IE406 Machine Learning - Assignment 3

202201140* | *Harsh Gajjar

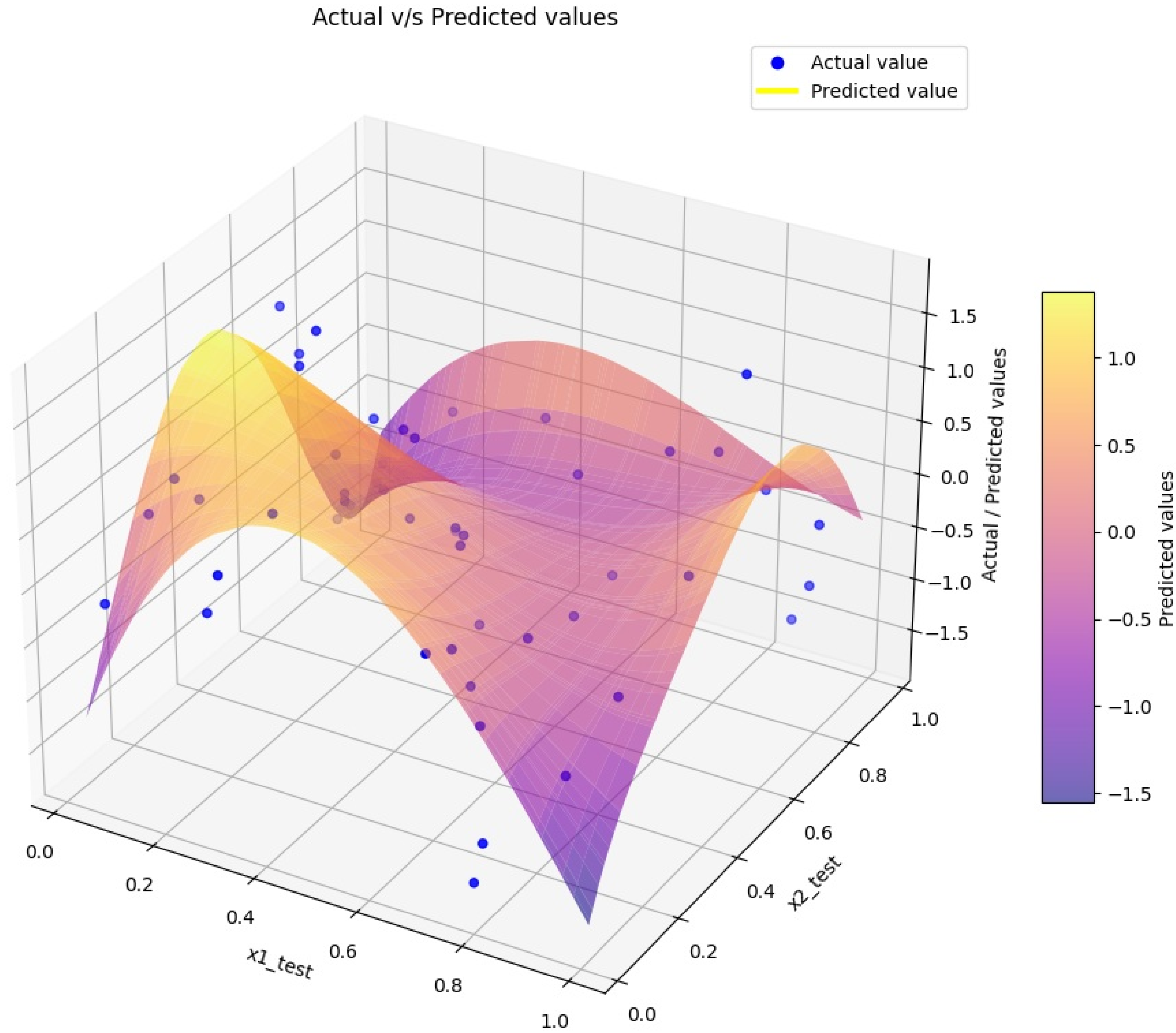
Importing libraries

In [15]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
from mpl_toolkits.mplot3d import Axes3D
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

1. Generate 400 data points in R^2 from the uniform distribution U [0,1]. Construct the training set T = {(x11, x21,y1),(x21,x22,y2),……,(x100 1, x100 2,y100)} using the relation Yi = sin(2 π (xi1^2+ xi2^2)) + ϵi where ϵi ~ N(0,0.25). In the similar way, construct a testing set of size 50 i,e. Test = {(x’11, x’12, y’1),(x’21, x’22 ,y’2),……,(x’50 1, x’50 2, y’50)}. Estimate the regularized polynomial regression of order 6 with direct method and obtain the 3d plot on test set along with test data points. Find the NMSE, RMSE, MAE and R^2.

In [2]:
```
# generating training set
x1_train = np.random.uniform(0, 1, 400)
x2_train = np.random.uniform(0, 1, 400)
epsilon_train = np.random.normal(0, np.sqrt(0.25), 400)
y_train = np.sin(2 * np.pi * (x1_train**2 + x2_train**2)) + epsilon_train


# generating testing set
x1_test = np.random.uniform(0, 1, 50)
x2_test = np.random.uniform(0, 1, 50)
epsilon_test = np.random.normal(0, np.sqrt(0.25), 50)
y_test = np.sin(2 * np.pi * (x1_test**2 + x2_test**2)) + epsilon_test


# generating A matrix
def generate_A(x1, x2, degree):
    num_terms = (degree + 1) * (degree + 2) // 2
    A = np.zeros((len(x1), num_terms))
    idx = 0
    for i in range(degree + 1):
        for j in range(i + 1):
            if (i+j) <= degree:
                A[:, idx] = x1**j * x2**(i - j)
                idx += 1
    return A


# generating u matrix
def generate_u(A, Y, lambdaa):
    I = np.eye(A.shape[1])
    return np.linalg.inv(A.T @ A + lambdaa * I) @ A.T @ Y


lambdaa = 0.0001
A_train = generate_A(x1_train, x2_train, 6)
u = generate_u(A_train, y_train, lambdaa)


# plotting
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

x1_test_grid, x2_test_grid = np.meshgrid(np.sort(x1_test), np.sort(x2_test))
x1_test_flat = x1_test_grid.flatten()
x2_test_flat = x2_test_grid.flatten()
A_test_flat = generate_A(x1_test_flat, x2_test_flat, 6)
y_pred_test_flat = A_test_flat @ u
y_pred_test_grid = y_pred_test_flat.reshape(x1_test_grid.shape)

surf = ax.plot_surface(x1_test_grid, x2_test_grid, y_pred_test_grid, cmap='plasma', alpha=0.6, edgecolor='none'

ax.scatter(x1_test, x2_test, y_test, color='blue', label='Actual value', marker='o')
ax.set_xlabel('x1_test')
ax.set_ylabel('x2_test')
ax.set_zlabel('Actual / Predicted values')
```

```
ax.set_title('Actual v/s Predicted values')

legend_elements = [Line2D([0], [0], marker='o', color='w', markerfacecolor='b', markersize=8, label='Actual valu
                   Line2D([0], [0], color='yellow', lw=3, label='Predicted value')]
ax.legend(handles=legend_elements)

cbar = plt.colorbar(surf, shrink=0.5, aspect=10)
cbar.set_label('Predicted values')

plt.tight_layout()
plt.show()


# calculating errors
y_test_flat = np.sin(2 * np.pi * (x1_test_flat**2 + x2_test_flat**2))
rmse = np.sqrt(mean_squared_error(y_test_flat, y_pred_test_flat))
nmse = mean_squared_error(y_test_flat, y_pred_test_flat) / np.var(y_test_flat)
mae = mean_absolute_error(y_test_flat, y_pred_test_flat)
r2 = r2_score(y_test_flat, y_pred_test_flat)

print(f'RMSE: {rmse}')
print(f'NMSE: {nmse}')
print(f'MAE: {mae}')
print(f'R^2: {r2}')
```



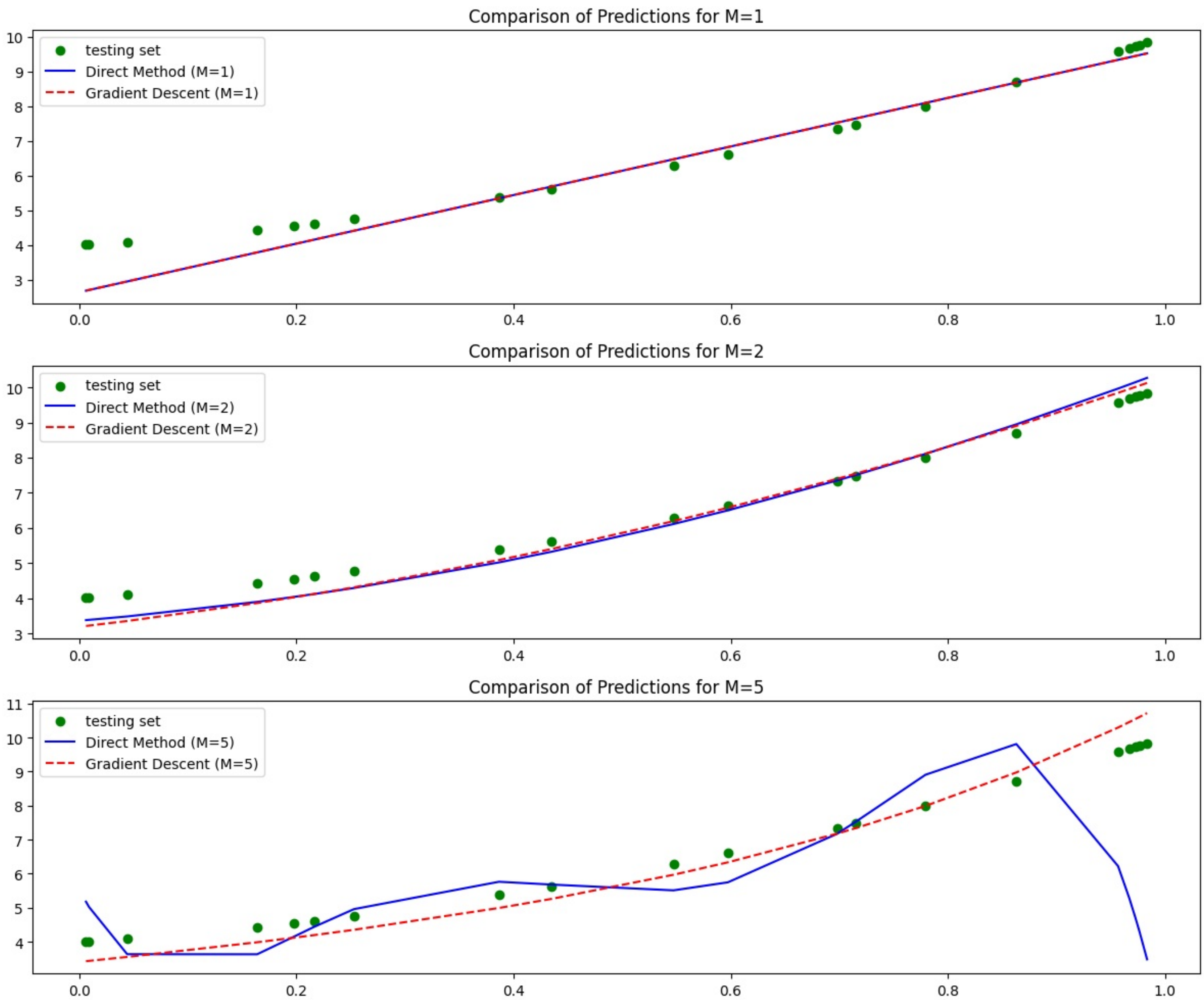

```
RMSE: 0.45864064609830923
NMSE: 0.42044754142666374
MAE: 0.36663865883798563
R^2: 0.5795524585733363
```

2. Consider the dataset 1. You will find the only one independent variable (Income in thousand dollars) and one target variable (Card Balance in hundred dollars). Train the polynomial regression model with M = 1, 2 and 5 using the gradient descent method and obtain the plots of predictions upon training set and test. Compare the predictions obtained by gradient descent method and direct method with in terms of RMSEs.

In [10]:
```
# reading data
train_data = pd.read_csv("./train.csv")
```

```python
test_data = pd.read_csv("./test.csv")


# training set
x_train = train_data['Income'].values
y_train = train_data['Balance'].values


# testing set
x_test = test_data['Income'].values
y_test = test_data['Balance'].values


# generating A matrix
def generate_A(x, degree):
    num_terms = degree + 1
    A = np.zeros((len(x), num_terms))
    for i in range(num_terms):
        A[:, i] = x**i
    return A

A1_train = generate_A(x_train, 1)
A2_train = generate_A(x_train, 2)
A5_train = generate_A(x_train, 5)

A1_test = generate_A(x_test, 1)
A2_test = generate_A(x_test, 2)
A5_test = generate_A(x_test, 5)


# generating u matrix (direct method)
def generate_u(A, Y):
    I = np.eye(A.shape[1])
    return np.linalg.inv(A.T @ A) @ A.T @ Y

u1 = generate_u(A1_train, y_train)
u2 = generate_u(A2_train, y_train)
u5 = generate_u(A5_train, y_train)


# generating u matrix (gradient descent method)
def generate_u_GD(A, Y):
    u = np.zeros(A.shape[1])
    alpha = 0.1
    for i in range(100):
        u = u - (alpha * A.T @ (A @ u - Y))
    return u

u1_gd = generate_u_GD(A1_train, y_train)
u2_gd = generate_u_GD(A2_train, y_train)
u5_gd = generate_u_GD(A5_train, y_train)


# predicting values
y_pred_test1 = A1_test @ u1
y_pred_test2 = A2_test @ u2
y_pred_test5 = A5_test @ u5

y_pred_test1_gd = A1_test @ u1_gd
y_pred_test2_gd = A2_test @ u2_gd
y_pred_test5_gd = A5_test @ u5_gd


# plotting
x_test_sorted = np.sort(x_test)
y_pred_test1_sorted = y_pred_test1[np.argsort(x_test)]
y_pred_test2_sorted = y_pred_test2[np.argsort(x_test)]
y_pred_test5_sorted = y_pred_test5[np.argsort(x_test)]
y_pred_test1_gd_sorted = y_pred_test1_gd[np.argsort(x_test)]
y_pred_test2_gd_sorted = y_pred_test2_gd[np.argsort(x_test)]
y_pred_test5_gd_sorted = y_pred_test5_gd[np.argsort(x_test)]

plt.figure(figsize=(12, 10))

plt.subplot(3, 1, 1)
plt.scatter(x_test, y_test, label='testing set', color='g')
plt.plot(x_test_sorted, y_pred_test1_sorted, label='Direct Method (M=1)', color='b')
plt.plot(x_test_sorted, y_pred_test1_gd_sorted, label='Gradient Descent (M=1)', color='r', linestyle='--')
plt.title('Comparison of Predictions for M=1')
plt.legend()

plt.subplot(3, 1, 2)
plt.scatter(x_test, y_test, label='testing set', color='g')
```

```python
plt.plot(x_test_sorted, y_pred_test2_sorted, label='Direct Method (M=2)', color='b')
plt.plot(x_test_sorted, y_pred_test2_gd_sorted, label='Gradient Descent (M=2)', color='r', linestyle='--')
plt.title('Comparison of Predictions for M=2')
plt.legend()

plt.subplot(3, 1, 3)
plt.scatter(x_test, y_test, label='testing set', color='g')
plt.plot(x_test_sorted, y_pred_test5_sorted, label='Direct Method (M=5)', color='b')
plt.plot(x_test_sorted, y_pred_test5_gd_sorted, label='Gradient Descent (M=5)', color='r', linestyle='--')
plt.title('Comparison of Predictions for M=5')
plt.legend()

plt.tight_layout()
plt.show()


# calculating errors
rmse1 = np.sqrt(mean_squared_error(y_test, y_pred_test1))
rmse1_gd = np.sqrt(mean_squared_error(y_test, y_pred_test1_gd))
rmse2 = np.sqrt(mean_squared_error(y_test, y_pred_test2))
rmse2_gd = np.sqrt(mean_squared_error(y_test, y_pred_test2_gd))
rmse5 = np.sqrt(mean_squared_error(y_test, y_pred_test5))
rmse5_gd = np.sqrt(mean_squared_error(y_test, y_pred_test5_gd))

print(f"RMSE for M=1 (Direct Method): {rmse1}")
print(f"RMSE for M=1 (Gradient Descent): {rmse1_gd}")
print(f"RMSE for M=2 (Direct Method): {rmse2}")
print(f"RMSE for M=2 (Gradient Descent): {rmse2_gd}")
print(f"RMSE for M=5 (Direct Method): {rmse5}")
print(f"RMSE for M=5 (Gradient Descent): {rmse5_gd}")
```



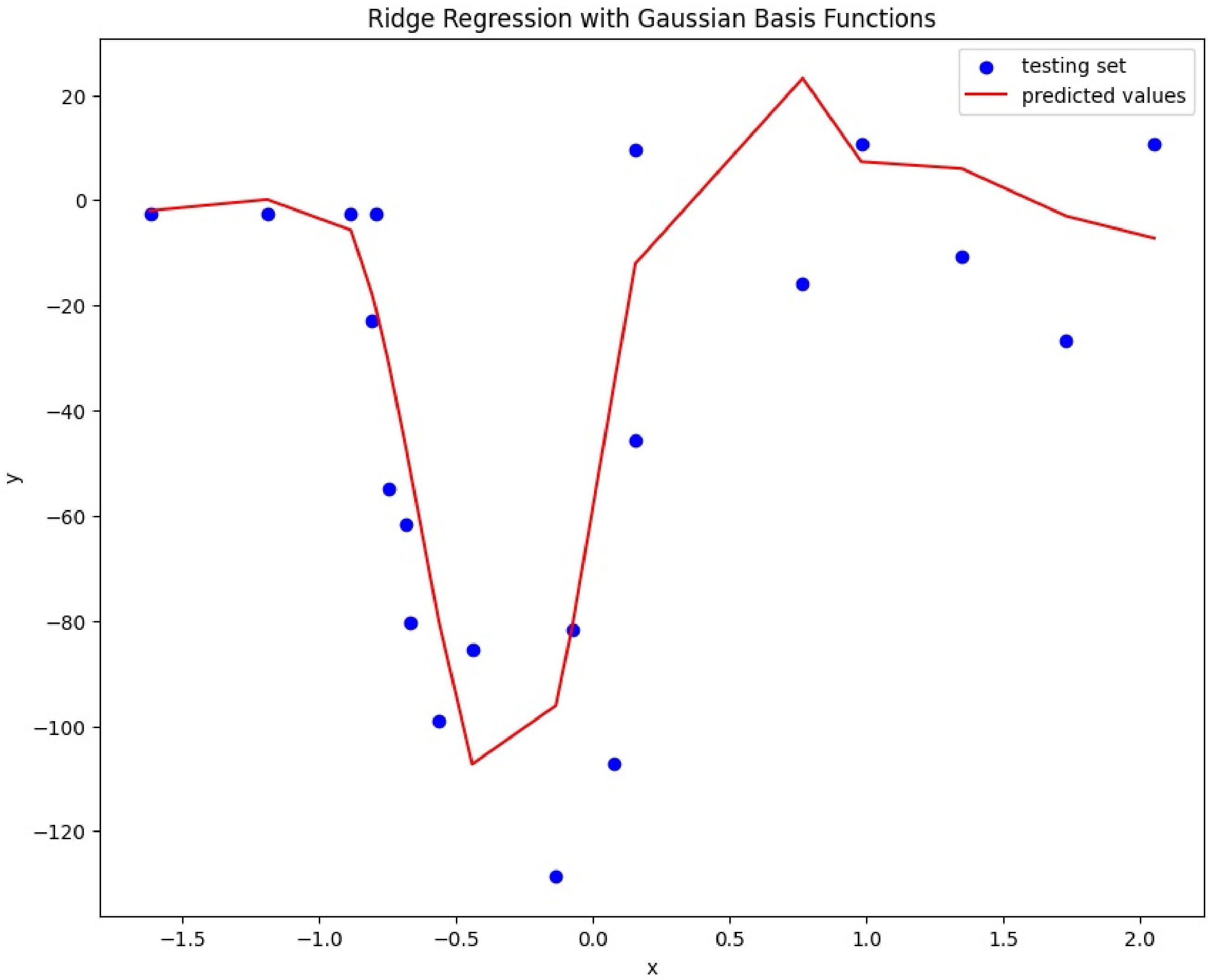

```
RMSE for M=1 (Direct Method): 0.5625799860234558
RMSE for M=1 (Gradient Descent): 0.5618039742097394
RMSE for M=2 (Direct Method): 0.4116989672523791
RMSE for M=2 (Gradient Descent): 0.4167016951572596
RMSE for M=5 (Direct Method): 2.577442141182683
RMSE for M=5 (Gradient Descent): 0.5252278917648591
```

3. Consider the motorcycle dataset. Estimate a regularized least square regression model (Also called Ridge Regression model) with Gaussian basis functions. Obtain the plot of estimated functions along with data points. Also obtain the RMSE, MAE,

NMSE and R^2 for evaluating the quality of fit.

In [26]:
```python
# reading data
df = pd.read_csv("./motorcycle.csv")
x = df['x'].values.reshape(-1, 1)
y = df['y'].values


# Standardizing the features (mean = 0, variance = 1)
scaler = StandardScaler()
x = scaler.fit_transform(x)


# Splitting into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x_scaled, y, test_size=0.15, random_state=42)


# Gaussian Basis Function
def gaussian_basis(x, centers, width):
    result = []
    for c in centers:
        basis = np.exp(-0.5 * ((x - c) / width) ** 2)
        result.append(basis)
    return np.column_stack(result)


# Parameters for Gaussian Basis Functions
n_basis = 25
centers = np.linspace(min(x_train), max(x_train), n_basis)
width = 0.25  # np.std(x_train)


# generating A matrix
A_train = gaussian_basis(x_train, centers, width)
A_test = gaussian_basis(x_test, centers, width)


# Regularization parameter
lambdaa = 1.0
I = np.identity(A_train.shape[1])


# Ridge regression weight calculation
w = np.linalg.inv(A_train.T @ A_train + lambdaa * I) @ A_train.T @ y_train


# Predictions on test data
y_pred_test = A_test @ w


# Calculating errors
rmse = np.sqrt(mean_squared_error(y_test, y_pred_test))
mae = mean_absolute_error(y_test, y_pred_test)
nmse = mean_squared_error(y_test, y_pred_test) / np.var(y_test)
r2 = r2_score(y_test, y_pred_test)

print("For testing set\n")
print(f'RMSE: {rmse}')
print(f'MAE: {mae}')
print(f'NMSE: {nmse}')
print(f'R^2: {r2}')


# Plotting
sorted_indices = np.argsort(x_test.flatten())
x_test_sorted = x_test[sorted_indices]
y_test_sorted = y_test[sorted_indices]
y_pred_test_sorted = y_pred_test[sorted_indices]

plt.figure(figsize=(10, 8))
plt.scatter(x_test_sorted, y_test_sorted, color='blue', label='testing set')
plt.plot(x_test_sorted, y_pred_test_sorted, color='red', label='predicted values')
plt.title('Ridge Regression with Gaussian Basis Functions')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

```
For testing set

RMSE: 25.983232929703888
MAE: 20.071051870894955
NMSE: 0.3642313052484549
R^2: 0.6357686947515451
```



In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js