Weekly Assignment 3 – Machine Learning

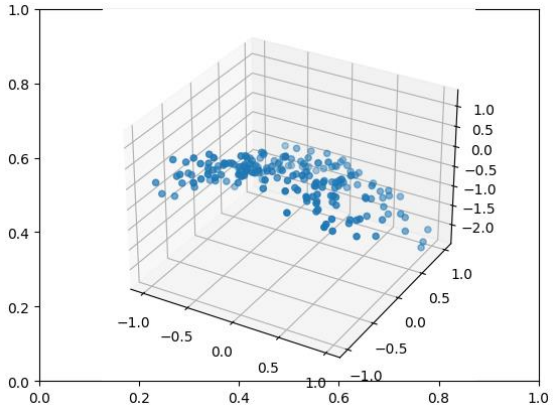
Paraic O'Reilly – 19335497 Data: # id:1--1--1

(i)

(a) Create a 3D plot.

The data for this assignment consisted of 3 columns. Two features X1 and X2 and a target variable y. Using the pandas python library, the csv file was stored in a data frame, then each column was put in a separate variable.

Using a python library, matplotlib.pyplot , the three variables were graphed on a 3D plot. X1 placed on the X-axis, X2 on the Y-axis, and y on the Z-axis. The resulting 3D plot is shown below.



The training data, takes the shape of a quadratic curve.

(b) Polynomial features, Training, and Parameters

Polynomial features were added using the function polynomials:

```
def polynomials(degree,X):
 poly = PolynomialFeatures(degree=degree)
 X_poly = poly.fit_transform(X)
 return X_poly
```

Once the new features were created a range of values for the penalty parameter C was made. Values from 0.001 to 10,000 were tested. This range of values was iterated through testing different values of C on the Lasso regression model as it tried to fit the newly created features (X_poly).

The intercept and model coefficients (which are automatically stored by the model) were then retrieved from each iteration of the model and stored. These are the parameters for the model. Smaller values of C are supposed to increase the cost penalty at each iteration which tends the parameters towards 0. Lasso regression as a model attempts to lower less important features towards 0, so features that are better predictors are identified.

The parameters for each iteration of C are stored in the table below.

Parameters (0,10)

C	θ_0	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_8	θ_9	θ_{10}
0.01	-0.311	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.1	-0.311	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	-0.311	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	-0.201	0.0	0.0	-0.867	-0.379	0.0	0.0	0.0	0.0	0.0	0.0
100	0.043	0.0	-0.0	-0.997	-0.778	-0.0	-0.0	-0.0	-0.0	-0.021	-0.004
1000	-0.017	0.0	0.042	-0.961	-0.782	0.0514	-0.049	0.0	0.0	-0.159	-0.070
100000	-0.005	0.0	0.051	-0.845	-0.775	0.11	-0.123	0.0,	0.077	-0.228	0.612

Parameters (11, 21)

C	θ_{11}	θ_{12}	θ_{13}	θ_{14}	θ_{15}	θ_{16}	θ_{17}	θ_{18}	θ_{19}	θ_{20}	θ_{21}
0.01	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
100	-0.163	-0.052	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1000	-0.206	-0.256	-0.0	0.075	0.0	0.0	-0.031	-0.003	0.0	0.0	0.0
10000	-0.215,	-0.346	0.015	0.078	0.087	0.039	-0.162	-0.088	0.06	0.113	0.456

There is no change in parameter values for the iterations where C is 0.01, 0.1 or 1, as the model has reduced all parameters to 0.

From 10 to 10,000, there are slight increases in parameters. Parameters 3 and 4 are the only relevant parameters for 10 and are relatively high. As the values of C increase other parameters slowly increase however none are ever as high as 3 and 4. This is because parameters 3 and 4 are enough to accurately fit the data, however as the values for C increase and there is less regularisation, irrelevant features increase, and the model begins to overfit. This is visible in the minute increases of other parameters and in the graphs of the next section.

(c) Generating and graphing predictions

Predictions were generated using the function create_test. The function is show below.

```
grid = np.linspace(-3,3) #-5,5
X_test = []
for i in grid:
 for j in grid:
   X_test.append([i, j])
X_test = np.array(X_test)
X_test_poly = polynomials(5,X_test)
return X_test, X_test_poly
```

This creates predictions on a grid using a nested for loop. As the dataset for the first feature had a range of [-1,1], the grid generated values from [-3,3] to test the robustness of the model. Polynomial features were created for up to a power of 5 from these test values and the model then attempts to predict them. This is then displayed against the training data for the model. The training data is represented by a scatter plot of red dots and the generated predictions are represented by a see through blue surface plot.

As the value of C increases, we can see the model gradually begin to overfit, as regularisation becomes less prevalent.

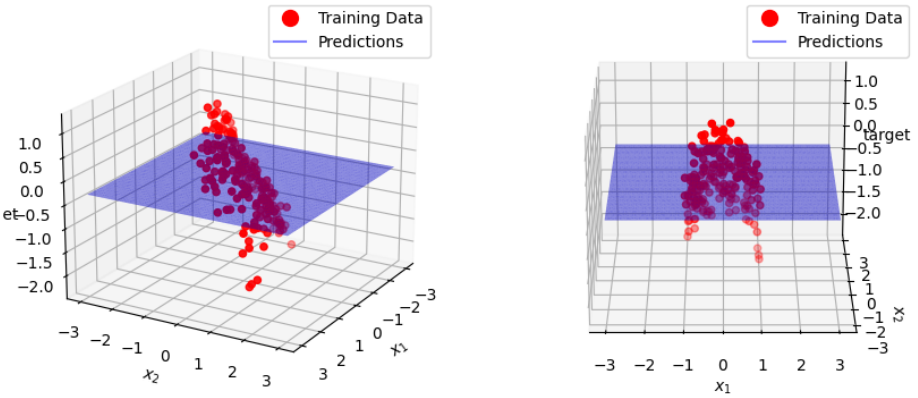
Initially, the model is underfitting at C=1. The predictions are a flat plane and do not capture the dimensionality of the data.

At C=10, we see the data predictions match the shape of the data. Here, the only non-zero parameters are 3 and 4 and both have relatively large values.

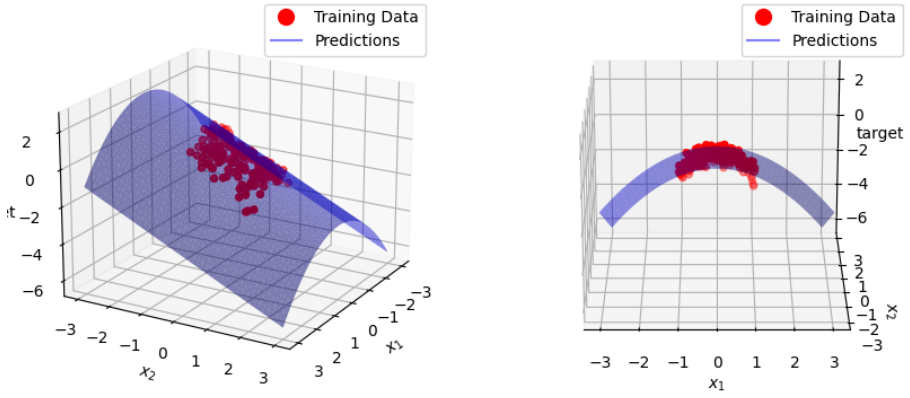
For values of C=100, 1000, 10,000 , we begin to see the affects of overfitting. The higher values of C do not eliminate lesser features by reducing them to 0. This leads to spurious parameters increasing and this is seen in the odd curves and valleys in the predictions that do not fit the original training data at all.

The graphs for each value of C are displayed below.

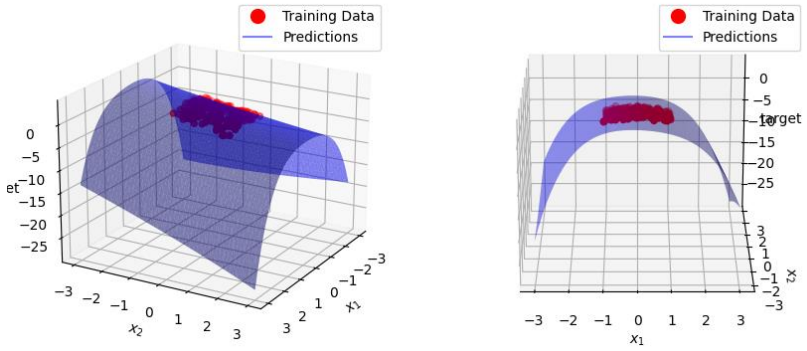
C <= 1



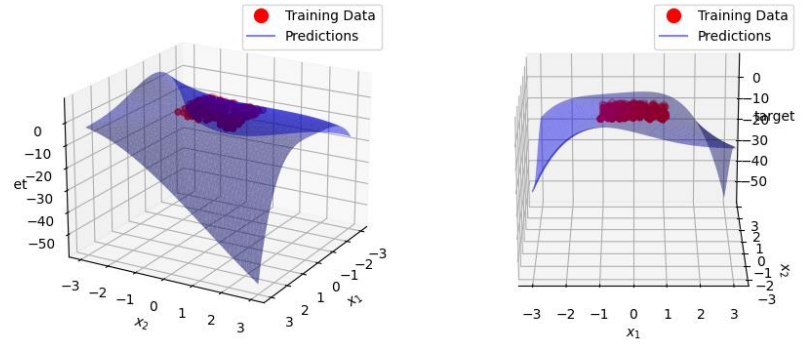
C = 10



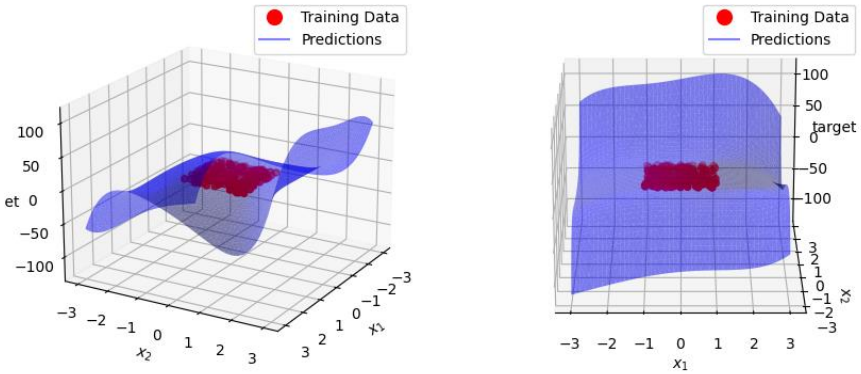
C=100



C=1000



C=10000



(d)

Overfitting occurs when the model fits to the training data too well and captures spurious trends or noise in the data, which are not part of the underlying patterns that govern the data.

Underfitting is when the model fails to capture the underlying pattern of the data. This is because the model is too simple and fails to grasp the complexity of the data.

The penalty parameter C helps balance underfitting and overfitting by managing regularisation. A high C value will minimise regularisation whereas a low value for C has high regularisation. For Lasso Regression models, lower values of C tend the parameters towards 0. This helps combat overfitting as it generalises the model more as parameters are reduced and less are used. This is known as feature selection. This risks underfitting however as losing too many parameters may make the model too simple. Higher values of C combat underfitting as the model can fit to the data more closely, although this risks capturing noise.

This can be seen in both the table of parameters and data visualisations above. The graph for C=1, shows a flat plane as the model is underfitting. This correlates with the table as all features bar the bias are 0. At C=10, the model is capturing the underlying pattern of the data well, and the training data's visualisation aligns with the prediction's visualisations. Three and four are the only parameters that are non-zero as they are the only needed features. As C increases and regularisation drops, other parameters start incrementally increasing resulting in the model overfitting.

(e)

Using the same function that created the Lasso model, but with a Ridge model called and a different range of selected values for C the following parameters were generated. A slightly different range was selected for the Ridge model, as it does not perform feature selection, but it will reduce some features to near 0. Therefore, lower values of C were selected in the range so the transition between the model underfitting and then overfitting the training data could be visualised. The starting value chosen was 0.0001, as it gave a near flat plain for predictions. After that, values went up by a factor of 10 to view the effects of overfitting.

The resulting parameter values are displayed in the tables below.

Parameters (0 – 10)

C	θ_0	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_8	θ_9	θ_{10}
0.00001	-0.3109	0.0	0.0	-0.001	0.0	-0.0	0.0	0.0	0.0	0.0	0.0
0.0001	-0.3097	0.0	0.0	-0.013	-0.003	0.0	0.0	0.0	-0.004	0.0	0.008
0.001	-0.298	0.0	0.004	-0.111	-0.03	0.002	0.0	0.0	-0.035	-0.004	-0.066
0.01	-0.216	0.0	0.023	-0.434	-0.196	-0.001	-0.006	0.006	-0.118	-0.023	-0.227
.1	0.088	0.0	0.35	-0.751	-0.462	0.004	-0.02	0.02	-0.107	-0.072	-0.241
1	-0.033	0.0	0.033	-0.887	-0.646	0.059	-0.077	0.037	-0.024	-0.134	-0.277
10	-0.009	0.0	0.48	-0.862	-0.75	0.106	-0.127	0.011	0.08	-0.228	-0.533

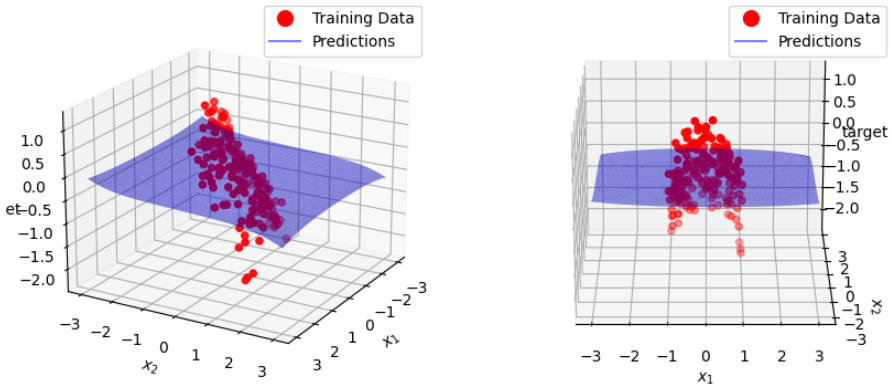
Parameters (11 – 21)

C	θ_{11}	θ_{12}	θ_{13}	θ_{14}	θ_{15}	θ_{16}	θ_{17}	θ_{18}	θ_{19}	θ_{20}	θ_{21}
0.00001	-0.001	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0.001
0.0001	-0.003	0.0	-0.001	0.0	0.0	0.0	-0.003	0.0	-0.003	0.0	-0.006
0.001	-0.027	-0.003	-0.012	0.001	0.002	-0.002	-0.022	-0.003	-0.021	-0.004	-0.047
0.01	-0.172	-0.024	-0.075	-0.001	0.004	-0.006	-0.067	-0.016	-0.059	-0.022	-0.148
.1	0.363	-0.092	-0.138	0.015	0,014	-0.001	-0.04	-0.046	0.01	-0.051	0.054
1	0.321	-0.252	-.069	0.075	0.054	0.008	-0.073	-0.104	0.076	0.018	0.134
10	0.239	-0.34	-0.024	0.083	0.094	0.038	-0.171	-0.114	0.075	0.127	0.386

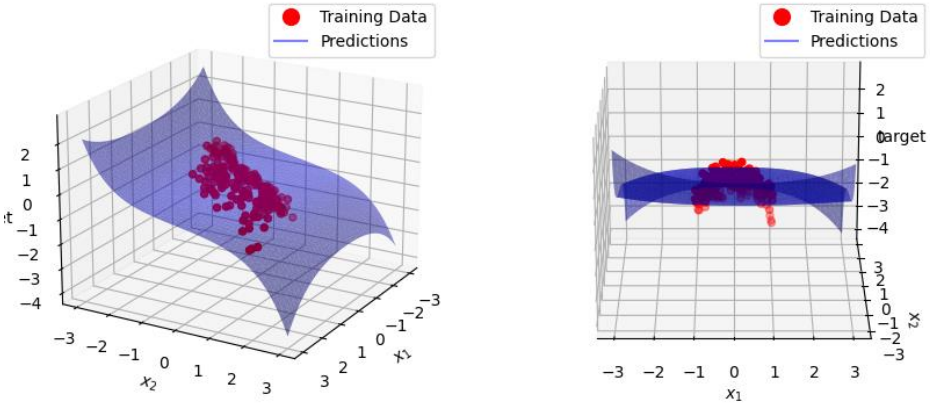
The predictions and training data were graphed using the same functions from the previous sections, the only difference being the updated range of values for C and the Ridge model.

The graphs are shown below.

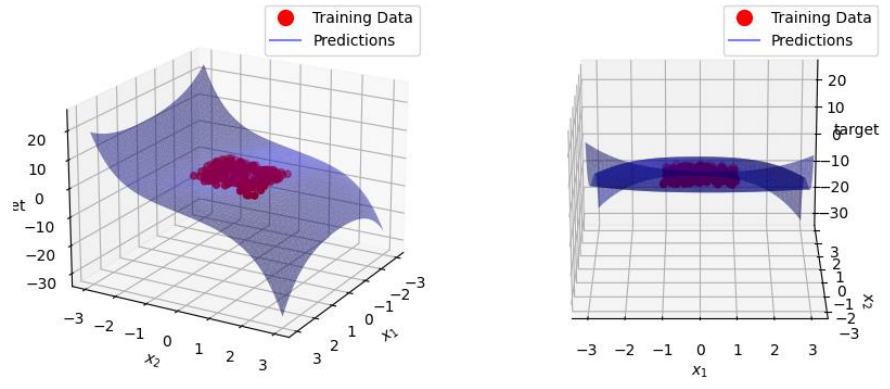
C = 0.00001



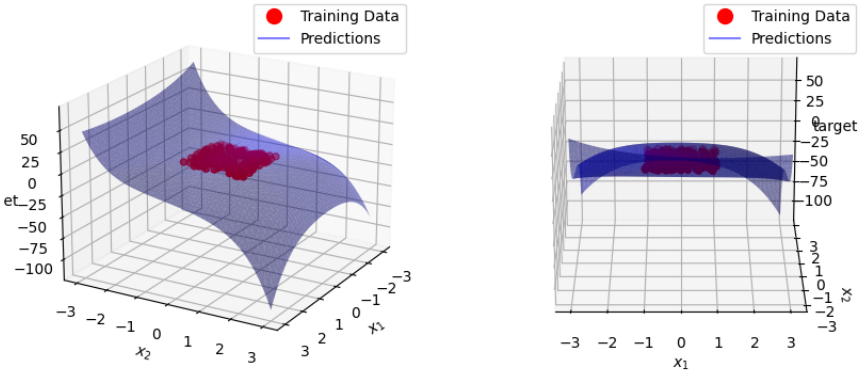
C = 0.0001



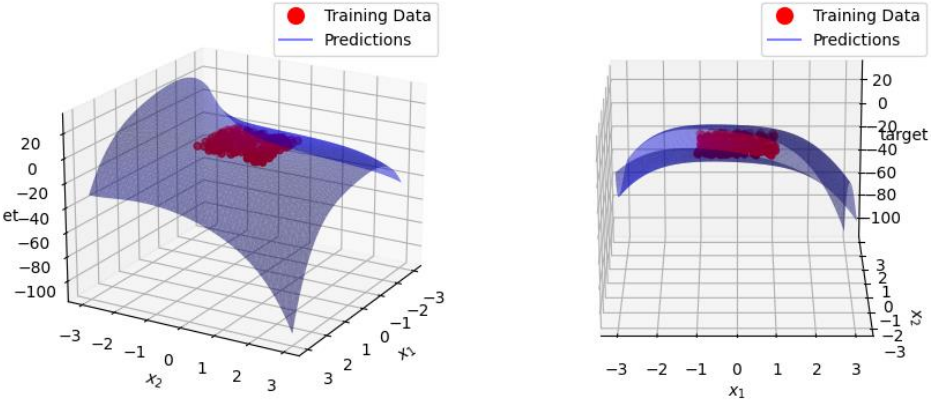
C = 0.001



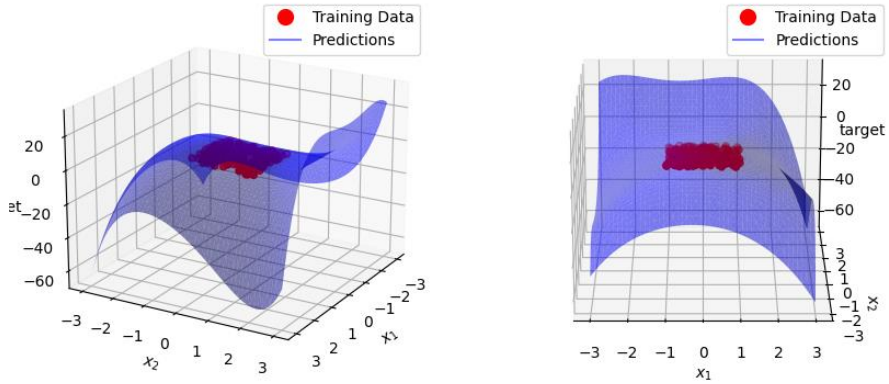
C = 0.01


Training Data
Predictions
Training Data
Predictions

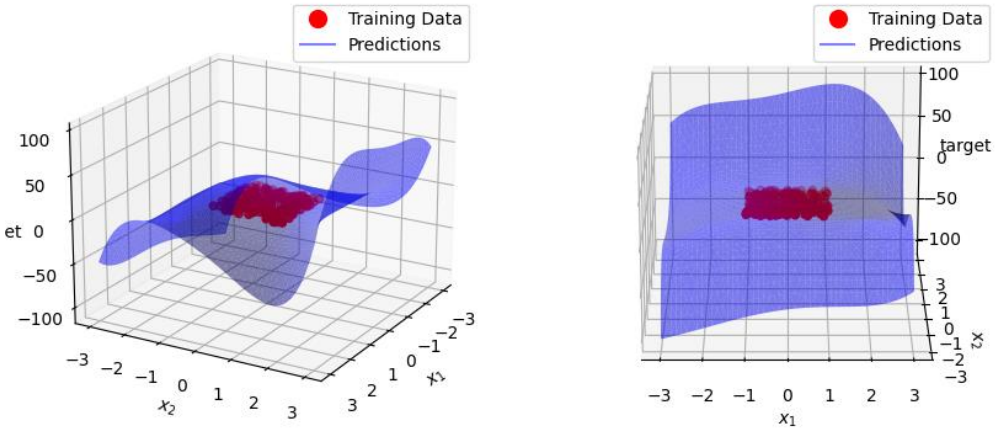
C = 0.1


Training Data
Predictions
Training Data
Predictions

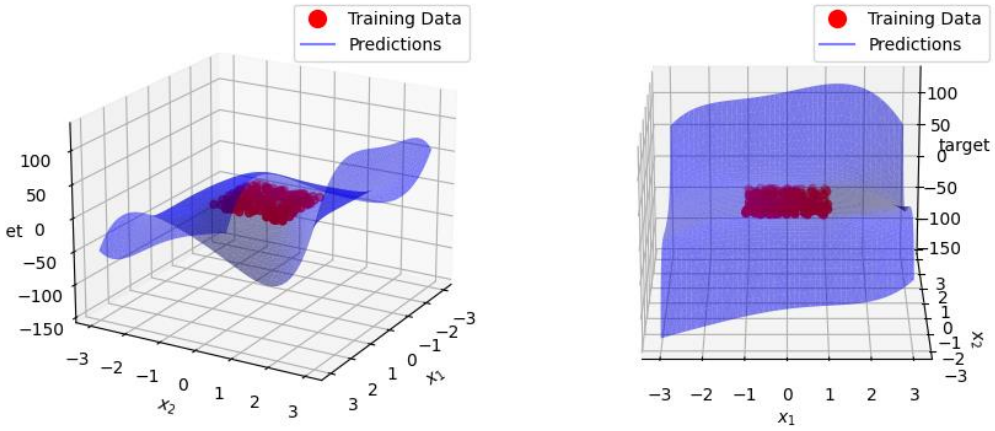
C = 1


Training Data
Predictions
Training Data
Predictions

C= 10



C = 100



It is slightly more difficult to tell which model best fits the data here then with the Lasso model here. It appears when C=1, or C=10 the model predictions visually match the data the most however there is more overfitting than the Lasso visualisations. This aligns with the recorded parameter values as the recorded values for parameters 3 and 4 are highest which was the same for the highest performing Lasso model. However, as half the other parameters are non-zero and the other have are significant (higher than +/- 0.1), the model is fitting too closely.

(ii)

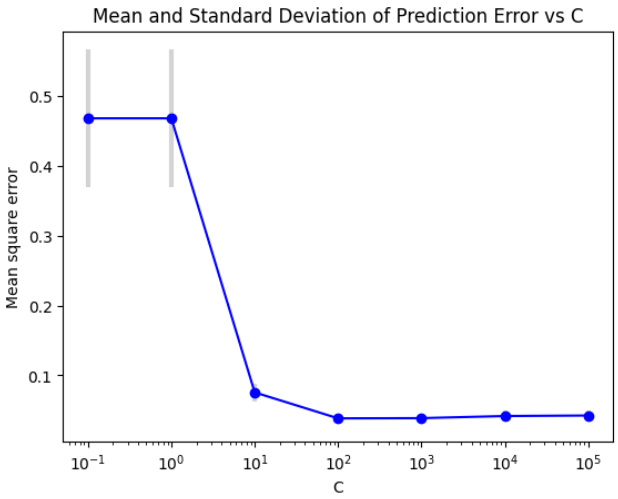
(a) Lasso Error Bar

For this section K-cross validation with 5 was used to help select a value for C. Different values of C were tested on a model using cross validation. At each iteration the mean and standard deviation were stored.

```
for C in parameters:
  model = Lasso(alpha=(1/(2*C)))
```

```
temp = []
kf = KFold(n_splits=5)
for train, test in kf.split(X_poly):
  model.fit(X_poly[train], y[train])
  ypred = model.predict(X_poly[test])
```

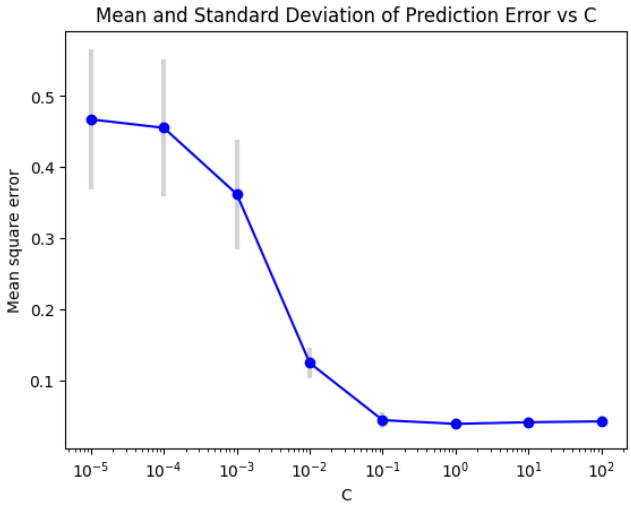
The values of C were chosen by starting at the first value of C to reduce all parameters to 0 and then to increase upward by a factor of 10 until there was only minor changes in the graph.



The value of C, I would choose is 100, as it minimises the mean squared error the most. However, based on our visualisations from earlier sections, C=100 was overfitting slightly and C=10 was matched the training data better.

(b) Ridge Error Bar

The same functions were used to generate the mean squared error and standard deviation using 5-fold cross validation although Ridge regression models were used and a different range of values for C were chosen.



With the Ridge model, the value of C, I would use is C =1 as it has the lowest error. This doesn't match the visualisations plotted earlier. Visually, C=0.1 had the least overfitting and matched the pattern of the data better. However, C=1 has less overfitting then the value selected by cross validation of the lasso model.

Appendix

```
import pandas as pd
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt

# read in data
df = pd.read_csv("data.csv")
# check if the data loaded in correctly
print(df.head())

#Store the data in features and target variables
X1 = df.iloc[:,0]
X2 = df.iloc[:,1]
y = df.iloc[:,2]

# Create the plot
fig , ax = plt.subplots()

# Load variables into the 3D scatter plot
ax = plt.axes(projection ="3d")
ax.scatter3D(X1,X2,y,)
```

```
from sklearn.linear_model import Lasso, Ridge
from sklearn.preprocessing import PolynomialFeatures
```

```
def plot(X1, X2, X_test, y, predictions):
    # Combine X1 and X2
    X = np.column_stack((X1, X2))

    # Create a grid for generating predictions
    grid_size = 100
    grid = np.linspace(-5, 5, grid_size)


    # Training data:
    fig = plt.figure(figsize=(10, 8))

    # First plot
    ax1 = fig.add_subplot(121, projection='3d')
    ax1.scatter(X1, X2, y, c='red', label = 'Training Data')
    #vmin=-5, vmax=5, color='#0000ff80'
    ax1.plot_trisurf(X_test[:,0], X_test[:,1], predictions, vmin=-100,
vmax=100, color='#0000ff80',label = 'Predictions on Test Data')
    ax1.zaxis.set_rotate_label(False)
    ax1.set_xlabel('$x_1$')
    ax1.set_ylabel('$x_2$')
    ax1.set_zlabel('target', rotation=0)
    ax1.view_init(elev=20, azim=30)  # Set the view angle for the first plot
    #ax1.legend()

    # Second plot
    ax2 = fig.add_subplot(122, projection='3d')
    ax2.scatter(X1, X2, y, c='red', label = 'Training Data')
    ax2.plot_trisurf(X_test[:,0], X_test[:,1], predictions, vmin=-100,
vmax=100, color='#0000ff80',label = 'Predictions on Test Data')
    #ax2.plot_trisurf(X1, X2, predictions, vmin=-5, vmax=5, color='#0000ff80')
    ax2.zaxis.set_rotate_label(False)

    ax2.set_xlabel('$x_1$')
    ax2.set_ylabel('$x_2$')
    ax2.set_zlabel('target', rotation=0)
    ax2.view_init(elev=20, azim=-90)  # Set a different view angle for the
second plot

    proxy_artists = [plt.Line2D([0], [0], linestyle="none", marker='o',
markersize=10, color='red'),
                plt.Line2D([0], [0], linestyle="-", color='#0000ff80')]

    # Add legend to the plots
    ax1.legend(proxy_artists, ['Training Data', 'Predictions'])
    ax2.legend(proxy_artists, ['Training Data', 'Predictions'])
```

```
# Creates a new variable with polynomial features of X
def polynomials(degree,X):
 poly = PolynomialFeatures(degree=degree)
 X_poly = poly.fit_transform(X)
 return X_poly

def Create_Test():
 grid = np.linspace(-3,3) #-5,5
 X_test = []
 for i in grid:
  for j in grid:
    X_test.append([i, j])
 X_test = np.array(X_test)
 X_test_poly = polynomials(5,X_test)
 return X_test, X_test_poly

# Trains lasso models with different values for C, calls previous two
functions
# Prints paramater values and plots results vs training data
def train_lasso(X1,X2,y):
    #Create variables
    X = np.column_stack((X1,X2))
    X_poly = polynomials(5,X)
    X_test, X_test_poly = Create_Test()

    #range of regularisation terms to be tested
    test_values = [0.001, 0.01,1, 10, 100, 1000,10000]

    print("Lasso Result:\n")
    #Loop through each test value creating a new model for each
    for C in test_values:
     #Fit the model to X and Y and set regularisation term
     model = Lasso(alpha=1/(2*C))
     model.fit(X_poly,y)
     predictions = model.predict(X_test_poly)

     #Store the paramaters the intercept
     print(f"Paramaters for alpha = {C}",)
     print(list(model.coef_))
     print(f"Intercept = {model.intercept_}\n")

     print(f"Plot for C = {C}")
     plot(X1,X2,X_test,y,predictions)

     plt.show()

#Does the same as the lasso function except uses the Ridge regression model
def train_ridge(X1,X2,y):
```

```
 #Create variables
 X = np.column_stack((X1,X2))
 X_poly = polynomials(5,X)
 X_test, X_test_poly = Create_Test()

 #range of regularisation terms to be tested
 test_values = [0.00001,0.0001,.001,.01,.1,1,10,100]

 #Loop through each test value creating a new model for each
 print("Ridge results:\n")
 for C in test_values:
    #Fit the model to X and Y and set regularisation term
    model = Ridge(alpha=(1/(C*2)))
    model.fit(X_poly,y)
    predictions = model.predict(X_test_poly)

    #Store the paramaters the intercept
    print(f"Paramaters for alpha = {C}",)
    #print(list(model.coef_))

    coefficients = list(model.coef_)
    rounded_coefficients = [round(coef, 3) for coef in coefficients]
    print(rounded_coefficients)
    print(f"Intercept = {model.intercept_}\n")

    print(f"Plot for C = {C}")
    plot(X1,X2,X_test,y,predictions)
```

```
train_lasso(X1,X2,y)
train_ridge(X1,X2,y)
```

```
# - Cross Validation to decide hyperparamater C
# Uses 5 fold CV to plot C vs error bar.
# Plot mean and standard deviation of prediction error vs C

from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import mean_squared_error

#Ridge regression version
def test_for_C_R(X1,X2,y):
 #Create variables
 X = np.column_stack((X1,X2))
 mean_error = []
 std_error = []
 X_poly = polynomials(5,X)
 X_test, X_test_poly = Create_Test()
```

```
 #Range of paramaters to be tested
 parameters = [0.00001,0.0001,.001,.01,.1,1,10,100,1000]

 #Loop through testing models
 for C in parameters:
  model = Ridge(alpha=(1/(2*C)))
  temp = []
  kf = KFold(n_splits=5)
  for train, test in kf.split(X_poly):
    model.fit(X_poly[train], y[train])
    ypred = model.predict(X_poly[test])
    temp.append(mean_squared_error(y[test], ypred))
  mean_error.append(np.array(temp).mean())
  std_error.append(np.array(temp).std())

 plt.errorbar(parameters, mean_error, yerr=std_error, fmt='o-', color='b',
ecolor='lightgray', elinewidth=3, capsize=0)
 plt.xlabel('C')
 plt.ylabel('Mean square error')
 plt.xscale('log')
 plt.title('Mean and Standard Deviation of Prediction Error vs C')
 plt.show()

#Lasso regression version
def test_for_C_L(X1,X2,y):
 X = np.column_stack((X1,X2))

 mean_error = []
 std_error = []
 X_poly = polynomials(5,X)
 X_test, X_test_poly = Create_Test()

 mean_error = []
 std_error = []
 parameters = [0.1, 1, 10, 100, 1000, 10000,100000]
 for C in parameters:
  model = Lasso(alpha=(1/(2*C)))
  temp = []
  kf = KFold(n_splits=5)
  for train, test in kf.split(X_poly):
    model.fit(X_poly[train], y[train])
    ypred = model.predict(X_poly[test])
    temp.append(mean_squared_error(y[test], ypred))
  mean_error.append(np.array(temp).mean())
  std_error.append(np.array(temp).std())
```

```
 plt.errorbar(parameters, mean_error, yerr=std_error, fmt='o-', color='b',
ecolor='lightgray', elinewidth=3, capsize=0)
 plt.xlabel('C')
 plt.ylabel('Mean square error')
 plt.xscale('log')
 plt.title('Mean and Standard Deviation of Prediction Error vs C')
 plt.show()
```

```
#Call functions
print("Error Bar for Lasso Regression:\n")
test_for_C_L(X1,X2,y)
print("Error Bar for Ridge Regression:\n")
test_for_C_R(X1,X2,y)
```