CS5691: Pattern Recognition and Machine Learning
Assignment #2

Topics: LDA, GMM, DBSCAN **Deadline:** 28 April 2023, 11:55 PM

Teammate 1: Siddhagavali Shital Bhiku (50 % of contribution) **Roll number:** ME20B166
Teammate 2: Srinivas Chowdary Ramineni (50 % of contribution) **Roll number:** ME20B174

- **For any doubts regarding questions 1 and 2**, you can mail cs22s013@smail.iitm.ac.in and cs21s043@smail.iitm.ac.in
- **For any doubts regarding question 3**, you can mail cs21d015@smail.iitm.ac.in and cs22s015@smail.iitm.ac.in
- Please refer to the **Additional Resources** tab on the Course webpage for basic programming instructions.
- This assignment has to be completed in teams of 2. Collaborations outside the team are strictly prohibited.
- Any kind of plagiarism will be dealt with severely. These include copying text or code from any online sources. These will lead to disciplinary actions according to institute guidelines. Acknowledge any and every resource used.
- Be precise with your explanations. Unnecessary verbosity will be penalized.
- Check the Moodle discussion forums regularly for updates regarding the assignment.
- You should submit a zip file titled **'rollnumber1_rollnumber2.zip'** on Moodle where rollnumber1 and rollnumber2 are your institute roll numbers. Your assignment will **NOT** be graded if it does not contain all of the following:
    1. Type your solutions in the provided LaTeX template file and title this file as **'Report.pdf'**. **State your respective contributions in terms of percentage at the beginning of the report clearly.** Also, embed the result figures in your LaTeX solutions.
    2. Clearly name your source code for all the programs in **individual Google Colab files**. Please submit your code only as Google Colab file (.ipynb format). Also, embed the result figures in your Colab code files.
- We highly recommend using `Python 3.6+` and standard libraries like `NumPy, Matplotlib, Pandas, Seaborn`. Please use `Python 3.6+` as the only standard programming language to code your assignments. Please note: the TAs will only be able to assist you with doubts related to Python.
- You are expected to code all algorithms from scratch. **You cannot use standard inbuilt libraries for algorithms until and unless asked explicitly**.
- **Any graph that you plot is unacceptable for grading unless it labels the x-axis and y-axis clearly.**

- Please note that the TAs will **only** clarify doubts regarding problem statements. The TAs won't discuss any prospective solution or verify your solution or give hints.
- Please refer to the CS5691 PRML course handout for the late penalty instruction guidelines.

---

1. **[Linear Discriminant Analysis (LDA), Principal Component Analysis (PCA)]** You will implement dimensionality reduction techniques (LDA, PCA) as part of this question for the dataset1 provided here.

   Note that you have to implement **LDA from scratch** without using any predefined libraries (i.e. sklearn, scipy) . However, you can use **predefined libraries to implement PCA.**

   (a) (2 marks) Use Linear Discriminant analysis (LDA) to convert dataset1 into the two-dimensional dataset and then visualize the obtained dataset. Also, perform an analysis on how results will change if we perform normalization (i.e., zero mean, unit variance normalization) on the initial dataset before applying LDA.

Solution:

Part (a):

Listing 1: Linear Discriminant analysis (LDA) to convert dataset1 into the two-dimensional dataset and then visualize the obtained database and normalization dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from numpy.linalg import eig


# Load the dataset
df = pd.read_csv('dataset_1.csv', sep='\t')

# Drop rows with missing values
df.dropna(inplace=True)

# Extract features and target
X = df.iloc[:, :64].values
y = df.iloc[:, -1].values

# Perform normalization analysis on original data
```

```
X_means = np.mean(X, axis=0)
X_stds = np.std(X, axis=0)
X_min = np.min(X, axis=0)
X_max = np.max(X, axis=0)

# Print results
print("Original Data Mean: ", X_means)
print("Original Data Standard Deviation: ", X_stds)
print("Original Data Minimum: ", X_min)
print("Original Data Maximum: ", X_max)

# Perform LDA

def lda(X, y, n_components=2, reg_param=0.0):
    """
    Linear Discriminant Analysis (LDA)
    Args:
        X (array-like): Input data matrix of shape (n_samples
            , n_features)
        y (array-like): Target vector of shape (n_samples,)
        n_components (int): Number of components to keep (
            default=2)
        reg_param (float): Regularization parameter (default
            =0.0)
    Returns:
        X_lda (array-like): LDA-transformed data of shape (
            n_samples, n_components)
    """
    classes = np.unique(y)
    n_classes = len(classes)
    n_features = X.shape[1]
    X_mean_class = np.zeros((n_classes, n_features))
    for i, c in enumerate(classes):
        X_mean_class[i] = np.mean(X[y == c], axis=0)
    X_mean_total = np.mean(X, axis=0)
    Sb = np.zeros((n_features, n_features))
    Sw = np.zeros((n_features, n_features))
    for i, c in enumerate(classes):
        class_count = X[y == c].shape[0]
        X_mean_diff = (X_mean_class[i] - X_mean_total).
            reshape(-1, 1)
        Sb += class_count * np.dot(X_mean_diff, X_mean_diff.T
            )
        X_c = X[y == c] - X_mean_class[i]
```

```
            Sw += np.dot(X_c.T, X_c)
        Sw += reg_param * np.eye(n_features)
        eigenvalues, eigenvectors = eig(np.dot(np.linalg.inv(Sw),
            Sb))
        eig_pairs = [(np.abs(eigenvalues[i]), eigenvectors[:, i])
            for i in range(n_features)]
        eig_pairs.sort(key=lambda x: x[0], reverse=True)
        W = np.hstack([eig_pairs[i][1].reshape(-1, 1) for i in
            range(n_components)])
        X_lda = np.dot(X, W)
        return X_lda

# Perform LDA with regularization
X_lda = lda(X, y, n_components=2, reg_param=1e-4)

# Convert LDA data to real numbers
X_lda_real = np.real(X_lda)

# Visualization with seaborn
sns.scatterplot(x=X_lda[:, 0], y=X_lda[:, 1], hue=y)
plt.xlabel('LDA Component 1')
plt.ylabel('LDA Component 2')
plt.title('Original dataset')
plt.show()

# Add small epsilon value to prevent division by zero
X_normalized = (X - np.mean(X, axis=0)) / (np.std(X, axis=0)
    + 1e-8)

# Replace any resulting NaN values with zeros
X_normalized[np.isnan(X_normalized)] = 0

# Perform normalization analysis on original data
X_norm_means = np.mean(X_normalized, axis=0)
X_norm_stds = np.std(X_normalized, axis=0)
X_norm_min = np.min(X_normalized, axis=0)
X_norm_max = np.max(X_normalized, axis=0)

# Print results
print("Normalized Data Mean: ", X_norm_means)
print("Normalized Data Standard Deviation: ", X_norm_stds)
print("Normalized Data Minimum: ", X_norm_min)
print("Normalized Data Maximum: ", X_norm_max)
```

```
# Perform normalization analysis on LDA data
lda_normalized_means = np.mean(X_lda_real, axis=0)
lda_normalized_stds = np.std(X_lda_real, axis=0)
lda_normalized_min = np.min(X_lda_real, axis=0)
lda_normalized_max = np.max(X_lda_real, axis=0)

# Print results
print("Normalized LD1 Mean: ", lda_normalized_means[0])
print("Normalized LD2 Mean: ", lda_normalized_means[1])
print("Normalized LD1 Standard Deviation: ",
    lda_normalized_stds[0])
print("Normalized LD2 Standard Deviation: ",
    lda_normalized_stds[1])
print("Normalized LD1 Minimum: ", lda_normalized_min[0])
print("Normalized LD2 Minimum: ", lda_normalized_min[1])
print("Normalized LD1 Maximum: ", lda_normalized_max[0])
print("Normalized LD2 Maximum: ", lda_normalized_max[1])

# Perform LDA with regularization

## Add small epsilon value to prevent division by zero
# X_normalized = (X - np.mean(X, axis=0)) / (np.std(X, axis
    =0) + 1e-8)

## Replace any resulting NaN values with zeros
# X_normalized[np.isnan(X_normalized)] = 0
X_lda_real = lda(X_normalized, y, n_components=2, reg_param=1
    e-4)

# Convert LDA data to real numbers
X_lda_real = lda(X_normalized, y, n_components=2, reg_param=1
    e-4)
X_lda_real = np.real(X_lda_real)

# Plot the LDA data
sns.scatterplot(x=X_lda_real[:, 0], y=X_lda_real[:, 1], hue=y
    , palette='Set1')
plt.xlabel('LDA Component 1 (Normalized)')
plt.ylabel('LDA Component 2 (Normalized)')
plt.title("Normalized dataset")
plt.legend(title='Target', loc='best')
plt.show()
```

Output:

Statistical analaysis of original data

```
Original Data Mean:  [0.00000000e+00 3.19029851e-01
   5.40671642e+00 1.21361940e+01
 1.04477612e+01 3.82089552e+00 3.84328358e-01 0.00000000e+00
 3.73134328e-03 2.06902985e+00 1.00932836e+01 1.27444030e+01
 1.26809701e+01 8.43470149e+00 8.58208955e-01 0.00000000e+00
 5.59701493e-03 3.10634328e+00 9.69962687e+00 8.28917910e+00
 9.33768657e+00 8.48880597e+00 1.54664179e+00 0.00000000e+00
 1.86567164e-03 2.75559701e+00 7.85447761e+00 7.07835821e+00
 8.74813433e+00 6.58208955e+00 2.33395522e+00 0.00000000e+00
 0.00000000e+00 2.35261194e+00 6.54664179e+00 7.22014925e+00
 8.12873134e+00 5.53171642e+00 2.49440299e+00 0.00000000e+00
 0.00000000e+00 1.49067164e+00 7.91791045e+00 7.95708955e+00
 7.46082090e+00 6.40858209e+00 2.36753731e+00 5.59701493e-03
 5.59701493e-03 7.68656716e-01 9.61380597e+00 1.17294776e+01
 1.20000000e+01 1.05037313e+01 3.86940299e+00 4.53358209e-01
 1.86567164e-03 3.15298507e-01 5.46455224e+00 1.21940299e+01
 1.31623134e+01 8.61380597e+00 3.77611940e+00 1.14552239e+00]
Original Data Standard Deviation:  [0.         0.90029628
   4.8463973  4.61443586 4.51046406 4.71873172
 1.3472113  0.         0.08630622 3.40336913 5.82892553
   3.91132665
 4.21804181 5.81905557 2.07922555 0.         0.09642112
   3.72490069
 5.64209921 6.21747916 6.48158683 5.7801663  2.54101943 0.
 0.04315311 3.29262609 6.12930465 6.65805384 6.84898376
   5.32846273
 3.32427245 0.         0.         3.08413016 5.94483878
   6.6655192
 6.69011823 5.28752771 3.67067423 0.         0.
   2.2485291
 6.23848014 6.59007806 6.44250942 5.84557501 3.51292792
   0.07460354
 0.12945933 1.77740453 5.58333371 4.82603601 4.05742364
   5.35040669
 4.57092903 1.48803308 0.04315311 1.01822438 5.02184157
   4.64521518
 3.04777835 5.55217079 5.59369553 3.23306416]
Original Data Minimum:  [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. 0.
   0. 0. 0. 0.
```

```
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Original Data Maximum: [ 0. 5. 16. 16. 16. 16. 11. 0. 2.
   15. 16. 16. 16. 16. 12. 0. 2. 16.
 16. 16. 16. 16. 14. 0. 1. 14. 16. 16. 16. 16. 12. 0. 0.
    11. 16. 16.
 16. 16. 14. 0. 0. 9. 16. 16. 16. 16. 16. 1. 3. 16. 16.
    16. 16. 16.
 16. 13. 1. 8. 16. 16. 16. 16. 16. 16.]
```

Statistical analaysis of Normalized data before applying LDA.

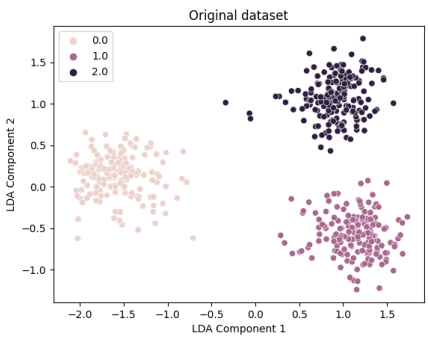
```
Normalized Data Mean: [ 0.00000000e+00 3.97691830e-17
   -1.19307549e-16 1.19307549e-16
 -1.52448535e-16 -5.30255773e-17 3.31409858e-17 0.00000000e
    +00
  3.31409858e-18 6.62819716e-17 -1.55762633e-16 -5.96537745e
    -17
 -3.97691830e-17 1.39192140e-16 -6.62819716e-17 0.00000000e
    +00
 -3.31409858e-18 -2.98268872e-17 9.44518096e-17 6.62819716e
    -17
  5.30255773e-17 -1.29249845e-16 9.27947603e-17 0.00000000e
    +00
 -3.31409858e-18 0.00000000e+00 -1.32563943e-17 8.61665631e
    -17
  1.39192140e-16 1.06051155e-16 -6.62819716e-17 0.00000000e
    +00
  0.00000000e+00 -1.98845915e-17 -5.30255773e-17 -3.97691830e
    -17
  1.20964598e-16 -3.64550844e-17 3.81121337e-17 0.00000000e
    +00
  0.00000000e+00 -3.97691830e-17 -6.62819716e-17 7.62242674e
    -17
  0.00000000e+00 7.29101688e-17 2.31986901e-17 6.62819716e
    -18
 -9.94229574e-18 3.97691830e-17 -1.01080007e-16 -6.62819716e
    -18
  1.98845915e-17 1.72333126e-16 -1.32563943e-17 9.94229574e
    -18
  0.00000000e+00 0.00000000e+00 2.65127886e-17 1.98845915e
    -17
 -2.18730506e-16 -1.25935746e-16 -6.62819716e-18 1.32563943e
    -17]
Normalized Data Standard Deviation: [0. 0.99999999
   1. 1. 1. 1.
```

```
 0.99999999  0.          0.99999988  1.          1.          1.
 1.          1.          1.          0.          0.9999999   1.
 1.          1.          1.          1.          1.          0.
 0.99999977  1.          1.          1.          1.          1.
 1.          0.          0.          1.          1.          1.
 1.          1.          1.          0.          0.          1.
 1.          1.          1.          1.          1.
    0.99999987
 0.99999992  0.99999999  1.          1.          1.          1.
 1.          0.99999999  0.99999977  0.99999999  1.          1.
 1.          1.          1.          1.         ]
Normalized Data Minimum:  [ 0.         -0.35436095
   -1.11561559 -2.63004934 -2.31633842 -0.80972934
 -0.28527697  0.         -0.04323377 -0.60793577 -1.73158561
    -3.25833256
 -3.00636426 -1.44949664 -0.41275414  0.         -0.0580476
    -0.83393989
 -1.71915213 -1.33320577 -1.44064822 -1.4686093  -0.6086698
      0.
 -0.04323376 -0.83689946 -1.28146308 -1.06312721 -1.27728939
    -1.23526988
 -0.70209504  0.          0.         -0.76281214 -1.10123117
    -1.08320883
 -1.21503553 -1.04618202 -0.67954899  0.          0.
    -0.66295412
 -1.26920504 -1.20743479 -1.15806131 -1.09631338 -0.67394987
    -0.07502344
 -0.04323377 -0.4324602  -1.72187558 -2.43045795 -2.95754179
    -1.96316503
 -0.84652441 -0.30466944 -0.04323376 -0.30965523 -1.08815703
    -2.62507319
 -4.31865834 -1.55143029 -0.67506702 -0.35431477]
Normalized Data Maximum:  [ 0.          5.19936627
   2.18580585  0.83733008  1.23096841  2.58101227
  7.87973763  0.         23.13006433  3.79946154  1.01334566
     0.83235109
  0.78686509  1.30009044  5.35862547  0.         20.68429467
     3.46147663
  1.11667181  1.24018444  1.02788308  1.29947715  4.90092991
     0.
 23.13006165  3.41502577  1.32894722  1.33997742  1.0588236
     1.76747233
  2.90771737  0.          0.          2.80383368  1.59017907
     1.31720433
```

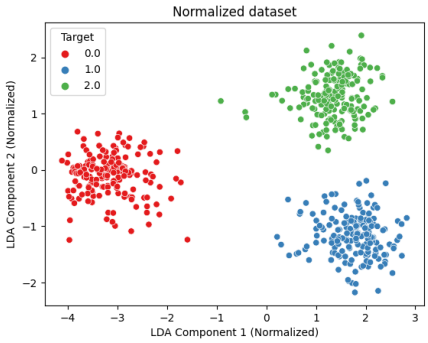
```
  1.1765515    1.97980685   3.13446421   0.           0.
               3.3396625
  1.2955222    1.22045754   1.32544301   1.64079973   3.88065538
    13.32916423
 23.13006523   8.56942973   1.14379587   0.88489236   0.98584726
    1.02726184
  2.65385809   8.43169541  23.13006165   7.54715915   2.09792515
     0.81933129
  0.93106723   1.33032544   2.18529602   4.59455082]
```

Statistical analaysis of Normalized data after applying LDA.

```
Normalized LD1 Mean:   0.16864713887644622
Normalized LD2 Mean:   0.20144017455807836
Normalized LD1 Standard Deviation:   1.2610842718474609
Normalized LD2 Standard Deviation:   0.7179690242869715
Normalized LD1 Minimum:   -2.1076820231305455
Normalized LD2 Minimum:   -1.2425747048458362
Normalized LD1 Maximum:   1.730941127166309
Normalized LD2 Maximum:   1.785067239840377
```



(a) Original data after applying LDA.



(b) Normalized data after applying LDA.

Figure 1: Visualization data after applying LDA.

Listing 2: Visualization of Statistical analaysis of orginal and normalized data

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

def plotHistogrmas(data, norm_data):
  fig, axes = plt.subplots(ncols=2, figsize=(12, 6))
```

```
    # Original data histogram
    sns.histplot(X_means, ax=axes[0], kde=True, color='skyblue'
        )
    axes[0].set_title('Original Data')

    # Normalized data histogram
    sns.histplot(X_norm_means, ax=axes[1], kde=True, color='
        orange')
    axes[1].set_title('Normalized Data')
    plt.show()

print("Plot of mean of data")
plotHistogrmas(X_means, X_norm_means)

print("Plot of std of data")
plotHistogrmas(X_stds, X_norm_stds)

print("Plot of min of data")
plotHistogrmas(X_min, X_norm_min)

print("Plot of max of data")
plotHistogrmas(X_max, X_norm_max)
```

Visualization of original data and Normalized data before applying LDA.

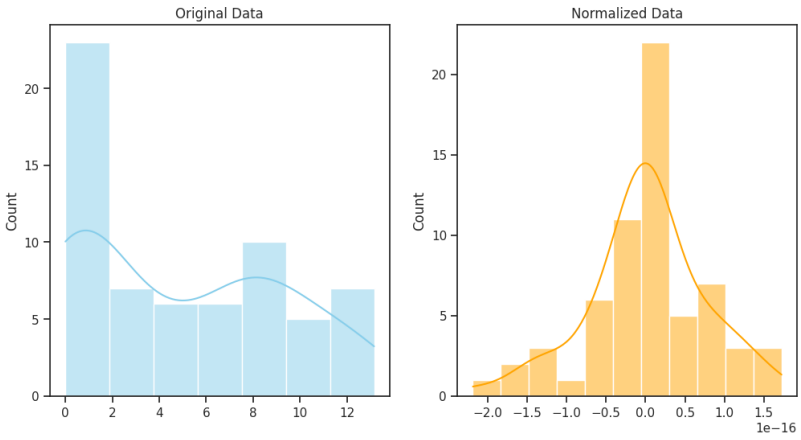


Figure 2: Visualization of mean of data before applying LDA.

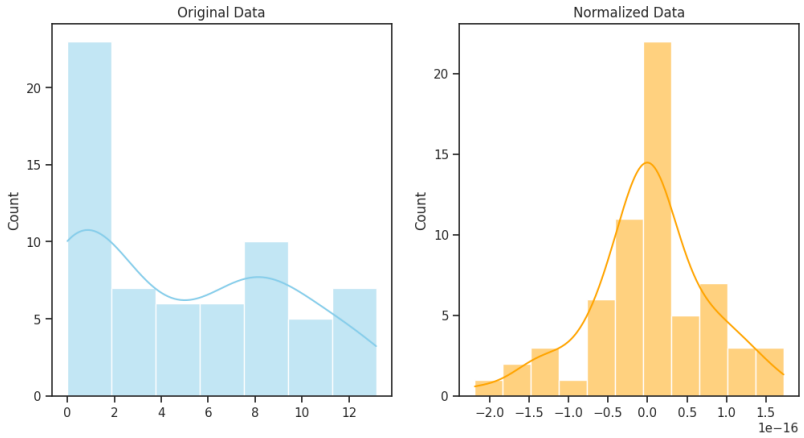


Figure 3: Visualization of std of data before applying LDA.

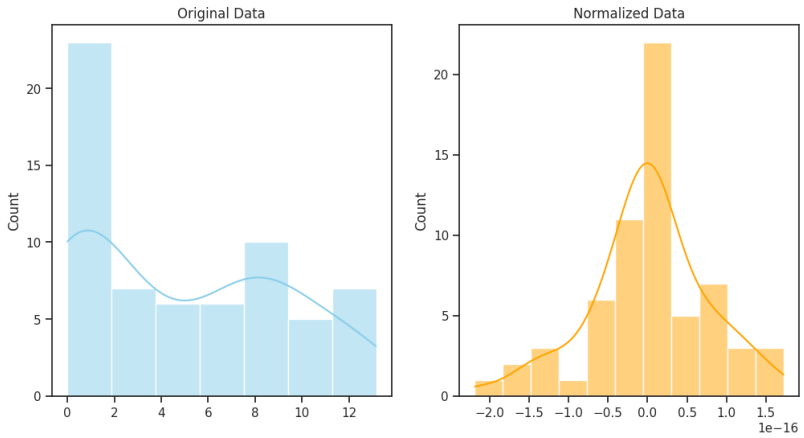


Figure 4: Visualization of min of data before applying LDA.

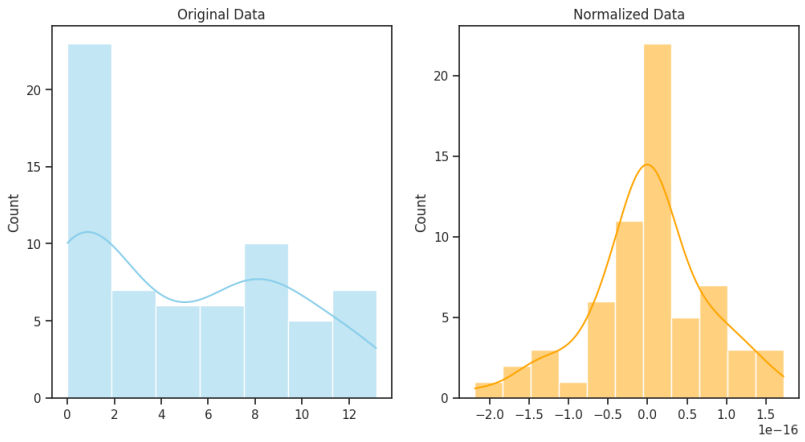


Figure 5: Visualization of max of data before applying LDA.

(b) (1.5 marks) Use PCA to convert dataset1 into two-dimensional data and then visualize the obtained dataset. Now, compare and contrast the visualizations of the final datasets obtained using LDA and PCA.

Solution:

Part (b):

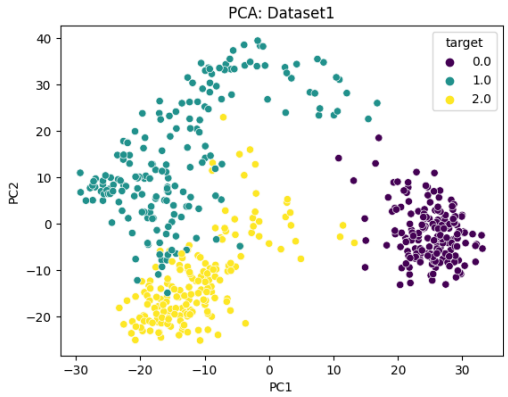
Listing 3: Using PCA to convert dataset1 into two-dimensional data and then visualize the obtained dataset.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

# Load dataset1 without headers
df = pd.read_csv('./drive/MyDrive/cs5691_A2_DB1/dataset_1.csv
    ', sep='\t', header=None)

# Extract features and target
X = df.iloc[:, :-1].values  # Exclude the last column as the
    target
y = df.iloc[:, -1].values

# Perform PCA
```

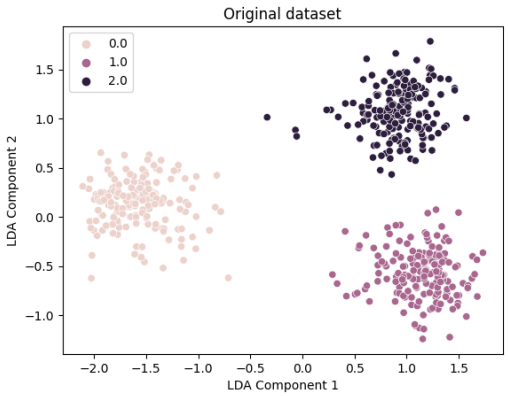
```
pca = PCA(n_components=2)
pca_data = pca.fit_transform(X)
pca_df = pd.DataFrame(data=pca_data, columns=['PC1', 'PC2'])
pca_df['target'] = y

# Visualize PCA data
sns.scatterplot(x='PC1', y='PC2', hue='target', data=pca_df,
    palette='viridis')
plt.title('PCA: Dataset1')
plt.show()
```

Output:

Visualization of data after applying PCA and LDA.



(a) Visualization of data after applying PCA.



(b) Visualization of data after applying LDA.

Figure 6: Comparing plots of data obtained by PCA and LDA.

- Visualizations obtained using LDA and PCA have distinct characteristics. The LDA plot displays well-separated clusters of data points. Conversely, the PCA plot shows a distribution of data points in a lower-dimensional space with overlapping regions.

- Despite their differences, both LDA and PCA visualizations reduce the dimensionality of the data and capture its main patterns or structures. However, the LDA plot provides a more effective separation of data points than the PCA plot, suggesting LDA's potential for capturing discriminative features of the data relevant for class separation. In contrast, PCA focuses on capturing overall variability in the data, which may result in overlapping data points.

- The differences in the visualizations can be attributed to the underlying assumptions and techniques of LDA and PCA. LDA is a supervised technique that considers class labels during data point projection, while PCA is an unsupervised technique that captures overall variability in the data without considering class labels.
- In part (a) as you can see we have did visualization by plotting histograms of statistic terms like mean, std etc.. We have plotted histogram for each term and we observed that original data is distributed randomly and normalized data is following normal distribution.

(c) (1.5 marks) Randomly shuffle and split the obtained dataset from part (a) into a training set (80%) and testing set (20%). Now build the Bayes classifier using the training set and report the following:

- Accuracy on both train and test data.
- Plot of the test data along with your classification boundary.
- confusion matrices on both train and test data.

Solution:

Part (c):

Listing 4: Applying Bayes classifier on obtained data in Part (a)

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

import numpy as np

def bayes_classifier(x_train, y_train, x_test):
    # Calculate class probabilities
    class_probabilities = {}
    for class_label in np.unique(y_train):
        class_probabilities[class_label] = np.sum(y_train ==
            class_label) / len(y_train)

    # Calculate mean and standard deviation for each feature
        and each class
```

```
    means = {}
    stds = {}
    for class_label in np.unique(y_train):
        class_data = x_train[y_train == class_label]
        means[class_label] = np.mean(class_data, axis=0)
        stds[class_label] = np.std(class_data, axis=0)

    # Classify test data
    y_pred = []
    for i in range(len(x_test)):
        # Calculate probability for each class
        probabilities = {}
        for class_label in np.unique(y_train):
            p = class_probabilities[class_label]
            for j in range(len(x_test[i])):
                p *= 1 / (np.sqrt(2 * np.pi) * stds[
                    class_label][j]) * np.exp(-0.5 * ((x_test[
                    i][j] - means[class_label][j]) / stds[
                    class_label][j]) ** 2)
            probabilities[class_label] = p

        # Select class with highest probability
        y_pred.append(max(probabilities, key=probabilities.
            get))

    return y_pred

def accuracy(y_true, y_pred):
    return np.mean(y_true == y_pred)

def plot_confusion_matrix(y_true, y_pred, c):
    unique_labels = np.unique(np.concatenate((y_true, y_pred)
        ))
    n_labels = len(unique_labels)
    confusion_mat = np.zeros((n_labels, n_labels))
    for i in range(n_labels):
        for j in range(n_labels):
            confusion_mat[i, j] = np.sum((y_true ==
                unique_labels[i]) & (y_pred == unique_labels[j
                ]))
    print(confusion_mat)
    total = np.sum(confusion_mat)
    percentages = (confusion_mat / total) * 100
    new_column=[]
```

```
for i in range(len(confusion_mat)):
  new_column.append(100*confusion_mat[i][i]/np.sum(
      confusion_mat[i]))
print(new_column)
new_row=[]
confusion_mat=confusion_mat.T
for i in range(len(confusion_mat)):
  new_row.append(100*confusion_mat[i][i]/np.sum(
      confusion_mat[i]))
l=np.sum(new_row)/(len(new_row))
new_row.append(np.sum(new_row)/(len(new_row)))
confusion_mat=confusion_mat.T
print(new_row)
percentage=np.hstack((confusion_mat,np.array([new_column
    ]).T))
percentage=np.vstack((percentage,np.array([new_row])))
confusion_mat=percentage

#Create heatmap with annotations
fig,ax = plt.subplots(figsize=(8,8))  #Set figsize to
    increase the size of cells
im = ax.imshow(confusion_mat,cmap='Dark2')

#Loop over data dimensions and create text
for i in range(len(confusion_mat)):
    for j in range(len(confusion_mat[0])):
        #Add confusion matrix text
        if i!=len(confusion_mat)-1 and j!=len(
            confusion_mat)-1:
          ax.text(j,i+0.15,f'{percentages[i,j]:.2f}%',ha=
              "center",va="center",color="black",fontsize
              =9)
          ax.text(j,i,f'{confusion_mat[i,j]:}',ha="center
              ",va="center",color="black")
        else:
          ax.text(j,i,f'{confusion_mat[i,j]:.1f}'+'%',ha=
              "center",va="center",color="black")
          ax.text(j,i+0.15,f'{100-confusion_mat[i,j]:.1f}
              '+'%', ha="center",va="center",color="black"
              ,fontsize=9)
#Set ticks and axis labels
ax.set_xticks(np.arange(len(confusion_mat)-1))
ax.set_yticks(np.arange(len(confusion_mat)-1))
ax.set_xticklabels(['class1','class2','class3'])
```

```
    ax.set_yticklabels(['class1','class2','class3'])
    ax.set_xlabel('Target Class')
    ax.set_ylabel('Output Class')
    ax.xaxis.set_label_coords(0.5,-0.08)
    ax.yaxis.set_label_coords(-0.2,0.5)
    ax.grid(False)
    ax.set_title(f"Confusion Matrix with accuracy {l:1f}%")
    fig.colorbar(im)
    plt.show()

df = pd.read_csv('./drive/MyDrive/cs5691_A2_DB1/dataset_1.csv
    ', sep='\t', header=None)

X = df.iloc[:, :64].values
y = df.iloc[:, -1].values

X_lda = lda(X, y, n_components=2, reg_param=1e-4)
x = np.real(X_lda)

# Shuffle the data and split into training and testing sets
perm = np.random.permutation(len(x))
x, y = x[perm], y[perm]
train_size = int(0.8 * len(x))
x_lda_train, y_train = x[:train_size], y[:train_size]
x_lda_test, y_test = x[train_size:], y[train_size:]

y_pred_train = bayes_classifier(x_lda_train, y_train,
    x_lda_train)
y_pred_test = bayes_classifier(x_lda_train, y_train,
    x_lda_test)


# Calculate training and testing accuracy
train_acc = accuracy(y_train, y_pred_train)
test_acc = accuracy(y_test, y_pred_test)
print('Training Accuracy:', train_acc)
print('Testing Accuracy:', test_acc)

# Plot classification boundary
x_min, x_max = x_lda_test[:, 0].min() - 1, x_lda_test[:, 0].
    max() + 1
y_min, y_max = x_lda_test[:, 1].min() - 1, x_lda_test[:, 1].
    max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1), np.arange(
```

```
    y_min, y_max, 0.1))
Z = np.array(bayes_classifier(x_lda_train, y_train, np.c_[xx.
    ravel(), yy.ravel()]))
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.4)
plt.scatter(x_lda_test[:, 0], x_lda_test[:, 1], c=y_pred_test
    , s=20, edgecolor='k')
plt.title('Classification boundary for LDA-transformed test
    data')
plt.xlabel('LDA component 1')
plt.ylabel('LDA component 2')
plt.show()

# Train data
print('Confusion Matrix (Train Data):')
plot_confusion_matrix(y_train, y_pred_train, "training data")

# Test data
print('Confusion Matrix (Test Data):')
plot_confusion_matrix(y_test,y_pred_test, "test data")
```

Output:

Accuracy on both train and test data.

```
Training Accuracy: 1.0
Testing Accuracy: 1.0
```

Plot of the test data along with your classiffication boundary.

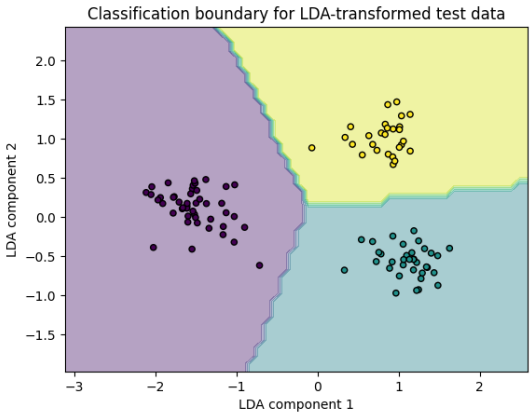
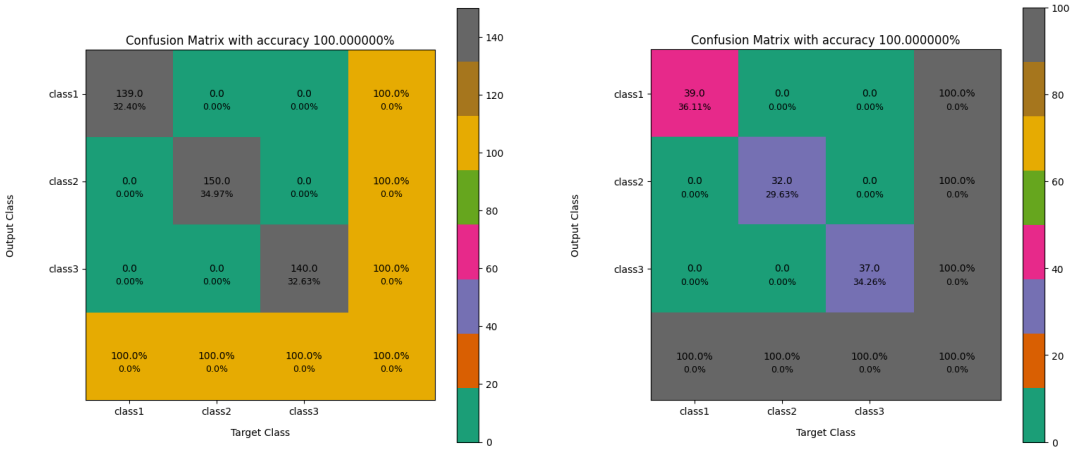


Figure 7: classiffication boundary of test data.

confusion matrices on both train and test data.



(a) confusion matrix of training data.

(b) confusion matrix of test data.

Figure 8: confusion matrices on both train and test data.

```
Confusion Matrix (Train Data):
[[139.   0.   0.]
 [  0. 150.   0.]
 [  0.   0. 140.]]
[100.0, 100.0, 100.0]
[100.0, 100.0, 100.0, 100.0]

Confusion Matrix (Test Data):
[[39.  0.  0.]
 [ 0. 32.  0.]
 [ 0.  0. 37.]]
[100.0, 100.0, 100.0]
[100.0, 100.0, 100.0, 100.0]
```

2. **[DBSCAN]** In this Question, you are supposed to implement **DBSCAN algorithm from scratch** on dataset2 provided here and dataset3 provided here. You also need to compare and contrast your observations from above with K-Means applied on both datasets. **However, you can use predefined libraries to implement K-means.**

   (a) (1 mark) Visualize the data in dataset2. Then, find a suitable **range of values for epsilon** (a hyperparameter in DBSCAN algorithm) by using the 'Elbow Curve' of Datapoints plotted between K-Distance vs Epsilon. For simplicity, take only integer values for epsilon. **You can use predefined libraries to implement K-distance.**

Solution:

Part (a):

Listing 5: Visualizing the data in dataset2.

```
import matplotlib.pyplot as plt
import numpy as np
X = np.loadtxt("dataset2.csv", delimiter=",")
plt.scatter(X[:,0],X[:,1],color='k',s=8)
plt.show()
```

Output:

Visualization of dataset2.

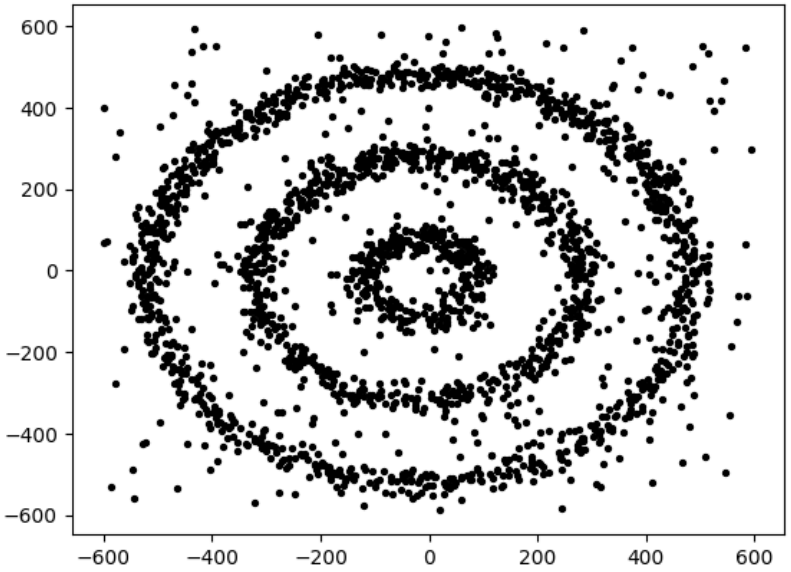


Figure 9: Visualization of dataset2

Listing 6: 'Elbow Curve' of Datapoints plotted between K-Distance vs Epsilon.

```
from sklearn.neighbors import NearestNeighbors
import numpy as np
import matplotlib.pyplot as plt
def eps_range(X,k):
  nbrs=NearestNeighbors(n_neighbors=k).fit(X)
```

```
    d,i=nbrs.kneighbors(X)
    kd=d[:,k-1]
    sorted_d=np.sort(kd)
    x=[]
    y=[]
    for i in range(0,len(sorted_d)):
        if i%1==0:
            x.append(i)
            y.append(sorted_d[i])
    x=np.array(x)
    y=np.array(y)
    dy=np.gradient(y,x)
    d2y=np.gradient(dy,x)
    inf_coord=np.where(np.diff(np.sign(d2y))>0)[0]+1
    y=y[inf_coord]
    epsilon_range=[]
    for k in y:
        if int(k) not in epsilon_range:
            epsilon_range.append(int(k))
    print(epsilon_range)
    return epsilon_range

from sklearn.neighbors import NearestNeighbors
import numpy as np
import matplotlib.pyplot as plt
X = np.loadtxt("dataset2.csv", delimiter=",")
nbrs = NearestNeighbors(n_neighbors=2,algorithm='auto').fit(X
    )
d,i=nbrs.kneighbors(X)
kd=d[:,1]
sorted_d=np.sort(kd)
plt.plot(sorted_d)
plt.xlabel('Index')
plt.ylabel('K-Distance')
plt.show()
```

Output:

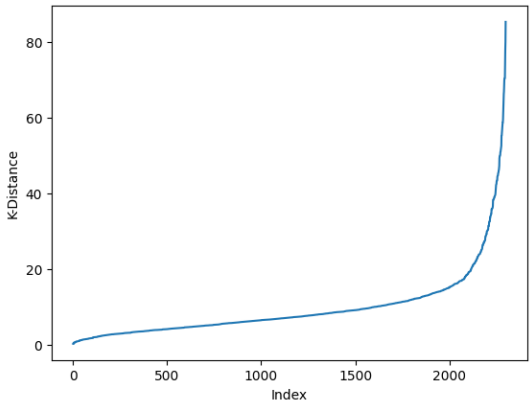


Figure 10: Elbow Curve

The nature of the curve changes sharply for epsilon values from 15 to 40
Suitable epsilon range = [15,40]

(b) (2 marks) Implement DBSCAN with the above suitable range of values of epsilon and detect the optimal value of epsilon, which gives the best clustering visually on the dataset. Show a visualization of the clusters formed for the best value of epsilon.

Solution:

Part (b):

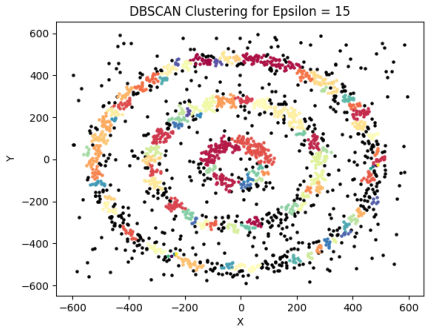
Listing 7: Implementing DBSCAN with the above suitable range of values of epsilon and detect the optimal value of epsilon

```
def dbscan(X,e,num):
  points=[]
  d=np.linalg.norm(X[:,np.newaxis] - X,axis=2)
  labels=np.full(X.shape[0],-1)
  for i in range(X.shape[0]):
      if np.sum(d[i]<=e)>=num:
          points.append(i)
  c=0
  for i in points:
```

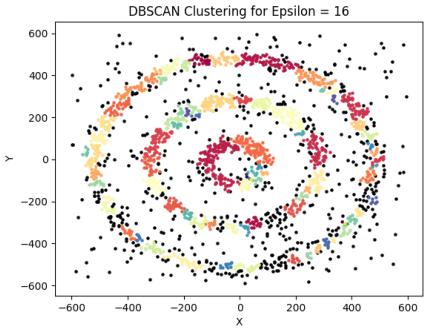
```
        if labels[i]==-1:
            n=[i]
            c+=1
            labels[i]=c
            while n:
                neighbor=n.pop(0)
                neighbor_indices=np.where(d[neighbor]<=e)[0]
                if len(neighbor_indices)>=num:
                    for idx in neighbor_indices:
                        if labels[idx]==-1:
                            n.append(idx)
                            labels[idx]=c
                else:
                    for idx in neighbor_indices:
                        if labels[idx]==-1:
                            labels[idx]=c
    return labels

def partb(X,no,erange):
    for e in erange:
        r=dbscan(X,e,no)
        clusters=np.unique(r)
        c=plt.cm.Spectral(np.linspace(0,1,len(clusters)))
        for i,cluster in enumerate(clusters):
            if cluster==-1:
                color='k'
            else:
                color=c[i]
            plt.scatter(X[r==cluster,0],X[r==cluster,1],c=color,s
                =5)
        plt.xlabel('X')
        plt.ylabel('Y')
        plt.title(f'DBSCAN Clustering for Epsilon = {e}')
        print('No of clusters :',len(set(r))-(1 if -1 in r else
            0))
        plt.show()

X = np.loadtxt("dataset2.csv", delimiter=",")
erange=[]
for i in range(15,41):
    erange.append(i)
partb(X,5,erange)
```

Output:

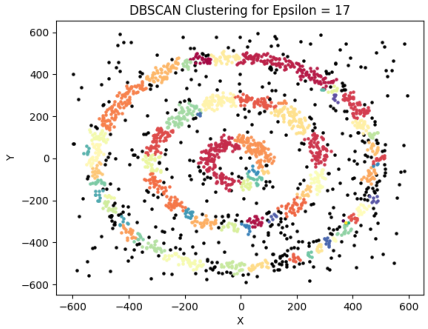
visualization of the clusters formed using range epsilone values found.



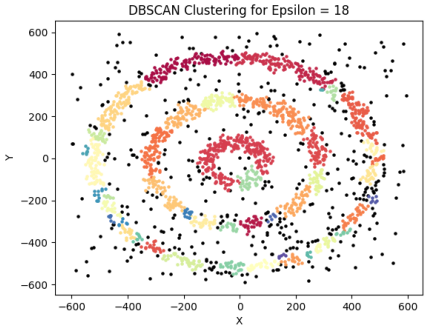
(a) No of clusters : 98.



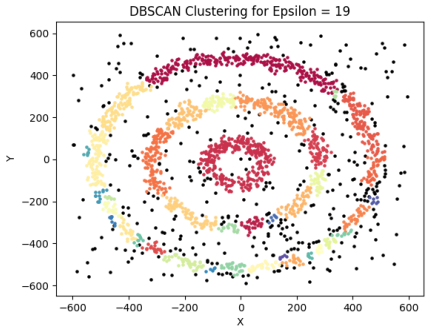
(b) No of clusters : 82.



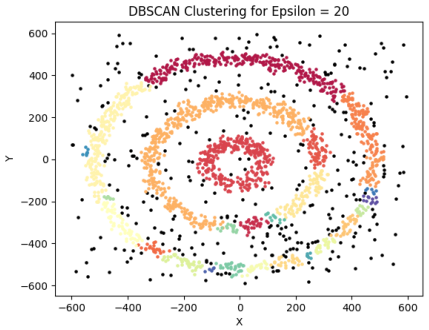
(a) No of clusters : 69.



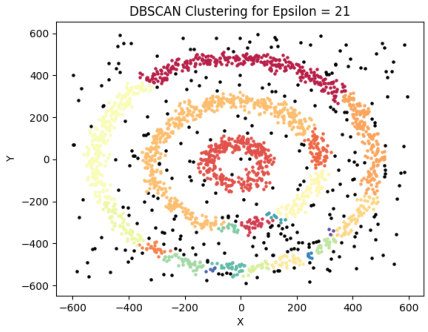
(b) No of clusters : 42.



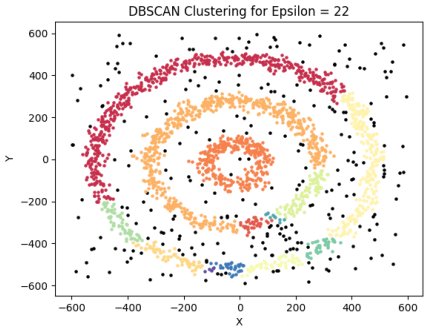
(a) No of clusters : 38.



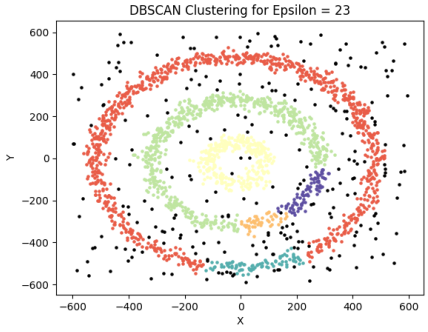
(b) No of clusters : 26.



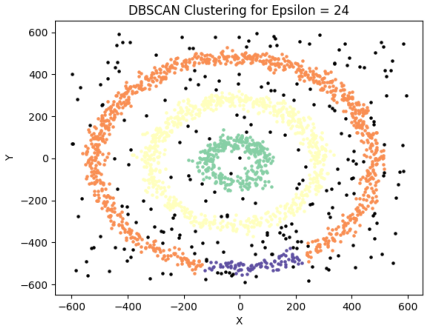
(a) No of clusters : 21.



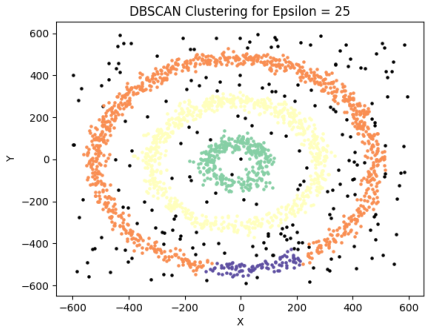
(b) No of clusters : 13.



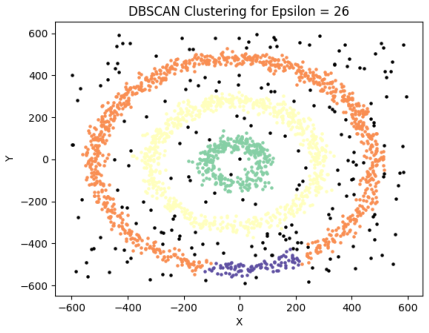
(a) No of clusters : 6.



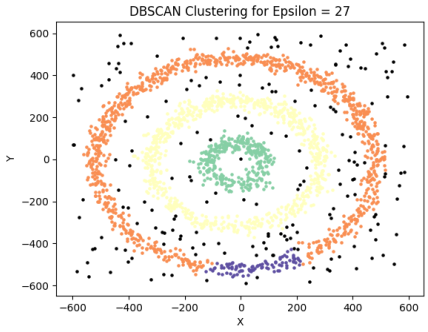
(b) No of clusters : 4.



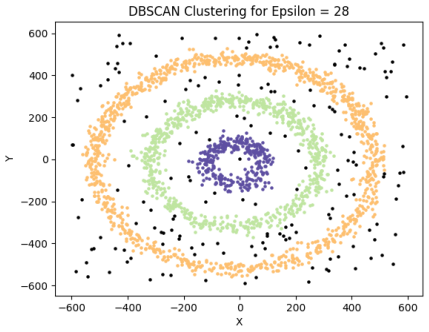
(a) No of clusters : 4.



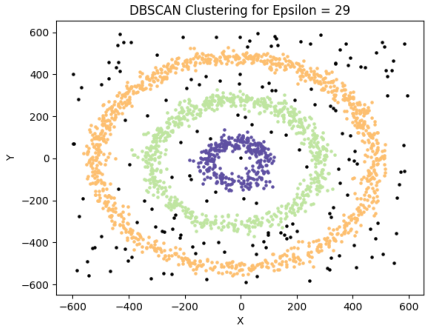
(b) No of clusters : 4.



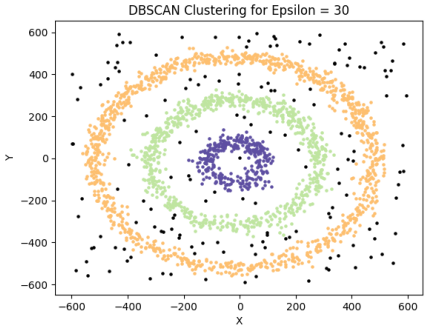
(a) No of clusters : 4.



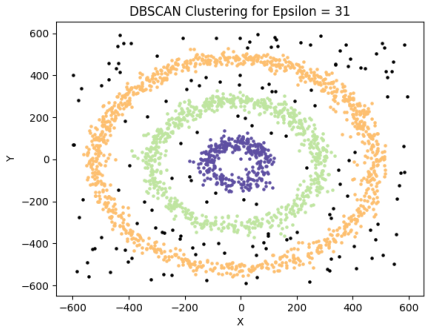
(b) No of clusters : 3.



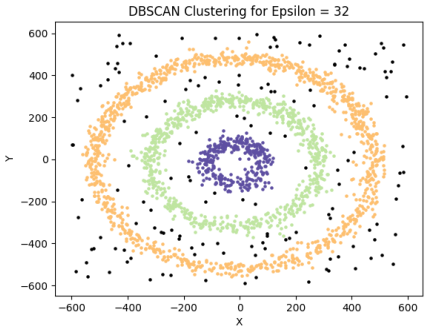
(a) No of clusters : 3.



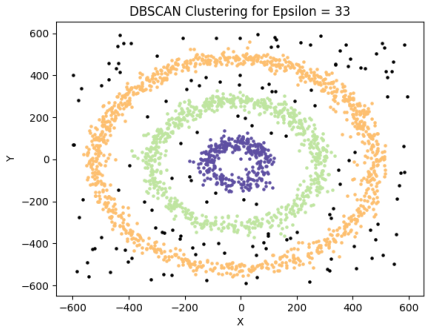
(b) No of clusters : 3.



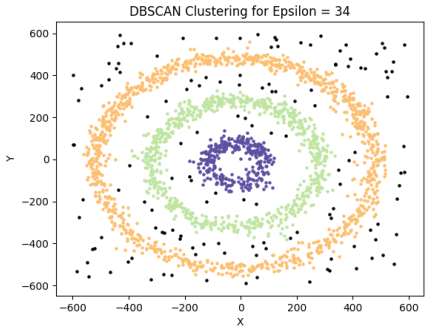
(a) No of clusters : 3.



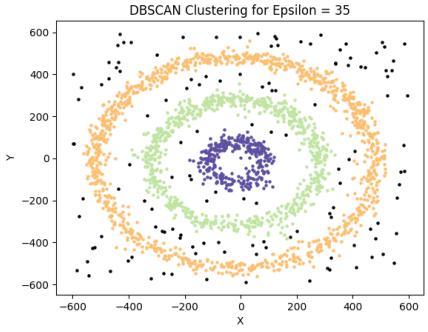
(b) No of clusters : 3.



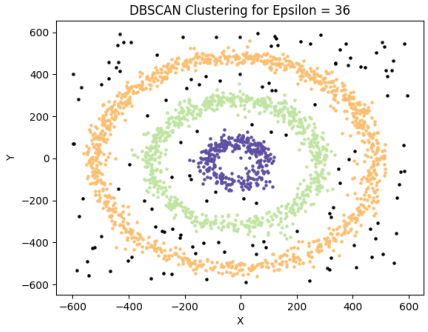
(a) No of clusters : 3.



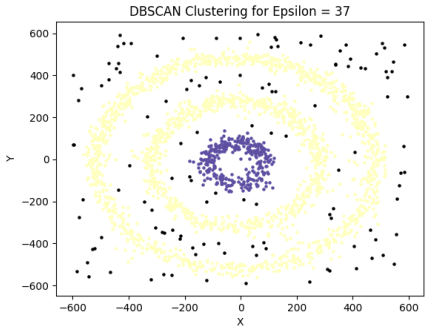
(b) No of clusters : 3.



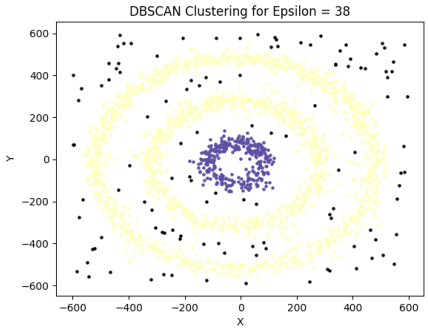
(a) No of clusters : 3.



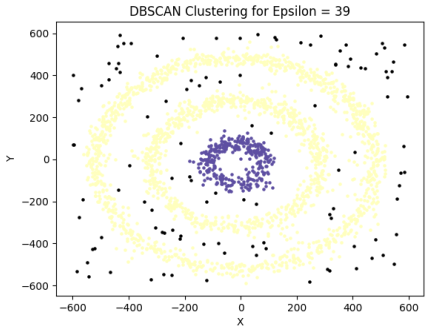
(b) No of clusters : 3.



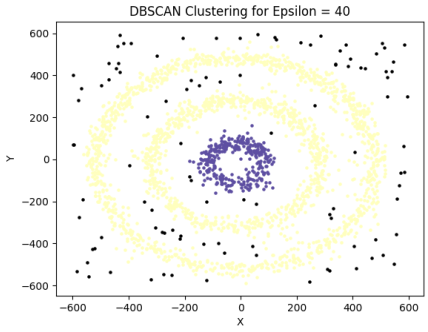
(a) No of clusters : 2.



(b) No of clusters : 2.



(a) No of clusters : 2.



(b) No of clusters : 2.

Looking at the raw data a human can easily conclude that there are three clusters. The DBSCAN clustering for epsilon = 28 to 36 gives three clusters.Given that for epsilon=28 the DBSCAN algorithm just shifts from 4 clusters to 3 clusters and after epsilon=36 the algorithm shifts on to 2 clusters. The optimal value of epsilon is 32 for which we get best clustering visually on the dataset.

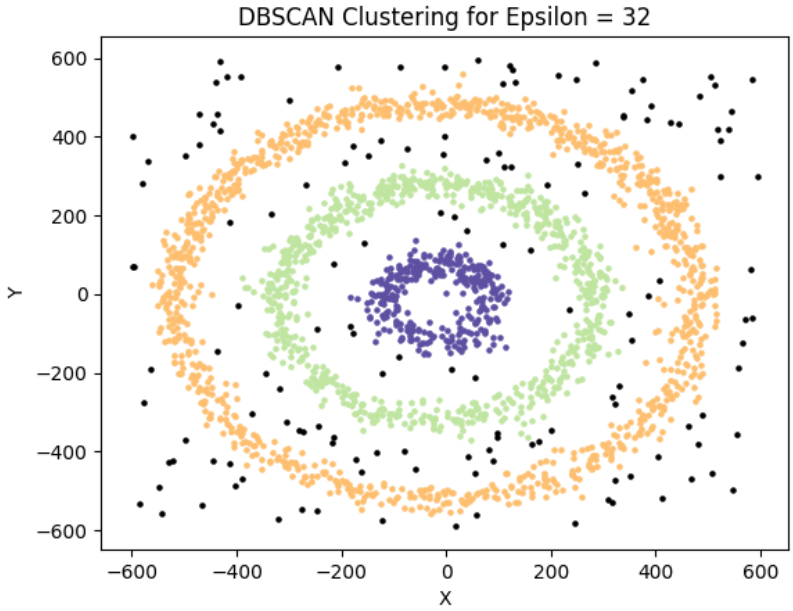


Figure 24: Best clustering with epsilon=32

(c) (1.5 marks) Implement K-Means and use it on dataset2 with value of K (number of clusters) set to the optimum number of clusters that you get from (b) above. Suggest various techniques to improve the clustering by KMeans in this case.

Solution:

Part (c):

Listing 8: Implementing K-Means and use it on dataset2 with value of K

```
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt
X=np.loadtxt("dataset2.csv",delimiter=",")
model=KMeans(n_clusters=3)
model.fit(X)
l=model.labels_
centroids=model.cluster_centers_
colors=['r','b','g']
for i in range(len(X)):
    plt.scatter(X[i,0],X[i,1],color=colors[l[i]],alpha=0.4,s
        =9)
i=0
colors=['darkred','darkblue','darkgreen']
for c in centroids:
  plt.scatter(c[0],c[1],marker='x',color=colors[i],s=300)
  i+=1
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Kmeans clustering')
plt.show()
```

Output:

K (number of clusters) set to the optimum number of clusters obtained in part (b).

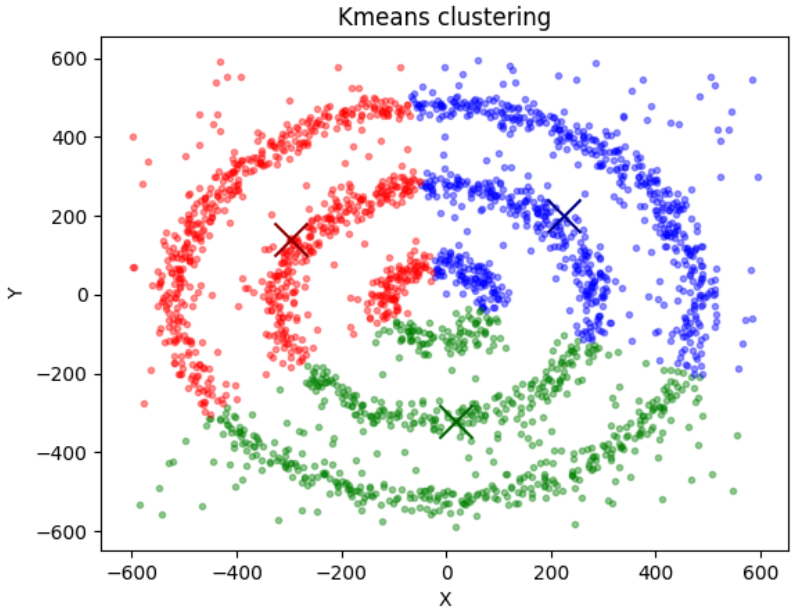


Figure 25: Clusters obtained using KMEAN Algorithem

- The standard Kmeans algorithm for k =3 does not cluster points the way DBSCAN did , actually it is far off.
- Some methods to improve the clustering would be to start off with better initialization of centroids, or change the distance metrics from euclidean to cosine similarity.
- When we tried the above methods for this data the result was not that different from the standard Kmeans algorithm.
- But when we converted the data from cartesian to polar coordinates and used Kmeans algorithm the clustering was really good,very close to DBSCAN.
- So in our case , the clusters being annular,changing the coordinate system is the best way to improve Kmeans clustering .

Listing 9: Implementing K-Means and use it on dataset2 with value of K

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Load data
X=np.loadtxt("dataset2.csv", delimiter=",")
r=np.sqrt(X[:,0]*X[:,0] + X[:,1]*X[:,1])
```

```
theta=np.arctan2(X[:, 1], X[:, 0])
X_polar=np.column_stack((r, theta))
# Perform clustering using KMeans
model=KMeans(n_clusters=3)
#instead of xcoord and ycoord it is now r and theta
model.fit(X_polar)
l=model.labels_
centroids=model.cluster_centers_
# Plot the results
colors=['r','b','g']
for i in range(len(X)):
    plt.scatter(X[i,0],X[i,1],color=colors[l[i]],alpha=0.4,s
        =9)
i = 0
colors=['darkred','darkblue','darkgreen']
for c in centroids:
    x=c[0] * np.cos(c[1])
    y=c[0] * np.sin(c[1])
    plt.scatter(x,y,marker='x',color=colors[i],s=300)
    i+=1
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Kmeans clustering using polar coordinates')
plt.show()
```

Output:

K (number of clusters) set to the optimum number of clusters = 3 obtained in part (b) using polar coordinates.

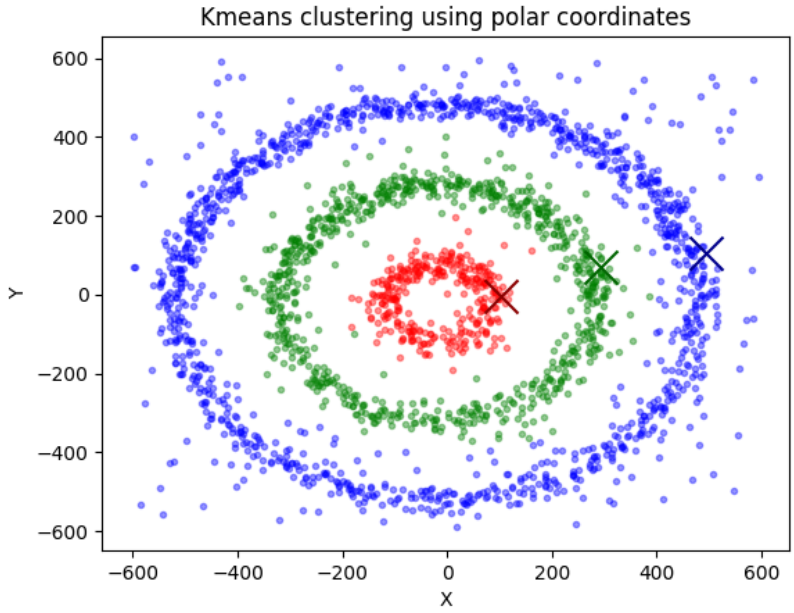


Figure 26: Clusters obtained using KMEANS Algorithem (polar coordinates)

Analysis and Observations

(d) (1.5 marks) Show a visualization of the data in dataset3. Use your implementation of DBSCAN with `minPts=15` on dataset3. Plot 'Elbow curve' to get an optimal range of values for `eps`. Detect the optimal value of epsilon which gives the best clustering visually on the dataset. Show a visualization of the clusters formed for the best value of epsilon.

Solution:

Part (d):

Listing 10: Visualizing the data in dataset3.

```
import matplotlib.pyplot as plt
X=np.loadtxt("dataset3.csv",delimiter=",")
plt.scatter(X[:,0],X[:,1],color='k',s=8)
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```

Visualizing the data in dataset3.

Output:

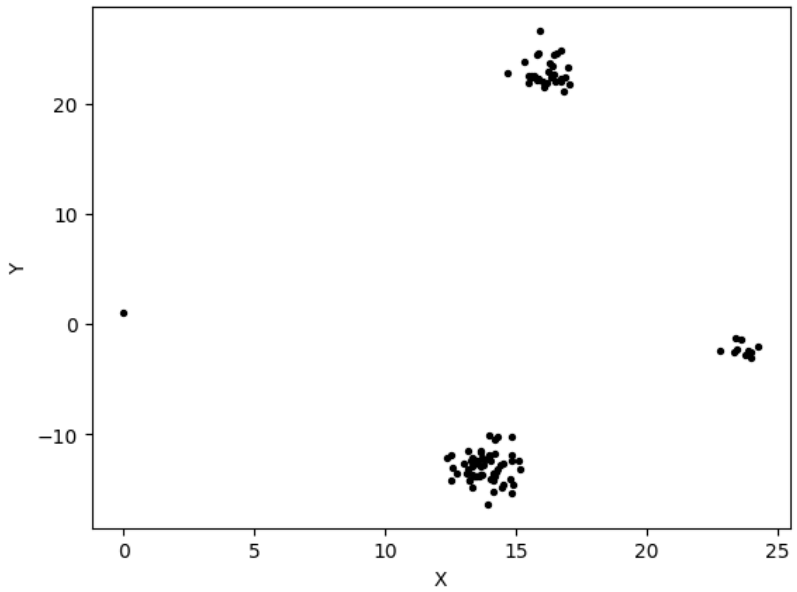


Figure 27: Visualizing the data in dataset3.

Listing 11: 'Elbow Curve' of Datapoints plotted between K-Distance vs Epsilon.

'Elbow Curve' of Datapoints plotted between K-Distance vs Epsilon.

Output:

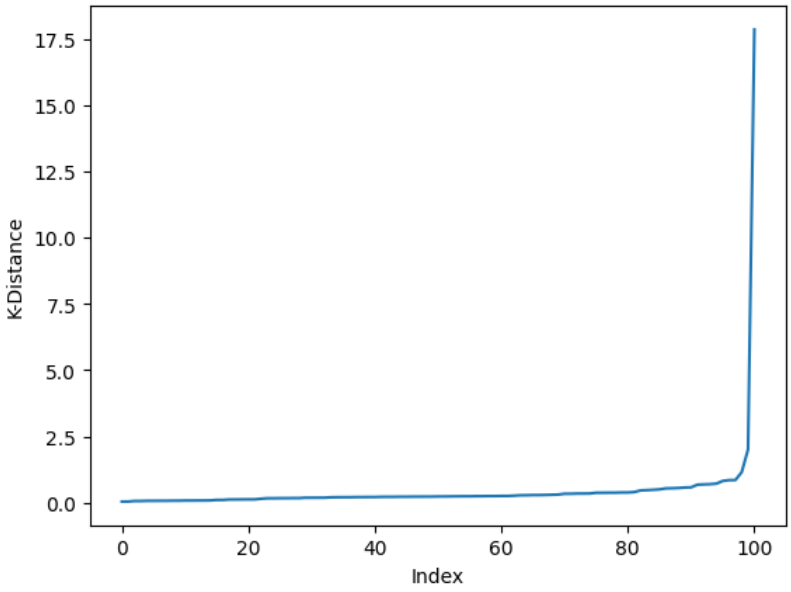


Figure 28: Elbow Curve

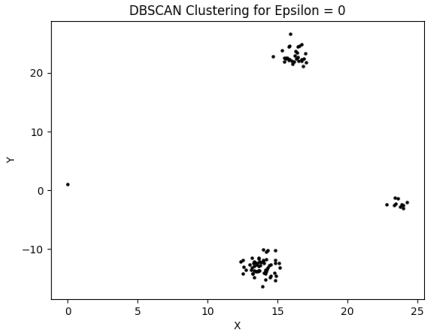
The nature of the curve changes sharply for epsilon values from 0.5 to 5 Suitable epsilon range = [0,5]

Listing 12: Visualization of Statistical analaysis of orginal and normalized data

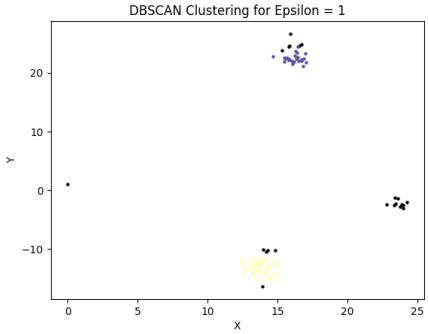
```
X=np.loadtxt("dataset3.csv",delimiter=",")
erange=[]
for i in range(0,6):
  erange.append(i)
partb(X,15,erange)
```

visualization of the clusters formed using range epsilone values found.

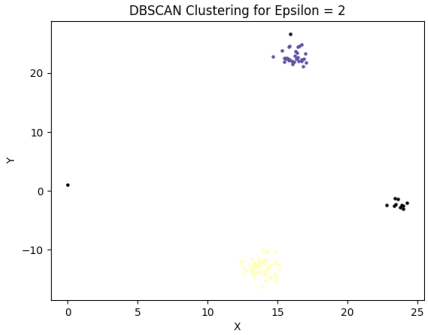
Output:



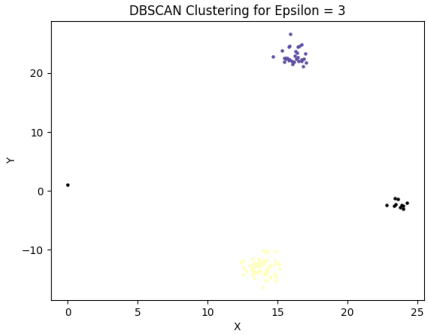
(a) DBSCAN clustering for epsilon 0.



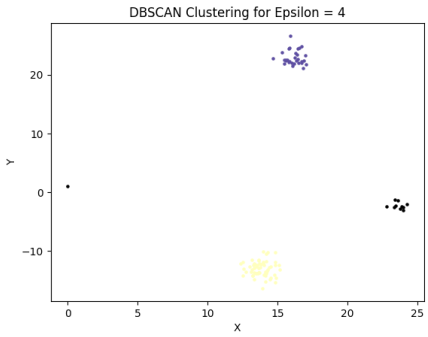
(b) DBSCAN clustering for epsilon 1.



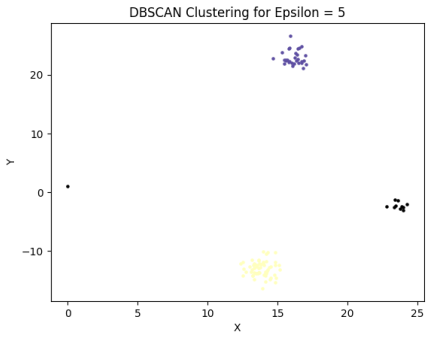
(a) DBSCAN clustering for epsilon 2.



(b) DBSCAN clustering for epsilon 3.



(a) DBSCAN clustering for epsilon 4.



(b) DBSCAN clustering for epsilon 5.

for epsilon = 3,4,5 the algorithm picks up 2 out of 3 clusters and classifies the rest as noise Optimal value of epsilon = 4

(e) (1 mark) Now perform KMeans with K=3. Write your observations for obtained results in (d) and (e). Did we give you bad initialization values?

Solution:

Listing 13: Implementing K-Means and use it on dataset3 with value of K

```
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt
X=np.loadtxt("dataset3.csv",delimiter=",")
model=KMeans(n_clusters=3)
model.fit(X)
l=model.labels_
centroids=model.cluster_centers_
colors=['r','b','g']
for i in range(len(X)):
    plt.scatter(X[i,0],X[i,1],color=colors[l[i]],alpha=0.3,s
        =8)
i=0
colors=['maroon','darkblue','darkgreen']
for c in centroids:
  plt.scatter(c[0],c[1],marker='x',color=colors[i],s=300)
  i+=1
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Clusterings using kmeans')
plt.show()
```

Output:

K (number of clusters) set to the optimum number of clusters

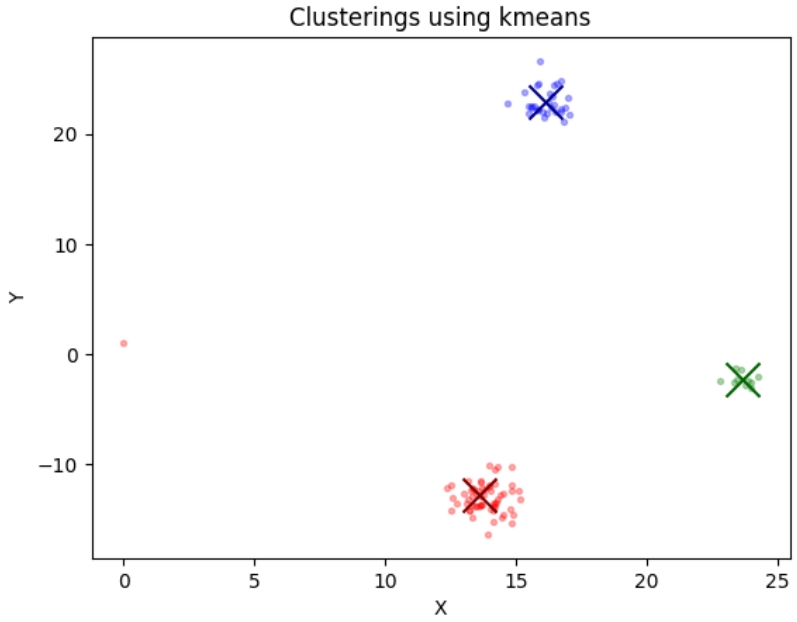


Figure 32: Clusters obtained using KMEAN Algorithem

- The Kmeans algorithm does a better job than the DBSCAN.DBSCAN detects only 2 of the three clusters and classifies the rest as noise.

- the initialization value for DBSCAN is bad, for minpts=15 the algorithm is not able to form one of the three clusters because the number of points in that cluster are lower than 15.

- the initialization for Kmeans is perfect because there are only 3 clusters

If we reduce minpts to 10 and run the same algorithm we get perfect clustering for epsilon = 3,4,5 and the algorithm doesn't not include the outlier point into any of the clusters which the Kmeans algorithm does.

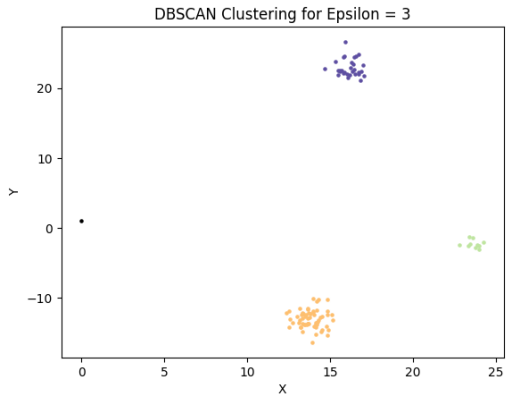
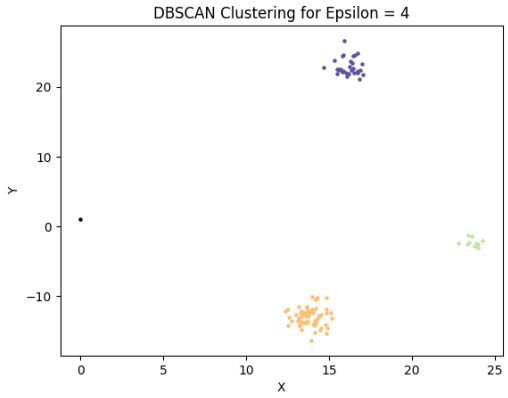
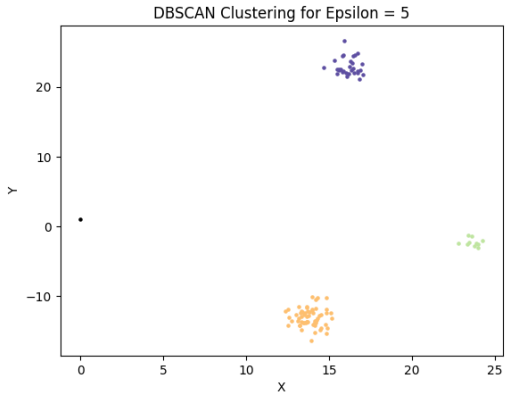


Figure 33: DBSCAN clustering for epsilon 3.



(a) DBSCAN clustering for epsilon 4.



(b) DBSCAN clustering for epsilon 5.

(f) (1 mark) Based on all your learnings from this question, state the relative pros and cons of KMeans vs DBSCAN.

Solution:

- In Kmeans, the number of clusters must be predetermined, whereas in DBSCAN it is not necessary to do so. DBSCAN is a good choice when the number of clusters is unknown.
- To use DBSCAN, we must specify the minimum number of points, and if it is not accurate, we may miss some clusters, as demonstrated in part d of this question.
- When using DBSCAN, we must exercise caution when selecting the value of

epsilon, as demonstrated in parts b and d. With the same minimum points, different values of epsilon result in varying clusters.

- Kmeans assigns all points to a cluster, resulting in the classification of noise, which is undesirable. In contrast, DBSCAN can easily exclude noise.

3. **[GMM]** In this question, you are supposed to implement the Expectation-Maximization algorithm for Gaussian mixture models on the given dataset4. The data can be found here.

   (a) (3 marks) Implement EM for GMM and plot the log-likelihood as a function of iterations.

Solution:

Part (a):

Listing 14: Ploting the log-likelihood as a function of iterations and dffererent numbers of Gaussians

```
import numpy as np
import matplotlib.pyplot as plt
import math
#define functions required for gmm

#multivariate normal func the return probab density
def multivariate(x,mean,cov):
  d=len(x)
  x=np.matrix(x-mean)
  det_cov=np.linalg.det(cov)
  inv_cov=np.linalg.inv(cov)
  prob=np.exp(x*inv_cov*x.T*-0.5)
  prob=prob*1/(math.pow((2*np.pi),d/2)*math.pow(det_cov,0.5))
  return prob
#logliklihood function
def loglike(x,mean,cov,weights):
  r,c=x.shape
  log=0
  for i in range(r):
    p=0
    for k in range(len(weights)):
      p+=weights[k]*multivariate(x[i],mean[k],cov[k])
    log+=np.log(p)
  return log
def posteriorcalc(x,mean,cov,weights):
    r,c =x.shape
```

```
    posterior=np.zeros((r,len(weights)))
    for n in range(r):
        for k in range(len(weights)):
            posterior[n,k]=weights[k]*multivariate(x[n],mean[
                k],cov[k])
        posterior[n] /= posterior[n].sum()
    return posterior
def mx_likelihood(x,posterior):
    r,c=x.shape
    K=posterior.shape[1]
    mean=np.zeros((r,c))
    cov=np.zeros((K,c,c))
    weights=np.zeros(K)
    for k in range(K):
        Nk=posterior[:,k].sum()
        weights[k]=Nk/r
        mean[k]=posterior[:,k].dot(x)/Nk
        for n in range(r):
            xn = X[n]-mean[k]
            cov[k]+=posterior[n,k]*np.outer(xn,xn)
        cov[k]= cov[k]/Nk
    return mean,cov,weights
```

Listing 15: Ploting the log-likelihood as a function of iterations and dffererent numbers of Gaussians using Kmean alogrithem

```
from sklearn.cluster import KMeans
X = np.loadtxt("dataset4.csv", delimiter=",")
log=[]
posteriors=[]
means=[]
covs=[]
for k in range(2, 7):
    # Initialize the model parameters
    kmeans=KMeans(n_clusters=k,random_state=0).fit(X)
    mean =kmeans.cluster_centers_
    # mean = np.random.randn(k, X.shape[1])
    cov = np.array([np.eye(X.shape[1]) for _ in range(k)])
    weights = np.ones(k) / k
    # Run EM algorithm
    iterations=200
    log_values=np.zeros(iterations)
    for i in range(iterations):
        posterior=posteriorcalc(X,mean,cov,weights)
```

```
            mean,cov,weights=mx_likelihood(X,posterior)
            log_values[i]=loglike(X,mean,cov,weights)
            if i>0 and np.abs(log_values[i]-log_values[i-1])<1e
                -7:
                break
        log.append(log_values[:i+1])
        means.append(mean)
        covs.append(cov)
        posteriors.append(posterior)
```

Listing 16: Ploting the log-likelihood as a function of iterations and dffererent numbers of Gaussiansusing random mean

```
from sklearn.cluster import KMeans
X = np.loadtxt("dataset4.csv", delimiter=",")
log=[]
posteriors=[]
means=[]
covs=[]
for k in range(2, 7):
    mean=np.array([np.mean(X,axis=0) +np.random.randn(X.shape
        [1])*0.001 for i in range(k)])
    cov = np.array([np.eye(X.shape[1]) for _ in range(k)])
    weights = np.ones(k) / k
    # Run EM algorithm
    iterations=200
    log_values=np.zeros(iterations)
    for i in range(iterations):
        posterior=posteriorcalc(X,mean,cov,weights)
        mean,cov,weights=mx_likelihood(X,posterior)
        log_values[i]=loglike(X,mean,cov,weights)
        if i>0 and np.abs(log_values[i]-log_values[i-1])<1e
            -7:
            break
    log.append(log_values[:i+1])
    means.append(mean)
    covs.append(cov)
    posteriors.append(posterior)
```

Listing 17: Ploting the log-likelihood as a function of iterations and dffererent numbers of Gaussians

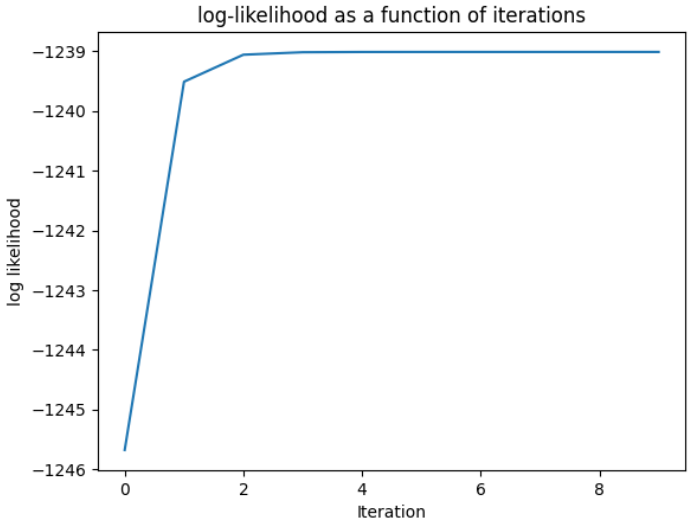
```
plt.plot(log[0])
plt.xlabel('Iteration')
plt.ylabel('log_likelihood')
```

```
plt.title('log-likelihood as a function of iterations')
plt.show()
for k in range(len(log)):
  plt.plot(log[k],label=k+2)
plt.xlabel('Iteration')
plt.legend()
plt.ylabel('log likelihood')
plt.title('log likelihood as a function of iterations for
   different values of k')
plt.show()
```

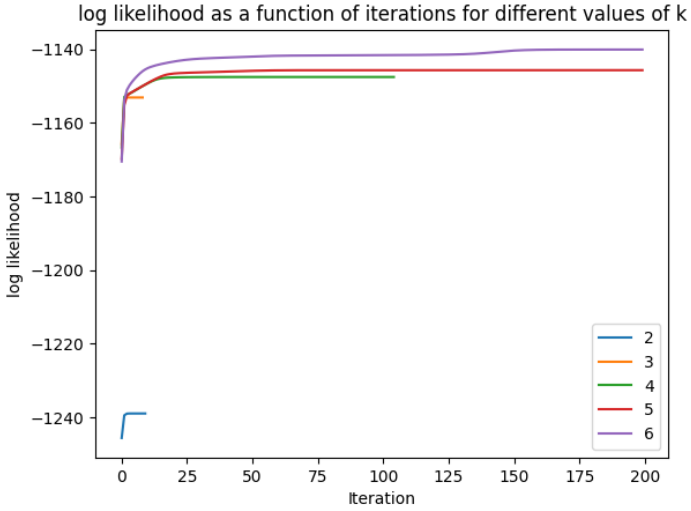
Note: Here I am also printing output for EM for different numbers of Gaussians (k)(Try 2,3,4,5,6). **Note:** Here we are using KMEAN and randome mean algorithem to compare and analyse Log-likelihood as a function of iterations.

Output:

Log-likelihood as a function of iterations means are initialized using Kmeans

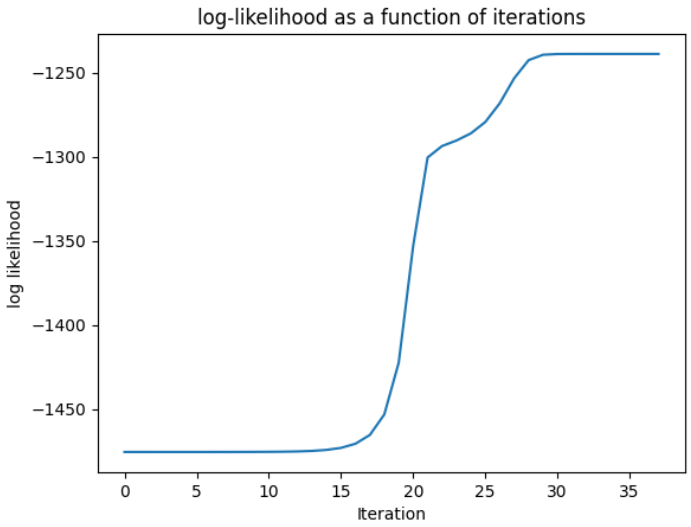


(a) Log-likelihood as a function of iterations K = 2.

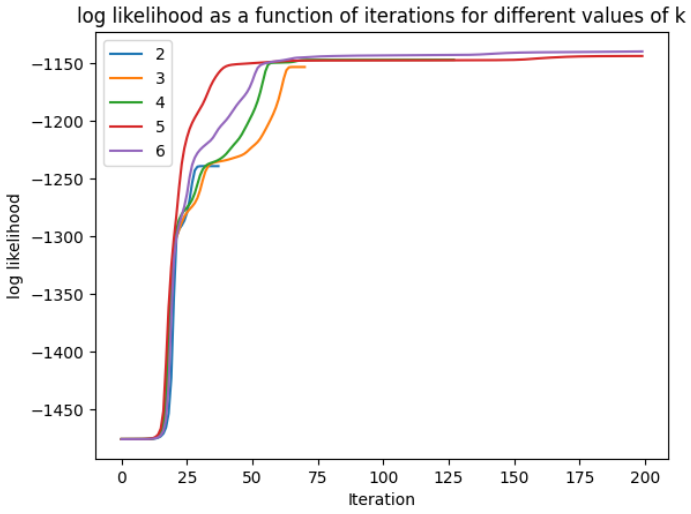


(b) Log likelihood as a function of iteration for different values of k.

Log-likelihood as a function of iterations means are initialized randomly



(a) Log-likelihood as a function of iterations K = 2.



(b) Log likelihood as a function of iteration for different values of k.

(b) (2 marks) Run EM for different numbers of Gaussians (k)(Try 2,3,4,5,6). Plot figures that can help in visualization and also log likelihood as a function of iteration for different values of k. Report the observations.

Solution:

Part (b):

Listing 18: visualization and also log likelihood as a function of iteration for differentvalues of k using k mean.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal

def plot_cluster(X, posterior, mean, covariance):
    K = posterior.shape[1]
    color1 = ['#fff37f', '#5a7fdf', '#c85bd3', '#ffae7d', '#
        ff7c8f', '#63d87d']
    color2 = ['#7f710c', '#1b307c', '#48015e', '#7d3e18', '
        #722026', '#1e5b23']
    for k in range(K):
        plt.scatter(X[:,0][posterior[:,k]>0.5], X[:,1][
            posterior[:,k]>0.5], color=color1[k], alpha=0.4)
        rv = multivariate_normal(mean=mean[k], cov=covariance
            [k])
        x, y = np.mgrid[-1:6:.01, -3:4:.01]
        pos = np.dstack((x, y))
        plt.contour(x, y, rv.pdf(pos), colors=color2[k])
    for k in range(K):
        plt.scatter(mean[k,0], mean[k,1], color=color2[k],
            marker='x', s=150)


for i in range(len(means)):
  plot_cluster(X,posteriors[i],means[i],covs[i])
  plt.xlabel('X')
  plt.ylabel('Y')
  plt.title(f'Clusters and their corresponding means(X) when
     k = {i+2}')
  plt.show()
```

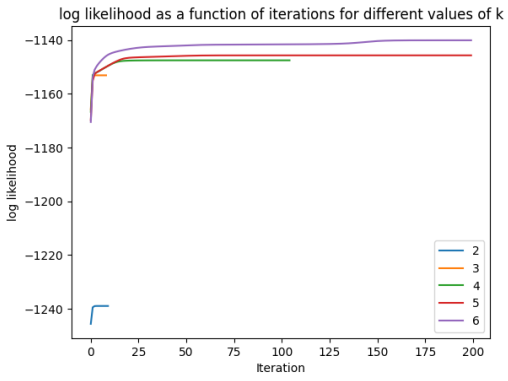
Listing 19: visualization and also log likelihood as a function of iteration for different values of k using random mean.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal

def plot_cluster(X, posterior, mean, covariance):
    K = posterior.shape[1]
```

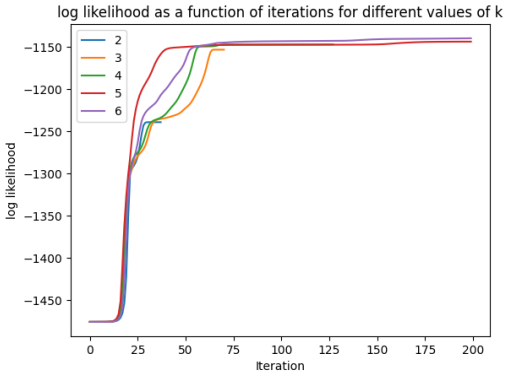
```
    color1 = ['#fff37f', '#5a7fdf', '#c85bd3', '#ffae7d', '#
        ff7c8f', '#63d87d']
    color2 = ['#7f710c', '#1b307c', '#48015e', '#7d3e18', '
        #722026', '#1e5b23']
    for k in range(K):
        plt.scatter(X[:,0][posterior[:,k]>0.5], X[:,1][
            posterior[:,k]>0.5], color=color1[k], alpha=0.4)
        rv = multivariate_normal(mean=mean[k], cov=covariance
            [k])
        x, y = np.mgrid[-1:6:.01, -3:4:.01]
        pos = np.dstack((x, y))
        plt.contour(x, y, rv.pdf(pos), colors=color2[k])
    for k in range(K):
        plt.scatter(mean[k,0], mean[k,1], color=color2[k],
            marker='x', s=150)


for i in range(len(means)):
  plot_cluster(X,posteriors[i],means[i],covs[i])
  plt.xlabel('X')
  plt.ylabel('Y')
  plt.title(f'Clusters and their corresponding means(X) when
      k = {i+2}')
  plt.show()
```

Output:

Log-likelihood Vs iterations for different values of k like 2,3,4,5,6.

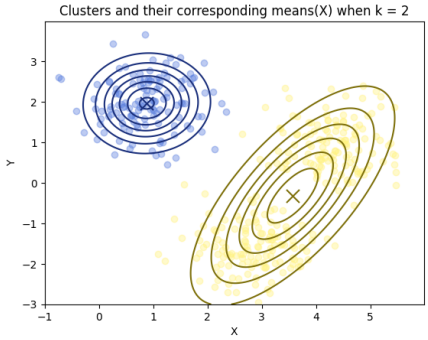


(a) Log likelihood as a function of iteration for different values of k (kmean).

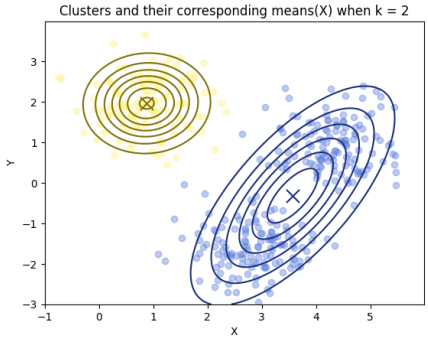


(b) Log likelihood as a function of iteration for different values of k (randome mean).

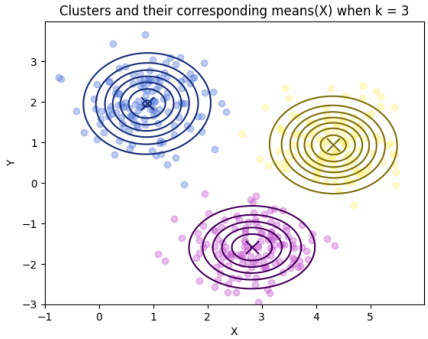
visualization for different values of k.



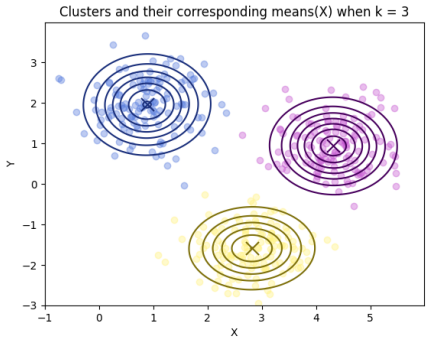
(a) Clusters and their corresponding means(X) when k=2 (kmean)



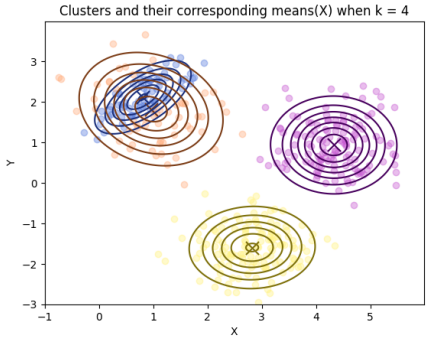
(b) Clusters and their corresponding means(X) when k=2 (random mean)



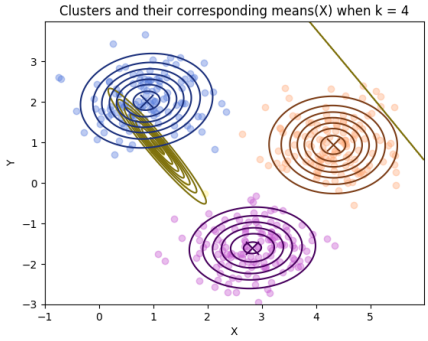
(a) Clusters and their corresponding means(X) when k=3 (kmean)



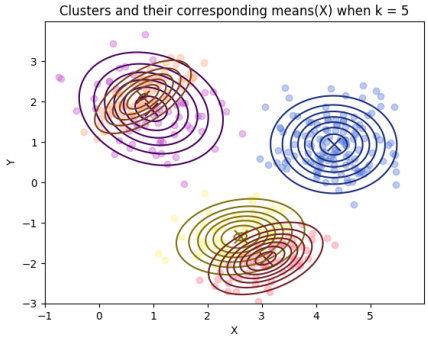
(b) Clusters and their corresponding means(X) when k=3 (random mean)



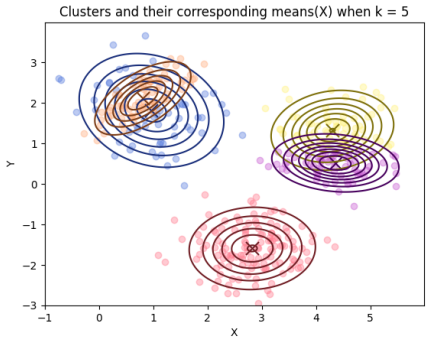
(a) Clusters and their corresponding means(X) when k=4 (kmean)



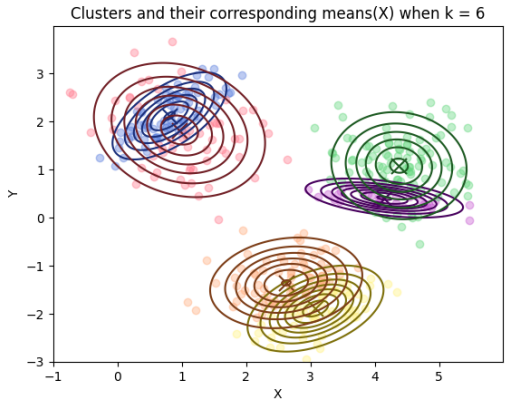
(b) Clusters and their corresponding means(X) when k=4 (random mean)



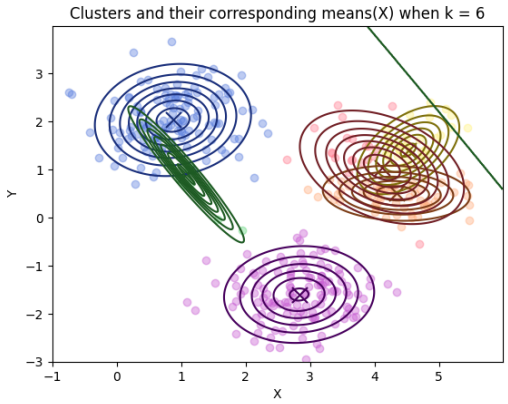
(a) Clusters and their corresponding means(X) when k=5 (kmean)



(b) Clusters and their corresponding means(X) when k=5 (random mean)



(a) Clusters and their corresponding means(X) when k=6 (kmean)



(b) Clusters and their corresponding means(X) when k=6 (random mean)

- Visually, the best clustering is obtained for K=3 irrespective of the method used for initializing the means.
- for both methods the number of iterations required for k=2,3 is less.
- When using random means initialization, a higher number of iterations is required for the log likelihood to reach convergence.
- When using K-means initialization, the log likelihood not only starts off close to the final value but also converges quickly.
- When using random initialization, the centroid values can vary each time the code is run, especially for higher values of K. On the other hand, with Kmeans initialization, the centroid values are more consistent and do not change significantly.

(c) (2 marks) Find the optimal k. There are several metrics like Silhouette score, Distance between GMMs, and Bayesian information criterion (BIC), or even you can use log-likelihood from the last question to infer. Give a clear explanation for your decision.
Note: **You can use third-party libraries - sklearn or any other only in this subsection.**

Solution:

Part (c):

Listing 20: Analaysis using log- likelihood

```
lx=[]
ly=[]
for i in range(len(log)):
    lx.append(i+2)
    ly.append(np.max(log[i]))
plt.plot(lx,ly,"bo-")
plt.xlabel("k - number of clusters")
plt.ylabel("Max log likelihood")
plt.title("Max log likelihood vs k")
plt.show()
```

Output:

Analysis using loglikelihood

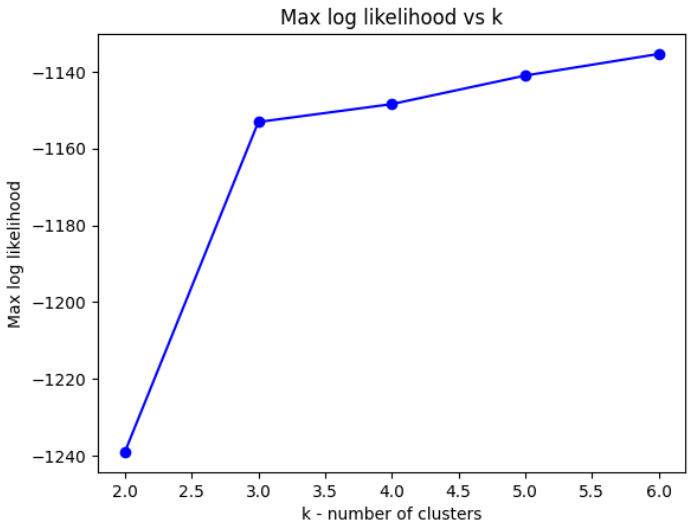


Figure 43: Plot Max log likelihood Vs k (Analysis using loglikelihood)

- We can see from the max log likelihood vs k graph that the value increases significantly as we go from 2 to 3 but there isnt any significant change after that

Listing 21: Analysis using Bayesian information criterion (BIC)

```
from sklearn.mixture import GaussianMixture
X=np.loadtxt("dataset4.csv",delimiter=",")
b=[]
for k in range(2,7):
    model=GaussianMixture(k)
    model.fit(X)
    b.append(model.bic(X))
plt.plot(range(2,7),b,"bo-")
plt.xlabel("k - number of clusters")
plt.ylabel('BIC')
plt.title("BIC vs k")
plt.show()
```

Output:

Analysis using Bayesian information criterion (BIC).

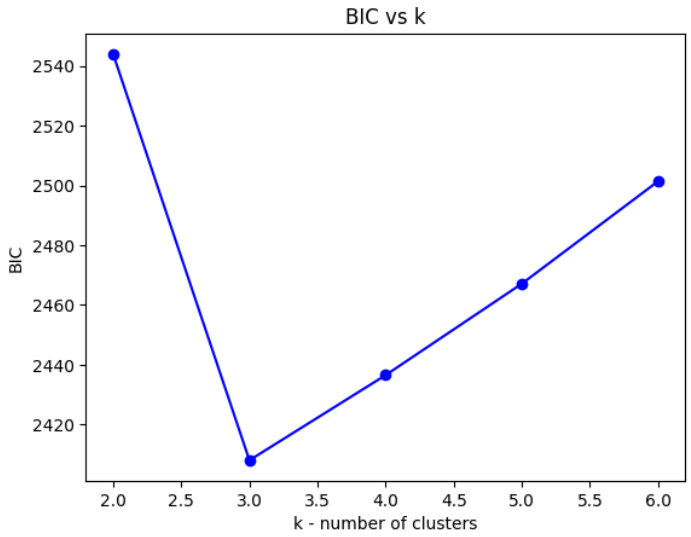


Figure 44: Plot BIC Vs k (Bayesian information criterion (BIC))

- BIC is dependent on both how the model fits and the complexity of the model.
- low score indicates that the fit is good and the complexity of the model is low

- high score means that the fit is bad and the complexity is higher.
- As we can see from the BIC vs k plot for k = 3 we get the lowest value of BIC thus indicating the optimal value of k = 3.

Listing 22: Analysis using Silhouette score

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
X=np.loadtxt("dataset4.csv",delimiter=",")
scores=[]
for k in range(2,7):
    kmeans=KMeans(k).fit(X)
    score=silhouette_score(X,kmeans.labels_)
    scores.append(score)
    print(f"Silhouette score for k={k}: {score:.5f}")
plt.plot(range(2,7),scores,"bo-")
plt.xlabel("k - number of clusters")
plt.ylabel("Silhouette score")
plt.title("Silhouette score vs k")
plt.show()
```

Output:

Analysis using silhouette score

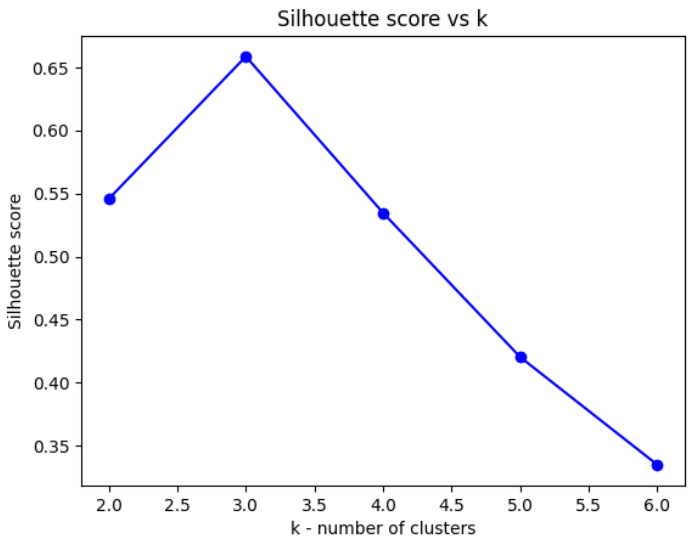


Figure 45: Plot silhouette score Vs k (silhouette score)

- Silhouette score tries to compute how close a point is to its own cluster vs other clusters. it ranges from -1 to 1
- A high score indicates that the data points in a cluster are well seperated from other clusters.
- From the Silhouette score vs k plot we can see that for k = 3 the score is maximum hence optimal value of k = 3

From the points mentioned above we can conclude optimal value of K = 3