Week 3 Answers

Question i

a

The plot displays a collection of blue dot markers representing data points. The x-axis corresponds to feature 1, the y-axis to feature 2, and the z-axis to the target values. It is clear that the data points lie on a curved plane by using the interactive plot. So the model should be a non-linear model.

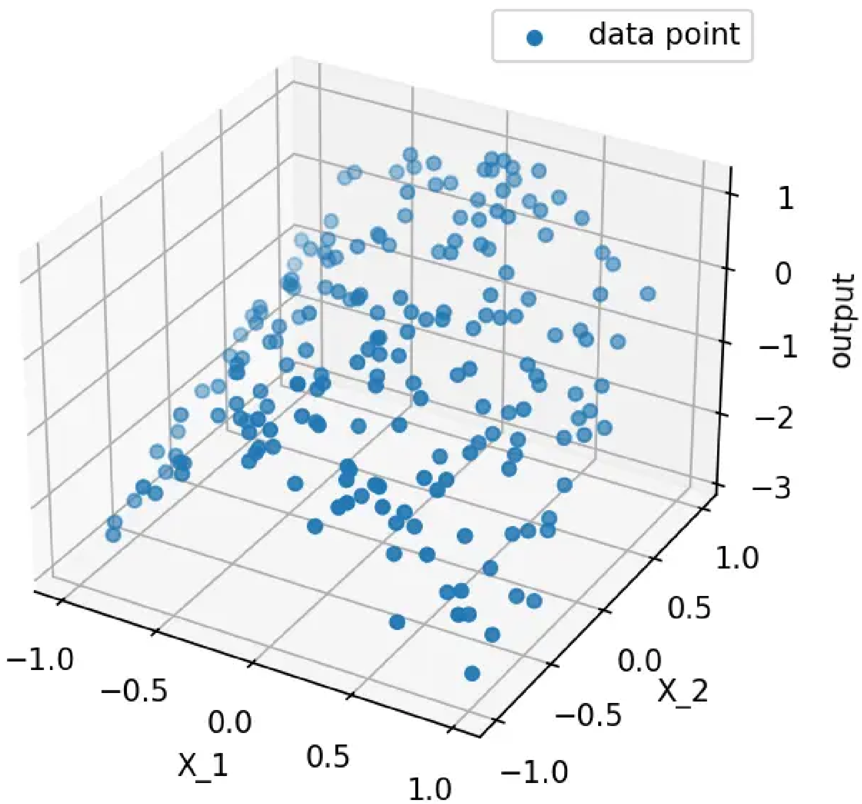


Figure 1: Plot 3D data points

Table 1: Coefficients of Lasso Regression with Different C

C	Bias	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
1	-0.625	0	0	0	0	0	0	0	0	0
10	-0.189	0	0.815	-1.401	0	0	0	0	0	0
100	-0.040	0	0.892	-1.882	-0.003	0	0	0	-0.018	0.082
1000	-0.017	0.138	0.856	-1.833	-0.065	-0.044	0	0	-0.092	0.154

x_1^4	$x_1^3x_2$	$x_1^2x_2^2$	$x_1x_2^3$	x_2^4	x_1^5	$x_1^4x_2$	$x_1^3x_2^2$	$x_1^2x_2^3$	$x_1x_2^4$	x_2^5
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
-0.127	0	-0.022	0	0	0	0.025	0.054	0.016	-0.037	0

From the table, we can conclude:

- With $C = 1$,

$$y = -0.625$$

- With $C = 10$,

$$y = -0.189 + 0.815x_2 - 1.401x_1^2$$

- With $C = 100$,

$$y = -0.04 + 0.892x_2 - 1.882x_1^2 - 0.003x_1x_2 - 0.018x_1x_2^2 + 0.082x_2^3$$

- With $C = 1000$,

$$\begin{aligned} y = &-0.017 + 0.138x_1 + 0.856x_2 - 1.833x_1^2 - 0.065x_1x_2 - 0.044x_2^2 \\ &-0.092x_1x_2^2 + 0.154x_2^3 - 0.127x_1^4 - 0.022x_1^2x_2^2 \\ &+0.025x_1^4x_2 + 0.054x_1^3x_2^2 + 0.016x_1^2x_2^3 - 0.037x_1x_2^4 \end{aligned}$$

From the feature weights, it can be observed that as C increases, α and bias decrease, resulting in less penalization on misclassified points. Consequently, the feature weights increase as the model focuses on fitting the data points more accurately, leading to a greater number of non-zero weights. This indicates that the model becomes more complex, capturing more details of data, which may increase the risk of overfitting if not managed properly.

c

The figure displays four Lasso prediction plots, each illustrating the scattered original data points and a prediction plane produced by the model for different values of C. In the first plot (Lasso $C = 1$), the prediction is a flat plane, indicating that the model is underfitting the data. As C increases to 10, the prediction becomes a curved plane that fits the data much more accurately, showing the model's ability to capture the underlying pattern. In the subsequent plots, as C increases further to 100 and 1000, the prediction plane remains similarly curved, with minimal noticeable changes. This suggests that beyond a certain point, increasing C no longer significantly impacts the model's fit, as it has already captured the data's structure effectively.

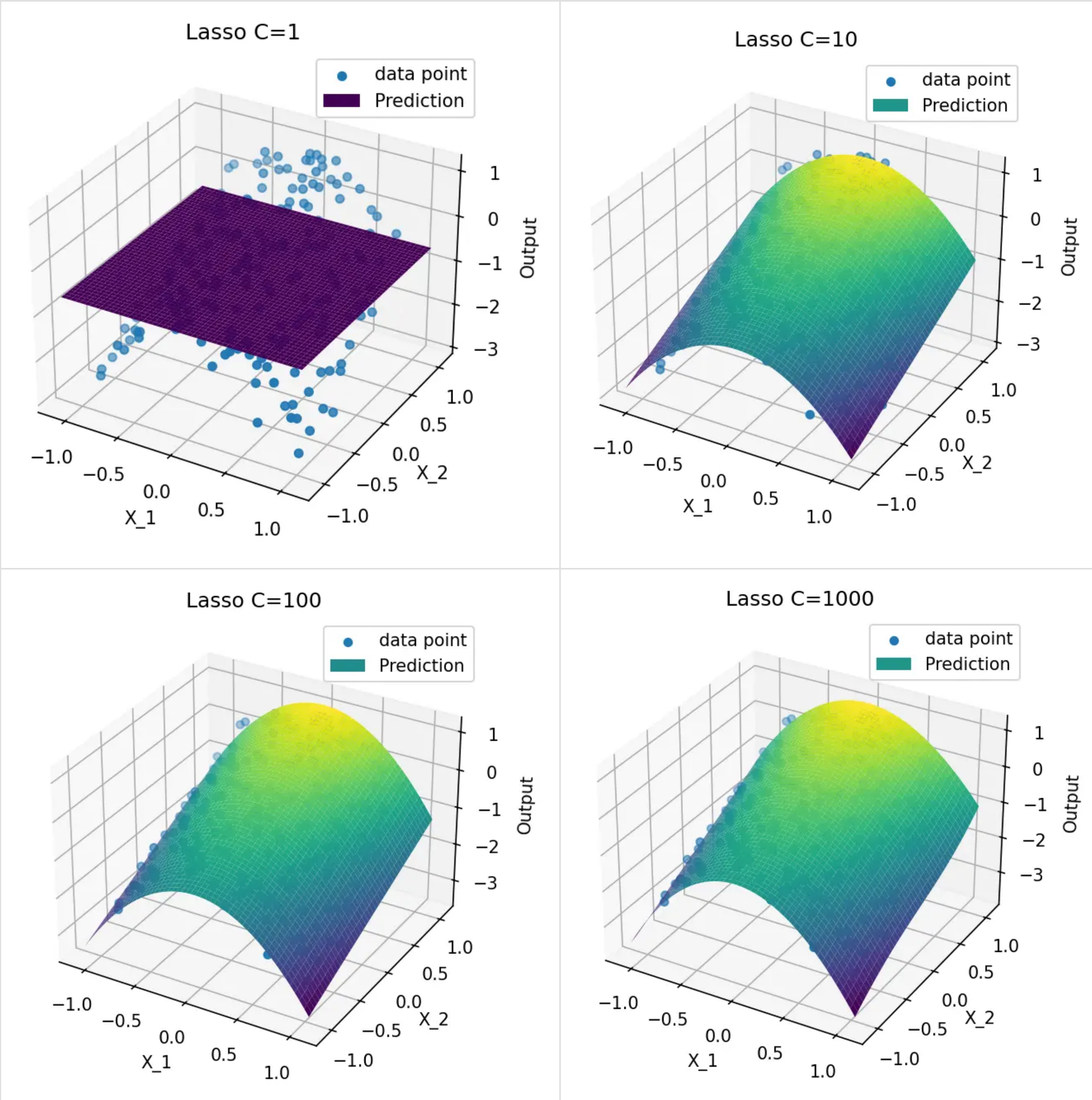


Figure 2: Plots of Lasso Predictions with various C and data points

d

When C is too small, the penalty applied to misclassification is quite large. As a result, the model prioritizes minimizing this penalty rather than fitting the data accurately, leading to underfitting. This is evident in the top-left plot above, where the model produces a flat plane that does not capture the underlying pattern of the data.

Conversely, when C is too large, the penalty becomes minimal, allowing the model to fit the data as closely as possible. This can lead to overfitting, where the model becomes overly complex and fails to generalize to new data. This is illustrated in the bottom-right plot above, where the model fits the data points very precisely but may not perform well on unseen data.

e

With the respect to $\alpha = \frac{1}{2C}$, the ridge regression model is trained on the values of C in range of `[0.001, 0.01, 1, 10]`. The coefficients and intercept of trained models are presented in the following table.

Table 2: Coefficients of Ridge Regression with Different C

C	**Bias**	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
0.001	-0.605	-0.014	0.089	-0.053	0.013	0.009	-0.008	0.025	-0.006	0.058
0.01	-0.466	-0.040	0.358	-0.362	0.046	0.034	-0.014	0.088	-0.015	0.218
1	-0.082	0.040	0.783	-1.388	-0.069	0.020	-0.042	0.039	-0.131	0.311
10	-0.028	0.126	0.762	-1.743	-0.156	-0.059	-0.216	-0.082	-0.317	0.595

x_1^4	$x_1^3x_2$	$x_1^2x_2^2$	$x_1x_2^3$	x_2^4	x_1^5	$x_1^4x_2$	$x_1^3x_2^2$	$x_1^2x_2^3$	$x_1x_2^4$	x_2^5
-0.046	0.009	-0.016	0.009	0.011	-0.004	0.014	-0.004	0.015	-0.003	0.045
-0.310	0.042	-0.113	0.036	0.033	0.005	0.051	-0.004	0.045	-0.007	0.156
-0.532	-0.031	-0.189	0.021	-0.011	-0.029	0.022	0.171	0.015	-0.077	-0.096
-0.204	0.038	-0.079	0.113	0.068	0.038	0.136	0.559	0.107	-0.129	-0.409

From the table, we can conclude:

- With $C = 0.001$,

$$\begin{aligned} y = &-0.605 - 0.014x_1 + 0.089x_2 - 0.053x_1^2 + 0.013x_1x_2 + 0.009x_2^2 - 0.008x_1^3 \\ &+0.025x_1^2x_2 - 0.006x_1x_2^2 + 0.058x_2^3 - 0.046x_1^4 + 0.009x_1^3x_2 - 0.016x_1^2x_2^2 + 0.009x_1x_2^3 \\ &+0.011x_2^4 - 0.004x_1^5 + 0.014x_1^4x_2 - 0.004x_1^3x_2^2 + 0.015x_1^2x_2^3 - 0.003x_1x_2^4 + 0.045x_2^5 \end{aligned}$$

- With $C = 0.01$,

$$\begin{aligned} y = &-0.466 - 0.040x_1 + 0.358x_2 - 0.362x_1^2 + 0.046x_1x_2 + 0.034x_2^2 - 0.014x_1^3 \\ &+0.088x_1^2x_2 - 0.015x_1x_2^2 + 0.218x_2^3 - 0.310x_1^4 + 0.042x_1^3x_2 - 0.113x_1^2x_2^2 + 0.036x_1x_2^3 \\ &+0.033x_2^4 + 0.005x_1^5 + 0.051x_1^4x_2 - 0.004x_1^3x_2^2 + 0.045x_1^2x_2^3 - 0.007x_1x_2^4 + 0.156x_2^5 \end{aligned}$$

- With $C = 1$,

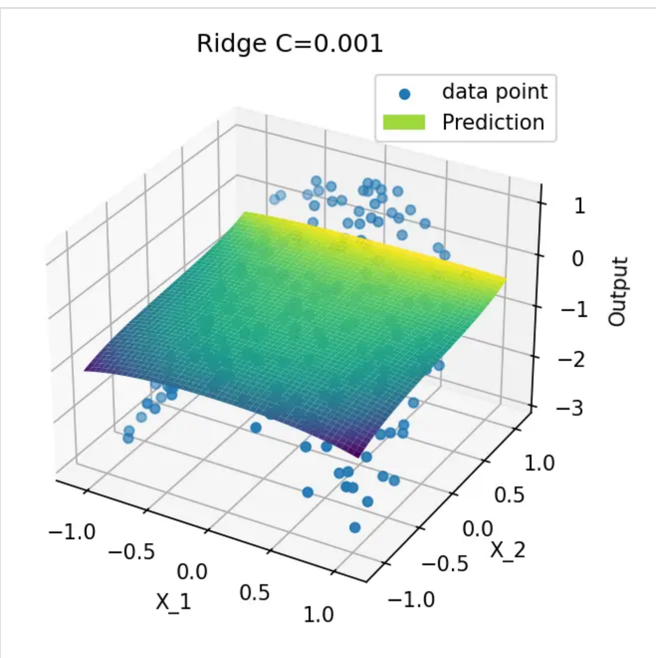
$$
\begin{aligned}
y = & -0.082 - 0.040x_1 + 0.783x_2 - 1.388x_1^2 - 0.069x_1x_2 + 0.020x_2^2 - 0.042x_1^3 \\
& +0.039x_1^2x_2 - 0.131x_1x_2^2 + 0.311x_2^3 - 0.532x_1^4 - 0.031x_1^3x_2 - 0.189x_1^2x_2^2 + 0.021x_1x_2^3 \\
& -0.011x_2^4 - 0.029x_1^5 + 0.022x_1^4x_2 + 0.171x_1^3x_2^2 + 0.015x_1^2x_2^3 - 0.077x_1x_2^4 - 0.96x_2^5
\end{aligned}
$$

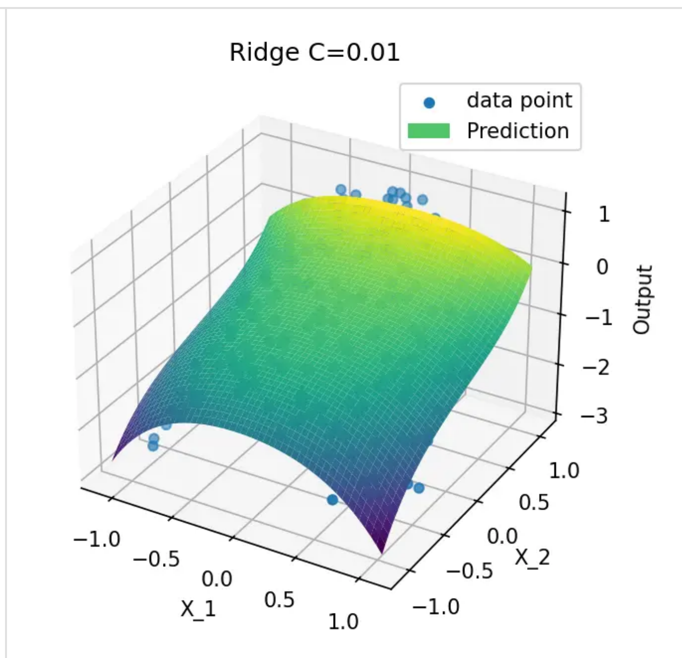
- With $C = 10$,

$$
\begin{aligned}
y = & -0.028 + 0.126x_1 + 0.762x_2 - 1.743x_1^2 - 0.156x_1x_2 - 0.059x_2^2 - 0.216x_1^3 \\
& -0.082x_1^2x_2 - 0.317x_1x_2^2 + 0.594x_2^3 - 0.204x_1^4 + 0.038x_1^3x_2 - 0.079x_1^2x_2^2 + 0.113x_1x_2^3 \\
& +0.068x_2^4 + 0.038x_1^5 + 0.136x_1^4x_2 + 0.559x_1^3x_2^2 + 0.106x_1^2x_2^3 - 0.129x_1x_2^4 - 0.409x_2^5
\end{aligned}
$$

As C increases, the α and bias decrease, which means less penalization applied on misclassified points. Consequently, the weights of features increases as the model is allowed to focus more on fitting the data precisely. This adjustment enables the model to capture more of the data's underlying patterns, reducing error but potentially increasing the risk of overfitting if C becomes too large.

The plots below display the predictions from the ridge regression model for different values of C . The trend observed in these plots is similar to that of the lasso regression plots. As C increases, the predictions made by the models transit from underfitting to overfitting. Initially, with smaller values of C, the model produces a flat plane, indicating underfitting. As C$ increases to 1, the model's prediction plane becomes more curved and fits the data points more precisely. The prediction plane barely changes when C reaches to 10 since it already fit data precisely.





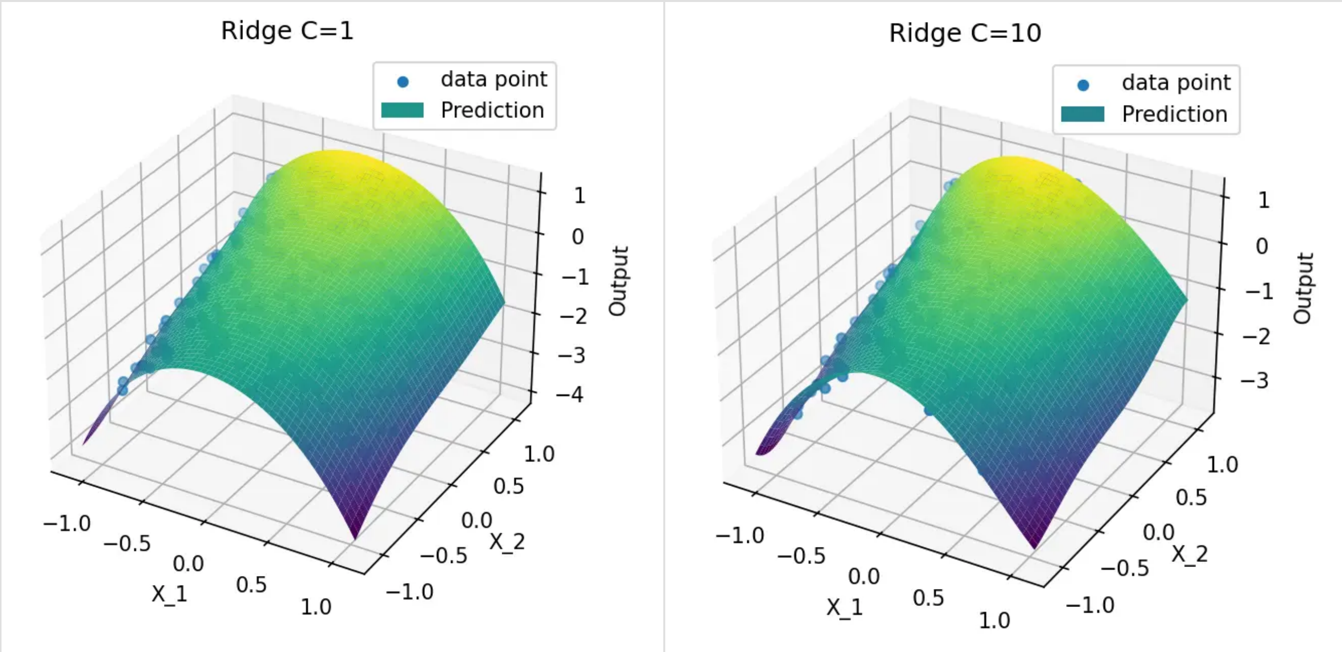


Figure 3: Plots of Ridge Predictions with various C and data points

Question ii

a

Initially, C values `[0.1, 1, 10, 100]` are tested shown in the top-left figure from Figure 4. The difference between $C = 0.1$ and $C = 1$ is negligible, and the mean squared error barely changes beyond $C = 10$. Therefore, $C = 0.1$ is removed and the upper limit is adjusted to $C = 50$.

Then, the range is updated to `[1, 5, 10, 20, 50]` in the top-right plot from Figure 4. Since the mean squared error remains stable beyond $C = 20$, the upper limit is reduced to $C = 20$ to allow a closer examination of C between 1 and 20.

From the last plot, the refined range `[1, 5, 7.5, 10, 20]` is used. It can be observed that the mean squared error stabilizes after $C = 7.5$, suggesting that further increases in C beyond this value yield minimal changes in performance.

These adjustments effectively demonstrate how varying C influences model performance and identify the optimal range where increasing C no longer impacts the MSE significantly.

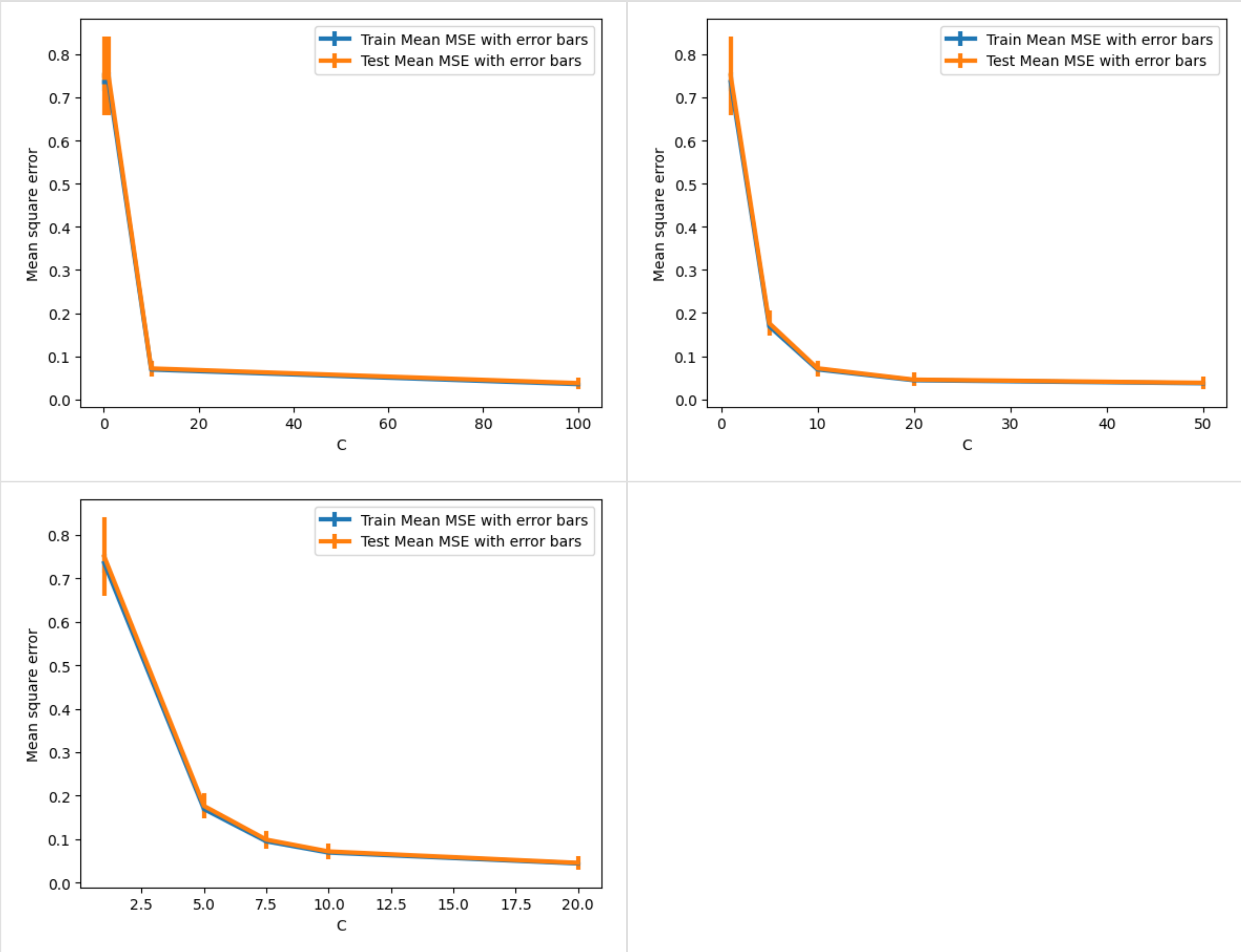


Figure 4: Plots of Lasso Predictions Errorbar

b

Based on the cross-validation data in (a), a value of $C = 7.5$ is recommended, as the mean squared error does not significantly decrease beyond this point. Choosing a higher value of C increases the risk of overfitting, as the model may become too complex and fail to generalize well to new data. Therefore, $C = 7.5$ provides a good balance between model accuracy and generalization.

c

Initially, C values `[0.1, 1, 10, 100]` are tested shown in the top-left figure from Figure 5. The mean squared error barely changes beyond $C = 10$. Therefore the upper limit is shrunk to $C = 10$ and explore more value between 0.1 and 1.

Then, the range is updated to `[0.1, 0.5, 1, 10]` in the top-right plot from Figure 5. Since the mean squared error remains stable beyond $C = 1$, the upper limit is reduced to $C = 1$.

Finally, the last plot has the refined range `[1, 5, 7.5, 10, 20]`. It can be observed that the mean squared error stabilizes after $C = 0.5$, suggesting that further increases in C beyond this value yield minimal changes in performance.

As a result, $C = 0.5$ is recommended as it effectively balances fitting the training data without a high risk of overfitting. This value provides a stable and efficient model performance while maintaining generalization capabilities.

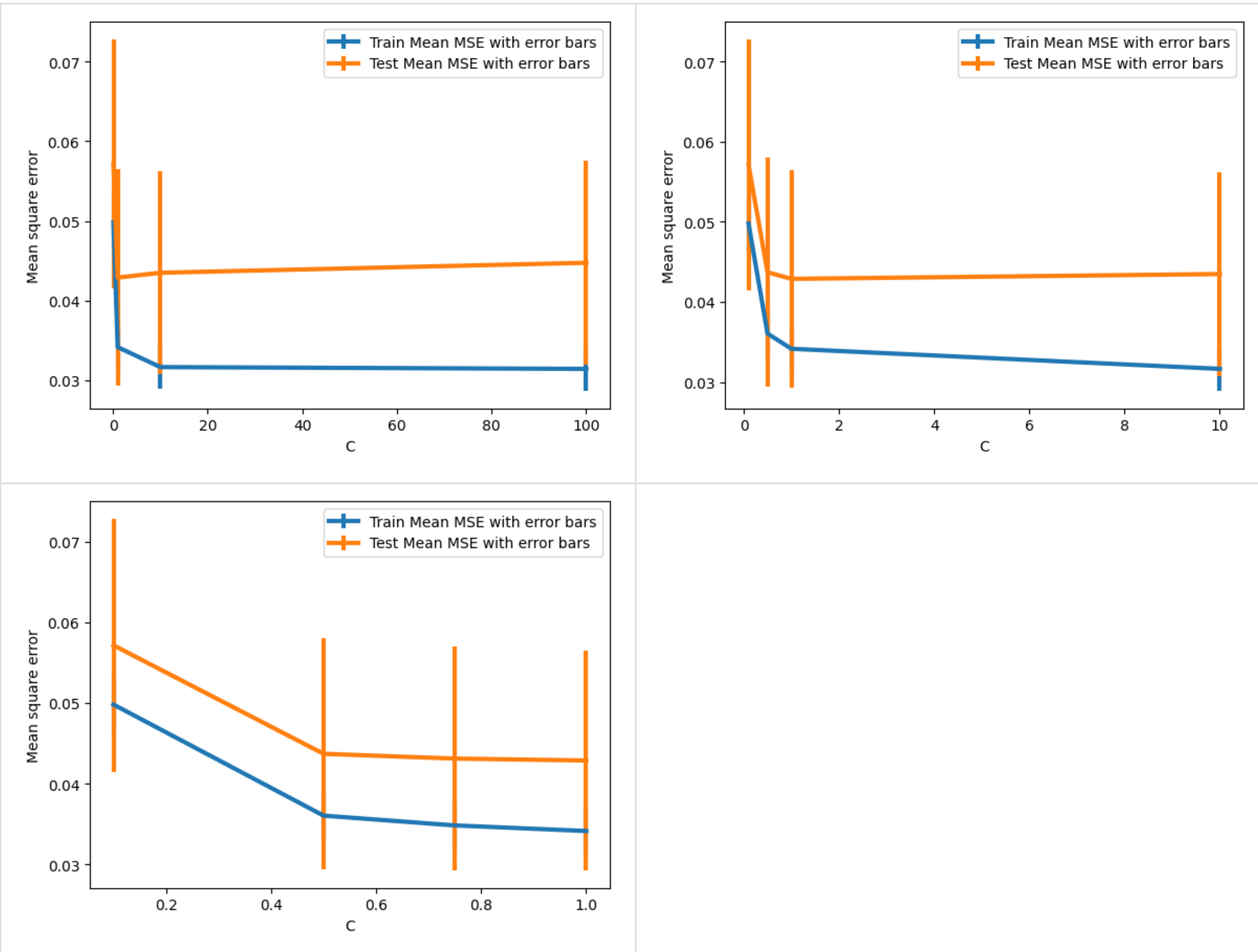


Figure 5: Plots of Ridge Predictions Errorbar

Appendix

```python
import numpy as np
import pandas as pd
from sklearn.linear_model import Lasso, Ridge
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import KFold
```

```
# id:6--12-6
df = pd.read_csv("week3.csv",sep=',')
x1=df.iloc[:,0]
x2=df.iloc[:,1]
X=np.column_stack((x1,x2))
y=df.iloc[:,2]

#(i)
#(a) plot 3d scatter
fig = plt.figure()
ax = fig.add_subplot(111, projection = '3d')
ax.scatter(X[:, 0], X[:, 1], y)
ax.set_xlabel('X_1')
ax.set_ylabel('X_2')
ax.set_zlabel('Target Value')
plt.show()

#(b) (c)
poly = PolynomialFeatures(5)
Xpoly5 = poly.fit_transform(X)
# split training and test data
Xpoly5_train, Xpoly5_test, y_train, y_test = train_test_split(Xpoly5, y,
test_size = 0.2, random_state=1)

## get all combinations of the two features with the power of 5
# feature_names = poly.get_feature_names_out(input_features=['x1', 'x2'])
# # Print the feature names
# for name in feature_names:
#     print(name)

# create grid little bigger than the range of original features
x1_grid = np.linspace(X[:, 0].min() - 0.1,  X[:, 0].max() + 0.1)
x2_grid = np.linspace(X[:, 1].min() - 0.1,  X[:, 1].max() + 0.1)
x1_grid, x2_grid = np.meshgrid(x1_grid, x2_grid)
print("feature 0 extented range:{}-{}".format(x1_grid.min(), x1_grid.max()))
print("feature 1 extented range:{}-{}".format(x2_grid.min(), x2_grid.max()))
# create polynomial features for the grid data so that
# it can be tested by a model trained with polynomial features up to the power
of 5
x_grid_test = np.c_[x1_grid.ravel(), x2_grid.ravel()]
x_grid_test = poly.fit_transform(x_grid_test)

Crange = [1,10,100,1000]
for C in Crange:
    model = Lasso(alpha = 1/(2*C), fit_intercept=True)
```

```
    model.fit(Xpoly5_train, y_train)
    ypred = model.predict(Xpoly5_test)

    # (b) report the parameters
    print("lasso C = {}: Coefficients: {} Intercept: {}".format(C,
model.coef_, model.intercept_))
    print("Mean Squared Error: ", mean_squared_error(y_test, ypred))

    fig = plt.figure()
    ax = fig.add_subplot(111, projection = '3d')
    ax.scatter(X[:, 0], X[:, 1], y)

    # (c) draw Xtest and ypred on a 3D surface
    ypred = model.predict(x_grid_test)
    ypred = ypred.reshape(x1_grid.shape)

    # Plot surface
    ax.plot_surface(x1_grid, x2_grid, ypred, cmap='viridis', edgecolor='none')

    # Labels
    ax.set_xlabel('X_1')
    ax.set_ylabel('X_2')
    ax.set_zlabel('Output')
    ax.set_title('C={}'.format(C))

    # Show plot
    plt.show()

# (e)
Crange = [0.001, 0.01, 1, 10]
for C in Crange:
    model = Ridge(alpha=1/(2*C))
    model.fit(Xpoly5_train, y_train)
    ypred = model.predict(Xpoly5_test)

    # (b) report the parameters
    print("Ridge C = {}: Coefficients: {} Intercept: {}".format(C,
model.coef_, model.intercept_))
    print("Mean Squared Error: ", mean_squared_error(y_test, ypred))

    fig = plt.figure()
    ax = fig.add_subplot(111, projection = '3d')
    ax.scatter(X[:, 0], X[:, 1], y)

    # (c) draw Xtest and ypred on a 3D surface
    ypred = model.predict(x_grid_test)
```

```python
    ypred = ypred.reshape(x1_grid.shape)

    # Plot surface
    ax.plot_surface(x1_grid, x2_grid, ypred, cmap='viridis', edgecolor='none')

    # Labels
    ax.set_xlabel('Feature 1')
    ax.set_ylabel('Feature 2')
    ax.set_zlabel('Predicted Value')
    ax.set_title('C={}'.format(C))

    # Show plot
    plt.show()

# (ii)
# (a) (b)
kf = KFold(n_splits=5)
def drawErrorBarLasso(Crange):
    train_mean_error=[]; train_std_error=[]
    test_mean_error=[]; test_std_error=[]
    for C in Crange:
        model = Lasso(alpha=1/(2*C))
        temp_train = []
        temp_test = []

        # iterate the 5-fold
        for train, test in kf.split(Xpoly5):
            model.fit(Xpoly5[train], y[train])

            ypred_train = model.predict(Xpoly5[train])
            temp_train.append(mean_squared_error(y[train], ypred_train))

            ypred_test = model.predict(Xpoly5[test])
            temp_test.append(mean_squared_error(y[test], ypred_test))

        train_mean_error.append(np.array(temp_train).mean())
        train_std_error.append(np.array(temp_train).std())
        test_mean_error.append(np.array(temp_test).mean())
        test_std_error.append(np.array(temp_test).std())

    plt.errorbar(Crange, train_mean_error, yerr=train_std_error, linewidth=3,
label='Train Mean MSE with error bars')
    plt.errorbar(Crange, test_mean_error, yerr=test_std_error, linewidth=3,
label='Test Mean MSE with error bars')
    plt.xlabel('C')
    plt.ylabel('Mean square error')
```

```python
    plt.legend(loc='best')
    plt.show()

drawErrorBarLasso([0.1, 1, 10, 100])
drawErrorBarLasso([1, 5, 10, 20, 50])
drawErrorBarLasso([1, 5, 7.5, 10, 20])

# (c)
# Crange = [0.1, 0.5, 1.5, 2, 2.5, 3, 10]
def drawErrorBarRidge(Crange):
    train_mean_error=[]; train_std_error=[]
    test_mean_error=[]; test_std_error=[]
    for C in Crange:
        model = Ridge(alpha=1/(2*C))
        temp_train = []
        temp_test = []

        # iterate the 5-fold
        for train, test in kf.split(Xpoly5):
            model.fit(Xpoly5[train], y[train])

            ypred_train = model.predict(Xpoly5[train])
            temp_train.append(mean_squared_error(y[train], ypred_train))

            ypred_test = model.predict(Xpoly5[test])
            temp_test.append(mean_squared_error(y[test], ypred_test))

        train_mean_error.append(np.array(temp_train).mean())
        train_std_error.append(np.array(temp_train).std())
        test_mean_error.append(np.array(temp_test).mean())
        test_std_error.append(np.array(temp_test).std())

    plt.errorbar(Crange, train_mean_error, yerr=train_std_error, linewidth=3,
label='Train Mean MSE with error bars')
    plt.errorbar(Crange, test_mean_error, yerr=test_std_error, linewidth=3,
label='Test Mean MSE with error bars')
    plt.xlabel('C')
    plt.ylabel('Mean square error')
    plt.legend(loc='best')
    plt.show()

drawErrorBarRidge([0.1, 1, 10, 100])
drawErrorBarRidge([0.1, 0.5, 1, 10])
drawErrorBarRidge([0.1, 0.5, 0.75, 1])
```