CSU44061 Machine Learning - Week 3

Faith Olopade - 21364066
Dataset # id:21-42--21

Appendix contains all the code.

Question (i)

(a): 3D Scatter Plot

Based on the 3D scatter plot of the training data, it appears that the data points follow a curved distribution rather than lying flat on a plane. This suggests a non-linear relationship between the features and the target variable, indicating that a simple linear model may not fully capture the complexity of the data. The curvature is particularly evident when viewing the plot from different angles.

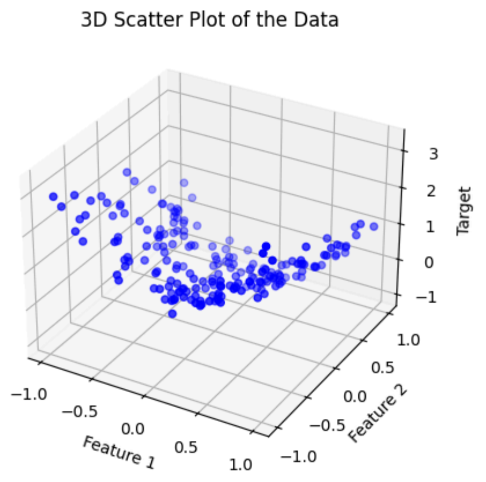

Figure 1: Scatter Plot of Training Data

(b): Lasso Regression Model

Lasso regression models were trained on the dataset along with polynomial features up to a degree 5 for the two input features . I used the PolynomialFeatures function from the sklearn package. The penalty values this lasso regression model was trained on the data for were [0.001, 1, 10,100, 1000]. The primary goal was to examine how the model coefficients change as C is varied and how polynomial feature complexity influences model behaviour.

Degree of Polynomial Feature = 1

	C	Coefficients of the Lasso Model	Intercept
1	0.001	0.0, -0.0, -0.0	0.600244

2	1	0.0, -0.0, -0.0	0.600244
3	10	0.0, -0.0, -0.7958632256359787	0.638911
4	100	0.0, -0.012125641385914388, -0.9350593326648267	0.645330
5	1000	0.0, -0.026641530029232193, -0.9489526789255559	0.645593

We find that the coefficients of the linear terms change very little when C rises from 0.001 to 1000. For low values of C, the coefficients stay close to zero; however, as C increases, the magnitude of the coefficients for the feature with higher power slightly increases. This shows that the model can fit more intricate patterns in the data as C is larger.

Degree of Polynomial Feature = 2

	C	Coefficients of the Lasso Model	Intercept
1	0.001	.0, -0.0, -0.0, 0.0, 0.0, 0.0	0.600244
2	1	0.0, -0.0, -0.0, 0.0, 0.0, 0.0	0.600244
3	10	0.0, -0.0, -0.8244545457357171, 1.4647286621471356, 0.0, 0.0	0.185942
4	100	0.0, -0.0036315817140275385, -0.9741594466703591, 2.0066249018789315, 0.0, 0.015211573744767054	0.020063
5	1000	0.0, -0.022163873617204285, -0.989054646254416, 2.054298844109457, 0.012803047759990702, 0.0701333817900854	-0.012400

The model provides quadratic interactions between features for degree 2 polynomials as C grows. Non-zero coefficients are seen for terms such as the square of the first feature and the interaction between the two initial features. This suggests that nonlinear interactions between the input variables are starting to be captured by the model.

Degree of Polynomial Feature = 3

	C	Coefficients of the Lasso Model	Intercept
1	0.001	0.0, -0.0, -0.0, 0.0, 0.0, 0.0, -0.0, -0.0, 0.0, -0.0	0.600244
2	1	0.0, -0.0, -0.0, 0.0, 0.0, 0.0, -0.0, -0.0, 0.0, -0.0	0.600244
3	10	0.0, -0.0, -0.8244545457357171, 1.4647286621471356, 0.0, 0.0, -0.0, -0.0, 0.0, -0.0	0.185942
4	100	0.0, -0.003631729288307697, -0.9741595822339331, 2.006624662480273, 0.0, 0.01521162705437309, -0.0, -0.0, -0.0, -0.0	0.020063
5	1000	0.0, -0.03811326206042145, -0.9910990944167374, 2.052410589236268, 0.01370401597403219, 0.07011950653410587, 0.01483432818409136, 0.005064186599239848, 0.020212317766216754, -0.0	-0.012315

Degree of Polynomial Feature = 4

	C	Coefficients of the Lasso Model	Intercept
1	0.001	0.0, -0.0, -0.0, 0.0, 0.0, 0.0, -0.0, -0.0, 0.0, -0.0, 0.0, 0.0, 0.0, 0.0, 0.0	0.600244
2	1	0.0, -0.0, -0.0, 0.0, 0.0, 0.0, -0.0, -0.0, 0.0, -0.0, 0.0, 0.0, 0.0, 0.0, 0.0	0.600244
3	10	0.0, -0.0, -0.8244545457357171, 1.4647286621471356, 0.0, 0.0, -0.0, -0.0, 0.0, -0.0, 0.0,	0.185942

		0.0, 0.0, 0.0, 0.0	
4	100	0.0, -0.003631729288307697, -0.9741595822339331, 2.006624662480273, 0.0, 0.01521162705437309, -0.0, -0.0, -0.0, -0.0, 0.0, 0.0, 0.0, 0.0, 0.0	0.020063
5	1000	0.0, -0.03469183958184042, -0.9916871684805458, 1.9882840667825594, 0.025644849582728006, 0.03250477665457457, 0.005440562047094556, 0.011799831426831057, 0.01880307602930709, 2480273, 0.0, 0.01521162705437309, -0.0, -0.0, -0.0, -0.0, 0.0, 0.0, 0.0, 0.0, 0.0	0.020063

Degree of Polynomial Feature = 5

	C	Coefficients of the Lasso Model	Intercept
1	0.001	0.0, -0.0, -0.0, 0.0, 0.0, 0.0, -0.0, -0.0, 0.0, -0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.0, -0.0, 0.0, -0.0, 0.0, -0.0	0.600244
2	1	0.0, -0.0, -0.0, 0.0, 0.0, 0.0, -0.0, -0.0, 0.0, -0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.0, -0.0, 0.0, -0.0, 0.0, -0.0	0.600244
3	10	0.0, -0.0, -0.8244545457357171, 1.4647286621471356, 0.0, 0.0, -0.0, -0.0, 0.0, -0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.0, -0.0, 0.0, -0.0, 0.0, -0.0	0.185942
4	100	0.0, -0.003631729288307697, -0.9741595822339331, 2.006624662480273, 0.0, 0.01521162705437309, -0.0, -0.0, -0.0, -0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.0, -0.0, 0.0, -0.0, 0.0, -0.0	0.020063
5	1000	0.0, -0.04035542912942993, -0.991815796530457, 1.998978200046954, 0.022545690587487646, 0.03498666541576832, 0.0, 0.008425430309599897, -0.0, -0.0, 0.015597278214036213, -0.014646, 0.03498666541576832, 0.0, 0.008425430309599897, -0.0, -0.0, 0.015597278214036213, -0.014548835338916188, 0.10294095409680215, -0.0, -0.0, -0.0, -0.0, 0.07306320822347685, -0.0, -0.0548835338916188, 0.10294095409680215, -0.0, -0.0, -0.0, -0.0, 0.07306320822347685, -0.0, -0.0, 0.0	0.001243

For polynomials of degree 3 and above, the model adds more interaction terms (such as cubic and quartic powers) as C rises. The coefficients are primarily 0 for lower values of C, but as C rises, non-zero coefficients for higher-order terms show up, suggesting that the model is fitting more intricate feature interactions. Nonetheless, a large number of these coefficients continue to be tiny, suggesting that Lasso's regularisation effectively encourages sparsity.

Increasing C means the number of parameters affecting the Lasso model increase, resulting in the chances of overfitting to increase.

(c): LR Predictions

I used tri_surf to plot predictions and the scatter plot in the matplotlib package. The predictions are shown as a smooth surface, while the training data points are plotted as red markers. This helps illustrate how well the model fits the data across different values of the regularisation parameter C. In figure 3 each plot shows the predictions as a surface while the training data points are overlaid using a scatter plot. This allows for a clear comparison between the model's predictions and the actual data. The training data is represented as red dots, while the predicted surface is shown in purple for clarity.

As C increases, the model becomes less regularised, leading to more complex surfaces that fit the training data more closely. In contrast, for smaller C values, the model is more regularised, resulting in under-fitting meaning the model is not able to fit all the data. This demonstrates how varying C affects the model's flexibility and its ability to capture the nuances in the data. However, with high C, there's a risk of overfitting, where the model starts to fit the noise in the data rather than the underlying pattern.

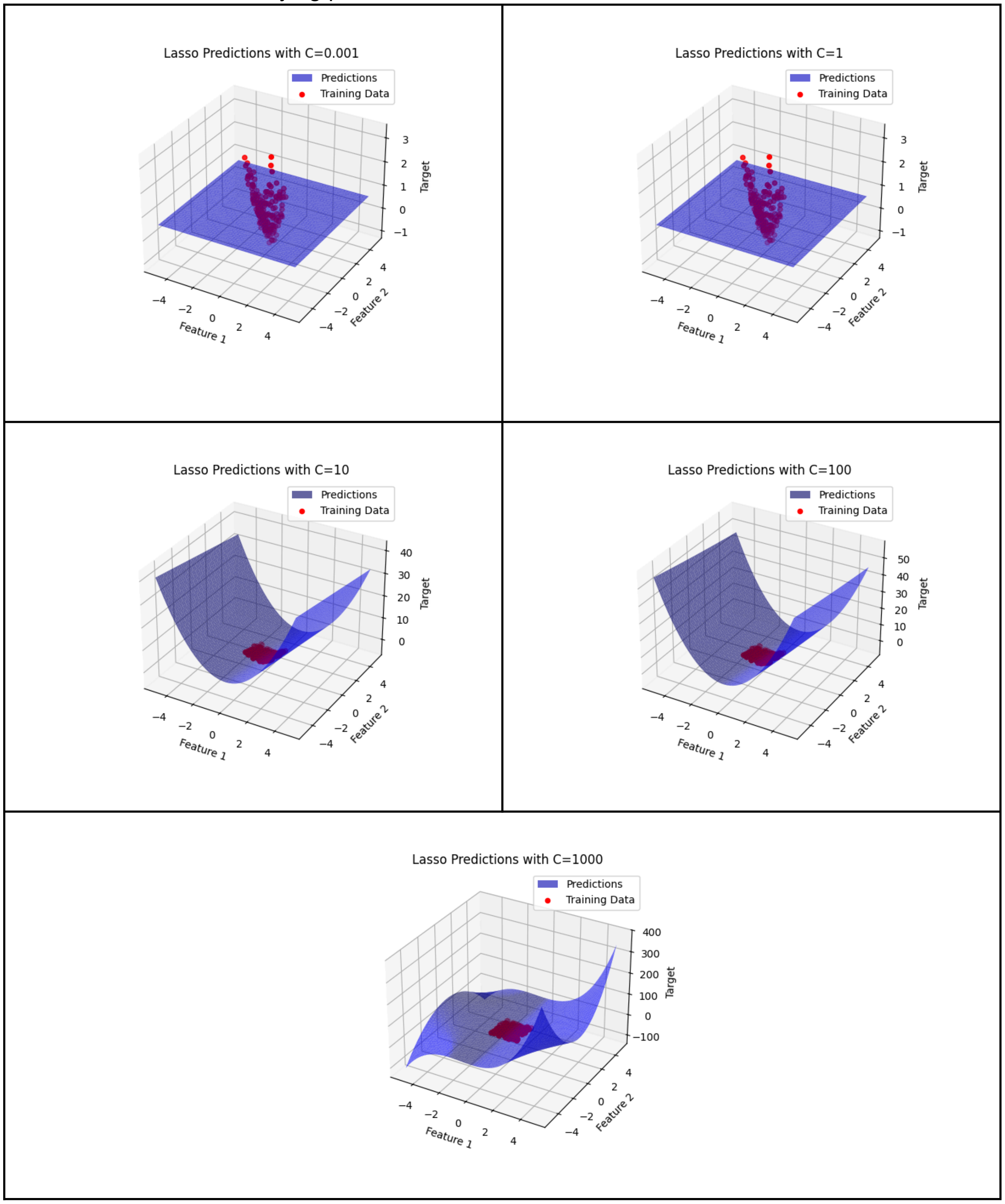

Figure 3: Lsso Regression Predictions for C = [0.001, 1, 10, 100, 1000]

(d): Under- and Over-Fitting

When a model in machine learning is too simplistic to capture the underlying patterns in the data, it is said to be underfitting; conversely, when a model is too complicated, it is said to be overfitting, fitting even the noise in the training data. In Lasso regression, both can be controlled by varying the regularisation parameter C.

By penalising large coefficients, Lasso regression introduces a regularisation term to stop overfitting. By adjusting C, one may control the trade-off between under- and overfitting. A lower C promotes simpler models with fewer non-zero coefficients and better regularisation. The penalty lessens as C rises, enabling the model to fit increasingly intricate patterns in the data but at the risk of overfitting.

- **Low C values (0.001, 1):** High regularisation leads to simpler models, as the coefficients typically stay zero for all degrees of polynomial features. Because the model is unable to reflect the intricate relationships present in the data, underfitting results.
- **Moderate C values (10, 100):** A better balance between regularisation and data fitting is reflected by the model's introduction of non-zero coefficients, particularly in higher-degree polynomials. Nonetheless, there is still control over the overfitting danger.
- **High C values (1000):** Overfitting is indicated by significant coefficients for higher-order terms, especially for polynomial degrees three and above. The model probably includes noise in its capture of even the smallest variances in the data.

Figure 3 shows this pattern, with higher C values producing increasingly complex surfaces that fit the training data closely but may not generalise well to new data, and lower C values producing smoother surfaces.

(e): Ridge Regression Model

In the same way as Lasso Regression models I trained Ridge regression models with polynomial features of varying degrees up to 5 and tested a range of regularisation parameters (C values ranging from 0.001 to 1000).

We observed that as C increased, the model became less regularised, resulting in larger coefficients for more complex terms, especially with higher-degree polynomial features.

Degree of Polynomial Feature = 1

	C	Coefficients of the Ridge Regression Model	Intercept
1	0.001	0.0, -0.0032735914536621133, -0.10786245176151169	0.605391
2	1	0.0, -0.02804073684676571, -0.9431285455344592	0.645270
3	10	0.0, -0.028232893414874692, -0.9497544239853373	0.645587
4	100	0.0, -0.028252253759511167, -0.9504221358950353	0.645619
5	1000	0.0, -0.028254191251825673, -0.9504889587265475	0.645622

Degree of Polynomial Feature = 2

	C	Coefficients of the Ridge Regression Model	Intercept
1	0.001	0.0, -0.003332878238567344, -0.10801643847908822, 0.06484354389875385, 0.004644230130543105, 0.008579276887554305	0.582493
2	1	0.0, -0.02507916013102304, -0.9820423024134676, 1.9980593736650565, 0.0219247782543221, 0.07985727986407108	0.001463
3	10	0.0, -0.02453394271374564, -0.989914535413131, 2.0529127001095855, 0.018064200746700107, 0.07672371186291013	-0.014138
4	100	0.0, -0.02447489522018194, -0.9907105569990018, 2.058565999710576, 0.017653995140061482, 0.07638390338781645	-0.015741
5	1000	0.0, -0.024468943023242595, -0.9907902486152714, 2.059133062031644, 0.017612722411355284, 0.07634964437914998	-0.015901

Degree of Polynomial Feature = 3

	C	Coefficients of the Ridge Regression Model	Intercept
1	0.001	0.0, -0.0032236761440216184, -0.10276300111511744, 0.0648724217020718, 0.00498478016267665, 0.008514713015219966, -0.0023782431009493695, -0.03217033763994992, 0.0038413449846794088, -0.061487641859073544	0.583740
2	1	0.0, -0.06726093846704032, -0.9521171442754036, 1.9910046244433344, 0.027563616426998835, 0.07971340235777725, 0.04513536754730787, 0.006817788091606499, 0.04393080931386581, -0.05817718416052037	0.001756
3	10	0.0, -0.0677471183909182, -0.9969146871166583, 2.0482308205721766, 0.02009316872544302, 0.07714785891470288, 0.048937780451424266, 0.021449508594427392, 0.03932030322768695, -0.002387052154941448	-0.014013
4	100	0.0, -0.06778562453706999, -1.0020053004335845, 2.054176677780965, 0.01925621617299915, 0.07686857784312905, 0.049344849607459426, 0.023037483083158762, 0.03880380464135166, 0.004112637247831607	-0.015637
5	1000	0.0, -0.0677893628726667, -1.002521377429119, 2.0547736400096364, 0.0191715102084778, 0.07684040785606451, 0.04938583979887338, 0.0231976247754544, 0.03875157249381347, 0.004773244819411295	-0.015799

Degree of Polynomial Feature = 4

	C	Coefficients of the Ridge Regression Model	Intercept
1	0.001	0.0, -0.0034382542439346706, -0.10277188770046354, 0.06304856229009341, 0.004625103150054056, 0.007791154035054834, -0.0025917250727540793, -0.03218438243753685, 0.0036466902784772696, -0.06145443907012488, 0.053668007057200956, 0.0035320636251404552, 0.026783111778547832, 0.0004343566997120235, 0.00908064451904389	0.570191
2	1	0.0, -0.054624820981065594, -0.9421763298708342, 1.5659362688576974, 0.09195983186408722, 0.06511607821450013, 0.00900004404659163, 0.035732594476346816, 0.03431705154479325, -0.07833918536511288, 0.4362123828711224, -0.08687533430225977, 0.22118222485245143, -0.04867259852061292, -0.07947484961278224	0.050504
3	10	0.0, -0.06297016207578908, -0.996796995740155, 1.9098595934626297, 0.13615015864314095, 0.15324432109458458, 0.03162585153653134, 0.04645010402077175, 0.033555505330238156, -0.012144395811565152, 0.1097310487567302, -0.12357907271654027, 0.14572769704978858, -0.07861022953552185, -0.1459427040963027	-0.003345

4	100	0.0, -0.06460953938415587, -1.0040012875671633, 1.9698733615142379, 0.14363016607075382, 0.16935232291024785, 0.035810715259245104, 0.04687405470349902, 0.033641201994848505, -0.0030121178819100592, 0.0495970292434736, -0.12783786015913381, 0.13351259902177168, -0.0850174423041734, -0.1584170194568595	-0.012420
5	1000	0.0, -0.06478815457018611, -1.0047494363288583, 1.9763360196567734, 0.1444310210230155, 0.17109458780283757, 0.03626335890355674, 0.046903021275036046, 0.03365360964299795, -0.002058470098415427, 0.04308434575803912, -0.1282627782777303, 0.1322146621880317, -0.08572721337014788, -0.15977287213031668	-0.013394

Degree of Polynomial Feature = 5

	C	Coefficients of the Ridge Regression Model	Intercept
1	0.001	0.0, -0.0034785952392677078, -0.10013787635406632, 0.06304161836820686, 0.004845686623137569, 0.007753459944345877, -0.002528955638623526, -0.03089652668516957, 0.003547356776224335, -0.05942263553179412, 0.05363415150243897, 0.0037138565644258353, 0.026756906037821385, 0.0005982424030815596, 0.009053947757577339, -0.0008633898627093329, -0.019205755765869206, 0.002755301118668513, -0.020529463935258722, 0.003788944320677217, -0.041162385378927295	0.570387
2	1	0.0, -0.049847504096495784, -0.9180048584843399, 1.5674970007253275, 0.0874837885422614, 0.07106947698524693, 0.03711494820245332, 0.0514059241597464, -0.010964523956618637, -0.2070112310844031, 0.43542531840014026, -0.07415789439265615, 0.21304273246928307, -0.050916637331615416, -0.0854219085900303, -0.058466703886004355, -0.05889726103448298, 0.09837236603987601, 0.053091281923668035, -0.00638603334164951, 0.11173353219380457	0.050529
3	10	0.0, -0.04105472536786421, -0.9604831896584264, 1.9294822245532863, 0.12373037361020572, 0.16165638826547443, 0.1016670627922968, 0.2483464819342638, -0.13510396037483996, -0.2798586602357653, 0.08146129741636036, -0.05603427766988912, 0.12292496658311845, -0.10887936083558043, -0.14869758006720538, -0.16382560895713733, -0.19399334684154032, 0.3074798930259541, -0.055978097355889246, -0.004241455360699332, 0.247485767835567	-0.005109
4	100	0.0, -0.03577131807211628, -0.9641933141638445, 1.998168606380406, 0.12989644222118957, 0.17774351760089835, 0.12660778224359348, 0.32307257933257305, -0.19384284036388882, -0.3261019166597907, 0.008126004594201758, -0.03817329817702415, 0.10644954355391625, -0.12990963727423646, -0.1592245405531899, -0.209068082334414, -0.24300981708897626, 0.3858821605085258, -0.10323860454695934, 0.011407081805614535, 0.3046484890227696	-0.014901
5	1000	0.0, -0.03514179876229874, -0.9643428366166165, 2.005717882156369, 0.1306174095869433, 0.17948037215331772, 0.13025402820995674, 0.3326711075314614, -0.20177215404235765, -0.3330079674699636, -6.895201884150134e-05, -0.03575016712267325, 0.10464944599827604, -0.13263966081831807, -0.16034675128848308, -0.2153323139047689, -0.2492929146549319, 0.3957908661002294, -0.10930183353087962, 0.01400470705783645, 0.31267259520476676	-0.015961

For degree 1, the coefficients stayed relatively small, showing the model favours simpler linear relationships with low-degree features. However, higher-degree features captured more complex patterns as C increased, with larger coefficients for high C values (e.g., 1000).

Lower C values kept higher-degree features close to zero, indicating underfitting, while higher C values helped fit more complex patterns but risked overfitting, particularly with degrees 4 and 5.

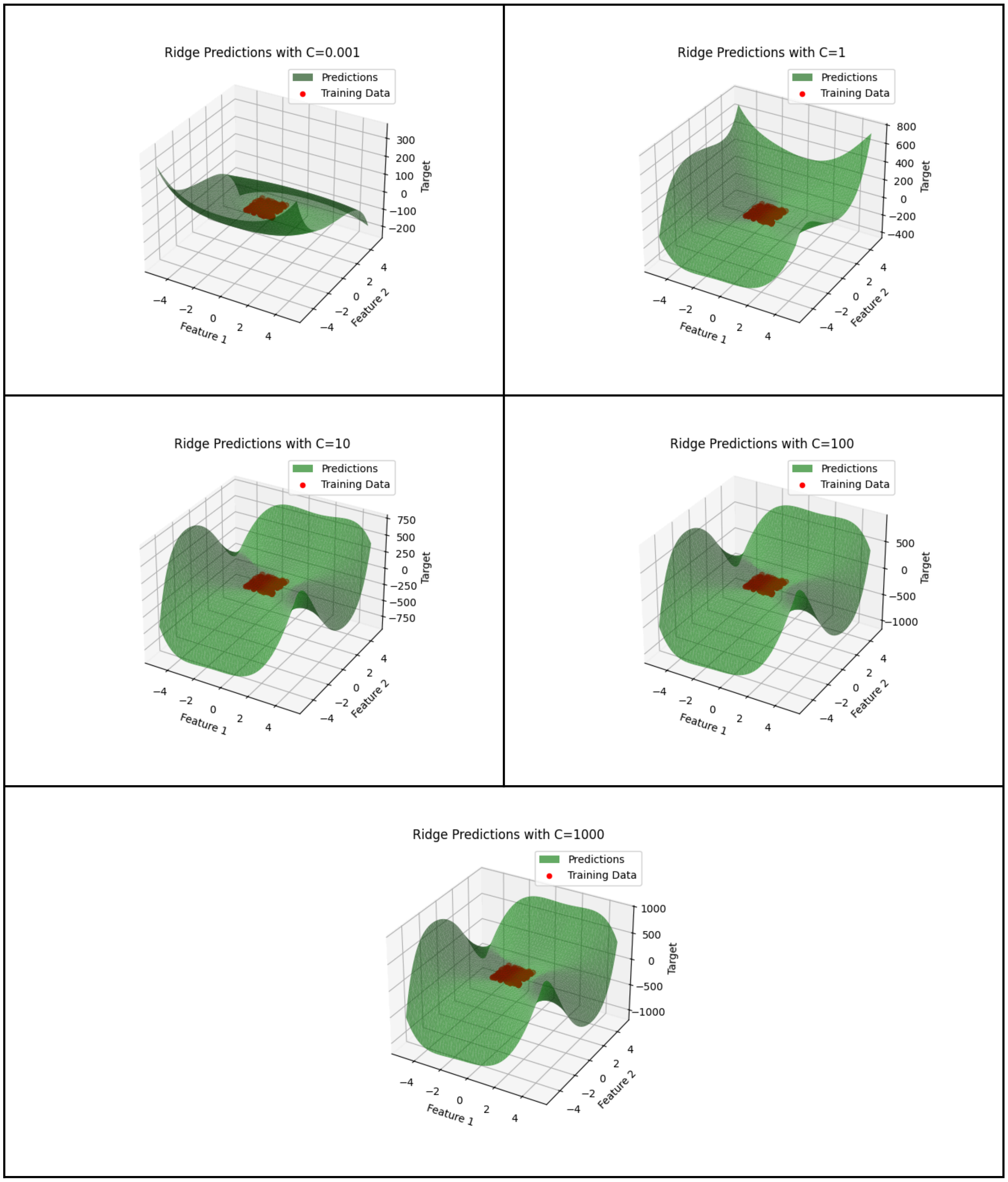

Figure 4: Ridge Regression Predictions for C = [0.001, 1, 10, 100, 1000]

In figure 4 we can see:

- As C increases, regularisation decreases, and the model becomes more complex.
- For very low C (strong regularisation), the model underfits, producing a simple, overly smooth surface.
- For higher C values, the model becomes more flexible but risks overfitting if regularisation is too weak.

Question (ii)

(a): LR 5-Fold Cross Validation

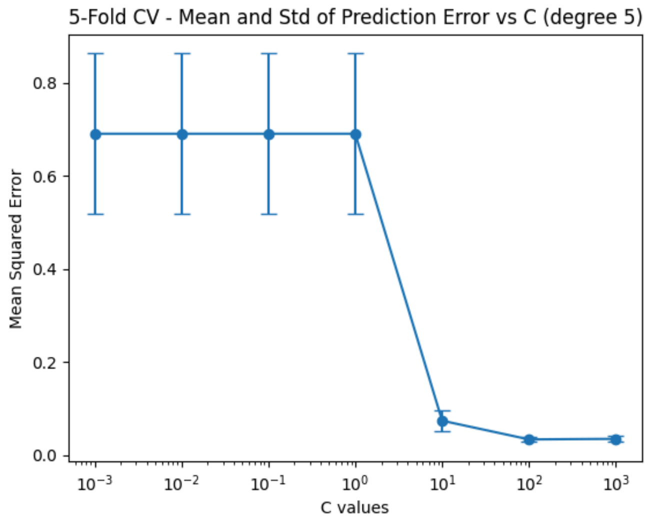

Figure 5: K-Fold Cross Validation

I determined the optimal value of the penalty term CCC in the Lasso Regression model using 5-fold cross-validation. I tested values of C from 0.001 to 1000, sticking to common choices like 0.001, 0.01, 0.1, 1 and 10, and also explored higher values like 100, 100, and 1000. The model's performance was measured using the mean and standard deviation of the predictions.

As shown in Figure 5, the Mean Square Error (MSE) was highest when C was small and decreased until C reached 10. Beyond that, up to C=1000, the MSE change was minimal.

(b): Values of C

Based on the cross-validation data, I recommend a value of C = 10 for this model. Cross-validation results showed that the Mean Squared Error (MSE) consistently decreased as C increased up to 10, and beyond that, the error rate plateaued. This suggests that increasing C beyond 10 doesn't significantly improve the model's performance and may risk overfitting by allowing the model to become too complex. C = 10 strikes a balance between fitting the training data well and avoiding overfitting, which makes it the optimal choice for this scenario.

(c): RR 5-Fold Cross Validation

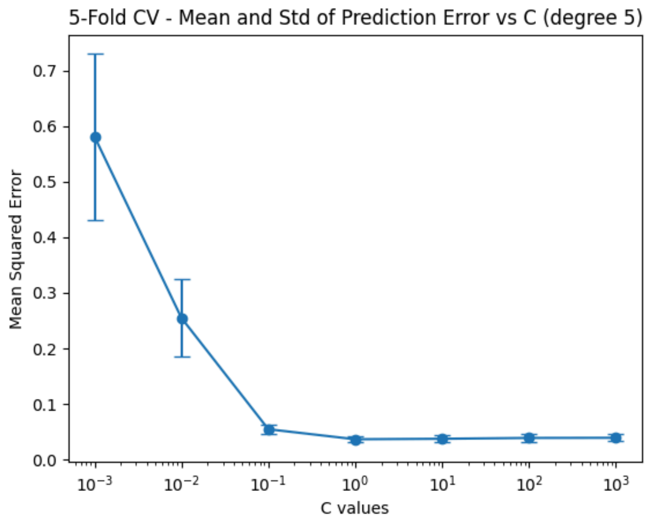

Figure 6: K-Fold Cross Validation

I applied Ridge Regression with 5-fold cross-validation to determine the best value for the regularisation parameter, C, assessing how different values impact prediction accuracy.

Cross-Validation Error vs C

I tested C values ranging from [0.001, 0.01, 0.1, 1, 10, 100, 1000], covering a spectrum from high to low regularisation. The plot (Figure 6) shows the mean squared error (MSE) with standard deviations for each value, created using errorbar from matplotlib.

Recommended Value of C

The MSE notably decreases as C increases up to 10, after which improvements level off. This suggests that while increasing C further won't significantly enhance performance, it may risk overfitting. Thus, C=10 offers the best balance between underfitting and overfitting.

Regularisation clearly helps improve generalisation, and selecting C=10 ensures optimal performance on unseen data.

Appendix

Question (i)

(a)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Load  dataset
data = pd.read_csv('week3.csv')

# Extract features and target from the dataset
X = data.iloc[:, :-1].values  # Features
y = data.iloc[:, -1].values   # Target

# 3D scatter plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Plot data points: X[:, 0] is the first feature, X[:, 1] is the
second feature, and y is the target
ax.scatter(X[:, 0], X[:, 1], y, c='b', marker='o')

# Axis labels
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
ax.set_zlabel('Target')

plt.title('3D Scatter Plot of the Data')
plt.show()
```

(b)

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

# Range of degrees and C values
degrees = range(1, 6)
C_values = [0.001, 1, 10, 100, 1000]

results = []

# Train models for each degree of polynomial features
for degree in degrees:
    poly = PolynomialFeatures(degree)
    X_poly = poly.fit_transform(X)

    for C in C_values:
        # Lasso uses alpha as the regularisation parameter, alpha = 1
/ (2 * C)
        alpha = 1 / (2 * C)
        model = Lasso(alpha=alpha)
        model.fit(X_poly, y)



        results.append({
            'Degree': degree,
            'C': C,
            'Coefficients': model.coef_,
            'Intercept': model.intercept_
        })

results_df = pd.DataFrame(results)

# Print results for each degree
for degree in degrees:
    print(f"Parameters for degree of polynomial feature = {degree}")
    subset = results_df[results_df['Degree'] == degree]
    print(subset[['C', 'Coefficients',
'Intercept']].to_string(index=False))
    print()
```

(c)

```
# Grid for the feature space
grid_range = np.linspace(-5, 5, 50)
X_test = []
for i in grid_range:
    for j in grid_range:
        X_test.append([i, j])
X_test = np.array(X_test)

# Transform grid using polynomial features
degree = 5  # Degree for polynomial expansion
poly = PolynomialFeatures(degree)
X_poly_test = poly.fit_transform(X_test)

# Fit Lasso model and generate predictions
C_values = [0.001, 1, 10, 100, 1000]  # Regularisation params
for C in C_values:
    alpha = 1 / (2 * C)
    model = Lasso(alpha=alpha)

    X_poly = poly.fit_transform(X)  # Transform training data features
    model.fit(X_poly, y)  # Fit model to the training data

    # Generate predictions on the grid
    predictions = model.predict(X_poly_test)

    # Plot for predictions
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    # Reshape predictions for surface plotting
    X1, X2 = X_test[:, 0], X_test[:, 1]
    ax.plot_trisurf(X1, X2, predictions, color='blue', alpha=0.6,
label='Predictions')  # Uniform color for predictions

    # Scatter plot for the original training data
    ax.scatter(X[:, 0], X[:, 1], y, c='r', marker='o', label='Training
Data')  # Keep training data in red

    ax.set_xlabel('Feature 1')
    ax.set_ylabel('Feature 2')
    ax.set_zlabel('Target')
    ax.set_title(f'Lasso Predictions with C={C}')

    plt.legend()

    plt.show()
```

(e)

```
from sklearn.linear_model import Ridge
#(b)
# Function to train and print Ridge regression results for various
polynomial degrees up to 5 and C values 0.001, 1, 10, 100 and 1000
def ridge_regression_with_poly_features(X, y, degrees, C_values):
    results = []
    for degree in degrees:
        poly = PolynomialFeatures(degree)
        X_poly = poly.fit_transform(X)

        for C in C_values:
            alpha = 1 / (2 * C)  # Ridge uses alpha as the
regularisation parameter
            model = Ridge(alpha=alpha)
            model.fit(X_poly, y)

            # Collect results for reporting
            results.append({
                'Degree': degree,
                'C': C,
                'Coefficients': model.coef_,
                'Intercept': model.intercept_
            })

    results_df = pd.DataFrame(results)
    return results_df

# Degrees and C values
degrees = range(1, 6)
C_values = [0.001, 1, 10, 100, 1000]

# Run Ridge Regression with polynomial features
ridge_results = ridge_regression_with_poly_features(X, y, degrees,
C_values)

# Results for each degree
for degree in degrees:
    print(f"Ridge Regression Parameters for degree of polynomial
feature = {degree}")
    subset = ridge_results[ridge_results['Degree'] == degree]
    print(subset[['C', 'Coefficients',
'Intercept']].to_string(index=False))
    print()

# (c) Predictions on a grid for Ridge regression and plot
grid_range = np.linspace(-5, 5, 50)
X_test = []
for i in grid_range:
    for j in grid_range:
        X_test.append([i, j])
X_test = np.array(X_test)
```

```
# Transform grid using polynomial features for plotting Ridge model
predictions
degree = 5  # Degree 5 polynomial expansion
poly = PolynomialFeatures(degree)
X_poly_test = poly.fit_transform(X_test)

# Fit Ridge model and generate predictions for different C values
for C in C_values:
    alpha = 1 / (2 * C)
    model = Ridge(alpha=alpha)

    # Fit Ridge model to the polynomial-transformed training data
    X_poly = poly.fit_transform(X)
    model.fit(X_poly, y)

    # Generate predictions on the grid
    predictions = model.predict(X_poly_test)

    # Plot predictions
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    # Reshape predictions for surface plotting
    X1, X2 = X_test[:, 0], X_test[:, 1]
    ax.plot_trisurf(X1, X2, predictions, color='green', alpha=0.6,
label='Predictions')

    # Scatter plot for the original training data
    ax.scatter(X[:, 0], X[:, 1], y, c='r', marker='o', label='Training
Data')

    ax.set_xlabel('Feature 1')
    ax.set_ylabel('Feature 2')
    ax.set_zlabel('Target')
    ax.set_title(f'Ridge Predictions with C={C}')
    plt.legend()
    plt.show()
```

Question (ii)

(a)

```
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error

# Function for cross-validation and plotting
def cross_validation_plot(X, y, c_values, degree):
    mean_error = []
    std_error = []

    # Polynomial feature transformation
    poly = PolynomialFeatures(degree=degree)
    X_poly = poly.fit_transform(X)

    # Perform 5-fold cross-validation
    kf = KFold(n_splits=5, shuffle=True, random_state=42)

    for C in c_values:
        alpha = 1 / (2 * C)  # Lasso uses alpha = 1 / (2 * C)
        model = Lasso(alpha=alpha)

        temp_errors = []
        for train_idx, test_idx in kf.split(X_poly):
            X_train, X_test = X_poly[train_idx], X_poly[test_idx]
            y_train, y_test = y[train_idx], y[test_idx]

            model.fit(X_train, y_train)
            y_pred = model.predict(X_test)

            # Calculate mean squared error for this fold
            error = mean_squared_error(y_test, y_pred)
            temp_errors.append(error)
        # Calculate  mean and standard deviation of the errors
        mean_error.append(np.mean(temp_errors))
        std_error.append(np.std(temp_errors))

    # Plot  results using error bars
    plt.errorbar(c_values, mean_error, yerr=std_error, fmt='-o',
capsize=5)
    plt.xlabel('C values')
    plt.ylabel('Mean Squared Error')
    plt.title(f'5-Fold CV - Mean and Std of Prediction Error vs C
(degree {degree})')
    plt.xscale('log')
    plt.show()



c_values = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
degree = 5  # Using polynomial features of degree 5 from  previous
part

# Run cross-validation and plot  results
cross_validation_plot(X, y, c_values, degree)
```

(c)

```
from sklearn.linear_model import Ridge
# Cross-validation and plotting function for Ridge Regression
def ridge_cross_validation_plot(X, y, c_values, degree):
    mean_error = []
    std_error = []

    # Polynomial feature transformation
    poly = PolynomialFeatures(degree=degree)
    X_poly = poly.fit_transform(X)

    # 5-fold cross-validation
    kf = KFold(n_splits=5, shuffle=True, random_state=42)

    for C in c_values:
        alpha = 1 / (2 * C)  # Ridge uses alpha = 1 / (2 * C)
        model = Ridge(alpha=alpha)

        temp_errors = []
        for train_idx, test_idx in kf.split(X_poly):
            X_train, X_test = X_poly[train_idx], X_poly[test_idx]
            y_train, y_test = y[train_idx], y[test_idx]

            model.fit(X_train, y_train)
            y_pred = model.predict(X_test)

            # ,MSE for this fold
            error = mean_squared_error(y_test, y_pred)
            temp_errors.append(error)

        # mean and standard deviation of the errors
        mean_error.append(np.mean(temp_errors))
        std_error.append(np.std(temp_errors))

    # Plot the results using error bars
    plt.errorbar(c_values, mean_error, yerr=std_error, fmt='-o',
capsize=5)
    plt.xlabel('C values')
    plt.ylabel('Mean Squared Error')
    plt.title(f'5-Fold CV - Mean and Std of Prediction Error vs C
(degree {degree})')
    plt.xscale('log')
    plt.show()

# C values
c_values = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
degree = 5  # Using polynomial features of degree 5 as in previous
parts
```

```
# Run cross-validation and plot the results for Ridge Regression
ridge_cross_validation_plot(X, y, c_values, degree)
```