Week 3 Assignment – Lasso Regression and Ridge Regression

Module: CS7CS4/CSU44061 Machine Learning

Student Name : Indrajeet Kumar
Student Number : 23345983

Part A : Training and Evaluating Lasso and Ridge Regression Models

1. Creating a 3D Scatter Plot of Downloaded Data

Firstly , we created a 3D Scatter plot using the matplotlib library to visualize the dataset. Here, the first feature i.e. X1 is mapped to X-axis , the second feature i.e. X2 is mapped to Y-axis and the target variable i.e. Y is mapped to Z-axis. Figure 1 displays the 3D scatterplot of features X1 and X2 in relation to the target variable.

The data shows a clear curved pattern, suggesting that the relationship between the features X and the dependent variable y is not a simple straight line but follows a more intricate and curved path.

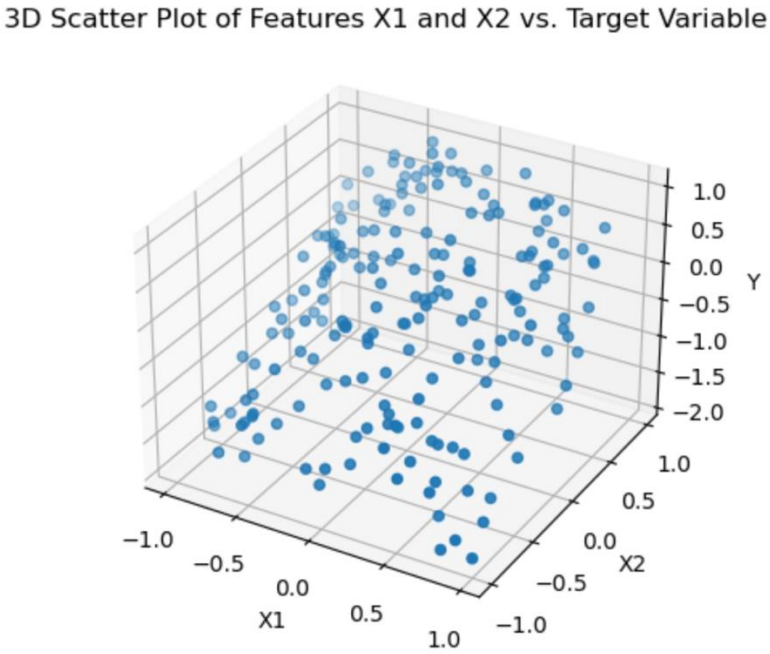

Figure 1 : 3D Scatter Plot of X1 and X2 features against target variable Y

2. Polynomial Feature Expansion and Lasso Regression Analysis

Lasso Regression models were then trained on the data. We start by enhancing our dataset with extra features, creating combinations of the original features up to the fifth power. This expansion allows us to account for potential non-linear patterns in the data. We then delve into training Lasso regression models using these newly generated polynomial features. To understand the impact of the regularization strength, we examine a wide range of C values, including 1, 10, and 100. The key aspect of our analysis is revealing which of these polynomial features, along with their corresponding powers, play a significant role in influencing the model's predictions. Figure 2 displays the parameters of the trained models.

	Penalty (C)	Intercept	Coefficients
0	1	-0.263738	[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]
1	10	-0.194317	[0.0, 0.0, 0.876, -0.406, 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]
2	100	0.008186	[0.0, -0.0, 1.01, -0.974, 0.0, -0.077, -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]

Figure 2 : Lasso regression Models Parameters

For C = 1 , The model shows a high degree of regularisation. The Intercept value is -0.263738 , none of the polynomial features up to degree 5 have no meaningful impact since all of the coefficients are zero. This suggests that the model underfits the data and is still simplistic.

For C = 10, the effect of regularisation decreases. The model now recognises the significance of the second-degree polynomial feature with a coefficient of 0.876, and the intercept is -0.194317. However, the coefficients of every other polynomial feature up to power 5 remain zero. This indicates a small loosening of the simplicity of the model.

For C = 100, the effect of regularisation becomes even milder. The Intercept value is 0.008186. In addition to the second-degree polynomial feature being significant, Higher-order polynomial features, also show non-zero coefficients. This implies that when the model fits the data, it gets more adaptable and begins to take into account a wider variety of variables.

The coefficients' variations with C show how the model's capacity to capture the underlying patterns in the data is traded off against its complexity. Models with fewer non-zero coefficients are simpler and exhibit higher regularisation when C values are smaller. On the other hand, higher C values cause the regularisation to loosen up and let the model include more characteristics, which increases complexity. This event highlights how important the regularisation parameter C is in managing the complexity of the model and, in turn, the way it performs in generalisation.

3. Analyzing Predictions and C Variations in 3D Data Plot

we first generate predictions for the target variable using each Lasso regression model developed in part (b). To create these predictions, we iterate over a grid of feature values using nested loops over a range from -5 to 5., allowing us to obtain predictions over a range of feature combinations. This grid of feature values extends beyond the range observed in the original dataset, ensuring comprehensive coverage.

We select a polynomial degree of 5 and proceed to work with Lasso regression models. Specifically, we focus on the impact of pre-defined C values, which include 1, 10, and 100. These particular 'C' values are integral in shaping the behaviour of the model.
For each value of 'C' in our **'c_values'** list, we fit a Lasso regression model to our dataset. We then generate predictions for our extensive feature grid **'X_test'** and create a 3D scatter plot. This plot effectively shows us the predictions on the **'X_test'** grid, represented as a surface plot, and the original training data points, which are

shown as data points using a scatter plot. This visual representation allows us to interpret how different 'C' values affect the model's predictions.

Each plot is carefully labelled to indicate the specific 'C' value being considered, making it easy to distinguish between them. Figures 3 to 5 showcase Lasso Regression predictions across various C values.

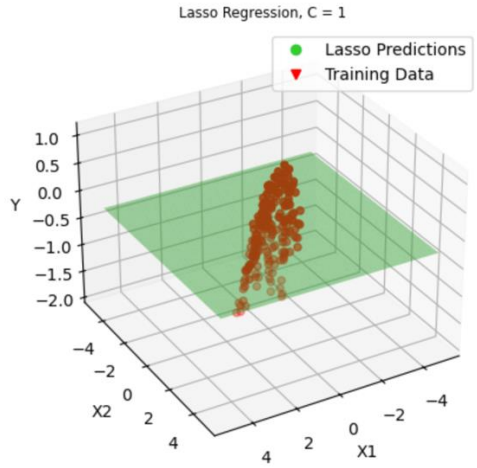

Figure 3 : Lasso Regression Prediction when C = 1

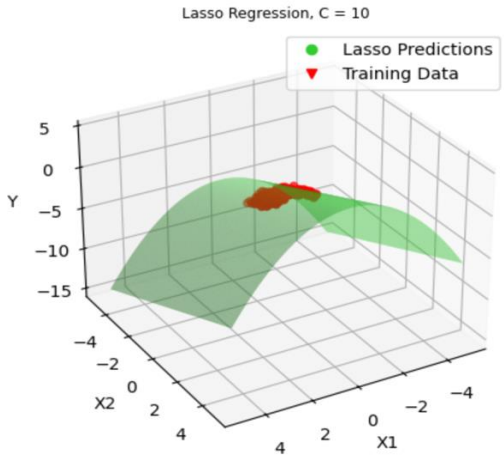

Figure 4 : Lasso Regression Prediction when C = 10

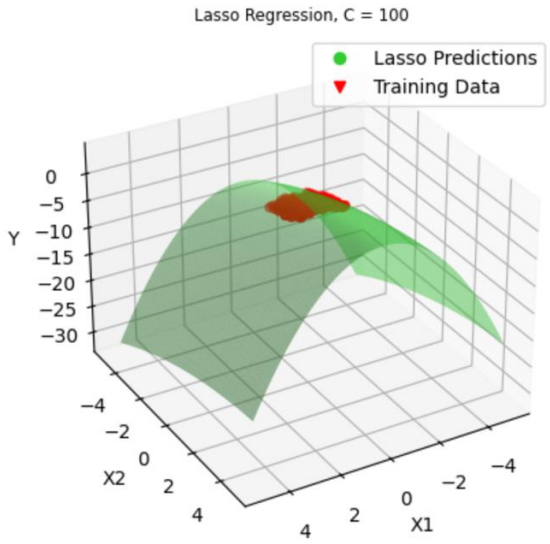

Figure 5 : Lasso Regression Prediction when C = 100

4. Managing the Trade-Off: Understanding Underfitting and Overfitting with Penalty Weight Parameter C

Underfitting: When a model is too basic to identify the underlying patterns in the data, underfitting takes place. It leads to subpar results on the test and training datasets. Low variance and significant bias characterise this.

Overfitting: Overfitting occurs when a model becomes too complicated and begins to fit the noise in the training set instead of the real patterns. With large variation and low bias, it performs terribly on unknown data yet remarkably well on training data.

For C = 1 : In this case, the model leans towards strong regularisation due to the impact of a relatively small C. The intercept value of -0.263738 indicates a conservative approach by the model. Additionally, the coefficients show that the model refrains from overemphasising any one feature or any of its higher powers. The coefficients are primarily composed of 0.0 values. This highly aggressive regularisation prevents overfitting. Because the model stays away from being very detailed, it is likely to underfit and may miss significant details in the data.

For C = 10 : The model's behaviour changes when we increase C to a value of 10. The intercept value changes to -0.194317, suggesting a minor movement in favour of a closer fit to the training set. The coefficients of the model show some non-zero values while maintaining some degree of constraint. In particular, it gives weight to feature combinations such as [0.876, -0.406], indicating a moderate allowance for higher-order terms. This is a more delicate balance to strike, which enables the model to pick up on subtler patterns in the data. This is similar to avoiding underfitting but avoiding overfitting at the same time.

For C = 100 : The approach of the model changes even more when C is increased to 100. The intercept value of 0.008186 indicates that the model is now leaning more in the direction of accepting the training data. The coefficients are more prominent, particularly those that relate to feature combinations like [1.01, -0.974], and [-0.077]. When a C value is high, the model becomes more accommodating and incorporates more intricate feature combinations into its forecasts. Since it allows for overfitting, this flexibility may be a double-edged sword. The risk of fitting noise in the training data exists, despite its exceptional ability to capture subtle patterns.

5. Ridge Regression Analysis

we replicate the steps from (b) and (c) but this time with Ridge Regression, which employs an L2 penalty in its cost function. Ridge Regression differs from Lasso in how it penalizes the magnitude of the coefficients.

we train Ridge Regression models for varying values of the penalty parameter C. Similar to Lasso, C determines the strength of regularization. We select a range of C values, including 1, 10, and 100, and fit Ridge Regression models to our data. The trained models for each C value will provide insights into the impact of C on the model parameters. Figure 6 displays the parameters of the Ridge Regression models.

	Penalty (C)	Intercept	Coefficients
0	1	-0.263738	[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]
1	10	-0.194317	[0.0, 0.0, 0.876, -0.406, 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]
2	100	0.008186	[0.0, -0.0, 1.01, -0.974, 0.0, -0.077, -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]

Figure 6 : Ridge Regression Models Parameters

we follow the same process as in the previous section to visualize the predictions of the Ridge Regression models. We generate predictions on a grid of feature values, extend it beyond the dataset's range, and use 3D plots to display the predictions and the training data. This visualization will help us understand how Ridge Regression performs under different C values and how it compares to Lasso Regression in terms of model parameters. Figure 7-9 showcase Ridge Regression predictions across various C values.

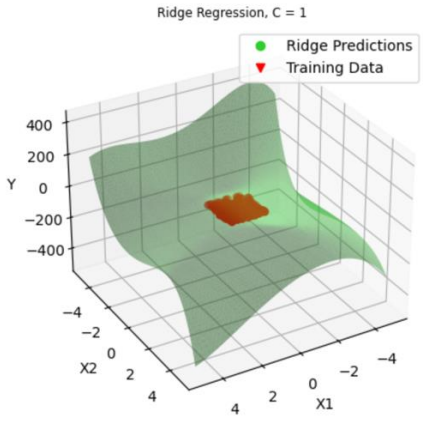

Figure 7 : Ridge Regression prediction when C = 1

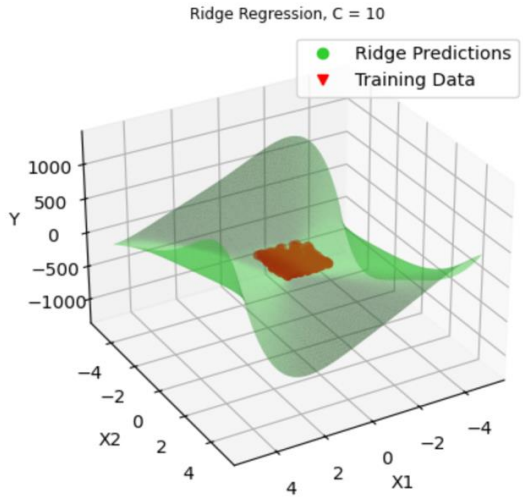

Figure 8 : Ridge Regression prediction when C = 10

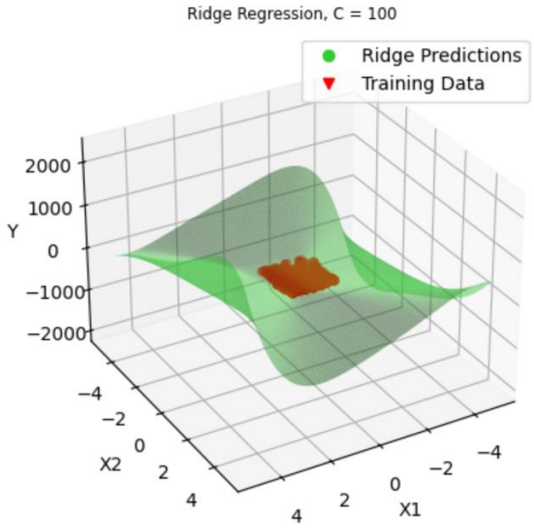

Figure 9 : Ridge Regression Prediction when C = 100

For C = 1, the intercept is 0.029166, and coefficients are non-zero, although some are still close to zero. Ridge Regression does not encourage coefficients to be exactly zero, but it shrinks them. For C = 10, the intercept is 0.073760, and coefficients become larger. Ridge Regression is less regularized compared to C = 1, and more coefficients deviate from zero. For C = 100, the intercept is 0.081987, and coefficients are even larger. Ridge Regression tends to keep more coefficients in the model, and the regularization is weaker.

A higher "C" number results in less regularisation for both Lasso and Ridge Regression. Ridge regression merely decreases coefficients towards zero but does not force them to be zero, whereas Lasso Regression tends to perform feature selection by driving some coefficients to exactly zero. Changes in "C" have an obvious effect on the Lasso model parameters' sparsity and the Ridge model's coefficient magnitudes. Stronger

regularisation is the outcome of a smaller "C" value, whereas weaker regularisation is the outcome of a greater "C" value in both situations.

Part B : Lasso And Ridge Regression with polynomial features using cross- validation

1. Exploring Regularization Strength (C) with 5-Fold Cross-Validation

We utilizes 5-fold Cross Validation to assess model performance. For each fold, the model is fitted on the training data , mean squared errors are calculated for both the training and test sets. We compile the results, including the C value, mean squared errors, and their standard deviations which is displayed in figure 10.

	C value	mse train results	mean mse train	std mse train	mse test	mean mse test	std mse test
0	1	[0.467, 0.478, 0.468, 0.492, 0.508]	0.483	0.015	[0.546, 0.51, 0.541, 0.453, 0.382]	0.486	0.062
1	10	[0.074, 0.067, 0.082, 0.073, 0.074]	0.074	0.004	[0.1, 0.078, 0.091, 0.076, 0.046]	0.078	0.018
2	100	[0.031, 0.031, 0.035, 0.034, 0.037]	0.034	0.002	[0.045, 0.051, 0.027, 0.033, 0.023]	0.036	0.011

Figure 10 : Mean Squared Error and Standard Deviations Across Different C Values

Subsequently, we extracted data from the results and utilized it to construct a visual representation of our analysis. The resulting plot, showcased in Figure 11, illustrates the variation in mean test mean squared errors, along with their corresponding error bars, providing insights into how the error evolves in relation to varying levels of regularization strength (C).

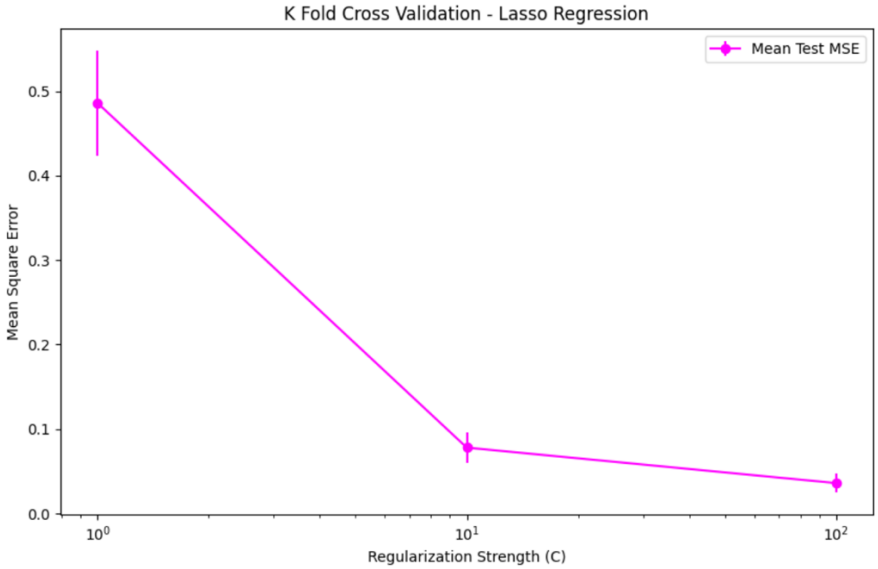

Figure 11: The Fluctuation of Mean Test Mean Squared Error Across Different C Values

2. Optimal Choice of C for Regularization

For C = 1 : As we can see from the figure 10, mean square error for training data is 0.483 and their standard deviation is 0.015 , For testing data mean square error is 0.486 and their standard deviation is 0.062.

For C = 10 : As we can see from the figure 10, mean square error for training data is 0.074 and their standard deviation is 0.004 , For testing data mean square error is 0.078 and their standard deviation is 0.018.

For C = 100 : As we can see from the figure 10, mean square error for training data is 0.034 and their standard deviation is 0.002 , For testing data mean square error is 0.036 and their standard deviation is 0.011.

Based on the results of this dataset's cross-validation, C = 10 appears to strike a good balance between variance and bias. It leads to better performance on the unseen data and also it prevents potential overfitting.

3. K Fold Cross Validation – Ridge Regression

We used 5-fold Cross Validation to assess our Ridge regression model's performance. The Ridge regression model was trained on the training data in each of the five folds, and we computed the mean squared errors for the training and test sets. Figure 12 presents the findings of this thorough assessment, together with the particular C values examined, the related mean squared errors, and the standard deviations that go along with them.

	C value	mse train results	mean mse train	std mse train	mse test	mean mse test	std mse test
0	1	[0.029, 0.03, 0.034, 0.033, 0.035]	0.032	0.002	[0.048, 0.048, 0.034, 0.037, 0.021]	0.038	0.010
1	10	[0.027, 0.028, 0.032, 0.031, 0.034]	0.031	0.002	[0.051, 0.048, 0.031, 0.04, 0.023]	0.039	0.010
2	100	[0.027, 0.028, 0.032, 0.031, 0.034]	0.030	0.002	[0.053, 0.048, 0.032, 0.042, 0.024]	0.040	0.011

Figure 12 : *Mean Squared Error and Standard Deviations Across Different C Values (Ridge Regression)*

We then went ahead and took useful information out of the Ridge Regression findings. Figure 13 illustrates the visual depiction that was made possible with the use of this data. The picture provides valuable insights into how the error profile changes as regularisation strength (C) is varied. It does this by graphically depicting the variation in mean test mean squared errors and informative error bars.

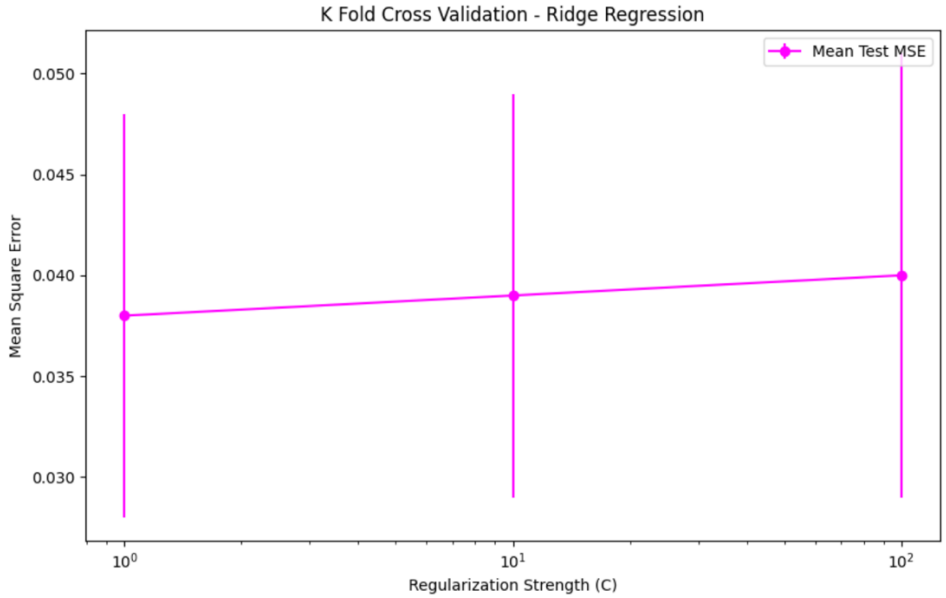

Figure 13: *The Fluctuation of Mean Test Mean Squared Error Across Different C Values (Ridge Regression)*

For C = 1, the mean square error on the training set is 0.032 with a standard deviation of 0.002, and the mean square error on the test set is 0.038 with a standard deviation of 0.010. For C = 10, the mean square error on the training set is 0.031 with a standard deviation of 0.002, and the mean square error on the test set is 0.039 with a standard deviation of 0.010. For C = 100, the mean square error on the training set is 0.030 with a standard deviation of 0.002, and the mean square error on the test set is 0.040 with a standard deviation of 0.011.

Considering these results, the choice of the optimal C value in Ridge regression, similar to Lasso, depends on the trade-off between bias and variance. C = 10 is the recommended choice for Ridge regression in this case.

Appendix Code

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


data = pd.read_csv("week3.csv")
# print(data.head())
data.columns = ['X1', 'X2', 'y']
#Extract Features

X1 = data.iloc[:,0]
X2 = data.iloc[:,1]
X = np.column_stack((X1,X2))
y= data.iloc[:,2]

# Create a 3D scatter plot
fig = plt.figure()
ax = fig.add_subplot(111, projection = '3d')
ax.scatter(X1, X2, y)

# Set labels for the axes
ax.set_xlabel('X1')
ax.set_ylabel('X2')
ax.set_zlabel('Y')

# Set a title for the plot
ax.set_title('3D Scatter Plot of Features X1 and X2 vs. Target Variable y')

# Display the plot
plt.show()

degree = 5
penalties = [1, 10, 100]

# Lasso Regression
lasso_results = []
regression_type = 'Lasso'

poly_features = PolynomialFeatures(degree=degree)
X_poly = poly_features.fit_transform(X)

for penalty in penalties:
    model = Lasso(alpha=1/(2 * penalty))
    model.fit(X_poly, y)
    result_dict = {
        'Penalty (C)': penalty,
        'Intercept': model.intercept_,
        'Coefficients': np.around(model.coef_, decimals=3),
```

```
    }
    lasso_results.append(result_dict)

lasso_results_df = pd.DataFrame(lasso_results)
print("Lasso Regression Results:")
print(lasso_results_df)

# Generate a grid of feature values
X_test = []
grid = np.linspace(-5, 5, 100)  # Adjust as needed
for i in grid:
    for j in grid:
        X_test.append([i, j])
X_test = np.array(X_test)

# Polynomial degree and penalty values
degree_poly = 5
c_values = [1, 10, 100]

# Lasso Regression
plot_colors = ['limegreen', 'red']

# Create polynomial features
Xpoly_Ridge = PolynomialFeatures(degree=degree_poly).fit_transform(X)
Xpoly_test = PolynomialFeatures(degree=degree_poly).fit_transform(X_test)

# Loop through penalty values and implement Lasso regression
for c_param in c_values:
    model = Lasso(alpha=1 / (2 * c_param))
    model.fit(Xpoly_Ridge, y)
    predictions = model.predict(Xpoly_test)

    # Create a 3D scatter plot
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    # Plot predictions
    ax.plot_trisurf(X_test[:, 0], X_test[:, 1], predictions, color=plot_colors[0], alpha=0.5)
    # Plot Training Data
    ax.scatter(X[:, 0], X[:, 1], y, color=plot_colors[1], label='Training Data')
    # Plot configuration
    ax.set_xlabel('X1')
    ax.set_ylabel('X2')
    ax.set_zlabel('Y')
    ax.set_title('Lasso Regression, C = {}'.format(c_param), fontdict={'fontsize': 8.5})
    # Legend
    scatter1_proxy = matplotlib.lines.Line2D([0], [0], linestyle="none", c=plot_colors[0], marker='o')
    scatter2_proxy = matplotlib.lines.Line2D([0], [0], linestyle="none", c=plot_colors[1], marker='v')
    ax.legend([scatter1_proxy, scatter2_proxy], ['Lasso Predictions', 'Training Data'], numpoints=1)
    ax.view_init(azim=60)
plt.show()
ridge_results = []
regression_type = 'Ridge'

poly_features = PolynomialFeatures(degree=degree)
X_poly = poly_features.fit_transform(X)
for penalty in penalties:
```

```
    model = Ridge(alpha=1/(2 * penalty))
    model.fit(X_poly, y)
    result_dict = {
      'Penalty (C)': penalty,
      'Intercept': model.intercept_,
      'Coefficients': np.around(model.coef_, decimals=3),
    }
    ridge_results.append(result_dict)
ridge_results_df = pd.DataFrame(ridge_results)
print(ridge_results_df)
# Generate a grid of feature values
X_test = []
grid = np.linspace(-5, 5, 100)  # Adjust as needed
for i in grid:
    for j in grid:
      X_test.append([i, j])
X_test = np.array(X_test)
# Polynomial degree and penalty values
degree_poly = 5
c_values = [1, 10, 100]
# Lasso Regression
plot_colors = ['limegreen', 'red']
# Create polynomial features
Xpoly_Ridge = PolynomialFeatures(degree=degree_poly).fit_transform(X)
Xpoly_test = PolynomialFeatures(degree=degree_poly).fit_transform(X_test)
# Loop through penalty values and implement Lasso regression
for c_param in c_values:
    model = Lasso(alpha=1 / (2 * c_param))
    model.fit(Xpoly_Ridge, y)
    predictions = model.predict(Xpoly_test)
    # Create a 3D scatter plot
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    # Plot predictions
    ax.plot_trisurf(X_test[:, 0], X_test[:, 1], predictions, color=plot_colors[0], alpha=0.5)
    # Plot Training Data
    ax.scatter(X[:, 0], X[:, 1], y, color=plot_colors[1], label='Training Data')
    # Plot configuration
    ax.set_xlabel('X1')
    ax.set_ylabel('X2')
    ax.set_zlabel('Y')
    ax.set_title('Lasso Regression, C = {}'.format(c_param), fontdict={'fontsize': 8.5})
    # Legend
    scatter1_proxy = matplotlib.lines.Line2D([0], [0], linestyle="none", c=plot_colors[0], marker='o')
    scatter2_proxy = matplotlib.lines.Line2D([0], [0], linestyle="none", c=plot_colors[1], marker='v')
    ax.legend([scatter1_proxy, scatter2_proxy], ['Lasso Predictions', 'Training Data'], numpoints=1)
    ax.view_init(azim=60)
plt.show()
ridge_results = []
regression_type = 'Ridge'
poly_features = PolynomialFeatures(degree=degree)
X_poly = poly_features.fit_transform(X)
for penalty in penalties:
    model = Ridge(alpha=1/(2 * penalty))
    model.fit(X_poly, y)
    result_dict = {
```

```
        'Penalty (C)': penalty,
        'Intercept': model.intercept_,
        'Coefficients': np.around(model.coef_, decimals=3),
    }
    ridge_results.append(result_dict)
ridge_results_df = pd.DataFrame(ridge_results)
print(ridge_results_df)
# Polynomial degree and penalty values
degree_poly = 5
c_values = [1, 10, 100]
# Ridge Regression
# Create polynomial features
Xpoly_Ridge = PolynomialFeatures(degree=degree_poly).fit_transform(X)
Xpoly_test_Ridge = PolynomialFeatures(degree=degree_poly).fit_transform(X_test)
# Loop through penalty values and implement Ridge regression
for c_param in c_values:
    model = Ridge(alpha=1/(2*c_param))
    model.fit(Xpoly_Ridge, y)
    predictions = model.predict(Xpoly_test_Ridge)
    # Create a 3D scatter plot
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    # Plot predictions
    ax.plot_trisurf(X_test[:, 0], X_test[:, 1], predictions, color=plot_colors[0], alpha=0.5)
    # Plot Training Data
    ax.scatter(X[:, 0], X[:, 1], y, color=plot_colors[1], label='Training Data')
    # Plot configuration
    ax.set_xlabel('X1')
    ax.set_ylabel('X2')
    ax.set_zlabel('Y')
    ax.set_title('Ridge Regression, C = {}'.format(c_param), fontdict={'fontsize': 8.5})
    # Legend
    scatter1_proxy = matplotlib.lines.Line2D([0], [0], linestyle="none", c=plot_colors[0], marker='o')
    scatter2_proxy = matplotlib.lines.Line2D([0], [0], linestyle="none", c=plot_colors[1], marker='v')
    ax.legend([scatter1_proxy, scatter2_proxy], ['Ridge Predictions', 'Training Data'], numpoints=1)
    ax.view_init(azim=60)
plt.show()
# Define hyperparameters
model_type = 'Lasso'  # Choose 'Lasso' or 'Ridge' as the regression model
C_values = [1, 10, 100]  # Regularization strength values
# Initialize empty lists to store results
mean_error = []
std_error = []
df_results = []
# Generate polynomial features
Xpoly = PolynomialFeatures(degree_poly).fit_transform(X)
# Loop through each C value
for C in C_values:
    mse_train_temp = []
    mse_test_temp = []
    # Create the regression model with the current C value
    if model_type == 'Lasso':
        model = Lasso(alpha=1/(2*C))
    elif model_type == 'Ridge':
        model = Ridge(alpha=1/(2*C))
    # Perform 5-fold cross-validation
```

```
    kf = KFold(n_splits=5)
    for train, test in kf.split(Xpoly):
        # Fit the model on the training set
        model.fit(Xpoly[train], y[train])
        # Predict on the training set and calculate mean squared error
        ypred_train = model.predict(Xpoly[train])
        mse_train = mean_squared_error(y[train], ypred_train)
        mse_train_temp.append(mse_train)
        # Predict on the test set and calculate mean squared error
        ypred_test = model.predict(Xpoly[test])
        mse_test = mean_squared_error(y[test], ypred_test)
        mse_test_temp.append(mse_test)
    # Create a dictionary of results including mean and variance
    result_dict = {
        'C value': C,
        'mse train results': np.around(mse_train_temp, decimals=3),
        'mean mse train': np.around(np.array(mse_train_temp).mean(), decimals=3),
        'std mse train': np.around(np.array(mse_train_temp).std(), decimals=3),
        'mse test': np.around(mse_test_temp, decimals=3),
        'mean mse test': np.around(np.array(mse_test_temp).mean(), decimals=3),
        'std mse test': np.around(np.array(mse_test_temp).std(), decimals=3)
    }
    df_results.append(result_dict)
# Create a DataFrame to store the results
df_kf_results = pd.DataFrame(df_results)
print(df_kf_results)
# Extract relevant data for plotting
C_values = df_kf_results['C value']
mean_mse_test = df_kf_results['mean mse test']
std_mse_test = df_kf_results['std mse test']
# Create the plot
plt.figure(figsize=(10, 6))
plt.errorbar(C_values, mean_mse_test, yerr=std_mse_test, fmt='-o', color='magenta', label='Mean
Test MSE')
# Set plot labels and title
plt.xlabel('Regularization Strength (C)')
plt.ylabel('Mean Square Error')
plt.title('K Fold Cross Validation - Lasso Regression')
# Customize the x-axis scale for better visualization
plt.xscale('log')  # Using a logarithmic scale for C values
# Add a legend
plt.legend()
# Show the plot
plt.show()
# ********** K Fold Cross Validation - Ridge Regression **********
# Define hyperparameters
model_type = 'Ridge'  # Choose 'Lasso' or 'Ridge' as the regression model
C_values = [1, 10, 100]  # Regularization strength values
# Initialize empty lists to store results
mean_error = []
std_error = []
df_results = []
# Generate polynomial features
Xpoly = PolynomialFeatures(degree_poly).fit_transform(X)
# Loop through each C value
for C in C_values:
```

```
    mse_train_temp = []
    mse_test_temp = []
    # Create the regression model with the current C value
    if model_type == 'Lasso':
        model = Lasso(alpha=1/(2*C))
    elif model_type == 'Ridge':
        model = Ridge(alpha=1/(2*C))
    # Perform 5-fold cross-validation
    kf = KFold(n_splits=5)

    for train, test in kf.split(Xpoly):
        # Fit the model on the training set
        model.fit(Xpoly[train], y[train])
        # Predict on the training set and calculate mean squared error
        ypred_train = model.predict(Xpoly[train])
        mse_train = mean_squared_error(y[train], ypred_train)
        mse_train_temp.append(mse_train)

        # Predict on the test set and calculate mean squared error
        ypred_test = model.predict(Xpoly[test])
        mse_test = mean_squared_error(y[test], ypred_test)
        mse_test_temp.append(mse_test)
    # Create a dictionary of results including mean and variance
    result_dict = {
        'C value': C,
        'mse train results': np.around(mse_train_temp, decimals=3),
        'mean mse train': np.around(np.array(mse_train_temp).mean(), decimals=3),
        'std mse train': np.around(np.array(mse_train_temp).std(), decimals=3),
        'mse test': np.around(mse_test_temp, decimals=3),
        'mean mse test': np.around(np.array(mse_test_temp).mean(), decimals=3),
        'std mse test': np.around(np.array(mse_test_temp).std(), decimals=3)
    }
    df_results.append(result_dict)
# Create a DataFrame to store the results
df_kf_results = pd.DataFrame(df_results)
print(df_kf_results)
# Extract relevant data for plotting
C_values = df_kf_results['C value']
mean_mse_test = df_kf_results['mean mse test']
std_mse_test = df_kf_results['std mse test']
# Create the plot
plt.figure(figsize=(10, 6))
plt.errorbar(C_values, mean_mse_test, yerr=std_mse_test, fmt='-o', color='magenta', label='Mean
Test MSE')
# Set plot labels and title
plt.xlabel('Regularization Strength (C)')
plt.ylabel('Mean Square Error')
plt.title('K Fold Cross Validation - Ridge Regression')
# Customize the x-axis scale for better visualization
plt.xscale('log')  # Using a logarithmic scale for C values
# Add a legend
plt.legend()

# Show the plot
plt.show()
```