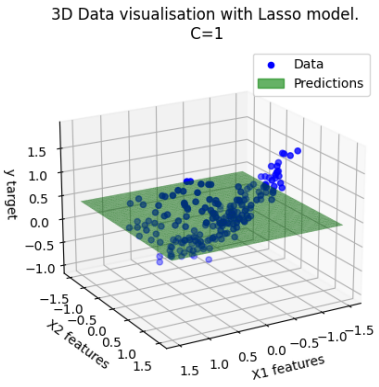
C: 100
Intercept: 0.03907829
Coefficients: [0. 0. 0.94152 0.88521 -0.01510 0. 0. 0. 0. 0. 0. -0. 0. -0. 0. 0. -0. -0. 0. 0. 0.]

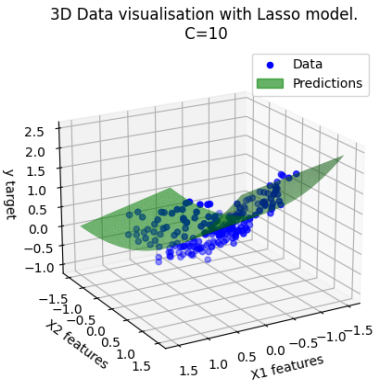
C: 1000
Intercept: 0.02549201
Coefficients: [0. -0.0441 0.93031 0.89433 0. 0.01151 0.08381 -0. 0. 0.03949 -0.00236 -0.11031 0.13452 0. -0.04831 0. -0.0874 -0.16147 0.0712 0.15023 -0.]
For C=1000 we see a big change from C=1. Much more features are included in the model.

c)
For each model in (b), predictions were generated for the target variable. A grid of feature values was created which extends beyond the range of values in the dataset. The predictions were then plotted against the training data. I used plot_trisurf() to plot the predictions.

See Figure 2 below for plotted predictions.

For the lasso model when C=1, we can observe that the penalty has had a big impact on the predictions, as seen in part (b). The resultant predictions lie flat on a plane at the intercept y=0.394793. As C increases, the penalty imposed decreases and we can see the predictions starting to fit the data more accurately. The development of the curve from a range of values for C is displayed clearly from the C values I used and as C increases, the curve gains more complexity. This is a result of the L1 penalty not shrinking the coefficients to 0 and thus affecting the model for predictions.





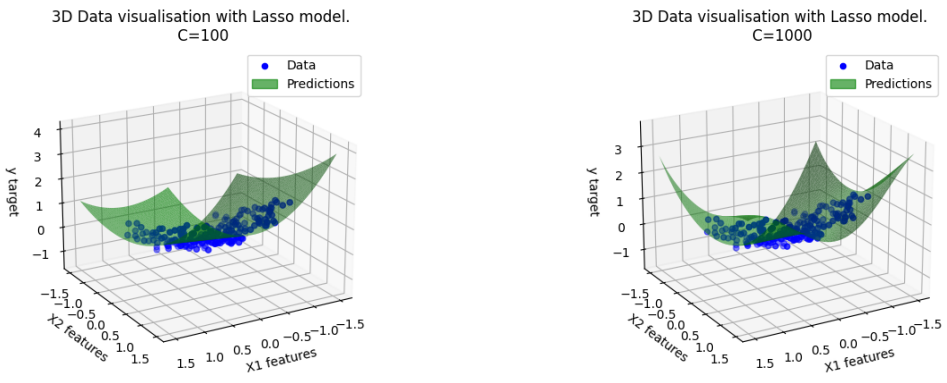


Figure 2. Lasso predictions with varying C penalty values fitted against original data.

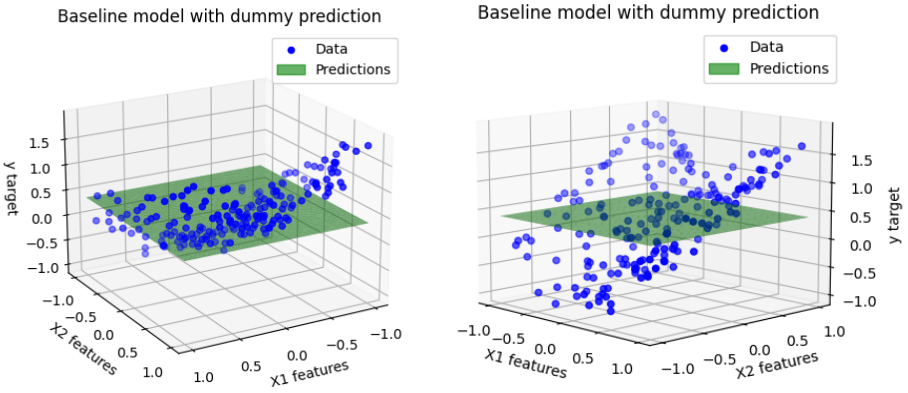


Figure 3. Baseline prediction using DummyRegressor (same graph different angles)

Comparison with a dummy predictor shows similarity to small values for C as the predictions seem planar but they are severely underfit to the training data.

d)
Under-fitting is when the model parameters for a model prediction are too simplistic to represent a justifiable prediction on a set of training data.
Over-fitting, on the contrary, is when the model parameters are too complex and predictions start to fit more closely to the noise in the data.
As we can see from parts (b) and (c), when we change the penalty C, this affects the complexity of our prediction model. When C is too small, we underfit the prediction to the data and this is seen by all parameter values being equal to 0. When C is too large, the opposite comes into effect and too many parameters are affected by the smaller weight of the penalty to the cost function. This is shown in both cases in the extremes when C=1 and C=1000. The shape of the prediction model when C=1 is a simple plane. The shape of the prediction model when C=1000 is a complex plane with many inflection points. When we choose a value for C in the middle of the extremes (in this case 1 and 1000), we can manage a trade-off between under and over-fitting the data.

e(b))
As done before in part b), the sklearn PolynomialFeatures function was used to add extra polynomial features to the training data. Ridge regression was applied to the new model with a range of values for C. (C=0.001,0.1,1,10,100,1000)
Below is the cost function for ridge regression:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \theta^T\theta/C$$

Ridge regression employs the L2 penalty rather than the L1 penalty seen in Lasso regression and due to this, the parameter values are encouraged to have smaller values.

In each of the following models, the intercept corresponds to θ_0 and the coefficients correspond to each of θ_{1-21}.

C: 0.001
Intercept: 0.3581944281840793
Coefficients: [0. -0.008 0.093 0.031 -0.006 0.007 -0.008 0.029 -0.005 0.057
0.025 -0.005 0.014 -0.003 0.006 -0.007 0.017 -0.004 0.018 -0.003 0.042]
For a very small value for C, we observe that almost all parameters still have values. In our lasso model, the parameters would shrink to zero if they were small enough but this is not the case when using ridge regression.

C: 0.1
Intercept: 0.10242435242852632
Coefficients: [0. -0.023 0.702 0.492 -0.036 -0.017 0.009 0.092 0.006 0.223
0.304 -0.048 0.151 0.045 -0.027 0.013 0. -0.022 -0.003 -0.006 0.055]
As C gets larger, we can see a change in all of the model parameters.

C: 1
Intercept: 0.051438195370957784
Coefficients: [0. -0.088 0.883 0.843 -0.129 -0.049 0.074 0.106 0.129 0.19
0.036 -0.062 0.256 0.211 -0.052 0.059 -0.185 -0.24 0.011 -0.011 -0.141]

C: 10
Intercept: 0.02339356475981097
Coefficients: [0. -0.237 0.885 1.135 -0.302 -0.062 0.354 0.294 0.537 0.275
-0.266 0.015 0.282 0.4 -0.068 -0.037 -0.467 -0.731 0.063 -0.146 -0.291]

C: 100
Intercept: 0.017432850754665774
Coefficients: [0. -0.326 0.858 1.177 -0.374 -0.064 0.59 0.44 0.756 0.347
-0.306 0.074 0.297 0.46 -0.074 -0.185 -0.623 -0.952 0.029 -0.251 -0.355]

C: 1000
Intercept: 0.01678745899677697
Coefficients: [0. -0.341 0.853 1.18 -0.384 -0.064 0.632 0.466 0.789 0.36
-0.308 0.084 0.299 0.468 -0.075 -0.213 -0.649 -0.985 0.021 -0.267 -0.366]
The change in parameter values from when C=100 to C=1000 is very small and we can establish from this that we have reached a value for C<100 that could be applied as an appropriate penalty for our ridge regression model.

e(c))

For the ridge model when C=0.001, we can observe that the penalty has had a big impact on the predictions, as seen in part (e(b)). The parameter values are all quite small and from figure 4, we see the predictions are not fitted very well to the data. As C increases, the penalty imposed decreases and we can see the predictions starting to fit the data more accurately. The development of the curve from a range of values for C is displayed clearly from the C values I used and as C increases, the curve gains more complexity. This is a result of the L2 penalty having less effect on the model parameters as our penalty from the cost function above ($\theta^T\theta/C$) has less impact for larger values of C. As we come to the models where C=100 and C=1000, we see little change from the graphs and this indicates to us, as mentioned above, that we have come to a value of C which fits the model well. If we keep increasing C, we run the risk of over-fitting the model.

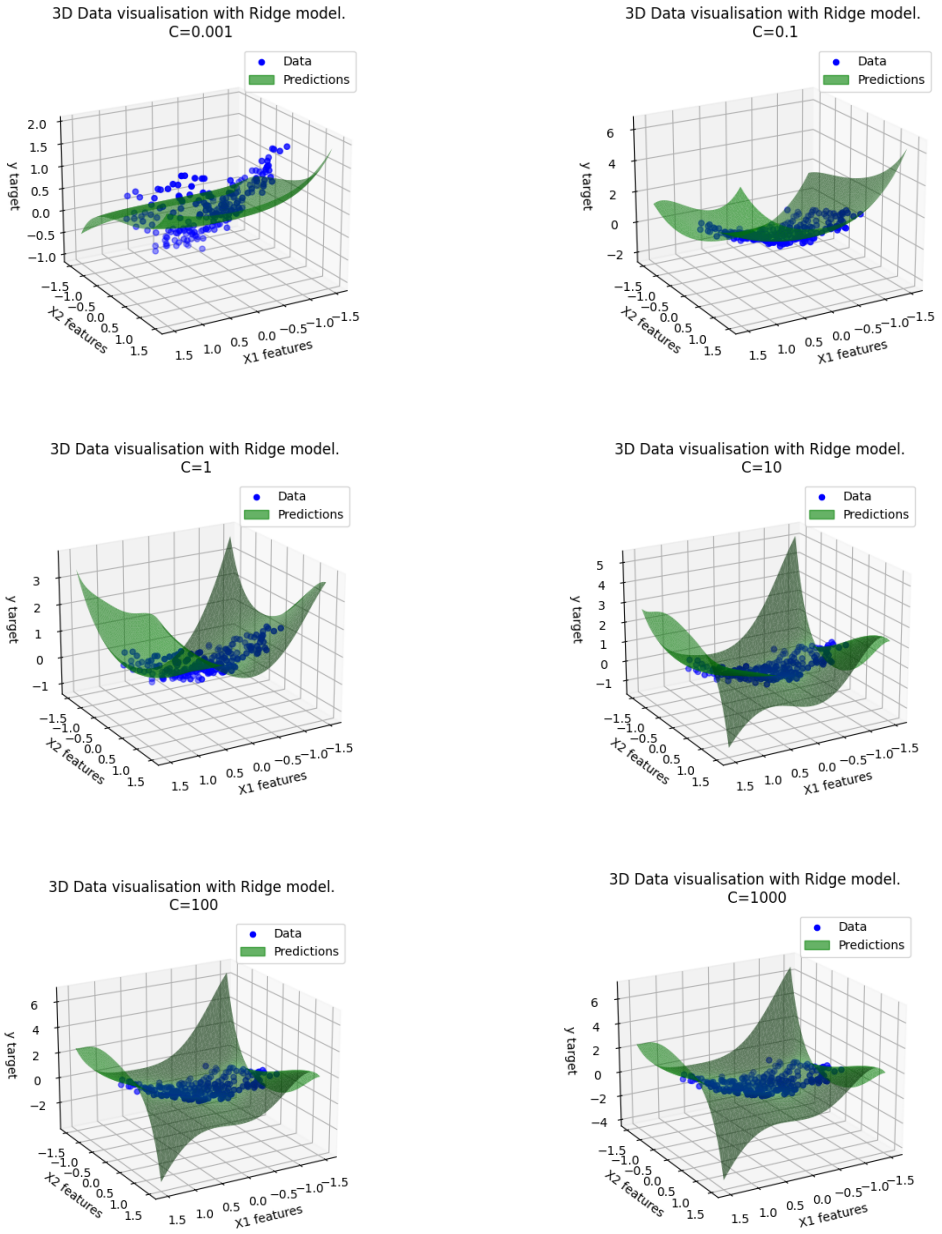


Figure 4. Ridge model predictions with varying C penalties fitted against original data.

ii)a)
Using 5-fold cross validation, I plotted the mean and standard deviation of the prediction error vs C for the Lasso model from part (i). Figure 5 shows the MSE plotted against different values for C. The vertical points show the severity of the error in the standard deviation. (Larger vertical line means more error)
The mean squared error is the average of all the error in the predictions with relation to the original data.

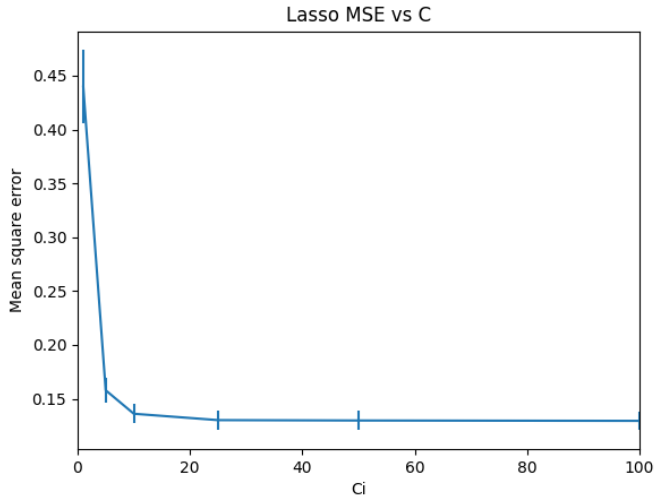


Figure 5. Mean squared error for Lasso model plotted against varying values of C.

b)
Based on the cross validation data, I would recommend choosing a value for C greater than 10 but not too large as to overfit the data. As we can see from figure 5, the mean squared error levels off after C becomes larger than 25. From this we can see when C has less effect on the cost function and so we can get an upper bound for C.

c(a)
Using 5-fold cross validation, I plotted the mean and standard deviation of the prediction error vs C for the Ridge model from part (i). Figure 5 shows the MSE plotted against different values for C. The vertical points show the severity of the error in the standard deviation. (Larger vertical line means more error)

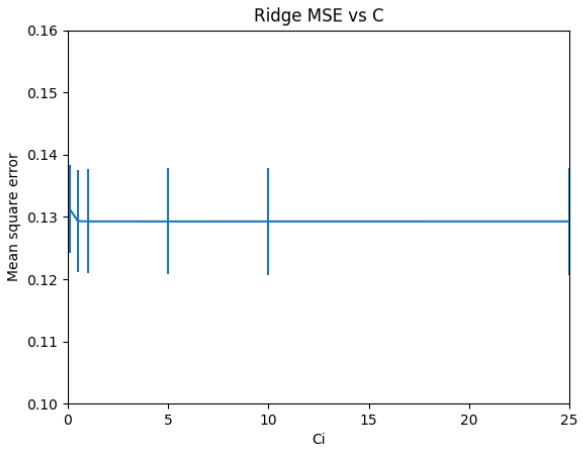


Figure 6. Mean squared error for Lasso model plotted against varying values of C.

c(b)
Based on the cross validation data, I would recommend choosing a value for C greater than 1 but not larger than 10 because as C increases, the predictions start to fit the noise in the data. In figure 4 we can see the effect of C being too large and thus overfitting the data. In figure 5, the mean squared error against C has very little variance as C increases so the smaller C is without impacting the error the better.

Appendix of code

```
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import pandas as pd
import numpy as np

from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
from sklearn.dummy import DummyRegressor

from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
# dataset #id:25-25-25

df = pd.read_csv("week3.csv")
X1 = df.iloc[:, 0]
X2 = df.iloc[:, 1]
X = np.column_stack((X1, X2))
y = df.iloc[:, 2]

Xtest = []
grid = np.linspace(-1.5, 1.5)
for i in grid:
   for j in grid:
       Xtest.append([i, j])
Xtest = np.array(Xtest)


def partIa():
   fig = plt.figure()
   ax = fig.add_subplot(111, projection='3d')
   ax.scatter(X[:, 0], X[:, 1], y, c='b', label='data')
   ax.set_xlabel("X1 features")
   ax.set_ylabel("X2 features")
```

```
    ax.set_zlabel("y target")
    ax.set_title("3D Visualisation of Data")
    ax.legend()
    ax.view_init(20, 60)
    plt.show()


def partIb():
    Xtrain, _, ytrain, _ = train_test_split(X, y, test_size=0.2)
    Xtrain_poly = PolynomialFeatures(5).fit_transform(Xtrain)
    c = [1, 10, 100, 1000]
    for i in c:
        model = Lasso(alpha=1/(2*i)).fit(Xtrain_poly, ytrain)
        print("C: " + str(i))
        print("intercept: " + str(model.intercept_))
        print("coefficients: " + str(model.coef_))

def partIc():
    Xtrain, _, ytrain, _ = train_test_split(X, y, test_size=0.2)
    Xtrain_poly = PolynomialFeatures(5).fit_transform(Xtrain)
    Xtest_poly = PolynomialFeatures(5).fit_transform(Xtest)

    dummy = DummyRegressor(strategy="mean").fit(Xtrain_poly, ytrain)
    ydummy = dummy.predict(Xtest_poly)

    c = [1, 10, 100, 1000]
    for i in c:
        model = Lasso(alpha=1/(2*i)).fit(Xtrain_poly, ytrain)
        ypred = model.predict(Xtest_poly)
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        ax.plot_trisurf(Xtest[:, 0], Xtest[:, 1], ypred, color='g', alpha=.6)
        ax.scatter(X[:, 0], X[:, 1], y, color='b', label='Data')
        ax.set_xlabel('X1 features')
        ax.set_ylabel('X2 features')
        ax.set_zlabel('y target')
        ax.set_title('3D Data visualisation with Lasso model.\n C=' + str(i))
        handles, _ = ax.get_legend_handles_labels()
        patch = mpatches.Patch(color='g', label='Predictions', alpha=0.6)
        handles.append(patch)
        ax.legend(handles=handles)
        ax.view_init(20, 60)
        plt.show()
    # Dummy Baseline
    fig = plt.figure()
```

```
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_trisurf(Xtest[:, 0], Xtest[:, 1], ydummy, color='g', alpha=.6)
    ax.scatter(X[:, 0], X[:, 1], y, color='b', label='Data')
    ax.set_xlabel('X1 features')
    ax.set_ylabel('X2 features')
    ax.set_zlabel('y target')
    ax.set_title('Baseline model with dummy prediction')
    handles, _ = ax.get_legend_handles_labels()
    patch = mpatches.Patch(color='g', label='Predictions', alpha=0.6)
    handles.append(patch)
    ax.legend(handles=handles)
    ax.view_init(20, 60)
    plt.show()


def partIeb():
    Xtrain, _, ytrain, _ = train_test_split(X, y, test_size=0.2)
    Xtrain_poly = PolynomialFeatures(5).fit_transform(Xtrain)
    c = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
    for i in c:
        model = Ridge(alpha=1/(2*i)).fit(Xtrain_poly, ytrain)
        print("C: " + str(i))
        print("intercept: " + str(model.intercept_))
        print("coefficients: " + str(np.round(model.coef_, decimals=3)))
    return

def partIec():
    Xtrain, _, ytrain, _ = train_test_split(X, y, test_size=0.2)
    Xtrain_poly = PolynomialFeatures(5).fit_transform(Xtrain)
    Xtest_poly = PolynomialFeatures(5).fit_transform(Xtest)

    dummy = DummyRegressor(strategy="mean").fit(Xtrain_poly, ytrain)
    ydummy = dummy.predict(Xtest_poly)
    print(len(ydummy))

    c = [0.001, 0.1, 1, 10, 100, 1000]
    for i in c:
        model = Ridge(alpha=1/(2*i)).fit(Xtrain_poly, ytrain)
        ypred = model.predict(Xtest_poly)
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        ax.plot_trisurf(Xtest[:, 0], Xtest[:, 1], ypred, color='g', alpha=.6)
        ax.scatter(X[:, 0], X[:, 1], y, color='b', label='Data')
        ax.set_xlabel('X1 features')
        ax.set_ylabel('X2 features')
```

```
        ax.set_zlabel('y target')
        ax.set_title('3D Data visualisation with Ridge model.\n C=' + str(i))
        handles, _ = ax.get_legend_handles_labels()
        patch = mpatches.Patch(color='g', label='Predictions', alpha=0.6)
        handles.append(patch)
        ax.legend(handles=handles)
        ax.view_init(20, 60)
        plt.show()
    # Dummy Baseline
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_trisurf(Xtest[:, 0], Xtest[:, 1], ydummy, color='g', alpha=.6)
    ax.scatter(X[:, 0], X[:, 1], y, color='b', label='Data')
    ax.set_xlabel('X1 features')
    ax.set_ylabel('X2 features')
    ax.set_zlabel('y target')
    ax.set_title('Baseline model with dummy prediction')
    handles, _ = ax.get_legend_handles_labels()
    patch = mpatches.Patch(color='g', label='Predictions', alpha=0.6)
    handles.append(patch)
    ax.legend(handles=handles)
    ax.view_init(20, 60)
    plt.show()
    return

def partIIa():
    kf = KFold(n_splits=5)
    mean_error = []
    std_error = []
    Ci_range = [1, 5, 10, 25, 50, 100, 1000]
    for Ci in Ci_range:
        model = Lasso(alpha=1/(2*Ci))
        temp = []
        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_error.append(np.array(temp).mean())
        std_error.append(np.array(temp).std())
    plt.title("Lasso MSE vs C")
    plt.errorbar(Ci_range, mean_error, yerr=std_error)
    plt.xlabel('Ci')
    plt.ylabel('Mean square error')
    plt.xlim((0, 100))
    plt.show()
```

```
def partIIc():
    kf = KFold(n_splits=5)
    mean_error = []
    std_error = []
    Ci_range = [0.1, 0.5, 1, 5, 10, 25, 50, 100, 1000]
    for Ci in Ci_range:
        model = Ridge(alpha=1/(2*Ci))
        temp = []
        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_error.append(np.array(temp).mean())
        std_error.append(np.array(temp).std())
    plt.title("Ridge MSE vs C")
    plt.errorbar(Ci_range, mean_error, yerr=std_error)
    plt.xlabel('Ci')
    plt.ylabel('Mean square error')
    plt.xlim((0, 25))
    plt.ylim((0.1, 0.16))
    plt.show()

partIa()
partIb()
partIc()
partIeb()
partIec()
partIIa()
partIIc()
```