Machine Learning Week 3 Assignment - id:8-8-8

1 Lasso and Ridge Regression

(a) Figure 1 shows the scatter plot generated using the data with the features X_1, X_2, and y corresponding to the axes x, y, and z respectively. The scatter plot is shown from two angles, and the points are coloured with respect to the y feature (z axis). The colour bar shows the relation of colour to value of y. From the first angle it can be seen that the data lays on a curve. From the colour of the points it can be seen that this pattern is symmetrical with respect to the plane passing through $X_1 = 0$. Points with $|X_1|$ closer to 0 have lower y. From the second angle it can be seen that the y feature is also positively linearly correlated with the second feature. Points with higher $|X_2|$ have higher y. Finally, the data points have X values in the [-1,1] interval.

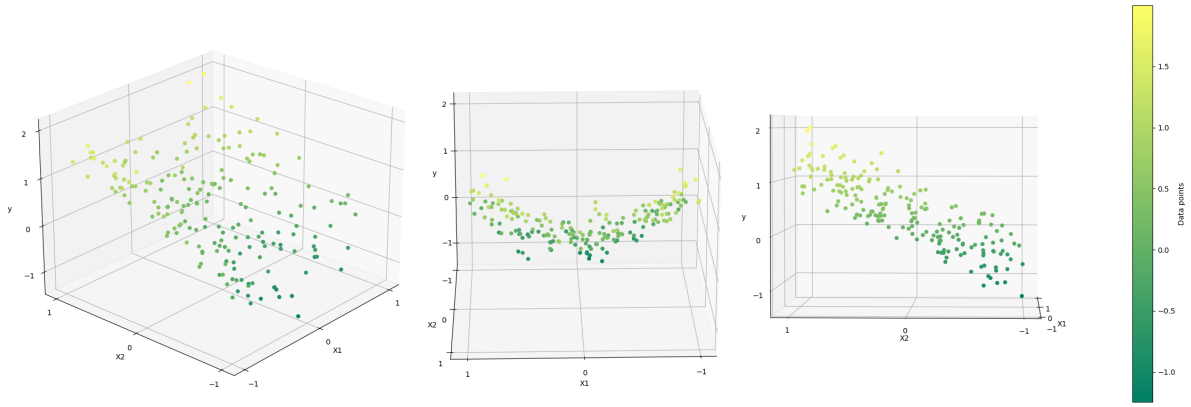

Figure 1: Data points plotted on X_1, X_2, y and colour-coordinated by y

(b) Table 1 shows the parameters of the trained Lasso Regression. The columns correspond to the (polynomial) feature and the rows correspond to the C values used. For readability, only data for features which were different from 0 in at least one instance are shown. All other features had coefficients equal to 0 for every C used. The first value for C was chosen so that all coefficients end up being zero. The intercept is 0.36. For $C = 10$ the only features used are X_2 and X_1^2. The coefficient for X_2 is positive. This is because, as stated earlier, y is positively linearly dependent on X_2. The coefficient for X_1^2 is also positive. This is because, as could be seen when looking at Figure 1, y is quadratically dependent on X_1^2, with the shape of the points distribution creating a convex curve. These two features seem to describe the data very well. This is encouraging, as this value for C was selected because it was the best value resulting from the cross-validation process which is presented in part 2 of the report. Finally, for $C = 1000$ there are 9 features included in the function for y. Most of these have very small coefficients, with the exception of X_2 and X_1^2. This is results in a function which behaves more or less similarly to the one generated with $C = 10$ on the [-1,1]x[-1,1] interval where the data point are, but which, as shown in figure 4, starts bending and twisting unnecessarily outside this range. This value for C was chosen as it is sufficiently big to cause a lot of non-zero coefficients and a function that is visibly overcomplicated.

	1	X_1	X_2	X_1^2	X_1X_2	$X_1X_2^2$	$X_1^3X_2$	$X_1^2X_2^2$	X_2^4	$X_1^3X_2^2$
C=1	0.3626	0	0	0	0	0	0	0	0	0
C=10	0.1961	0	0.8197	0.3715	0	0	0	0	0	0
C=1000	-0.0104	-0.0781	0.9941	0.9939	0.0270	0.1093	0.1247	0.0057	0.0072	0.1426

Table 1. *Parameters of the Lasso Regression Classifier with various C*

(c) Figures 2, 3, and 4 show the surface plots of the functions generated using the lasso regression with parameter C equal to 1, 10, and 1000 respectively. These plots correspond to each row of table 1 and offer a visual representation of the descriptions given in part (b). Each plot has a colour bar which serves as a legend for the two elements shown. The scatter plot is still being shown in hues of green-yellow, and the surface plot is shown in hues of pink-yellow. The colour bars show how the hues ranges of y-values which correspond to the different hues. of The first plot is simply a flat plane going through $y = 0.36$ which corresponds to all coefficients being set to 0. The second plot shows a surface which is curved and slanted at a similar and looks similar to the distribution of points on the scatterplot. The shape of this function does not change outside the [-1,1]x[-1,1] range. The third plot shows a surface which looks like the one in figure 3 on the [-1,1]x[-1,1] interval but as $|X_1|$ becomes larger it starts warping. Generally, it can be seen how the functions become more complex as C increases because the strength of regularisation decreases, allowing random variance in the data to be captured and amplified outside the range of the data points in the sample. Looking at the data points there is no reason to assume that the shape of the distribution would start changing and curving outside the [-1,1]x[-1,1] range. If more data points were added outside this range we would instead expect their y value to follow the pattern described at (a). If that were the case, the function generated using $C = 10$ would still fit the new data just as well. The predictions generated with $C = 1000$, however, would quickly deteriorate for these new data points, even though this model fits the existing data just as well as the $C = 10$ model.

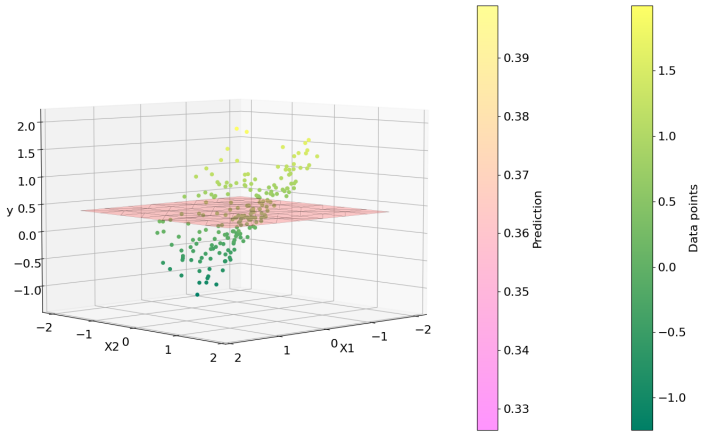

Figure 2: Surface plots showing target variable predictions for a lasso model with $C = 1$ alongside the plotted data points

(d) Under-fitting occurs when a model does not manage to capture the the trends and properties of a distribution sufficiently well, and instead ends up showing an over-simplified representation of the data. When this occurs, the coefficients of the features are too low, and the functions are too flat. This is what is happening in the $C = 1$ model. The coefficients are all set to 0 and the function is just a flat plane going through the middle of the data points. The only semblance of accuracy shown by this model is that it correctly identifies that the data points are all somewhere around $y = 0.36$. This is still better than a plane going, for example, through $y = 100$ and it's showing that the model is correctly identifying approximately where the data is, but doesn't manage to capture any of its properties. In the case of the $C = 1$ model, under-fitting is caused by a mistake on the part of the programmer (me). In setting C too low, the strong regularisation forces all coefficients to 0. However, under-fitting can also be caused by a data sample that is too small or biased to be representative of the population or the trend it is supposed to represent. For example, even

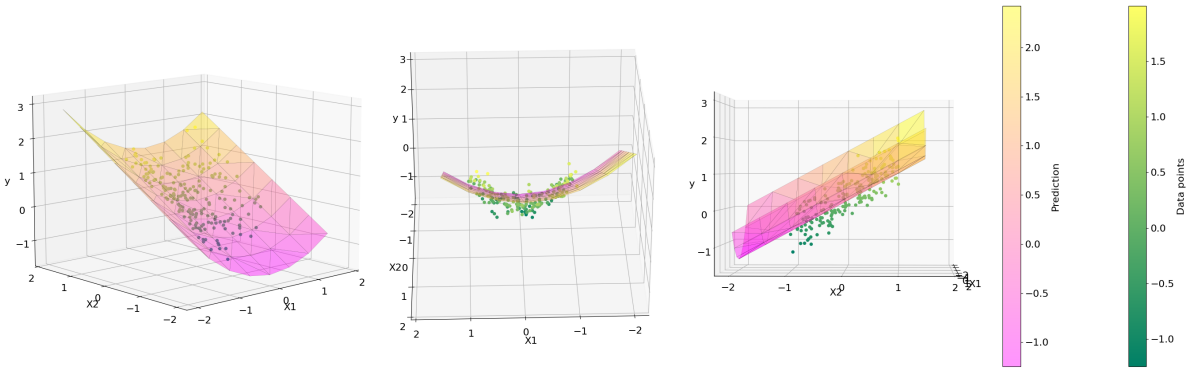

Figure 3: Surface plots showing target variable predictions for a lasso model with $C = 10$ alongside the plotted data points

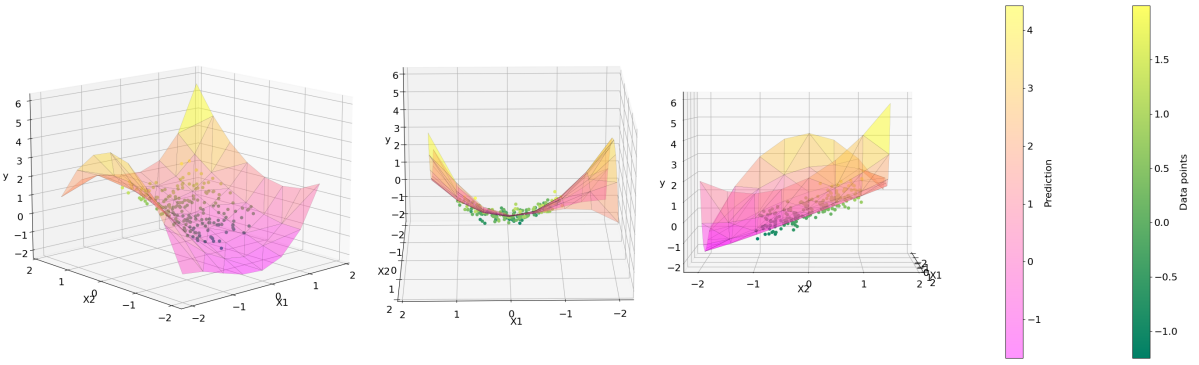

Figure 4: Surface plots showing target variable predictions for a lasso model with $C = 1000$ alongside the plotted data points

though I assume that the simple, elegant solution offered by the $C = 10$ model is the best one, that is not necessarily true. If this data set was a sample from a population that actually had a distribution similar to the function used by the $C = 1000$ model, then the $C = 10$ model would be under-fitting the data through no fault of my own. The assumption that this is not the case, and the preference for the simpler model is neccessary in order to satisfy Occam's razor. The between two models that fit the training data well, the simple model, which makes fewer assumptions about the population, is preferred. Violating this principle is what leads to over-fitting. A model that attempts to fit the training data perfectly is incorporating not only the general trends, but also the errors resulting from variance. Such a model would likely fail to generate accurate predictions for a test set different than the one it was trained on. Since the point of an ML model is to predict target values for new data whose real target values are not known in advance, failure resulted from overfitting renders a model useless. The model with $C = 1000$ is likely over-fitting the data and would probably and up performing poorly on a test values, especially outside the [-1,1]x[-1,1] range.

(e) Table 2 shows the parameters of the trained Ridge Regression. The columns correspond to the (polynomial) feature and the rows correspond to the C values used. The table is split into two sections for readability. The first value for C was chosen so that the coefficients are very small and the plot of target values predicted for the range [-1,1]x[-1,1] is flat. The intercept is the largest parameter, at 0.36, just like it was in the $C = 1$ lasso model. For $C = 0.03$, the largest coefficient is that of X_2, followed by X_1^2. Just like before, I chose to show the results for this value because it is the one chosen as the best after inspecting the cross-validation results. However, unlike with the Lasso Regeression, this time not only every feature has a non-zero coefficient, but the coefficients for X_2^3 and X_1^4 are almost as large as the one of X_1^2. $C = 30$ was chosen to exemplify over-fitting. X_2 and X_1^2 are still the most impactful features, both having coefficients above 1, with X_1^3, X_2^3, X_1^5, and $X_1^3 X_3^2$ all being well above 0 (in absolute value). Figure 5 shows the predictions generated by the $C = 0.00003$ model plotted alongside the training data. The plot

is visibly flat, and the model is under-fitting. Since the coefficients are not actually equal to 0, the plot does change its shape, but only well outside the range of the data set. Figure 6 shows the plot corresponding to the simplest well-performing model. It fits the data well but because of X_2^3, it starts curving excessively downwards and upwards for $X_2 < 0$ and $X_2 > 0$ respectively. Also, because of X_1^4, the slopes become excessively steep for $|X_1| > 1$. Figure 7 shows the plot of the $C = 30$ model.

	1	X_1	X_2	X_1^2	X_1X_2	X_2^2	X_1^3	$X_1^2X_2$	$X_1X_2^2$	X_2^3
C=0.00003	0.3617	-0.0003	0.0035	0.0009	-0.0002	0.0003	-0.0002	0.0012	0.0000	0.0020
C=0.03	0.1428	-0.0275	0.5878	0.3583	0.0231	-0.0041	-0.0242	0.1377	0.0255	0.2240
C=30	0.0022	0.0012	1.0600	1.0490	0.0666	-0.1594	-0.3506	0.0907	0.0928	-0.3469

	X_1^4	$X_1^3X_2$	$X_1^2X_2^2$	$X_1X_2^3$	X_2^4	X_1^5	$X_1^4X_2$	$X_1^3X_2^2$	$X_1^2X_2^3$	$X_1X_2^4$
C=0.00003	0.0007	-0.0002	0.0004	-0.0001	0.0003	-0.0002	0.0007	-0.0001	0.0007	0.0001
C=0.03	0.2591	0.0248	0.1070	0.0260	-0.0003	-0.0078	0.0555	0.0092	0.0433	0.0253
C=30	-0.1154	0.1404	0.1603	-0.0886	0.1410	0.2652	-0.0107	0.2667	-0.1235	-0.0141

Table 2. *Parameters of the Ridge Regression Classifier with various C*

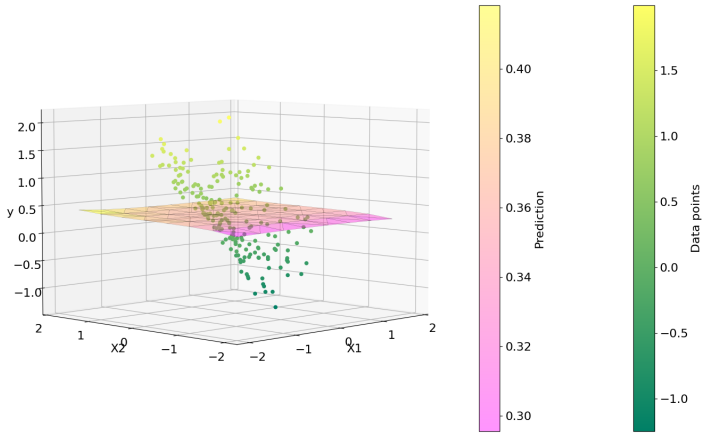

Figure 5: Surface plots showing target variable predictions for a ridge model with $C = 0.00003$ alongside the plotted data points

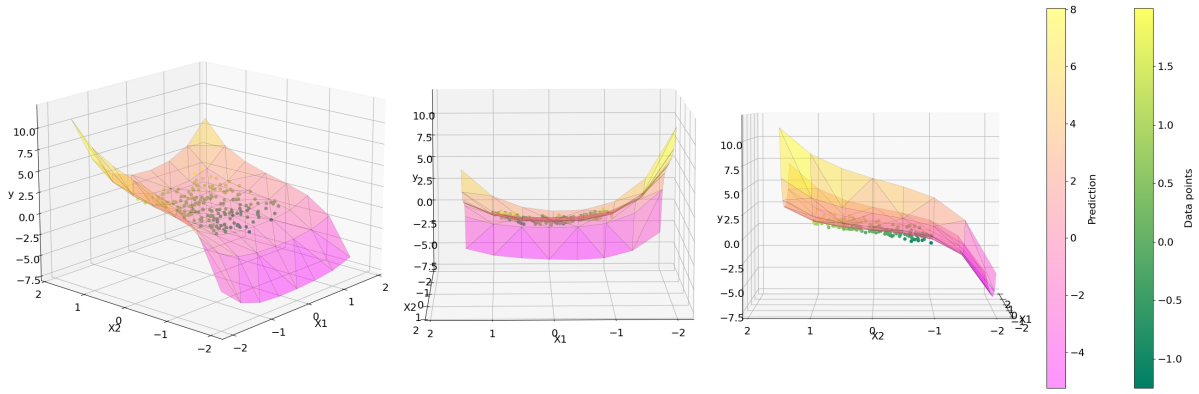

Figure 6: Surface plots showing target variable predictions for a ridge model with $C = 0.03$ alongside the plotted data points

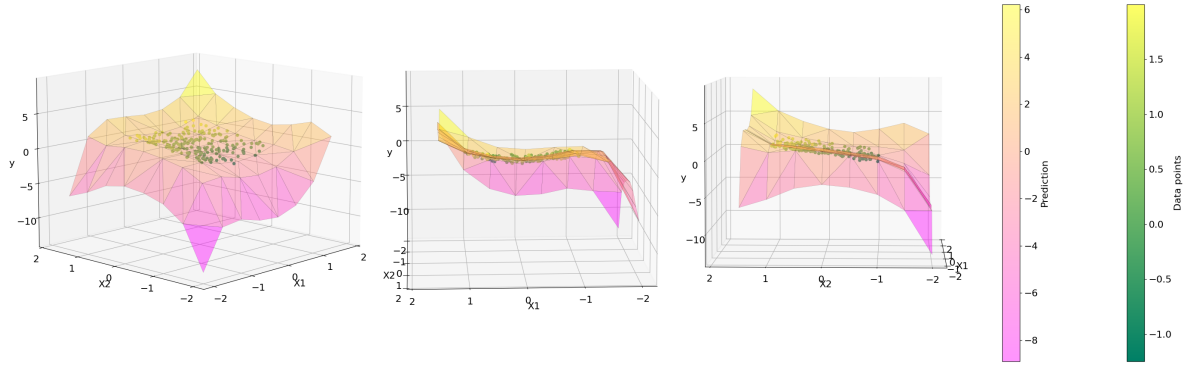

Figure 7: Surface plots showing target variable predictions for a ridge model with $C = 30$ alongside the plotted data points

2 Hyperparameter Tuning

(a) Figure 8 shows the plot of the MSE of the prediction errors for various values of C. I first took a large range of Cs and then narrowed it down close to the elbow of the plot line in order to find the best C.

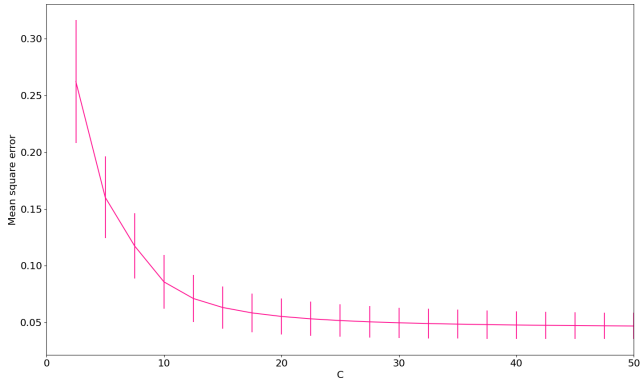

Figure 8: MSE of prediction errors for the Lasso Regression Classifier with various values of C

(b) The best value is the one which sufficiently minimises the MSE, but is small enough to not over-complicate the model. This value is corresponds to the elbow of the plot line. Looking at the figure, $C = 10$ seems to be a good value as it fulfils the criteria above.

(c) Figure 9 shows the plot of the MSE of the prediction errors for the Ridge Regression classifier for various values of C. I used the same process as described for (a) and applied the same principles. The value chosen was 0.03.

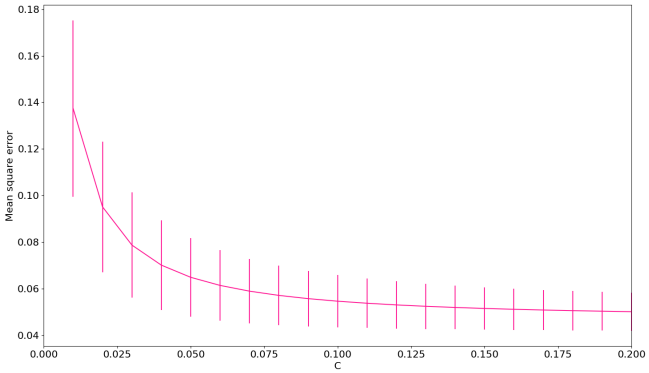

Figure 9: MSE of prediction errors for the Lasso Regression Classifier with various values of C

Appendix - Code

```
#id:8-8-8

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.colors as mcolors
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.linear_model import LogisticRegression, Lasso, Ridge
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error,
                            accuracy_score, f1_score,recall_score, precision_score
from sklearn.svm import LinearSVC
from sklearn.preprocessing import PolynomialFeatures
from IPython.display import display

#This method generates the scatter plot
def scatterPlot(X,y):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection = '3d')
    #Labeling axes and setting the ticks
    ax.set_xlabel('X1')
    ax.set_ylabel('X2')
    ax.set_zlabel('y')
    plt.xticks([-2, -1, 0, 1, 2])
    plt.yticks([-2, -1, 0, 1, 2])
    ax.set_zticks([-1, 0, 1, 2])
    #Plotting
    scatter = ax.scatter3D(X[:,0], X[:,1], y, c = (y),
                           cmap = plt.get_cmap('summer'),alpha = 1)
    #Adding a legend
    cbarS = plt.colorbar(scatter)
    cbarS.set_label('Data points')
    plt.show()

#This method trains the classifier, keeps track of parameters, and plots predictions
def classify(X,y,CRange,clf):
    # I am saving parameters to a dataframe to create coefficient tables
    Xdata = pd.DataFrame({
        'X1': X[:, 1],
        'X2': X[:, 2]})
    dfPoly = pd.DataFrame()
    for i, C in enumerate(CRange):
        #Training model
        clf.set_params(alpha=1 / (2 * C))
        clf.fit(X, y)
        #Saving coefficients
        d = dict(enumerate(clf.coef_.flatten(), 1))
        dfPoly = dfPoly.append(pd.DataFrame(d, index=['C=' + repr(C)]))
        #Saving intercept
        dfPoly.iloc[i, 0] = clf.intercept_
        #Plotting the prediction surface and data points
        Xtest = []
        for i in range(-20, 20, 5):
            for j in range(-20, 20, 5):
                Xtest.append([i / 10, j / 10])
```

```
        Xtest = np.array(Xtest)
        XtestPoly = poly.fit_transform(Xtest)
        yTest = clf.predict(XtestPoly)
        fig = plt.figure()
        plt.rcParams['font.size'] = '16'
        ax = fig.add_subplot(111, projection='3d')
        scatter = ax.scatter3D(X[:, 1], X[:, 2], y, c=(y),
                               cmap=plt.get_cmap('summer'), alpha=1)
        surface = ax.plot_trisurf(Xtest[:, 0], Xtest[:, 1], yTest,
                                  cmap=plt.get_cmap('spring'), alpha=0.42,
                                  edgecolor = 'black', linewidth = 0.2)
        #Adding legends
        cbarS=plt.colorbar(scatter)
        cbarF=plt.colorbar(surface)
        cbarS.set_label('Data points')
        cbarF.set_label('Prediction')
        #Setting labels and ticks
        ax.set_xlabel('X1')
        ax.set_ylabel('X2')
        ax.set_zlabel('y')
        plt.xticks([-2, -1, 0, 1, 2])
        plt.yticks([-2, -1, 0, 1, 2])
    plt.show()
    #Naming columns as their coresponding polynomial features and displaying the
     parameter dataframe

    dfPoly.columns = poly.get_feature_names_out(Xdata.columns)
    display(dfPoly)

#This method plots MSE for various Cs to select the best hyperparameter
def crossValidation(X,y,step,clf):
    mean_error = []
    std_error = []
    CRange = [x * step for x in range(1, 21)]
    for C in CRange:
        clf.set_params(alpha=1 / (2 * C))
        temp = []
        kf = KFold(n_splits=5)
        for train, test in kf.split(X):
            clf.fit(X[train], y[train])
            yPred = clf.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())
    fig = plt.figure()
    plt.rcParams['font.size'] = '16'
    plt.errorbar(CRange, mean_error, yerr=std_error,color = 'deeppink')
    plt.xlabel('C')
    plt.ylabel('Mean square error')
    plt.xlim(0, step*20)
    plt.show()

#Reading data from the .csv file and processing it
df = pd.read_csv("data.csv",header=None)
X1=df.iloc[:,0]
X2=df.iloc[:,1]
X=np.column_stack((X1,X2))
y=df.iloc[:,2]
```

```
#Creating the polynomial features
poly = PolynomialFeatures(degree=5)
XPoly = poly.fit_transform(X)

#Uncomment the desired classifier
clf,CRange,step = Lasso(), [1,10,1000], 2.5
#clf,CRange,step  = Ridge(), [0.00003,0.03,30],0.01


scatterPlot(X,y)
classify(XPoly,y,CRange,clf)
crossValidation(XPoly,y,step,clf)
```