| SPECIAL TOPICS IN PATTERN RECOGNITION |
(EEE8501.01-00)
April 2th, 2024

# Project#1

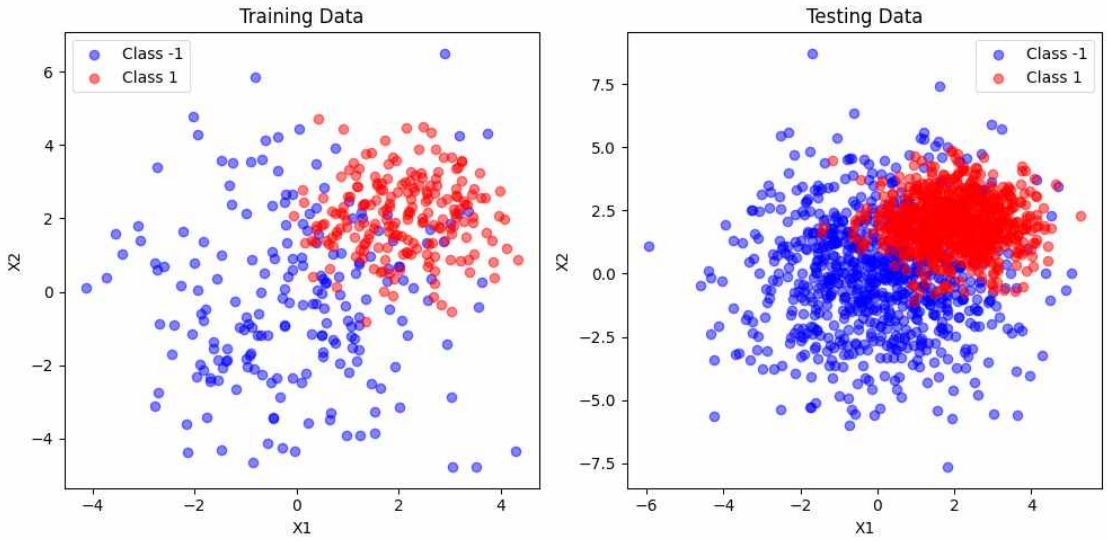
Yonsei University
Department of Electrical and Electronic Engineering
Hari Kang (ID: 2023321196)

---

## I. In the main report content (not in appendix), plot the training input features (Xtrain) and the test input features (Xtest) in subplot 1 and subplot 2 (with '-1 labeled points in blue and '+1' labeled points in red) respectively.

Two sets of samples, each consisting of 1000 instances, were prepared with distinct Gaussian distributions. Every data sample encompasses two features. For the first set of 1000 samples, they adhere to the distribution (X1,X2)~N([0,0],[[3, 0], [0, 5]]), and are assigned a label of -1. The second set of 1000 samples follows the distribution (X1,X2)~N([2,2],[[1,0],[0,1]]) and are labeled as 1. The collection of samples with mean [0,0] is designated as set a, and the collection with mean [2,2] as set b.

For constructing the training set, 200 samples from set a and 200 samples from set b were selected, totaling 400 samples. For the test set, 800 samples from each of sets a and b were selected, amounting to 1600 samples. Consequently, the training set comprises 400 samples, with 200 samples from each of the two classes, and the test set consists of 1600 samples, with 800 from each class.




## II. Perform classification by minimizing the RSS using linear regression model.

### A. Train the data using LR(minimizing RSS) without regularization.

The primary objective in finding the most efficient linear model for distinguishing between two sets of data is to identify the optimal weights, w, in the equation

y=wX+b. The optimal weights can be computed using the following equation:

$$W = (X^T X)^{-1} X^T y$$

Here, X represents the training data (including features), and y represents the target values.

Using Python, this formula can be implemented to develop and train the model. Following the training phase, predictions on the test data are carried out, and the Residual Sum of Squares (RSS) is calculated. Linear Regression Model is presented below:

```
def LR(X,y):
    X_bias = np.concatenate(np.ones((X.shape[0], 1)), X)
    return  np.linalg.inv(X_bias.T.dot(X_bias)).dot(X_bias.T).dot(y)
```

RSS is presented below:

```
def RSS(y_pred,y):
    return np.sum((y - y_pred) ** 2)
```

## B. Compute the predicted outputs of test data using optimized LR.

```
w = LR(X_train,y_train)
X_test_bias = np.c_[np.ones((X_test.shape[0], 1)), X_test]
y_test_pred = X_test_bias @ w
ktest = RSS(y_test_pred,y_test)
print(ktest)

X_train_bias = np.c_[np.ones((X_train.shape[0], 1)), X_train]
y_pred = X_train_bias @ w
ktrain = RSS(y_pred,y_train)
print(ktrain)
```

✓ 0.0s

```
890.7651937447724
200.85837829933627
```

```
y_test_pred[:10]
```

✓ 0.0s

```
array([[-1.23034793],
       [-0.0400722 ],
       [-0.55152803],
       [-0.11109687],
       [-0.73535764],
       [-0.69595149],
       [-1.28614292],
       [-1.37477233],
       [-0.38090378],
       [ 0.44324337]])
```

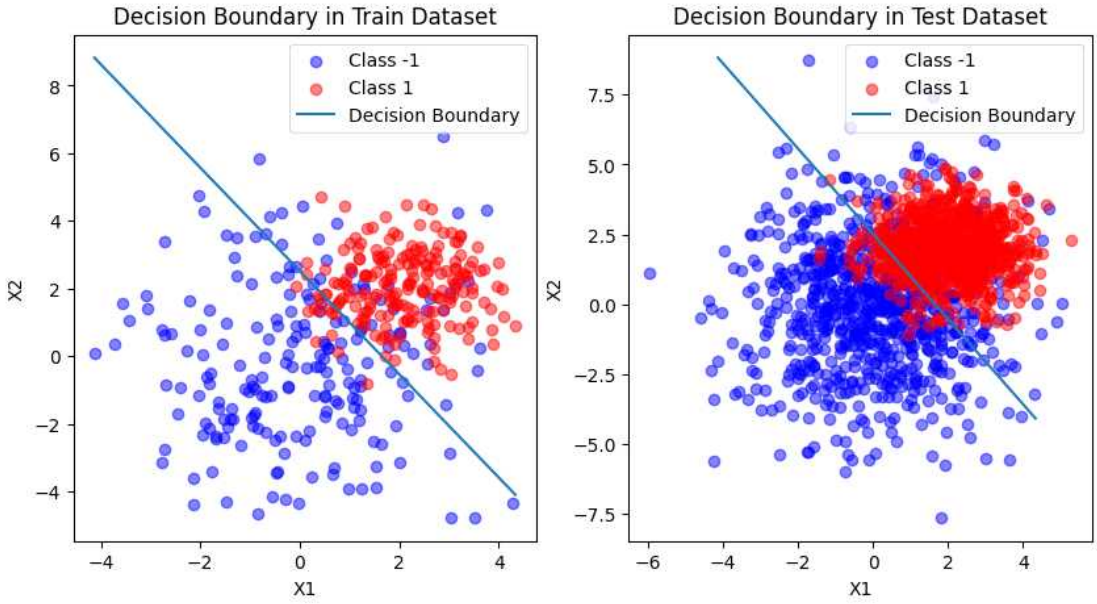
```
y_test[:10]
```

✓ 0.0s

```
array([[-1.],
       [-1.],
       [-1.],
       [-1.],
       [-1.],
       [-1.],
       [-1.],
       [-1.],
       [-1.],
       [-1.]])
```

We can predict output using our model. Using the optimal weights obtained through Linear Regression, the predicted values for both the train and test data were computed and their similarity to the actual values was evaluated using the Residual-Sum-of-Squares (RSS). Above is a representation of a portion of the test data's actual y values alongside the corresponding predicted values from the test data, which overall demonstrated unsatisfactory accuracy. With our dataset, The test error is 890.77. and train error is 200.86.

## C. plot the decision boundary for the optimal Linear regression model at threshold 0.




Upon examining the results for ten test data points and observing the values of the Residual-Sum-of-Squares (RSS), the performance appeared to be suboptimal. However, by analyzing the decision boundary, it can be confirmed that it optimally separates the training and testing data as effectively as possible. Given that there are only two feature dimensions, it is inevitable that some data points will not be perfectly separated, leading to unsatisfactory results. From this observation, it can be inferred that increasing the number of dimensions may improve accuracy.

## III. Perform classification using the RM model[1] for model orders ranging from 1 to 6.

### A. Train the data using the RM Model for regression without Regularization.

Before train the data, Let's clarify RM Model. According to 'example usage' in offered pdf, There is X, namely np.array([1,2],[3,4],[5,6]). The data can be interpreted as three sample data points, each with two feature dimensions. When this data is inputted into the RM (Regression Model) method with an order of 2, the result obtained is as follows:

```
[[ 1.  1.  2.  1.  4.  3.  6.]
 [ 1.  3.  4.  9. 16. 21. 28.]
 [ 1.  5.  6. 25. 36. 55. 66.]]
```

This shows that the data has transformed into three sample data points, each now having seven feature dimensions. From this, it can be inferred that this mechanism is suitable for datasets that are difficult to separate with fewer feature dimensions.

$$P = \begin{bmatrix} 1 & x_{1,1} & x_{2,1} & x_{1,1}^2 & x_{1,1}x_{2,1} & x_{2,1}^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{1,m} & x_{2,m} & x_{1,m}^2 & x_{1,m}x_{2,m} & x_{2,m}^2 \end{bmatrix}.$$

We can denote this concept as above matrix. And The optimal weights can be computed using the following equation:

$$W = P^T(P^T P)^{-1} y$$

and if we explain with code line by line, it is more easier to understand with equation below.

$$g_{RM}(\boldsymbol{w}, \boldsymbol{x}) = \boldsymbol{w}^T p_{RM}(\boldsymbol{x})$$
$$= w_0 + \underbrace{\sum_{k=1}^{r} w_k \left( \sum_{j=1}^{l} x_j \right)^k}_{\text{Linear}} + \sum_{k=1}^{r} \sum_{j=1}^{l} \beta_{k,j} x_j^k + \sum_{k=2}^{r} \underbrace{\left( \sum_{i=1}^{l} \gamma_{k,i} x_i \right)}_{\text{Combined power}} \left( \sum_{j=1}^{l} x_j \right)^{k-1} \quad (45)$$
$$l, r \geqslant 2,$$

```
def RM(X, order):
    # Build regressor matrix P (mxK):
    # order = desired order of approximation,
    # X = input matrix (mxl), K = number of parameters to be est.
    # m = number of data samples, l = input dimension.
    m, l = X.shape
    MM1 = []
    MM3 = []

    Msum = np.sum(X, axis=1) # this part means (sum_{j=1}^{l} x_j)^k eqaution inner part

    for i in range(1, order+1):
        M1 = np.zeros((m, l)) # store individual powers of input feature(a.k.a l)
        M3 = np.zeros((m, l)) # combined powers involving the sum of all features
        for k in range(l):
            M1[:, k] = X[:, k]**i  # i-th power of the k-th feature
            if i > 1:
                M3[:, k] = X[:, k] * Msum**(i-1) # (sum_{j=1}^{l} x_j)^{i-1}
        MM1.append(M1)
        if i > 1:
            MM3.append(M3)
    P = np.concatenate([np.ones((m, 1)), np.concatenate(MM1, axis=1), np.concatenate(MM3, axis=1)], axis=1) # concatenates a column of ones (for the intercept term)
    return P
```

```
LR_weights = {}
for i in range(1,7):
    train = P_train[f'P_train{i}']
    LR_weights[f'LR_train{i}'] = LR(train,y_train)
    print(f'{i}th order RM regression weight shape is ',LR_weights[f'LR_train{i}'].shape)
```

✓ 0.0s

```
1th order RM regression weight shape is  (3, 1)
2th order RM regression weight shape is  (7, 1)
3th order RM regression weight shape is  (11, 1)
4th order RM regression weight shape is  (15, 1)
5th order RM regression weight shape is  (19, 1)
6th order RM regression weight shape is  (23, 1)
```

## B. Compute the predicted ouputs of test data using RM regression.

Now, We get trained weight with different order of P. As the order increases, it can be observed that the feature dimension also increases.

After training, The predicted values, y_pred, have been obtained, and a portion of these is represented according to their respective orders.

```
1th order RM model
[[-1.23034793]
 [-0.0400722 ]
 [-0.55152803]
 [-0.11109687]
 [-0.73535764]
 [-0.69595149]
 [-1.28614292]
 [-1.37477233]
 [-0.38090378]
 [ 0.44324337]]
```

```
2th order RM model
[[-0.62137208]
 [-0.56185232]
 [ 0.19821744]
 [ 0.38072902]
 [-0.58761883]
 [-0.00802914]
 [-0.31944213]
 [-0.38850694]
 [ 0.29162651]
 [ 0.19514278]]
```

```
3th order RM model
[[-1.6896707 ]
 [-0.8885043 ]
 [-0.60273311]
 [ 0.40905198]
 [-1.42624719]
 [-0.72510033]
 [-1.67019733]
 [-1.67620637]
 [-0.29856705]
 [ 0.40191894]]
```

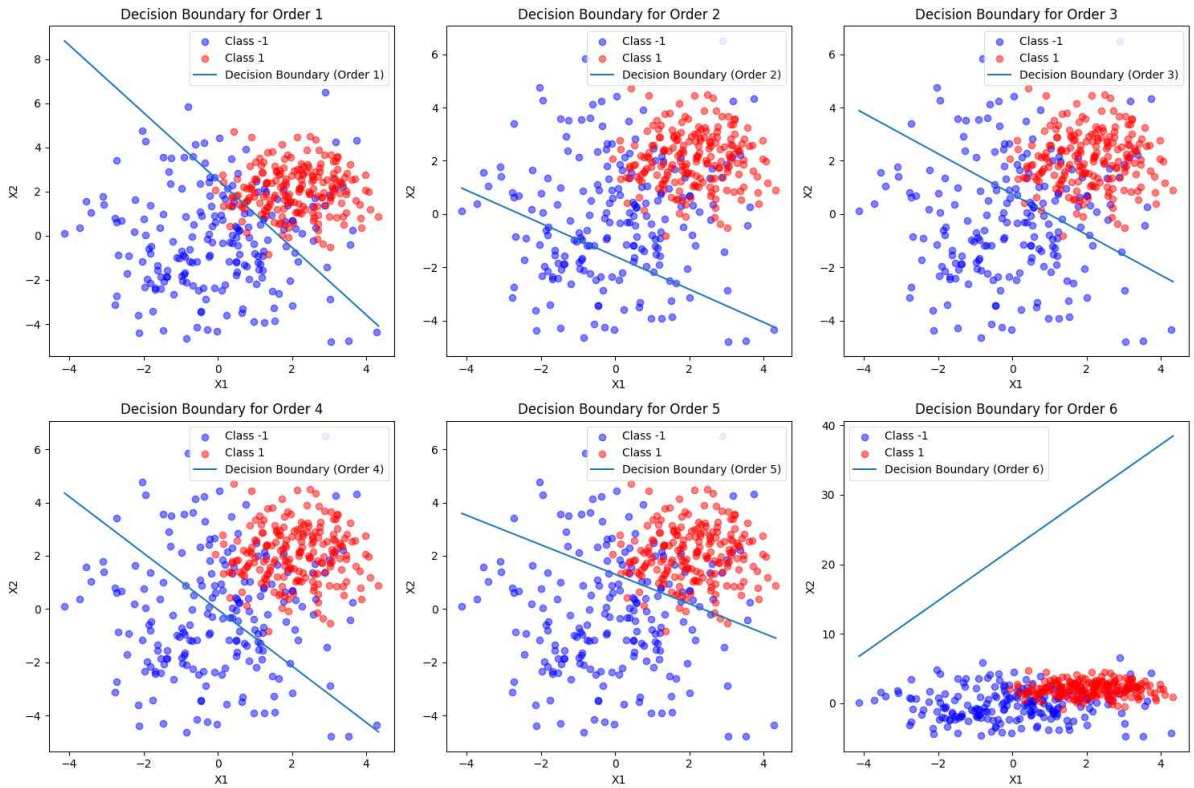
```
4th order RM model
[[-0.54998558]
 [-0.8953338 ]
 [-0.09835089]
 [ 0.30804511]
 [-0.64449398]
 [-0.43783671]
 [-0.46254993]
 [-0.37620221]
 [ 0.06751411]
 [ 0.64091916]]
```

```
5th order RM model
[[-1.69585931e+00]
 [-1.25179520e+00]
 [-1.06447909e+00]
 [-5.92059506e-03]
 [-1.77396385e+00]
 [-3.77504875e-01]
 [-1.18596833e+00]
 [-5.47668297e-01]
 [-7.31956249e-01]
 [ 1.42492398e-03]]
```

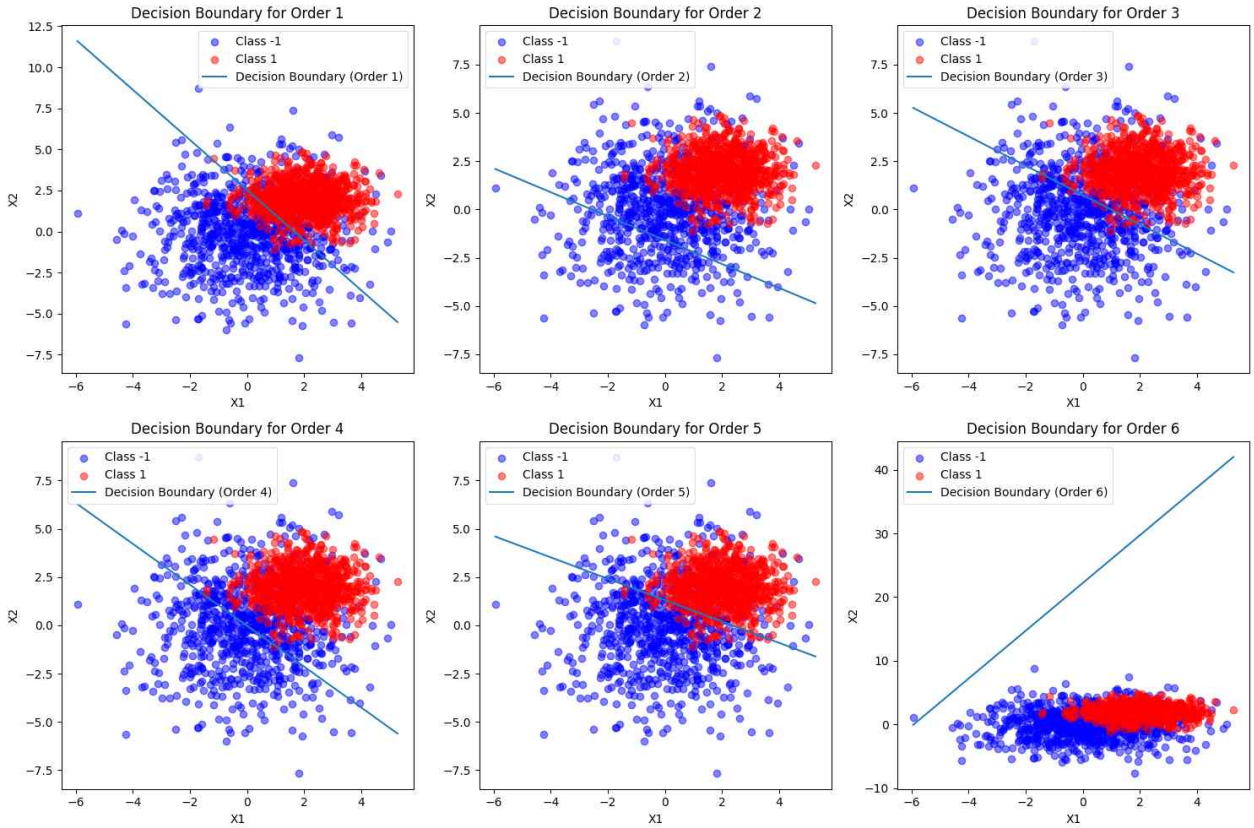
```
6th order RM model
[[ 1.49735506]
 [-0.87543503]
 [ 2.71669575]
 [ 1.85326477]
 [ 1.87012923]
 [ 1.09496638]
 [ 1.78957805]
 [ 0.6110908 ]
 [ 2.88361372]
 [ 1.02345012]]
```

```
y_test[:10]
✓ 0.0s
array([[-1.],
       [-1.],
       [-1.],
       [-1.],
       [-1.],
       [-1.],
       [-1.],
       [-1.],
       [-1.],
       [-1.]])
```

From the results, it can be observed that the fifth order demonstrates the best performance, indicating that a higher order does not necessarily equate to better results.

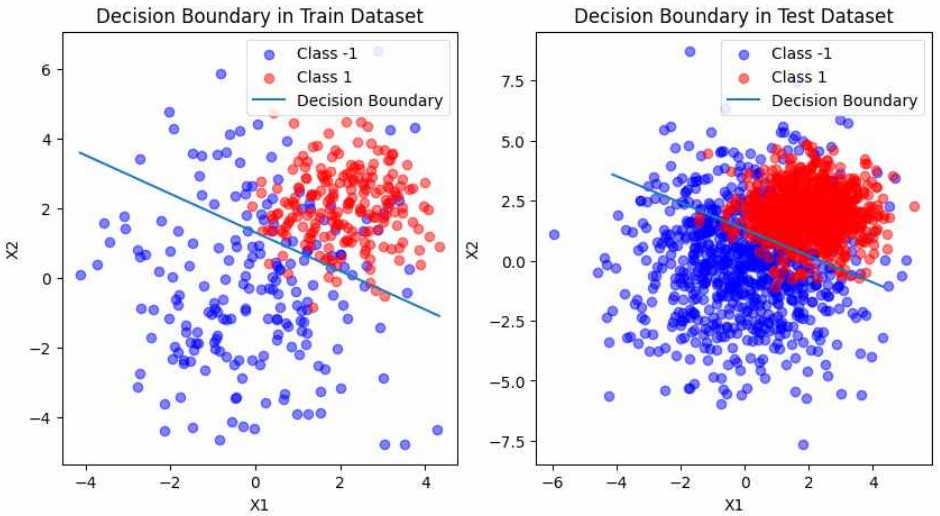



From the graphs, it can be observed that the 5th order polynomial regression model provides the best separation for the training data.




From the graphs, it can be observed that the 5th order polynomial regression model provides the best separation for the tesing data.

C. Plot Decision boundary for the optimal RM regression model at threshold 0 for an RM model at order 5.




## IV. Compare the test classification accuracies of the above classifiers.

A. calculate the test classification accuracy.

```
def classacc(ytest,ypred):
    y_pred_class = np.where(ypred >= 0, 1, -1)
    correct_classifications = y_test == y_pred_class
    accuracy = np.mean(correct_classifications)
    return accuracy
```

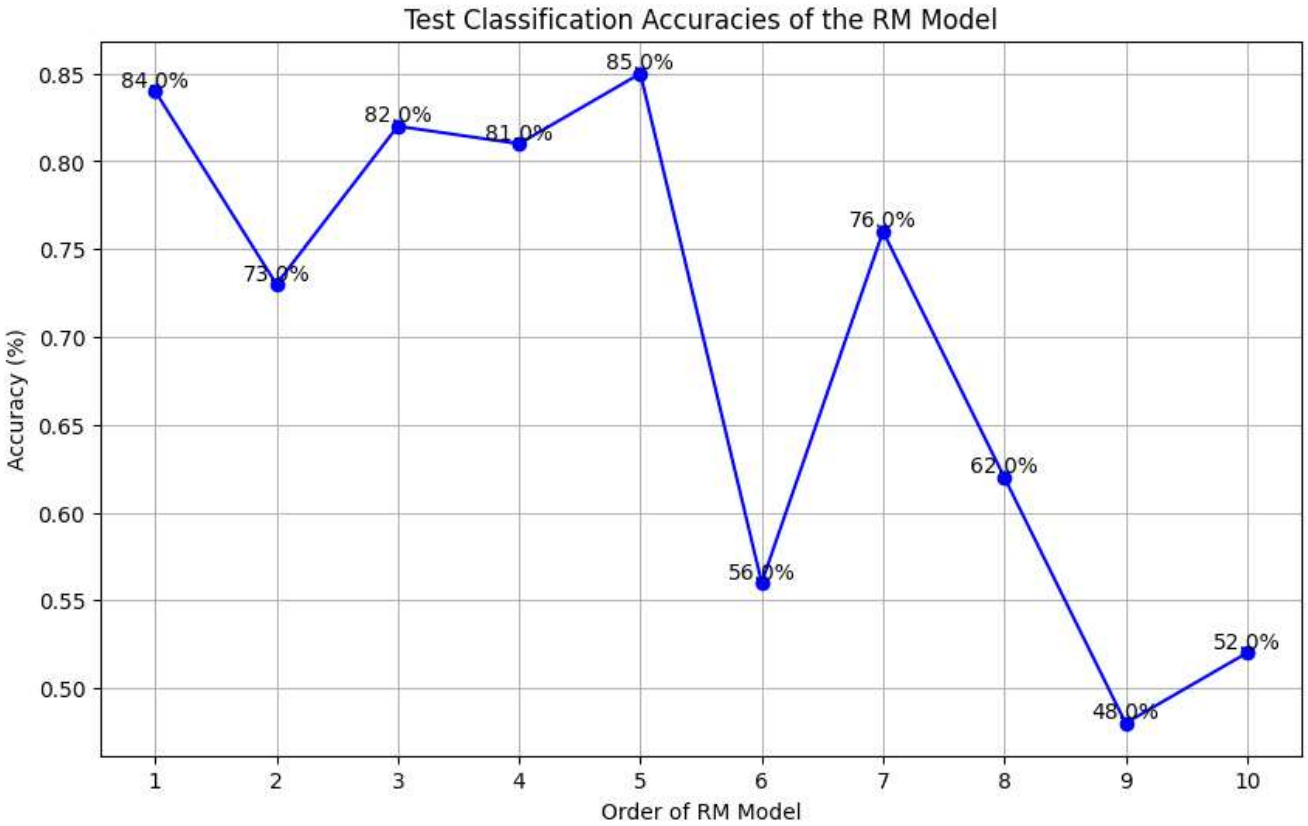
Since the predicted values are real numbers and the desired output is binary classes,

values greater than 0 are designated as class 1, while those less than 0 are assigned to class -1. These binary values are then compared to measure the accuracy.

RM 1th order RM Model acc is 83.62 %
RM 2th order RM Model acc is 72.56 %
RM 3th order RM Model acc is 82.19 %
RM 4th order RM Model acc is 80.50 %
RM 5th order RM Model acc is 85.06 %
RM 6th order RM Model acc is 55.94 %
RM 7th order RM Model acc is 76.25 %
RM 8th order RM Model acc is 61.62 %
RM 9th order RM Model acc is 48.50 %
RM 10th order RM Model acc is 52.25 %
LR Model acc is 83.62 %

The statement indicates the accuracy of the RM model for each order, as well as the accuracy of the LR model.

## B. plot the test classification accuracies of the RM model for orders 1 to 10.




When incrementally increasing the order from 1 to 10, it has been observed that the fifth order exhibits the highest performance. This indicates that merely increasing the order, i.e., augmenting the feature vectors used to describe the data, does not guarantee an improvement in performance. This observation underscores the importance of a judicious selection of model complexity in achieving optimal performance.

## V. Observations and Discussion : provide your observation and discussion regarding the above results.

For the dataset I created, linear regression (LR) exhibited an accuracy of 83.62%. The RM model, evaluated across orders from one to ten, demonstrated the following accuracies: RM 1st order at 83.62%, RM 2nd order at 72.56%, RM 3rd order at 82.19%, RM 4th order at 80.50%, RM 5th order at 85.06%, RM 6th order at 55.94%, RM 7th order at 76.25%, RM 8th order at 61.62%, RM 9th order at 48.50%, and RM 10th order at 52.25%. From these results, it is evident that the 5th order RM model exhibits the highest performance, followed closely by linear regression. Given the similar computational speed, it can be concluded that for this particular custom dataset, the 5th order RM model represents the most optimal choice.

The data I have generated, which demonstrates relatively high accuracy, appears to be easy to distinguish between classes. However, it can be inferred that applying RM regression to more intertwined datasets could potentially yield better performance by effectively separating features. Currently, the advent of various deep learning models certainly promises improved outcomes compared to this method. Nevertheless, it is evident that these advancements come at the cost of significantly increased computational resources. It can be conjectured that for complex datasets characterized by a high degree of entangled features and which do not follow a Gaussian distribution, the application of this model might prove challenging. However, Linear regression and RM regression, with their rapid computational speed, provide reasonable results for less complex data, underscoring their considerable value for specific datasets.

## VI. Appendix

```
import numpy as np
import matplotlib.pyplot as plt
# Define means and covariances
mu1 = np.array([2, 2])
sigma1 = np.array([[5, 0], [0, 1]])
mu2 = np.array([2, 5])
sigma2 = np.array([[1, 0], [0, 3]])
# Set random seed for reproducibility
np.random.seed(88)
# Generate data
r1 = np.random.multivariate_normal(mu1, sigma1, size=1000)
r2 = np.random.multivariate_normal(mu2, sigma2, size=1000)
X = np.concatenate((r1, r2), axis=0)
# Separate data for training and testing (mimicking the approach)
X_train = np.concatenate((X[:200], X[1000:1200]))
X_test = np.concatenate((X[200:1000], X[1201:2000]))
y_train = np.concatenate((-np.ones(200), np.ones(200)))
y_test = np.concatenate((-np.ones(800), np.ones(800)))
# Create the scatter plot
plt.figure(1)
plt.subplot(1, 2, 1)
plt.scatter(X[:, 0], X[:, 1], s=10, alpha=0.7) # Scatter plot
plt.plot(X_train[:, 0], X_train[:, 1], 'b+', label='Training Set - Class 1')
plt.plot(X_train[200:, 0], X_train[200:, 1], 'ro', label='Training Set - Class 2')
plt.legend()
plt.subplot(1, 2, 2)
plt.scatter(X_test[:, 0], X_test[:, 1], s=10, alpha=0.7) # Scatter plot
plt.plot(X_test[:, 0], X_test[:, 1], 'b+', label='Test Set - Class 1')
plt.plot(X_test[800:, 0], X_test[800:, 1], 'ro', label='Test Set - Class 2')
plt.legend()
plt.show()


import numpy as np
import matplotlib.pyplot as plt
np.random.seed(77)
# Parameters for the Gaussian distribution
# Mean and covariance for the training set
```

```
mean_data1 = [0, 0]
cov_data1 = [[3, 0], [0, 5]]  # Diagonal covariance, for independence

# Mean and covariance for the test set
mean_data2 = [2, 2]
cov_data2 = [[1, 0], [0, 1]]  # Diagonal covariance, for independence
# Generating samples
data1= np.random.multivariate_normal(mean_data1, cov_data1, 1000)
data2 = np.random.multivariate_normal(mean_data2, cov_data2, 1000)
classm1 = -np.ones((1000, 1))  # Class -1
class1 = np.ones((1000, 1))  # Class 1
# Separate data for training and testing (mimicking the approach)
X_train = np.concatenate((data1[:200], data2[:200]))
X_test = np.concatenate((data1[200:], data2[200:]))
y_train = np.concatenate((classm1[:200], class1[:200]))
y_test = np.concatenate((classm1[200:], class1[200:]))
# Plotting
plt.figure(figsize=(10, 5))

# Training data
plt.subplot(1, 2, 1)
plt.scatter(X_train[:200, 0], X_train[:200, 1], c='blue', label='Class -1', alpha=0.5)
plt.scatter(X_train[200:, 0], X_train[200:, 1], c='red', label='Class 1', alpha=0.5)
plt.title('Training Data')
plt.xlabel('X1')
plt.ylabel('X2')
plt.legend()

# Testing data
plt.subplot(1, 2, 2)
plt.scatter(X_test[:800, 0], X_test[:800, 1], c='blue', label='Class -1', alpha=0.5)
plt.scatter(X_test[800:, 0], X_test[800:, 1], c='red', label='Class 1', alpha=0.5)
plt.title('Testing Data')
plt.xlabel('X1')
plt.ylabel('X2')
plt.legend()

plt.tight_layout()
plt.show()

def LR(X,y):
    X_bias = np.c_[np.ones((X.shape[0], 1)), X]
    return  np.linalg.inv(X_bias.T.dot(X_bias)).dot(X_bias.T).dot(y)

# loss function
def RSS(y_pred,y):
    return np.sum((y - y_pred) ** 2)

w = LR(X_train,y_train)
X_test_bias = np.c_[np.ones((X_test.shape[0], 1)), X_test]
y_test_pred = X_test_bias @ w
ktest = RSS(y_test_pred,y_test)
print(ktest)

X_train_bias = np.c_[np.ones((X_train.shape[0], 1)), X_train]
y_pred = X_train_bias @ w
ktrain = RSS(y_pred,y_train)
print(ktrain)

# draw decision boundary
x_values = np.array([np.min(X_train[:, 0]), np.max(X_train[:, 0])])
y_values = -(w[1] / w[2]) * x_values - (w[0] / w[2])
plt.figure(figsize=(10, 5))

# Training data
```

```
plt.subplot(1, 2, 1)
plt.scatter(X_train[:200, 0], X_train[:200, 1], c='blue', label='Class -1', alpha=0.5)
plt.scatter(X_train[200:, 0], X_train[200:, 1], c='red', label='Class 1', alpha=0.5)
plt.plot(x_values, y_values, label="Decision Boundary")
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Decision Boundary in Train Dataset')
plt.legend()

# Testing data
plt.subplot(1, 2, 2)
plt.scatter(X_test[:800, 0], X_test[:800, 1], c='blue', label='Class -1', alpha=0.5)
plt.scatter(X_test[800:, 0], X_test[800:, 1], c='red', label='Class 1', alpha=0.5)
plt.plot(x_values, y_values, label="Decision Boundary")
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Decision Boundary in Test Dataset')
plt.legend()


plt.show()

def RM(X, order):
    # Build regressor matrix P (mxK):
    # order = desired order of approximation,
    # X = input matrix (mxl), K = number of parameters to be est.
    # m = number of data samples, l = input dimension.
    m, l = X.shape
    MM1 = []
    MM3 = []
    Msum = np.sum(X, axis=1)
    for i in range(1, order+1):
        M1 = np.zeros((m, l))
        M3 = np.zeros((m, l))
        for k in range(l):
            M1[:, k] = X[:, k]**i
            if i > 1:
                M3[:, k] = X[:, k] * Msum**(i-1)
        MM1.append(M1)
        if i > 1:
            MM3.append(M3)
    if MM3:
        P = np.concatenate([np.ones((m, 1)), np.concatenate(MM1, axis=1), np.concatenate(MM3, axis=1)], axis=1)
    else:
        P = np.concatenate([np.ones((m, 1)), np.concatenate(MM1, axis=1)], axis=1)
    return P

X = np.array([[1, 2], [3, 4], [5, 6]])
order = 2
P = RM(X, order)
print(P)

P_train = {}
for i in range(1,11):
    P_train[f'P_train{i}'] = RM(X_train,i)

def PLR(P,y):
    return np.linalg.inv(P.T @ P) @ P.T @ y

LR_weights = {}
for i in range(1,11):
    train = P_train[f'P_train{i}']
    LR_weights[f'LR_train{i}'] = PLR(train,y_train)
    print(f'{i}th order RM regression weight shape is ',LR_weights[f'LR_train{i}'].shape)
```

```
predicts = {}
P_test = {}
for i in range(1,11):
    P_test[f'P_test{i}'] = RM(X_test,i)
    test = P_test[f'P_test{i}']
    weight = LR_weights[f'LR_train{i}']
    predicts[f'order{i}_pred'] = test @ weight

# i=0
# for k,v in predicts.items():
#     i+=1
#     print(f"{i}th order RM model")
#     print(f"{v[:10]}")

# # draw decision boundary
# w = LR_weights['LR_train5']
# x_values = np.array([np.min(X_train[:, 0]), np.max(X_train[:, 0])])
# y_values = -(w[1] / w[2]) * x_values - (w[0] / w[2])
# plt.figure(figsize=(10, 5))

# # Training data
# plt.subplot(1, 2, 1)
# plt.scatter(X_train[:200, 0], X_train[:200, 1], c='blue', label='Class -1', alpha=0.5)
# plt.scatter(X_train[200:, 0], X_train[200:, 1], c='red', label='Class 1', alpha=0.5)
# plt.plot(x_values, y_values, label="Decision Boundary")
# plt.xlabel('X1')
# plt.ylabel('X2')
# plt.title('Decision Boundary in Train Dataset')
# plt.legend()

# # Testing data
# plt.subplot(1, 2, 2)
# plt.scatter(X_test[:800, 0], X_test[:800, 1], c='blue', label='Class -1', alpha=0.5)
# plt.scatter(X_test[800:, 0], X_test[800:, 1], c='red', label='Class 1', alpha=0.5)
# plt.plot(x_values, y_values, label="Decision Boundary")
# plt.xlabel('X1')
# plt.ylabel('X2')
# plt.title('Decision Boundary in Test Dataset')
# plt.legend()


# plt.show()
# plt.figure(figsize=(15, 10))

# # Loop through orders 1 to 6
# for i in range(1, 7):
#     w = LR_weights[f'LR_train{i}']
#     x_values = np.array([np.min(X_train[:, 0]), np.max(X_train[:, 0])])
#     y_values = -(w[1] / w[2]) * x_values - (w[0] / w[2])

#     # Training data plot
#     plt.subplot(2, 3, i)
#     plt.scatter(X_train[:200, 0], X_train[:200, 1], c='blue', label='Class -1', alpha=0.5)
#     plt.scatter(X_train[200:, 0], X_train[200:, 1], c='red', label='Class 1', alpha=0.5)
#     plt.plot(x_values, y_values, label=f"Decision Boundary (Order {i})")
#     plt.xlabel('X1')
#     plt.ylabel('X2')
#     plt.title(f'Decision Boundary for Order {i}')
#     plt.legend()

# plt.tight_layout()
# plt.show()
# plt.figure(figsize=(15, 10))

# # Loop through orders 1 to 6
```

```
# for i in range(1, 7):
#     w = LR_weights[f'LR_train{i}']
#     x_values = np.array([np.min(X_test[:, 0]), np.max(X_test[:, 0])])
#     y_values = -(w[1] / w[2]) * x_values - (w[0] / w[2])

#     # testing data plot
#     plt.subplot(2, 3, i)
#     plt.scatter(X_test[:800, 0], X_test[:800, 1], c='blue', label='Class -1', alpha=0.5)
#     plt.scatter(X_test[800:, 0], X_test[800:, 1], c='red', label='Class 1', alpha=0.5)
#     plt.plot(x_values, y_values, label=f"Decision Boundary (Order {i})")
#     plt.xlabel('X1')
#     plt.ylabel('X2')
#     plt.title(f'Decision Boundary for Order {i}')
#     plt.legend()

# plt.tight_layout()
# plt.show()

predicts = {}
P_test = {}
for i in range(1,7):
    P_test[f'P_test{i}'] = RM(X_test,i)
    test = P_test[f'P_test{i}']
    weight = LR_weights[f'LR_train{i}']
    predicts[f'order{i}_pred'] = test @ weight

def classacc(ytest,ypred):
    y_pred_class = np.where(ypred >= 0, 1, -1)
    correct_classifications = y_test == y_pred_class
    accuracy = np.mean(correct_classifications)
    return accuracy
i=0
for v in predicts.values():
    acc = classacc(y_test,v)
    i+=1
    print(f'RM {i}th order RM Model acc is ', f'{acc*100:.2f}','%')
def LR(X,y):
    X_bias = np.c_[np.ones((X.shape[0], 1)), X]
    return  np.linalg.inv(X_bias.T.dot(X_bias)).dot(X_bias.T).dot(y)
w = LR(X_train,y_train)
X_test_bias = np.c_[np.ones((X_test.shape[0], 1)), X_test]
y_test_pred = X_test_bias @ w

acc = classacc(y_test,y_test_pred)
print(f'LR Model acc is ', f'{acc*100:.2f}','%')
# Data for RM model orders and their accuracies
orders = list(range(1, 11))
accuracies = [0.84, 0.73, 0.82, 0.81, 0.85, 0.56, 0.76, 0.62, 0.48, 0.52]

# Plotting the accuracies
plt.figure(figsize=[10,6])
plt.plot(orders, accuracies, marker='o', linestyle='-', color='b')
plt.title('Test Classification Accuracies of the RM Model')
plt.xlabel('Order of RM Model')
plt.ylabel('Accuracy (%)')
plt.xticks(orders)
plt.grid(True)

for i, txt in enumerate(accuracies):
    plt.text(orders[i], accuracies[i], f"{txt*100:.1f}%", ha='center', va='bottom')
plt.show()
```