Georgia Institute of Technology

ECE 4252/8803: Fundamentals of Machine Learning (FunML)

Spring 2025

Homework Assignment # 4

Due: February 28, 2025 @8PM

Please read the following instructions carefully.

- The entire homework assignment is to be completed on this `ipython` notebook. It is designed to be used with `Google Colab`, but you may use other tools (e.g., Jupyter Lab) as well.
- Make sure that you execute all cells in a way so their output is printed beneath the corresponding cell. Thus, after successfully executing all cells properly, the resulting notebook has all the questions and your answers.
- Make sure you delete any scratch cells before you export this document as a PDF. Do not change the order of the questions and do not remove any part of the questions. Edit at the indicated places only.
- Print a PDF copy of the notebook with all its outputs printed. **Submit PDF on `Gradescope`.** Then, zip both **PDF** and **IPYNB** in a single **ZIP** file and submit it on `Canvas` under Assignments.
- Rename the PDF, IPYNB and ZIP file according to the format:
  LastName_FirstName_ECE_4252_8803_F24_assignment_4.zip
  LastName_FirstName_ECE_4252_8803_F24_assignment_4.pdf
  LastName_FirstName_ECE_4252_8803_F24_assignment_4.ipynb
- When submitting PDF on Gradescope, make sure to match each question to the corresponding pages. **Incorrect page assignment may lead to reduction of points.**
- It is encouraged for you to discuss homework problems amongst each other, but any copying is strictly prohibited and will be subject to Georgia Tech Honor Code.
- Late homework is not accepted unless arranged otherwise and in advance.
- Comment on your codes.

- Refer to the tutorial and the supplementary/reading materials that are posted on `Canvas` for the first lecture to help you with this assignment.
- **IMPORTANT:** Start your solution with a **BOLD RED** text that includes the words *solution* and the part of the problem you are working on. For example, start your solution for Part (c) of Problem 2 by having the first line as:
  Solution to Problem 2 Part (c). Failing to do so may result in a *20% penalty* of the total grade.

Assignment Objectives:

- Understand the intuition behind various clustering algorithms discussed in class
- Connect concepts related to different clustering algorithms
- Implement and evaluate clustering techniques
- Implement clustering algorithms on real world datasets

Guide for Exporting Ipython Notebook to PDF:

Remeber to convert your homework into PDF format before submitting it.
Here is a video summarizing how to export ipython Notebook into PDF.

- **[Method 1: Print to PDF]**
  After you run every cell and get their outputs, you can use **[File] -> [Print Preview]** , and then use your browser's print function. Choose **[Save as PDF]** to export this Ipython Notebook to PDF for submission.
  Note: Sometimes figures or texts are splited into different pages. Try to tweak the layout by adding empty lines to avoid this effect as much as you can.
- **[Method 2: GoFullPage Chrome Extension]**
  Install the extension and generate PDF file of the Ipython Notebook in the browser.

Note: Since we have embedded images in HW1, it's recommended to generate PDF using the first method. Also, Georgia Tech provides a student discount for Adobe Acrobat subscription. Further information can be found here.

Problem 1: K-Means and Gaussian Mixture Models (GMMs) on a Toy Example (20pts)

Suppose we are given a dataset with 7 training examples, each with three features as shown below.

Datapoint ($\mathbf{x}_i$)	Feature 1 Value ($\mathbf{x}_{i1}$)	Feature 2 Value ($\mathbf{x}_{i2}$)	Feature 3 Value ($\mathbf{x}_{i3}$)
1	1	0	1.5
2	1.25	2	0.25
3	0.25	3	4
4	2.5	1.5	0.5
5	0.75	3	1.75
6	0.5	0.5	0.5
7	3	0	2

In this problem, you will be running the K-means and the GMM clustering algorithms on this dataset step by step to gain an insight into the algorithms.

(a)

In this part, you will run the K-means clustering algorithm on the data above for two iterations. Fill out the tables below for each iteration.

Follow the steps highlighted in Lecture 11 (19-Feb-2024) Page 9 of the PDF where five steps are listed. The difference is that you do not have a convergence criterion but instead you will stop after the second iteration. Use $k = 2$ clusters. Initialize the centroids using the following mean values, $\mathbf{u}_1 = [0, 0, 1]^T$, $\mathbf{u}_2 = [1, 3, 0]^T$, and $\mathbf{u}_3 = [3, 2, 4]^T$ respectively. You may do the intermediate calculations on scratch paper using either a calculator or a computer program, but do your best to understand every step.

Write down the values for the distances of each of the datapoint from the means in each iteration and the resulting cluster assingments in the tables, respectively.

The point distances are calculated using the following formula:

$$\|\mathbf{x}_i - \mathbf{u}_k\|_2^2$$

The cluster assingments are obtained as below:

$$\underset{k}{\operatorname{argmin}} \|\mathbf{x}_i - \mathbf{u}_k\|_2^2$$

Provide also the means after each iteration. (all numbers round to at least 4 decimals.)

(b)

With the dataset above, you will use GMM in this part to determine the clustering assignment.

Use $K = 3$ clusters. Use the same initializations for the means, $\mathbf{u}_1 = [0, 0, 1]^T$, $\mathbf{u}_2 = [1, 3, 0]^T$, and $\mathbf{u}_3 = [3, 2, 4]^T$ respectively. Run 2 iterations of the GMM algorithm. Assume the initial priors to be equal, i.e., $p(\mathbf{u}_k, \Sigma_k) = 1/3$. Assume that the initial covariance matrices Σ_k to be 3×3 identity matrices. Refer to Lecture 12 (21-Feb-2024).

Fill in the table below after for each iteration. You may do the intermediate calculations on scratch paper using either a calculator or a computer program, but please show how you get the numbers.

Provide the class assignments, in addition to the means and covariance matrices for the two mixtures in each iteration.

The posterior for the datapoint $\mathbf{x}_i$ is obtained using the following formula:

$$p(\mathbf{u}_k, \Sigma_k \,|\mathbf{x}_i) = \frac{p(\mathbf{u}_k, \Sigma_k) \times \mathcal{N}(\mathbf{x}_i; \mathbf{u}_k, \Sigma_k)}{\sum_{k=1}^{K} p(\mathbf{u}_k, \Sigma_k) \times \mathcal{N}(\mathbf{x}_i; \mathbf{u}_k, \Sigma_k)}$$

where $p(\mathbf{u}_k, \Sigma_k)$ is the prior for k-th mixture, $\mathcal{N}(\mathbf{x}_i; \mathbf{u}_k, \Sigma_k)$ is the multivariate normal distribution given as following:

$$\mathcal{N}(\mathbf{x}_i; \mathbf{u}_k, \Sigma_k) = \frac{1}{\sqrt{(2\pi)^3 |\Sigma_k|}} \exp(-\frac{1}{2}(\mathbf{x}_i - \mathbf{u}_k)^T \Sigma_k^{-1} (\mathbf{x}_i - \mathbf{u}_k))$$

Note: $|\Sigma_k|$ refers to the determinant of covariance matrix, not the absolute value.

(all numbers round to at least 4 decimals.)

Problem 1 (a) Solution

First iteration:

Point Distances $||\mathbf{x}_i - \mathbf{u}_k||_2^2$

Datapoint ($\mathbf{x}_i$)	Distance from $\mathbf{u}_1$	Distance from $\mathbf{u}_2$	Distance from $\mathbf{u}_3$
1	1.25	11.25	14.25
2	6.125	1.125	17.125
3	18.0625	16.5625	8.5625
4	6.75	6.5	9.25
5	10.3125	3.0625	13.0625
6	.75	7.25	20.25
7	10	13	8

Class Assignments $\underset{k}{\operatorname{argmin}} ||\mathbf{x}_i - \mathbf{u}_k||_2^2$

Datapoint ($\mathbf{x}_i$)	Cluster Assignment (1 or 2 or 3
1	1
2	2
3	3
4	2
5	2
6	1
7	3

Second iteration:

Point Distances $||\mathbf{x}_i - \mathbf{u}_k||_2^2$

Class Assignments $\underset{k}{\operatorname{argmin}} ||\mathbf{x}_i - \mathbf{u}_k||_2^2$

Datapoint ($\mathbf{x}_i$)	Distance from $\mathbf{u}_1$	Distance from $\mathbf{u}_2$	Distance from $\mathbf{u}_3$
1	.375	5.3888	4.8906
2	3.875	.4306	7.9531
3	16.8125	12.2847	5.1406
4	4.875	1.5556	7.016
5	8.125	2.0972	4.5781
6	.375	3.8889	85.16
7	6.125	8.3055	5.1406

Datapoint ($\mathbf{x}_i$)	Cluster Assignment (1 or 2 or 3
1	1
2	2
3	3
4	2
5	2
6	1
7	3

Problem 1 (b) Solution

```
import numpy as np
from scipy.stats import multivariate_normal

# Set NumPy display options for cleaner output
np.set_printoptions(precision=4, suppress=True)

# Define dataset: 7 samples with 3 features each
data = np.array([
    [1, 0, 1.5],
    [1.25, 2, 0.25],
    [0.25, 3, 4],
    [2.5, 1.5, 0.5],
    [0.75, 3, 1.75],
    [0.5, 0.5, 0.5],
    [3, 0, 2]
])

# Define number of clusters
num_clusters = 3

# Initialize cluster parameters
# Initial means for each cluster
initial_means = np.array([
    [0, 0, 1],     # Mean for cluster 1
    [1, 3, 0],     # Mean for cluster 2
    [3, 2, 4]      # Mean for cluster 3
])

# Initial covariance matrix (identity matrix)
initial_cov = np.identity(3)

# Initial mixture weights (equal probability for each cluster)
initial_weights = np.ones(num_clusters) / num_clusters

# Helper function to compute multivariate Gaussian density
def compute_gaussian_density(x, mu, sigma):
```

```
        return multivariate_normal.pdf(x, mean=mu, cov=sigma)

    # Gaussian Mixture Model with Expectation-Maximization
    def fit_gmm(data, means, covariance, weights, max_iterations=2):
        n_samples = data.shape[0]

        for iter_num in range(max_iterations):
            # E-step: Calculate posterior probabilities
            responsibilities = np.zeros((n_samples, num_clusters))

            for i in range(n_samples):
                # Calculate unnormalized probabilities
                for k in range(num_clusters):
                    responsibilities[i, k] = weights[k] * compute_gaussian_density(data[i], mear

                # Normalize to get posterior probabilities
                responsibilities[i, :] /= np.sum(responsibilities[i, :])

            # M-step: Update parameters
            # Update means
            for k in range(num_clusters):
                numerator = np.sum(responsibilities[:, k].reshape(-1, 1) * data, axis=0)
                denominator = np.sum(responsibilities[:, k])
                means[k] = numerator / denominator

            # Update covariance (simplified to diagonal covariance)
            for k in range(num_clusters):
                deviations = data - means[k]
                weighted_cov = np.dot(responsibilities[:, k] * deviations.T, deviations) / np.su
                covariance = np.diag(np.diag(weighted_cov))  # Extract diagonal elements

            # Update mixture weights
            weights = np.mean(responsibilities, axis=0)

            # Print results for this iteration
            print(f"Iteration {iter_num+1}:")
            print(f"Posteriors: \n{responsibilities}")
            print(f"Updated Means: \n{means}")
            print(f"Updated Priors: {weights}")
            print(f"Updated Covariance: \n{covariance}\n")

        return means, covariance, weights, responsibilities

    # Run the GMM algorithm
    final_means, final_cov, final_weights, final_resp = fit_gmm(
        data, initial_means, initial_cov, initial_weights, max_iterations=2
    )
```

```
Iteration 1:
Posteriors:
[[0.9918 0.0067 0.0015]
 [0.0758 0.9239 0.0003]
```

```
 [0.0084 0.0178 0.9737]
 [0.1173 0.8668 0.0159]
 [0.0288 0.9537 0.0175]
 [0.9525 0.0474 0.    ]
 [0.2668 0.0081 0.7252]]
Updated Means:
[[1 0 1]
 [1 2 0]
 [1 1 3]]
Updated Priors: [0.3488 0.4035 0.2477]
Updated Covariance:
[[2.0098 0.     0.    ]
 [0.     2.7079 0.    ]
 [0.     0.     1.0561]]

Iteration 2:
Posteriors:
[[0.6928 0.1485 0.1587]
 [0.2431 0.7459 0.0109]
 [0.0125 0.0023 0.9852]
 [0.3658 0.6122 0.0219]
 [0.2728 0.4233 0.3038]
 [0.5432 0.4343 0.0225]
 [0.58   0.0775 0.3425]]
Updated Means:
[[1 0 1]
 [1 1 0]
 [0 2 2]]
Updated Priors: [0.3872 0.3492 0.2637]
Updated Covariance:
[[1.9687 0.     0.    ]
 [0.     1.8151 0.    ]
 [0.     0.     2.2393]]
```

First iteration:

Point Posteriors $p(\mathbf{u}_k, \Sigma_k \,|\mathbf{x}_i)$

Datapoint ($\mathbf{x}_i$)	Posterior Mixture 1	Posterior Mixture 2	Posterior Mixture 3
1	0.9918	0.0067	0.0015
2	0.0758	0.9239	0.0003
3	0.0084	0.0178	0.9737
4	0.1173	0.8668	0.0159
5	0.0288	0.9537	0.0175
6	0.9525	0.0474	0.0000
7	0.2668	0.0081	0.7252

Cluster Assignment $\underset{k}{\operatorname{argmax}}\, p(\mathbf{u}_k, \Sigma_k \,|$

Datapoint ($\mathbf{x}_i$)	Cluster Assignment (1 or
1	1
2	2
3	3
4	2
5	2
6	1
7	3

Second iteration:

Point Posteriors $p(\mathbf{u}_k, \Sigma_k \,|\mathbf{x}_i)$

Datapoint ($\mathbf{x}_i$)	Posterior Mixture 1	Posterior Mixture 2	Posterior Mixture 3
1	0.6928	0.1485	0.1587
2	0.2431	0.7459	0.0109
3	0.0125	0.0023	0.9852
4	0.3658	0.6122	0.0219
5	0.2728	0.4233	0.3038
6	0.5432	0.4343	0.0225
7	0.5800	0.0775	0.3425

Cluster Assignment $\underset{k}{\operatorname{argmax}}\, p(\mathbf{u}_k, \Sigma_k \,|$

Datapoint ($\mathbf{x}_i$)	Cluster Assignment (1 or
1	1
2	2
3	3
4	2
5	2
6	1
7	1

Problem 2: K-Means vs GMMs for Modeling Non-Spherical Distributions (20pts)

K-means is good when finding clusters of data sampled from gaussian distributions with zero correlation (and ideally equal variances in all feature directions). In cases where this is not true (i.e., gaussian distributions have correlated features and/or unequal variances), GMMs tend to perform superior to K-means, as you will hopefully see in this question.

As seen in Problem 1 above, running both K-means and GMMs requires the setup of an $N \times K$ dimensional table for each iteration, storing the point distances to the indiviudal cluster means for the former and point posteriors for the latter. N refers to the number of training examples while K is the number of clusters. The **un-normalized** posterior for the i-th datapoint having mean and covariance $\mathbf{u}_k$ and Σ_k is given as:

$$p(\mathbf{u}_k, \Sigma_k \,|\mathbf{x}_i) = \overbrace{p(\mathbf{u}_k, \Sigma_k)}^{\text{prior for mixture } k} \times \overbrace{p(\mathbf{x}_i|\mathbf{u}_k, \Sigma_k)}^{\text{likelihood for } \mathbf{x}_i \text{ given mxiture } k} \tag{4}$$

$$p(\mathbf{u}_k, \Sigma_k \,|\mathbf{x}_i) = p(\mathbf{u}_k, \Sigma_k) \times \mathcal{N}(\mathbf{x}_i; \mathbf{u}_k, \Sigma_k), \tag{5}$$

where $\mathcal{N}(\mathbf{x}_i; \mathbf{u}_k, \Sigma_k)$ is the multivariate normal distribution characterized by mean $\mathbf{u}_k$ and covariance Σ_k and

$$\mathcal{N}(\mathbf{x}_i; \mathbf{u}_k, \Sigma_k) = \frac{1}{\sqrt{(2\pi)^P |\Sigma|}} \exp(-\frac{1}{2}(\mathbf{x} - \mathbf{u})^T \Sigma^{-1} (\mathbf{x} - \mathbf{u})) \tag{6}$$

(a)

If we want to normalize Equation 4, what is the formula of denominator? After normalization, what is the range for normalized posterior?

(b)

As we have seen in Regression, it is often more convenient to compute these values in the natural log scale. In this part, take the log of both sides of Equation 4. Write down the resulting equation below.

(c)

Plug Equation 6 into the equation you derived in part (b). Simplify and write down a fully expanded expression for $\log p(\mathbf{u}_k, \Sigma_k \,|\mathbf{x}_i)$.

(d)

Considering the expression in part (c), it is possible to simplify the expression to represent K-means. In other words, K-means can be a special case of GMM. Explain exactly under what constraints and changes, if any, on the prior, means, covariance matrices, and the update process does this statement become true, i.e., K-means is a sepcial case of GMM.

(e)

The code below generates some data sampled from two $2-D$ gaussian distributions with different means and covariance matrices.

Using the `sklearn.cluster.KMeans` class for K-means and `sklearn.mixture.GaussianMixture` class for GMMs, set up a class object for each to run for 2 clusters on this data. Describe the results. Which one of K-means and GMMs captures the original distribution better? How do you relate this to what you explained in part (d)?

(f)

The code tagged with the text cell Visualization of Cluster with Centers contains usage of several built-in library functions called from the Kmeans import of sklearn like centroids , y_kmeans.transform etc.

Study the code cell properly as well as the output that it provides, and then use these functions (and your understanding) to find and remove the furthest outliers from both clusters until around 25% (~225 to 250) of the furthest datapoints, from the centroids, are removed and then re-cluster the new distribution using Kmeans and plot the new centroids (same visualization as code cell provided).

Note: To discount the effect of random initialization, you might have to execute the cell multiple times to get a consistent idea of the results.

Problem 2 (a) Solution

Denominator: $\sum_{k=1}^{K} p(\mathbf{u}_k, \Sigma_k) \times p(\mathbf{x}_i|\mathbf{u}_k, \Sigma_k)$ or $\sum_{k=1}^{K} p(\mathbf{u}_k, \Sigma_k)\mathcal{N}(\mathbf{x}_i; \mathbf{u}_k, \Sigma_k)$
Range: From 0 to 1

Problem 2 (b) Solution

$$\log(p(\mathbf{u}_k, \Sigma_k \mid \mathbf{x}_i)) = \log(p(\mathbf{u}_k, \Sigma_k) \times \mathcal{N}(\mathbf{x}_i; \mathbf{u}_k, \Sigma_k))$$

Problem 2 (c) Solution

$$\log p(u_k, \Sigma_k|x_i) = logp(u_k, \Sigma_k) - \frac{P}{2}log(2\pi) - \frac{1}{2}log|\Sigma_k| - \frac{1}{2}(x_i - u_k)^T\Sigma_k^{-1}(xi - u_k)$$

Problem 2 (d) Solution

Since the prior probabilities are considered equal for all clusters, the term $\log(p(\mathbf{u}_k, \Sigma_k))$ becomes constant and can be eliminated from the equation. The constant term $-\frac{P}{2}\log(2\pi)$ remains the same across all data points, making it irrelevant for k-means clustering. For a spherical distribution with no correlation between dimensions, the covariance matrices should be identity matrices, which causes the term $-\frac{1}{2}\log(|\Sigma_k|)$ to vanish. When using identity covariance matrices, the expression $\Sigma_k^{-1}(\mathbf{x}_i - \mathbf{u}_k)$ simplifies to the squared Euclidean distance between points and cluster centers, represented as $(\mathbf{x}_i - \mathbf{u}_k)^T(\mathbf{x}_i - \mathbf{u}_k)$ or $||\mathbf{x_i} - \mathbf{u_k}||_2^2$. This leaves only the distance term, which matches exactly with the objective function used in standard k-means clustering.

Problem 2 (e) Solution

K-means divides the dataset into two clusters using linear boundaries, as illustrated in the left figure. It imposes piecewise linear separations between clusters, which doesn't adequately represent the natural structure of the data.

GMM provides greater flexibility through its probabilistic approach. The right figure demonstrates how GMM more accurately captures the underlying distribution by adjusting to the actual dispersion of the data points.

This observation aligns with our analysis in part (d), which established that k-means represents a restricted version of GMM where all clusters are constrained to have identical variance and shape.

GMM proves superior in this scenario because it naturally accommodates varying distributions within the dataset.

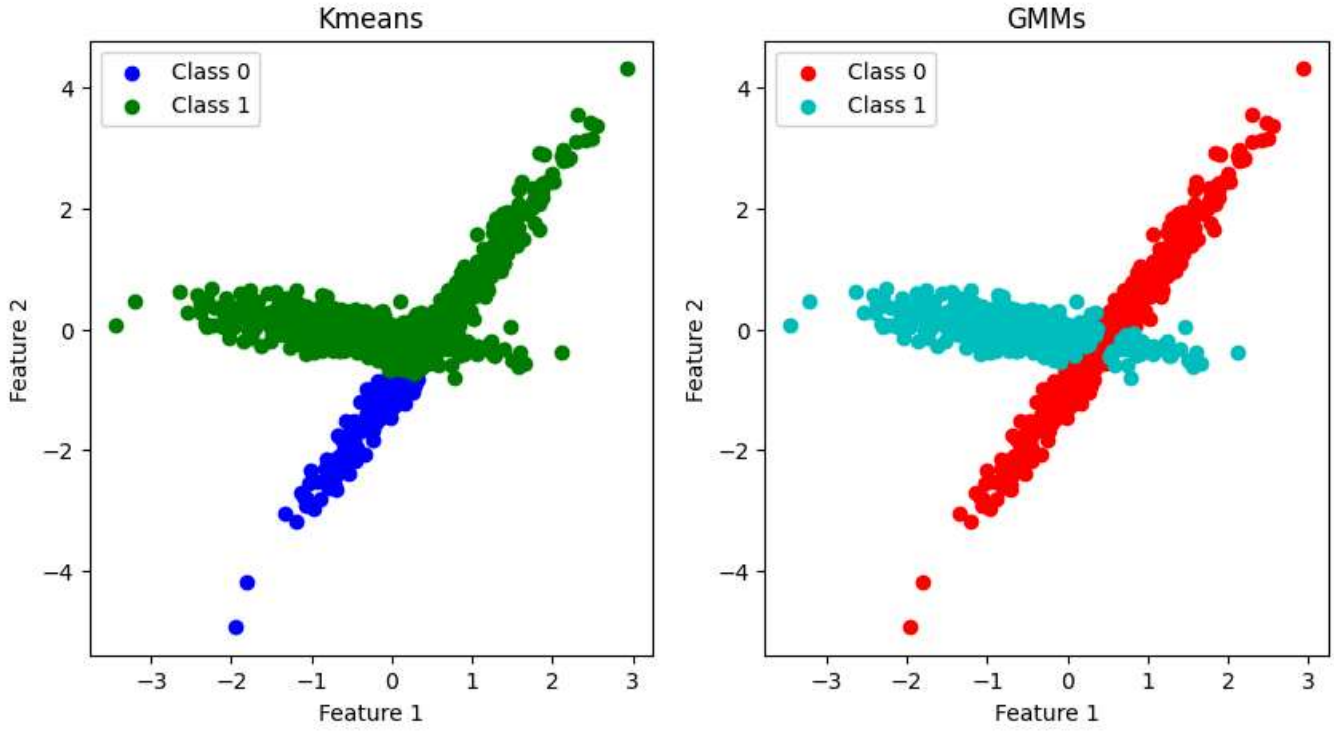
```
#### Do not change this cell. This a helper cell. Please execute it.

# imports and utlity functions
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import torch
from itertools import cycle
from sklearn import metrics
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.datasets import load_iris, load_wine
import random
from skimage import data, color


# generate colors for clustering via python generator
colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k']
color_generator = cycle(colors)

# helper functions

# evaluation metrics

# Dunn Index
# from the resource: https://en.wikipedia.org/wiki/Cluster_analysis
# the distance between two clusters can be any of the measurements, i.e. distance between ce

def delta(ck, cl):
    values = np.ones([len(ck), len(cl)]) * np.finfo(np.float32).max
    for i in range(len(ck)):
        for j in range(len(cl)):
            values[i, j] = np.linalg.norm(ck[i] - cl[j])

    return np.min(values)


def big_delta(ci):
    values = np.zeros([len(ci), len(ci)])

    for i in range(0, len(ci)):
        for j in range(0, len(ci)):
            values[i, j] = np.linalg.norm(ci[i] - ci[j])

    return np.max(values)

# Dunn Index
def dunn_index(X, cluster_labels):
```

```
        # A list containing a numpy array for each cluster
        # k_list[k] is np.array([N, p]) (N : number of samples in cluster k, p : sample dimensic
        k_list = []
        for k in np.unique(cluster_labels):
            k_list.append(X[cluster_labels == k])

        deltas = np.ones([len(k_list), len(k_list)]) * np.finfo(np.float32).max
        big_deltas = np.zeros([len(k_list), 1])
        l_range = list(range(0, len(k_list)))

        for k in l_range:
            for l in (l_range[0:k] + l_range[k + 1:]):
                deltas[k, l] = delta(k_list[k], k_list[l])

            big_deltas[k] = big_delta(k_list[k])

        di = np.min(deltas) / np.max(big_deltas)

        return di


    def delta_fast(ck, cl, distances):
        values = distances[np.where(ck)][:, np.where(cl)]
        values = values[np.nonzero(values)]

        return np.min(values)


    def big_delta_fast(ci, distances):
        values = distances[np.where(ci)][:, np.where(ci)]

        return np.max(values)


    def dunn_index_fast(X, cluster_labels):
        """Dunn Index - fast(using sklearn pairwise euclidean_distance function
        X: np.array
        np.array([N,p] of all samples
        cluster_labels: np.array
        np.array([N,]) labels of all samples
        """
        distances = euclidean_distances(X)
        ks = np.sort(np.unique(cluster_labels))

        deltas = np.ones([len(ks), len(ks)]) * np.finfo(np.float32).max
        big_deltas = np.zeros([len(ks), 1])
        l_range = list((range(0, len(ks))))

        for k in l_range:
            for l in (l_range[0:k] + l_range[k+1:]):
                deltas[k, l] = delta_fast((cluster_labels==ks[k]), (cluster_labels==ks[l]), dist
```

```
        big_deltas[k] = big_delta_fast(cluster_labels == ks[k], distances)

    di = np.min(deltas) / np.max(big_deltas)
    return di


# Cluster Purity
def purity_score(labels_true, labels_pred):
    # compute contingency matrix
    contingency_matrix = metrics.cluster.contingency_matrix(labels_true, labels_pred)

    return np.sum(np.amax(contingency_matrix, axis=0)) / np.sum(contingency_matrix)


# for Problem 2 (e)

from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture

mu_1 = np.array([0,0])
mu_2 = np.array([0,0])
cov1 = np.array([[1.95, 1],[1, 0.5]])
theta = np.radians(30)
c, s = np.cos(theta), np.sin(theta)
R = np.array(((c, -s), (s, c)))
X_1 = np.random.multivariate_normal(mu_1, cov1, 500)
X_2 = X_1.dot(R) - np.array([2,0]).reshape(1,-1)
X = np.concatenate((X_1, X_2), axis=0)
X = (X - X.mean(axis=0)) / X.std(axis=0)

#-----------------Don't change anything above-----------------------#
model_kmeans = KMeans(n_clusters=2, random_state=42)
model_gmms = GaussianMixture(n_components=2, random_state=42)
y_pred_kmeans = model_kmeans.fit_predict(X)
y_pred_gmms = model_gmms.fit_predict(X)
#-----------------Don't change anything below-----------------------#

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))

for cluster_id, color in zip(range(2), color_generator):
  data_x = X[y_pred_kmeans==cluster_id,0]
  data_y = X[y_pred_kmeans==cluster_id,1]
  ax1.scatter(data_x, data_y, color = color, label='Class {}'.format(cluster_id))
  ax1.legend()
  ax1.set_xlabel('Feature 1')
  ax1.set_ylabel('Feature 2')
  ax1.set_title('Kmeans')


for cluster_id, c in zip(range(2), color_generator):
```

```
    data_x = X[y_pred_gmms==cluster_id,0]
    data_y = X[y_pred_gmms==cluster_id,1]
    ax2.scatter(data_x, data_y, color = c, label='Class {}'.format(cluster_id))
    ax2.legend()
    ax2.set_xlabel('Feature 1')
    ax2.set_ylabel('Feature 2')
    ax2.set_title('GMMs')


plt.show()
```

```
<ipython-input-3-109f5de95e85>:12: RuntimeWarning: covariance is not symmetric positive-
  X_1 = np.random.multivariate_normal(mu_1, cov1, 500)
```



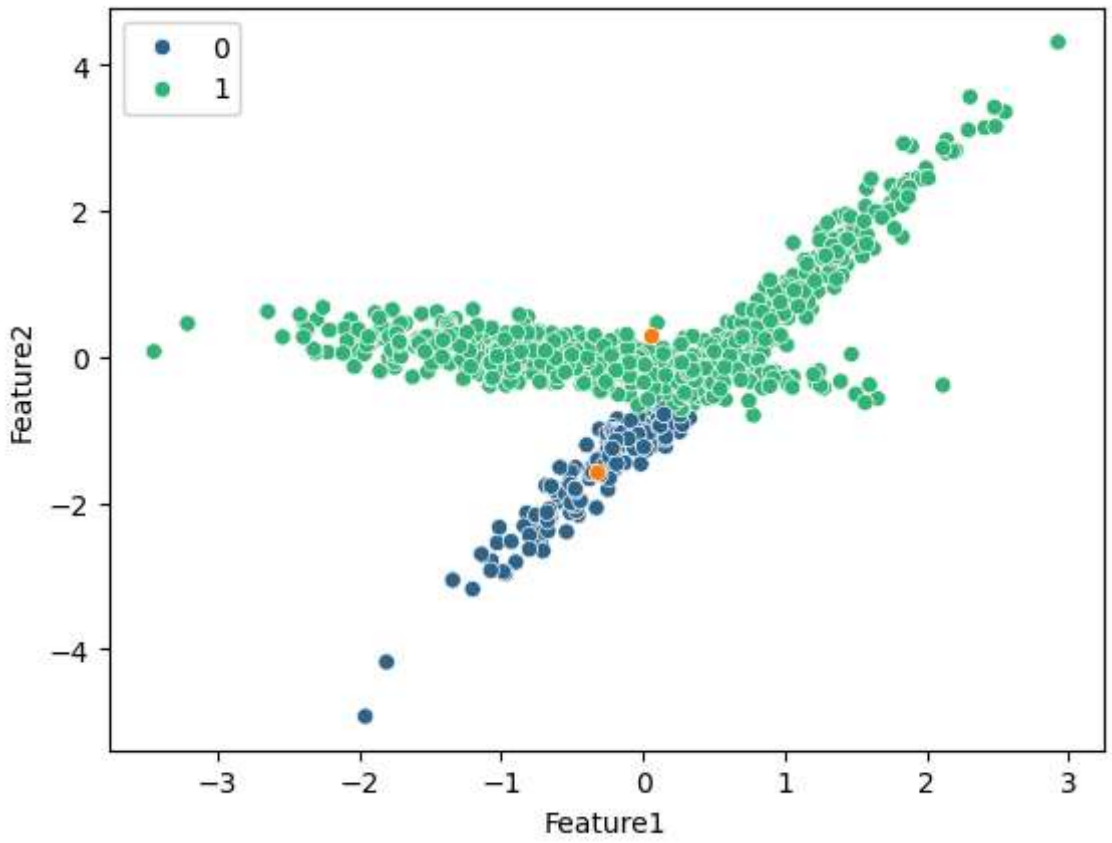
Problem 2 (f) Solution

Visualization of Cluster with Centers

```
import pandas as pd
import seaborn as sns
#### Do not change this cell and run it

#convert to pandas dataframe for easier visualization
X_df = pd.DataFrame(X,columns=['Feature1','Feature2'])
```

```
#fitting of the Kmeans model
y_kmeans = model_kmeans.fit(X)
#centroid location of cluster
centroids = y_kmeans.cluster_centers_
# squared distance to cluster center
X_dist = y_kmeans.transform(X)**2
#labels of each points
labels = y_kmeans.predict(X)
# do something useful to add label as an extra column to the dataframe for visualization
import pandas as pd
df1 = pd.DataFrame(X_dist.sum(axis=1).round(2), columns=['sqdist'])
df1['label'] = labels
#add cluster labels as another column
X_df['Clusters'] = y_kmeans.labels_
#plot the cluster with centroid
sns.scatterplot(x="Feature1", y="Feature2",hue = 'Clusters',  data=X_df,palette='viridis')
sns.scatterplot(x = centroids[:, 0],
    y = centroids[:, 1],)
```

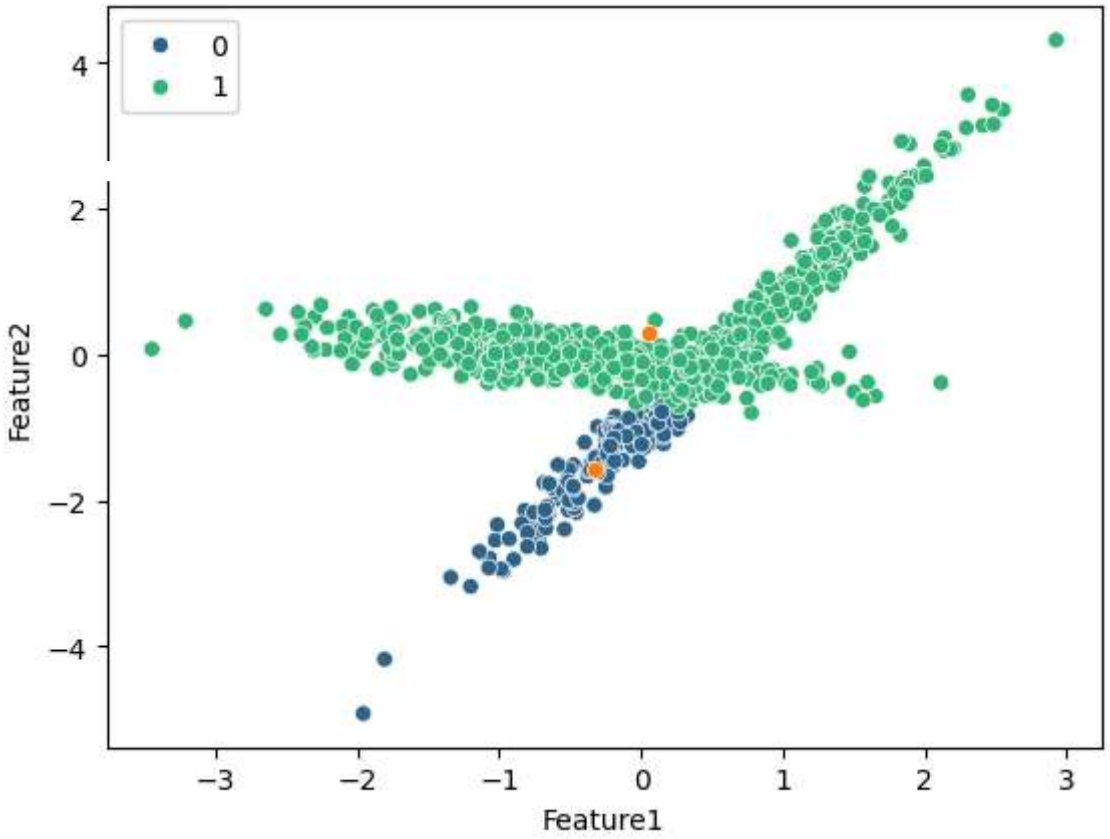
```
<Axes: xlabel='Feature1', ylabel='Feature2'>
```




```
##Run an iterative loop to remove (225-250) furthest outlier from each cluster and finally r
for _ in range(25):  # Iteratively eliminate approximately 25% of the dataset
   # Calculate the cumulative distance from each point to all centroids
   point_distances = X_dist.sum(axis=1)
```

```
    # Select the indices of the 225-250 most distant points
    farthest_point_indices = point_distances.argsort()[-250:]  # Get indices of the 250 most

    # Create a filtered dataset excluding the identified distant points
    X_filtered = np.delete(X, farthest_point_indices, axis=0)

    # Apply KMeans clustering to the refined dataset
    kmeans_algorithm = KMeans(n_clusters=2, random_state=42)
    cluster_assignments_filtered = kmeans_algorithm.fit_predict(X_filtered)

    # Extract the updated cluster centers
    cluster_centers = kmeans_algorithm.cluster_centers_

# Visualize the final clustering result after iterative outlier elimination
filtered_data_frame = pd.DataFrame(X_filtered, columns=['Feature1', 'Feature2'])
filtered_data_frame['Clusters'] = cluster_assignments_filtered
#-----------------Don't change anything below-----------------------#
sns.scatterplot(x="Feature1", y="Feature2",hue = 'Clusters',  data=X_df,palette='viridis')
sns.scatterplot(x = centroids[:, 0],
    y = centroids[:, 1],)
```

```
<Axes: xlabel='Feature1', ylabel='Feature2'>
```



Problem 3: Implementing K-Means (20pts)

In this problem, you are going to design a python class that implements the k-means algorithm. You are provided with a class template called `MyKmeans` that you have to fill in to implement various stages in the k-means workflow. Follow the steps outlined in the parts below to answer the question.

(a)

The initialization function in the class stores the training data (`X_train`), the number of clusters (`k`), and the number of iterations to run the k-means algorithm for (`num_iter`). The first step in the algorithm entails randomly selecting k data points in `X_train` to be the centroids, one for each cluster. Fill in code for the class function `__init_means()` below to do this. The output should be stored into the `self.means` variable to be used later in the fitting stage. Pay attention to the shape of this array.

(b)

Implement the `fit_predict()` function in the class definition by writing code to execute the Assignment and Update steps. For each iteration, the assignment step involves computing the euclidean distance of each data point in `X_train` to each of the k centroids selected in part(a) above. The result is essentially an $N \times K$ dimensional table where each column stores the euclidean distance of all data points to the centroid corresponding to the column. This is used to compute the cluster label for each datapoint by choosing the centroid (where centroid label $\in [1, K]$) corresponding to the smallest distance, resulting in a $N \times 1$ array.

(c)

Next, implement the Update step, where the $N \times 1$ label vector just created in part (b) is used to recompute the means for each of the k centroids, and thus update the `self.means` structure. Refer to Lecture 11 (19-Feb-2024) Page 9 for the details.

(d)

Execute the cell once you have completed all of the above to classify the `Iris` dataset you used in homework 1 using **two** preselected features. You may have to run the cell multiple times to discount the effect of random initialization and get consistent results.

Complete the code to plot the locations of the means of clusters. Plot the clustering results when using features 0 and 2.

Problem 3 (a)(b)(c) Solution

```
from sklearn.metrics import davies_bouldin_score, mutual_info_score, adjusted_rand_score

class MyKMeans:
  def __init__(self, X_train, k, num_iter=20):
```

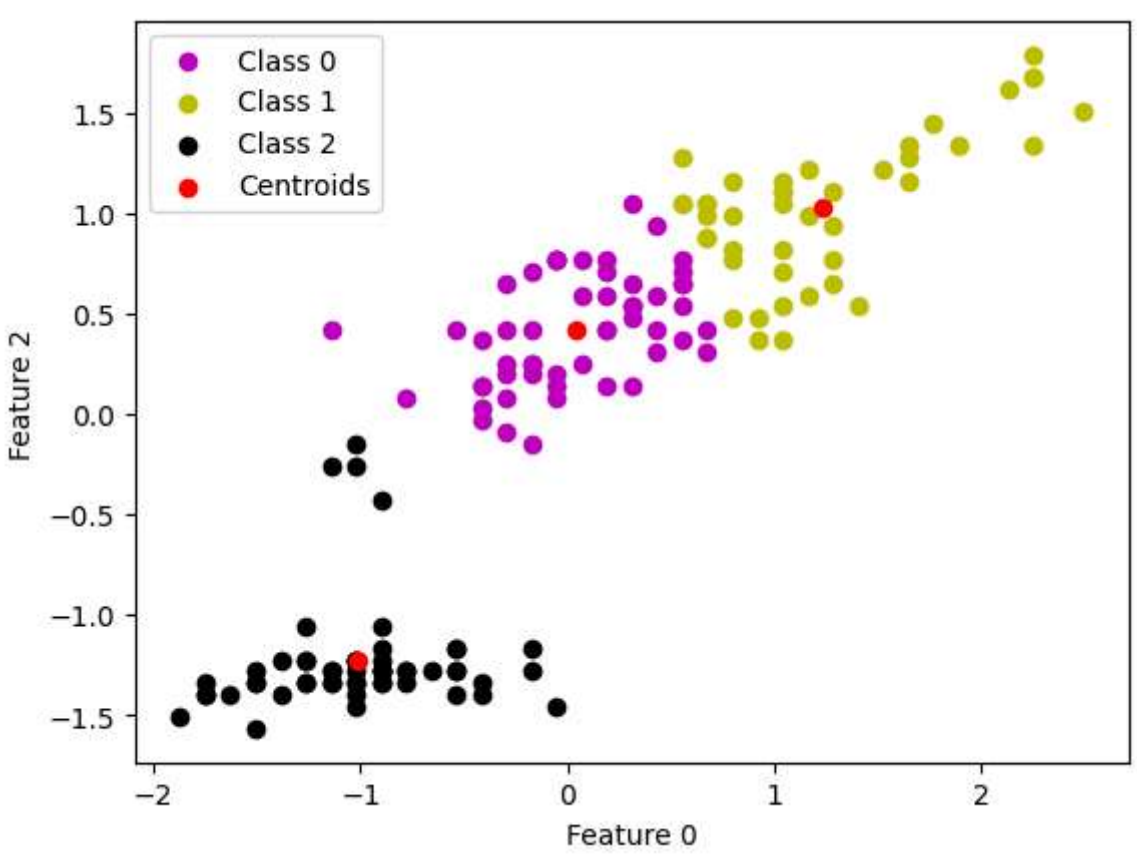
```
    """
    Parameters
    ----------
    X_train: ndarray of shape (number of samples, num of features).
      Training data array.

    k: int,
      number of clusters.

    num_iter:int
      number of steps to run algorithm for.
    """

    self.X_train = X_train
    self.k = k
    self.num_iter = num_iter

  def __init_means(self):
    """initialize means as an ndarray of shape (k, num of features)."""
    means = np.zeros((self.k, self.X_train.shape[1]))
    random = np.random.default_rng()
    means = random.choice(self.X_train, size=self.k, replace=False)
    self.means = means

  def fit_predict(self):
    """Runs the k means algorithm.

    Returns
    -------
    y_pred: ndarray of shape (num of samples, 1)
      array of predicted cluster labels for each data point.
    """

    self.__init_means()  # initialize means

    for iteration in range(self.num_iter):      # begin the algorithm

     # expectation phase from part (b)
     # Compute distances between each data point and every centroid
     point_centroid_distances = np.linalg.norm(self.X_train[:, np.newaxis] - self.means, axi
     # Determine cluster membership by finding nearest centroid
     y_pred = np.argmin(point_centroid_distances, axis=1)

     # maximization phase from part (c)
     updated_centroids = np.zeros_like(self.means)

     # Update each centroid based on the mean of its assigned points
     for cluster_idx in range(self.k):
         cluster_members = self.X_train[y_pred == cluster_idx]
         if cluster_members.size > 0:
             updated_centroids[cluster_idx] = np.mean(cluster_members, axis=0)
```

```
            else:
                # Handle empty clusters by selecting a random point
                updated_centroids[cluster_idx] = self.X_train[np.random.randint(self.X_train.sh

        # Evaluate if algorithm has converged
        if np.allclose(updated_centroids, self.means, atol=1e-4):
            break

        # Apply the updated centroids
        self.means = updated_centroids

    return y_pred
```

Problem 3 (d) Solution

```
# k means parameters
k = 3
num_iter = 20
feature_nums = [0,2]  # features to use

# load and preprocess the dataset
dataset = load_iris()
X, y = dataset.data, dataset.target
X = (X - X.mean(axis=0)) / X.std(axis=0)
X = X[:,feature_nums]

# fit the model
kmeans = MyKMeans(X, k=k, num_iter=num_iter)
y_pred = kmeans.fit_predict()


# plot data
fig, ax = plt.subplots()

for cluster_id in range(k):
  data_x = X[y_pred==cluster_id,0]
  data_y = X[y_pred==cluster_id,1]
  ax.scatter(data_x, data_y, color = next(color_generator), label='Class {}'.format(cluster_
  ax.legend()
#----------------Don't change anything above----------------------#

ax.scatter(kmeans.means[:,0], kmeans.means[:,1], color = 'red', label='Centroids')
ax.legend()
#----------------Don't change anything below----------------------#
plt.xlabel('Feature {}'.format(feature_nums[0]))
plt.ylabel('Feature {}'.format(feature_nums[1]))
plt.show()

print('Davies Bouldin Score: {:0.4f}'.format(davies_bouldin_score(X, y_pred)))
```

```
print('Dunn Index: {:0.4f}'.format(dunn_index(X, y_pred)))
print('Mutual Information Score: {:0.4f}'.format(mutual_info_score(y, y_pred)))
print('Rand Index: {:0.4f}'.format(adjusted_rand_score(y, y_pred)))
print('Purity Score: {:0.4f}'.format(purity_score(y, y_pred)))
```




```
Davies Bouldin Score: 0.6373
Dunn Index: 0.0672
Mutual Information Score: 0.6401
Rand Index: 0.5634
Purity Score: 0.8067
```

Problem 4: Implementing Gaussian Mixture Models (15pts)

In this problem, you will create a `MyGMMs` class that implements clustering using Gaussian Mixture Models (GMM). The class intialization function stores the training data array, `X_train`, the prespecified number of mixtures, `k`, and the number of iterations, `num_iteratons`. The means and covariances of the k-th cluster and the posteriors structure have already been initalized in the `__init_params()` function. Read every question very carefully before you start your solution.

(a)

Expectation Step: The analogue of the $N \times K$ dimesnional cluster distances table created in Problem 1 (b) above is the posteriors structure in GMMs. The entries γ_{ik} in this table store the (unnormalized) posterior probabilities of the parameters of the k-th mixture given a datapoint $\mathbf{x}_i$

(i.e., $\gamma_{ik} = p(\mathbf{u}_k, \Sigma_k \,|\mathbf{x}_i)$). Using the expression you derived in problem 2 (b), fill in the `fit_predict()` function below to compute the expectation step. We work with natural logs because they are numerically easier to deal with from the computer's point of view. Remember also that since you are working with logs, so you need to take the exponent of the final value and normalize before storing it as a posterior.

(b)

Maximization Step: Having completed the expectation step in the E-M algorithm you studied in class, you now have a complete $N \times K$ dimensional table of posterior values. We now move on to the maximization step where we are required to update the priors ($p(\mathbf{u}_k, \Sigma_k)$), the means ($\mathbf{u}_k$), and the covariances ($\Sigma_k$) for each of the K clusters.

(i) Using Lecture 12 (21-Feb-2024) as a reference, write down the expression for the mean of the k-th mixture in terms of γ_{ik} we computed in (a).

(ii) Again using Lecture 12 as the reference, write down the expression for the covariance of the k-th mixture in terms of γ_{ik} and $\mathbf{u}_k$ we computed above.

(iii) Once again using Lecture 12 as a reference, write down the expression for the new prior of the j-th mixture in terms of γ_{ik}.

(iv) Code steps (i - iii) into the `fit_predict()` function in the class template below. This completes the maximization step of the iteration.

Further, complete the code to calculate the assignments using the latest posterior table. Using these assignments, visualize the clustering at each iteration by setting the parameter visualize_learning accordingly. Do not forget to update the `self.parameter` and `self.priors` variables before the function returns the labels.

Execute the cell multiple times until you are able to get an idea of what the consistent results look like.

Plot the clustering results when using features 0 and 2.

Practical Tips:

1. The priors in equation 1 can underflow to zero in the log expression, resulting in nans. You may want to add a small value e.g., $1e-4$ to compensate for that.
2. In case the covariance matrix turns out to be singular or badly conditioned, taking the inverse will result in an error. You may want to use `numpy.linalg.pinv` rather than `numpy.linalg.inv`. This will calcualte the pseudo inverse.
3. You can improve the conditioning of the covariance matrices in the maximization step by adding an identity matrix scaled by a small number e.g., 1e-4.

Problem 4 (a) Solution

Answered in the code cell below

Problem 4 (b)(i) Solution

$$\mu_k = \frac{\Sigma_{i=1}^{N} \gamma_i^k x_i}{\Sigma_{i=1}^{N} \gamma_i^k}$$

Problem 4 (b)(ii) Solution

$$\Sigma_k^{t+1} = \frac{\Sigma_{i=1}^{N} (\gamma_i^k)^{(t)} (x_i - \mu_k^{(t+1)})(x_i - \mu_k^{(t+1)})^T}{\Sigma_{i=1}^{N} (\gamma_i^k)^{(t)}}$$

Problem 4 (b)(iii) Solution

$$\pi_k^{t+1} = \frac{1}{N} \Sigma_{i=1}^{N} (\gamma_i^k)^{(t)}$$

Problem 4 (b)(iv) Solution

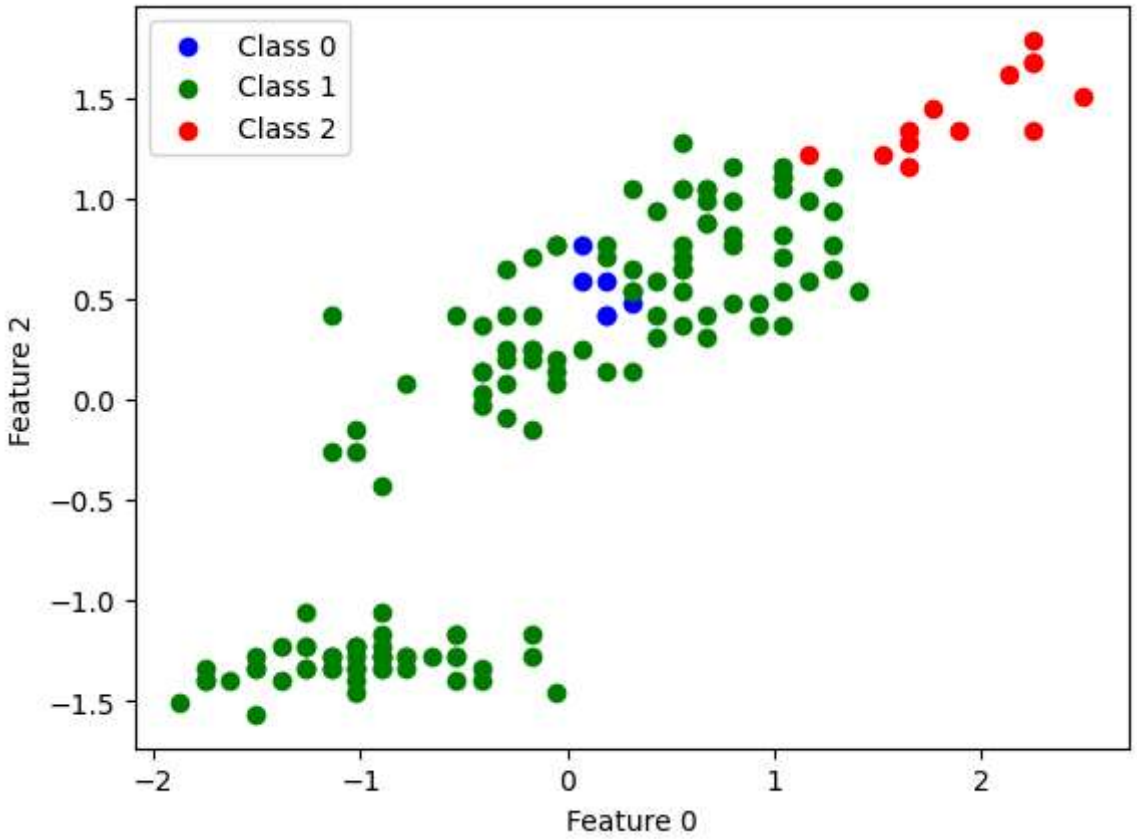
```
# for problem 4
from sklearn.metrics import davies_bouldin_score, mutual_info_score, adjusted_rand_score
from scipy.stats import multivariate_normal

# plot data
def my_plot_data(X, y_pred, feature_nums):
  fig, ax = plt.subplots()

  for cluster_id in range(k):
    data_x = X[y_pred==cluster_id,0]
    data_y = X[y_pred==cluster_id,1]
    ax.scatter(data_x, data_y, color = next(color_generator), label='Class {}'.format(cluste
    ax.legend()

  plt.xlabel('Feature {}'.format(feature_nums[0]))
  plt.ylabel('Feature {}'.format(feature_nums[1]))
  plt.show()

  print('Davies Bouldin Score: {:0.4f}'.format(davies_bouldin_score(X, y_pred)))
  print('Dunn Index: {:0.4f}'.format(dunn_index(X, y_pred)))
  print('Mutual Information Score: {:0.4f}'.format(mutual_info_score(y, y_pred)))
```

```
    print('Rand Index: {:0.4f}'.format(adjusted_rand_score(y, y_pred)))
    print('Purity Score: {:0.4f}'.format(purity_score(y, y_pred)))


class MyGMMs:
    def __init__(self, X_train, k, num_iter=20, visualize_learning=False):
        """
        Parameters
        ----------
        X_train: ndarray of shape (number of samples, num of features).
            Training data array.

        k: int,
            number of clusters.

        num_iter: int
            number of iteration to run E-M algorithm for.
        """

        self.X_train = X_train
        self.k = k
        self.num_iter = num_iter
        self.visualize_learning = visualize_learning
        self.__init_params()

    def __init_params(self):
        """Function initializes the means, covariances, and posteriors structure for
        the E-M algorithm"""


        # extract k and data matrices
        k = self.k
        X_train = self.X_train

        # Initialize priors as uniform
        self.priors = 1/k * np.ones((k,1))

        # intialize means and covariances
        self.parameters = [[] for i in range(k)]
        for i in range(k):
            self.parameters[i].append(np.random.randn(X_train.shape[1],1))  # initialize random
            temp = np.random.randn(X_train.shape[1],X_train.shape[1])
            self.parameters[i].append(temp.T.dot(temp)+1e-4*np.eye(X_train.shape[1]))  # initial

        # set up posterior structure
        self.posteriors = np.zeros((X_train.shape[0], k))

#-----------------Don't change anything above-----------------------#

    def fit_predict(self):
          """ Returns predicted cluster classes.
```

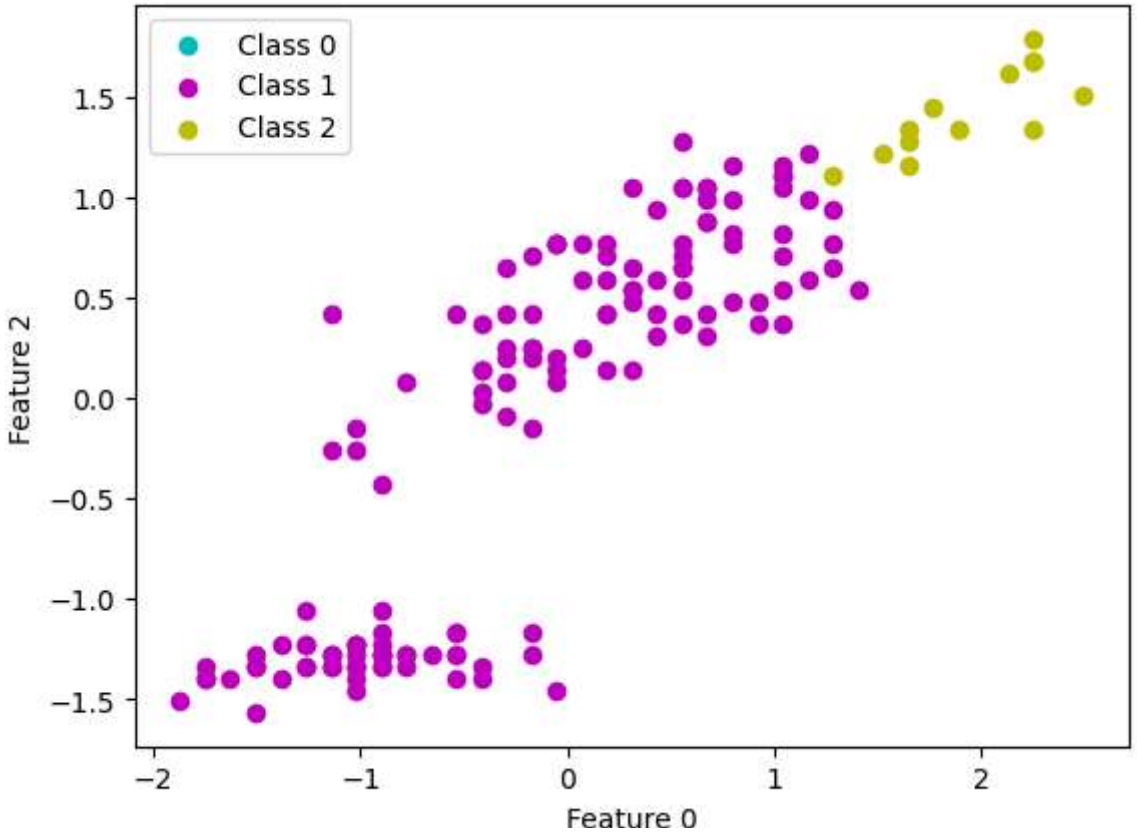
```
        Returns
        -------
        y_pred: ndarray of shape (number of samples, 1).
            Predicted classes for each data point in X_train.
        """
        X_train = self.X_train
        k = self.k
        num_samples = X_train.shape[0]
        num_features = X_train.shape[1]

        for iteration in range(self.num_iter):
            #posteriors
            for mixture_id in range(k):
                mean_k = self.parameters[mixture_id][0].flatten()
                cov_k = self.parameters[mixture_id][1]

                # orob denstiy
                pdf = multivariate_normal.pdf(X_train, mean=mean_k, cov=cov_k + 1e-4 * np.ey
                self.posteriors[:, mixture_id] = self.priors[mixture_id] * pdf

            # Normalize
            self.posteriors += 1e-4
            self.posteriors /= self.posteriors.sum(axis=1, keepdims=True)

            # Update parameters
            for mixture_id in range(k):
                gamma_k = self.posteriors[:, mixture_id]  # Posterior probabilities for k
                sum_gamma_k = gamma_k.sum()

                # new mean
                new_mean = np.sum(X_train * gamma_k[:, np.newaxis], axis=0) / sum_gamma_k
                self.parameters[mixture_id][0] = new_mean.reshape(-1, 1)

                # new covariance
                X_centered = X_train - new_mean
                new_cov = (gamma_k[:, np.newaxis] * X_centered).T @ X_centered / sum_gamma_k
                self.parameters[mixture_id][1] = new_cov + 1e-4 * np.eye(num_features)  # Re

                # new priors
                self.priors[mixture_id] = sum_gamma_k / num_samples

            if self.visualize_learning:
                print(f'The clustering for iteration: {iteration}')
                my_plot_data(X_train, self.posteriors.argmax(axis=1), feature_nums=[0, 2])


        y_pred = self.posteriors.argmax(axis=1)


        self.priors = self.priors
```

```
            self.parameters = self.parameters

            return y_pred


    #-----------------Don't change anything below-----------------------#

    # k means parameters
    k = 3                     # num of mixtures
    num_iter = 20
    feature_nums = [0,2]  # features to use

    # load and preprocess the dataset
    dataset = load_iris()
    X, y = dataset.data, dataset.target
    X = (X - X.mean(axis=0)) / X.std(axis=0)
    X = X[:, feature_nums]

    gmm = MyGMMs(X, k, num_iter=num_iter, visualize_learning=True)
    y_pred = gmm.fit_predict()

    print('Final Clustering Result')
    my_plot_data(X, y_pred, feature_nums = feature_nums)
```

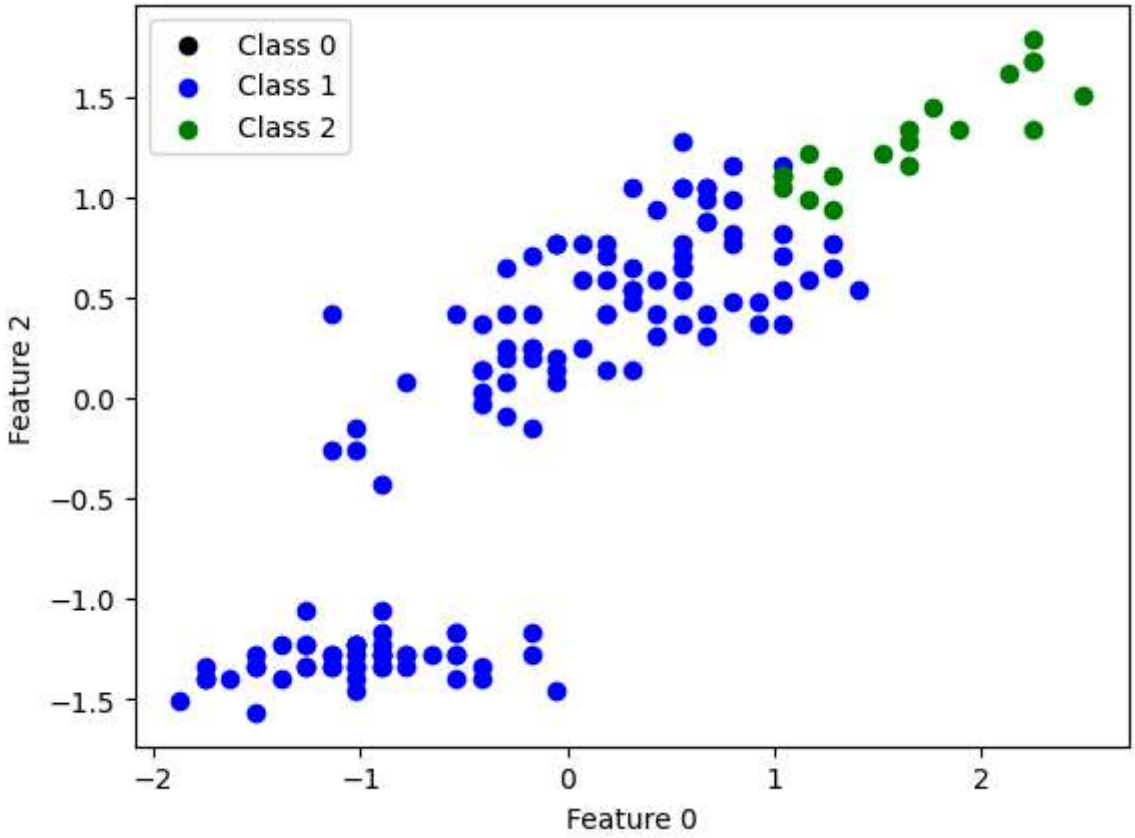
```
The clustering for iteration: 0
```



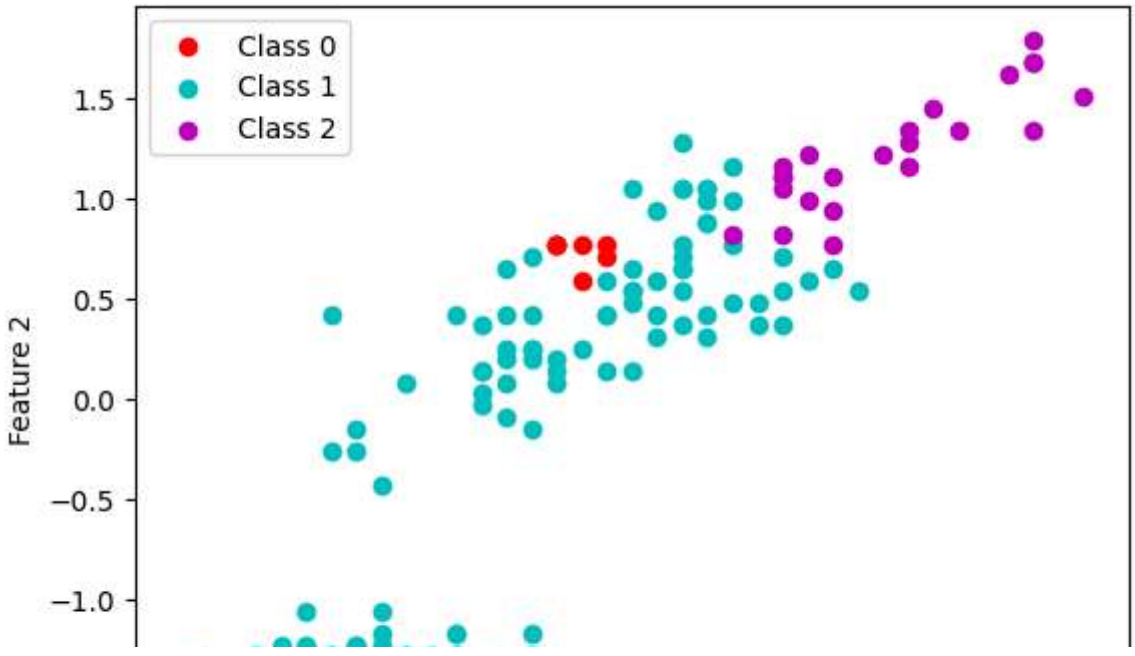

```
Davies Bouldin Score: 1.2984
Dunn Index: 0.0139
Mutual Information Score: 0.1232
Rand Index: 0.0330
Purity Score: 0.4467
The clustering for iteration: 1
```

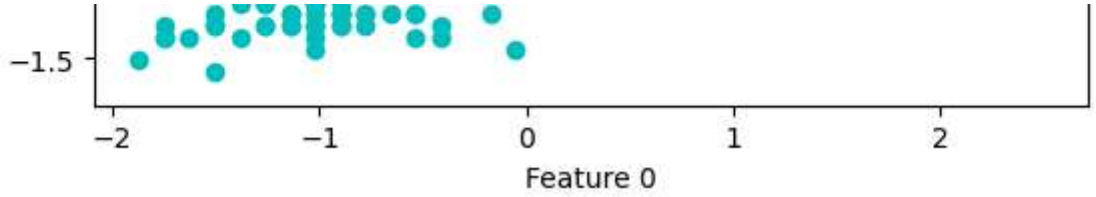


```
Davies Bouldin Score: 0.5900
Dunn Index: 0.0408
Mutual Information Score: 0.1037
Rand Index: 0.0304
Purity Score: 0.4200
The clustering for iteration: 2
```

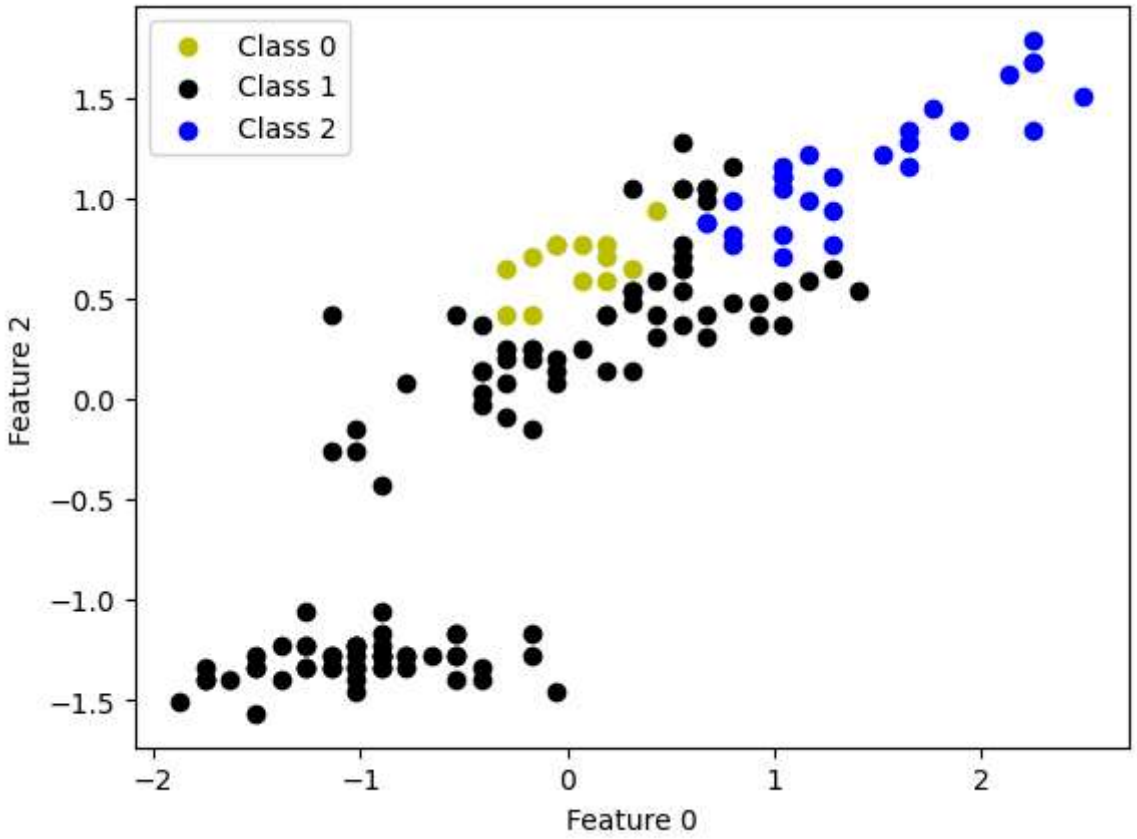



```
Davies Bouldin Score: 0.6663
Dunn Index: 0.0144
Mutual Information Score: 0.1586
Rand Index: 0.0691
Purity Score: 0.4600
The clustering for iteration: 3
```

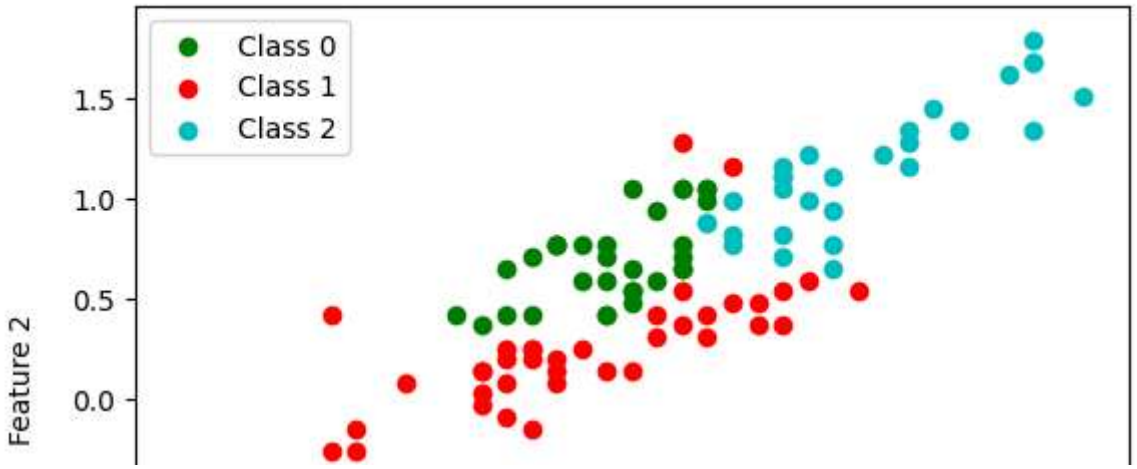


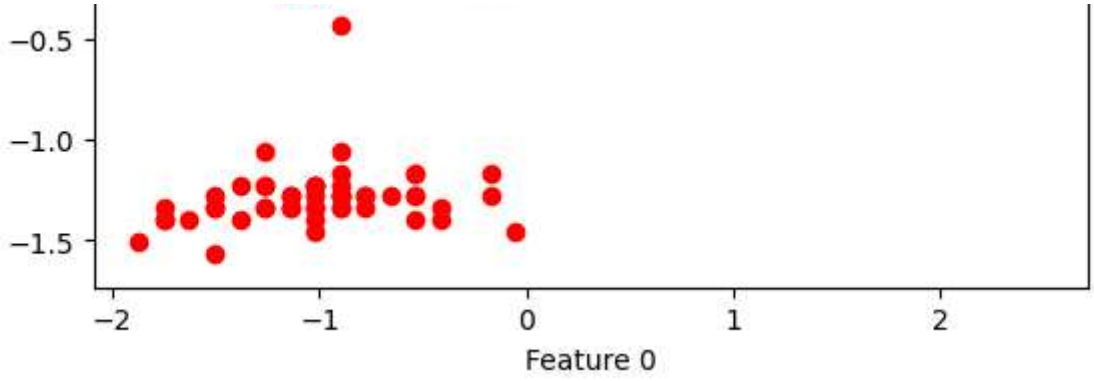



```
Davies Bouldin Score: 1.0120
Dunn Index: 0.0147
Mutual Information Score: 0.2368
Rand Index: 0.1319
Purity Score: 0.5200
The clustering for iteration: 4
```

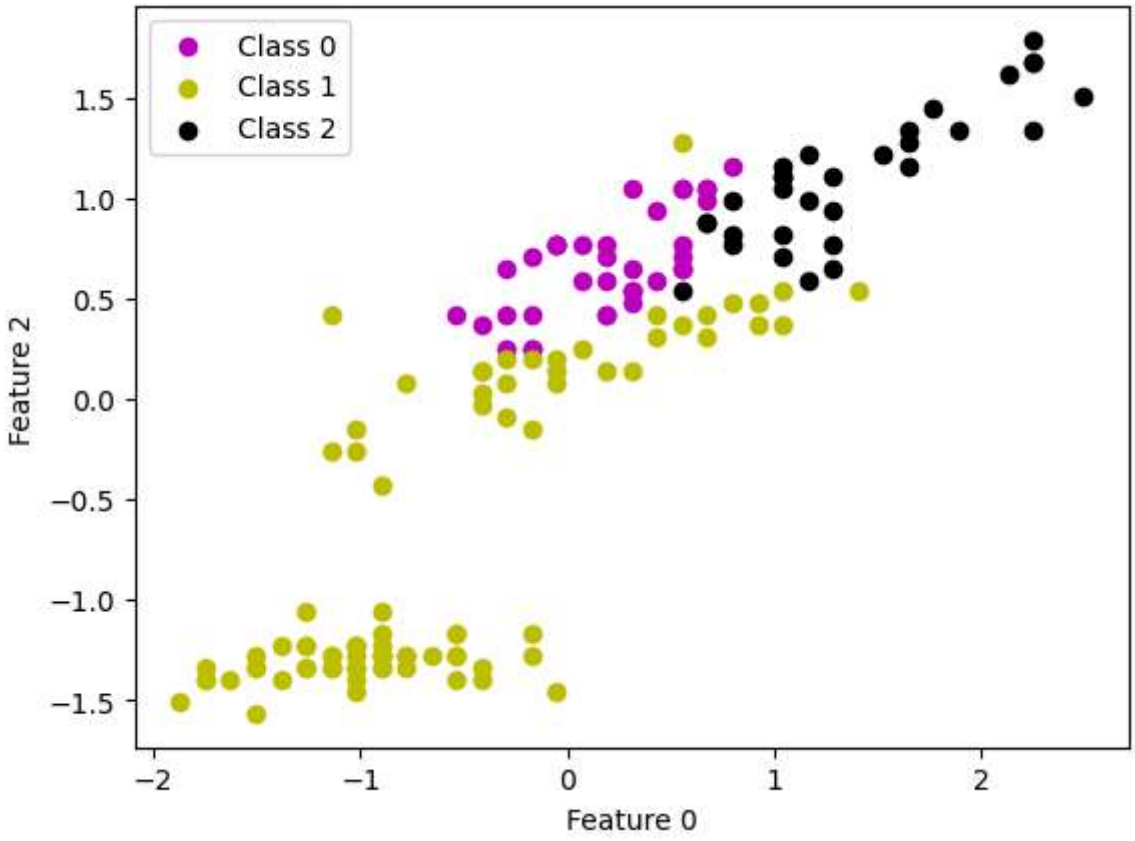



```
Davies Bouldin Score: 1.0286
Dunn Index: 0.0295
Mutual Information Score: 0.3112
Rand Index: 0.2204
Purity Score: 0.5800
The clustering for iteration: 5
```

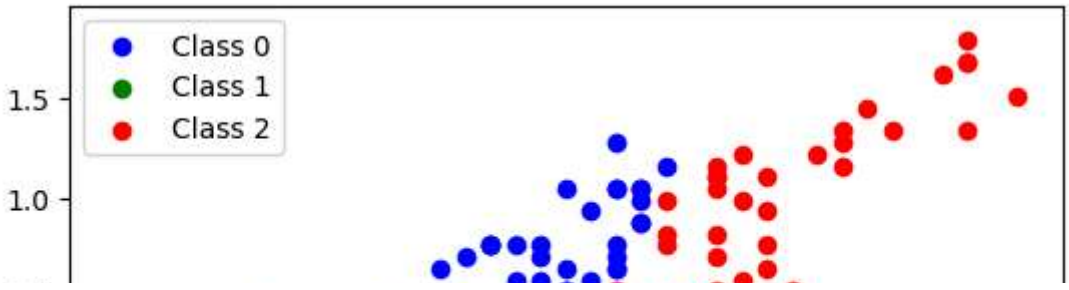


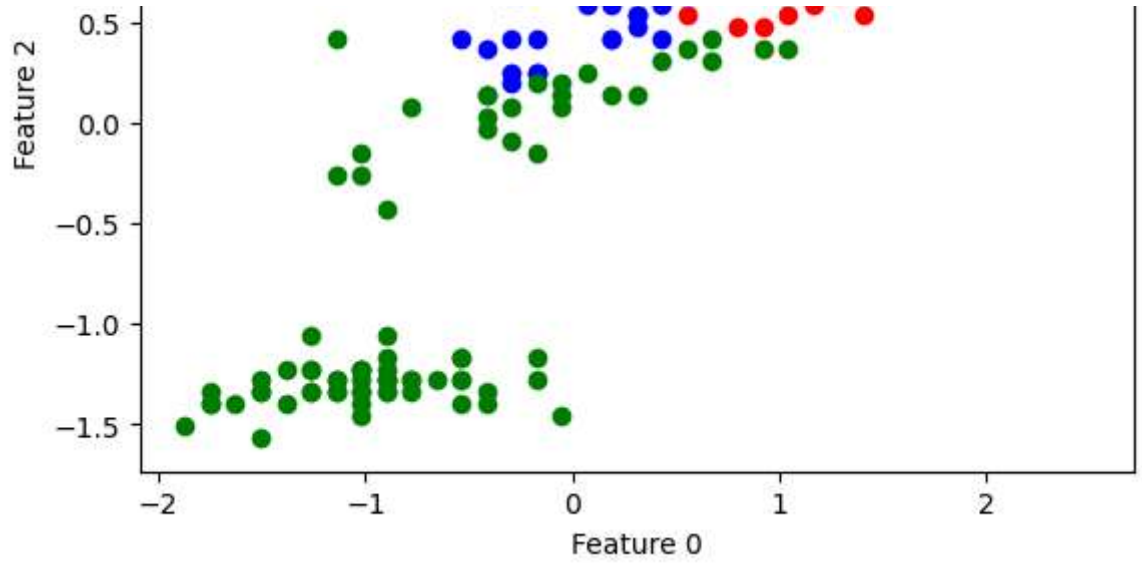



```
Davies Bouldin Score: 0.8360
Dunn Index: 0.0295
Mutual Information Score: 0.4317
Rand Index: 0.3312
Purity Score: 0.6467
The clustering for iteration: 6
```

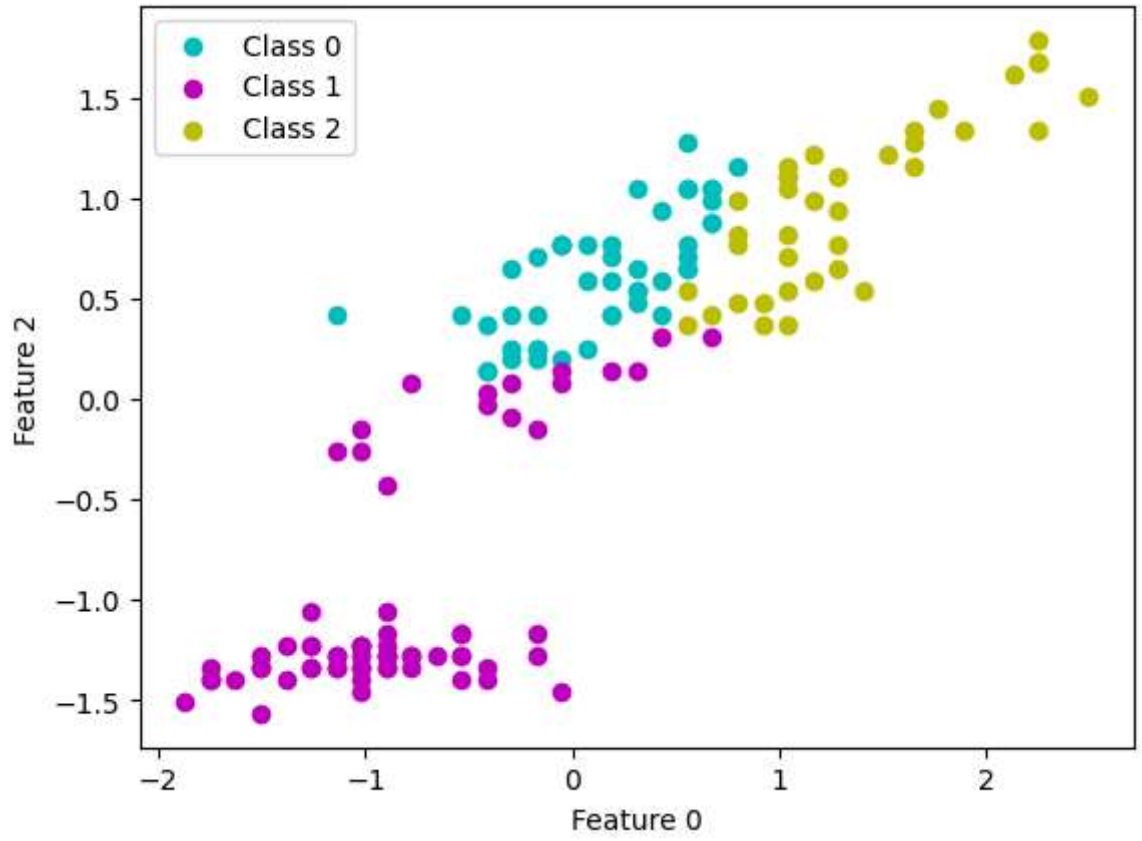



```
Davies Bouldin Score: 0.8156
Dunn Index: 0.0147
Mutual Information Score: 0.4340
Rand Index: 0.3390
Purity Score: 0.6533
The clustering for iteration: 7
```

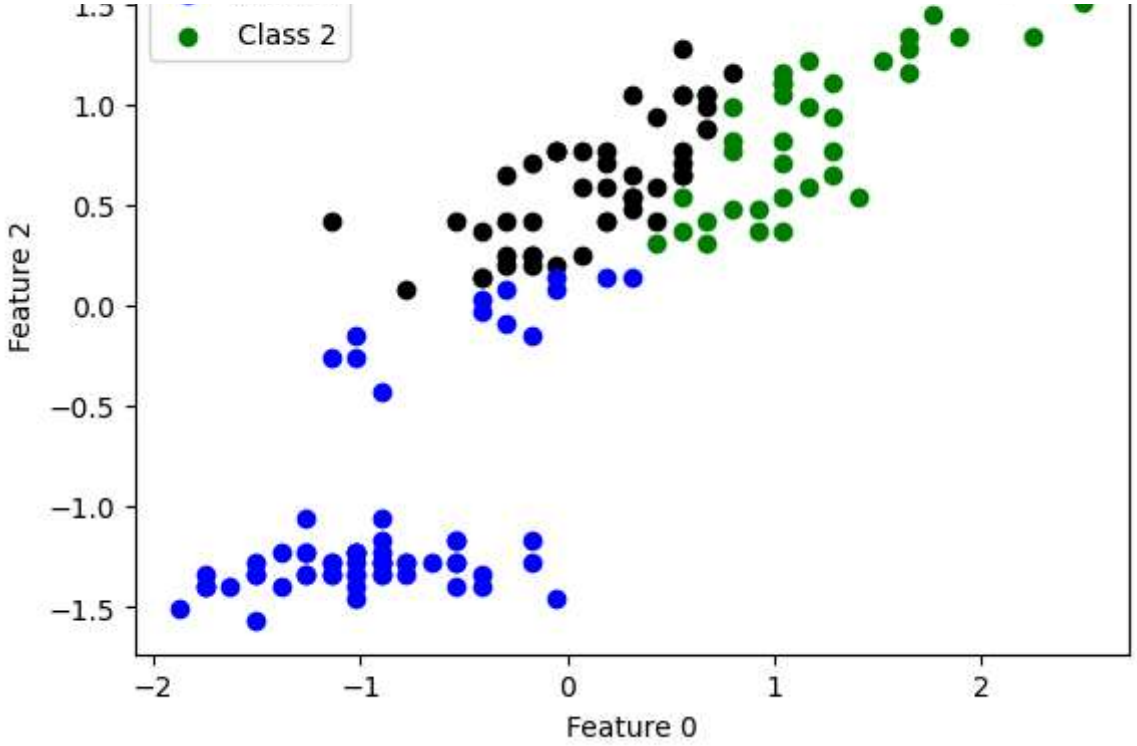




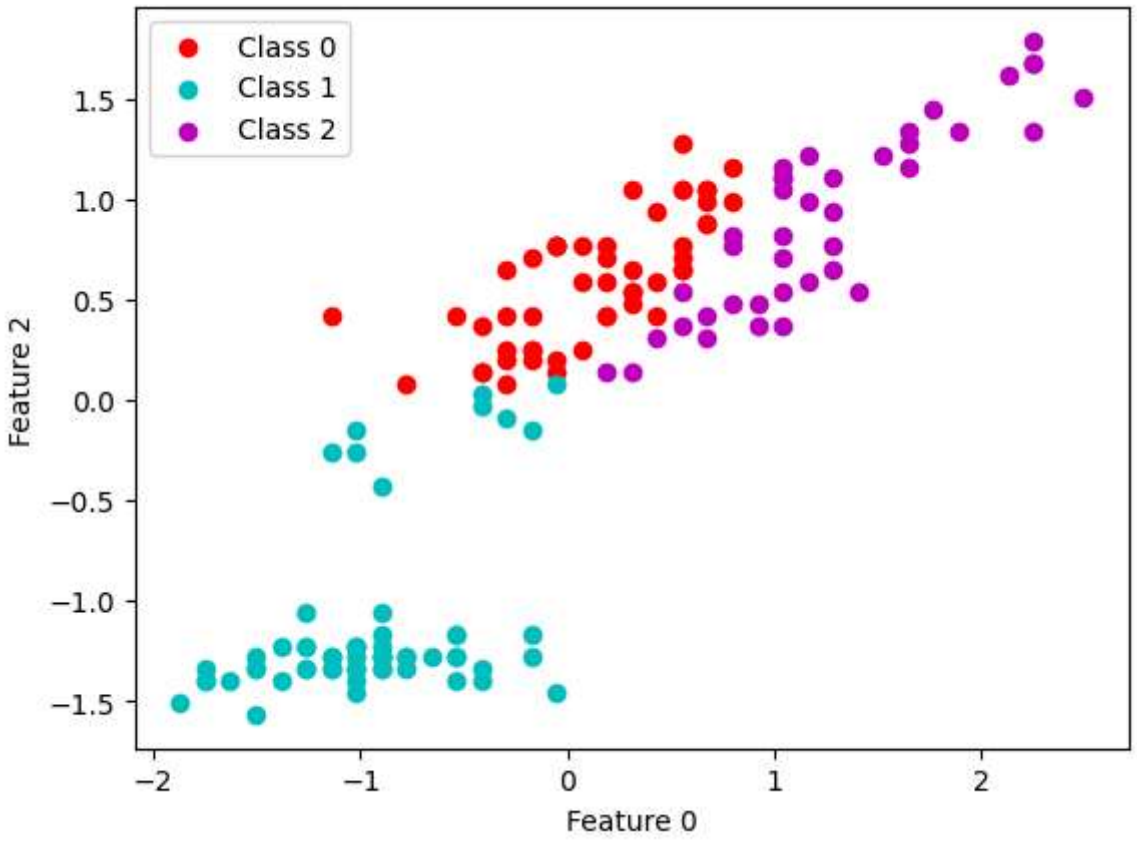

```
Davies Bouldin Score: 0.7598
Dunn Index: 0.0164
Mutual Information Score: 0.4376
Rand Index: 0.3460
Purity Score: 0.6600
The clustering for iteration: 8
```




```
Davies Bouldin Score: 0.7544
Dunn Index: 0.0182
Mutual Information Score: 0.4829
Rand Index: 0.3843
Purity Score: 0.6667
The clustering for iteration: 9
```

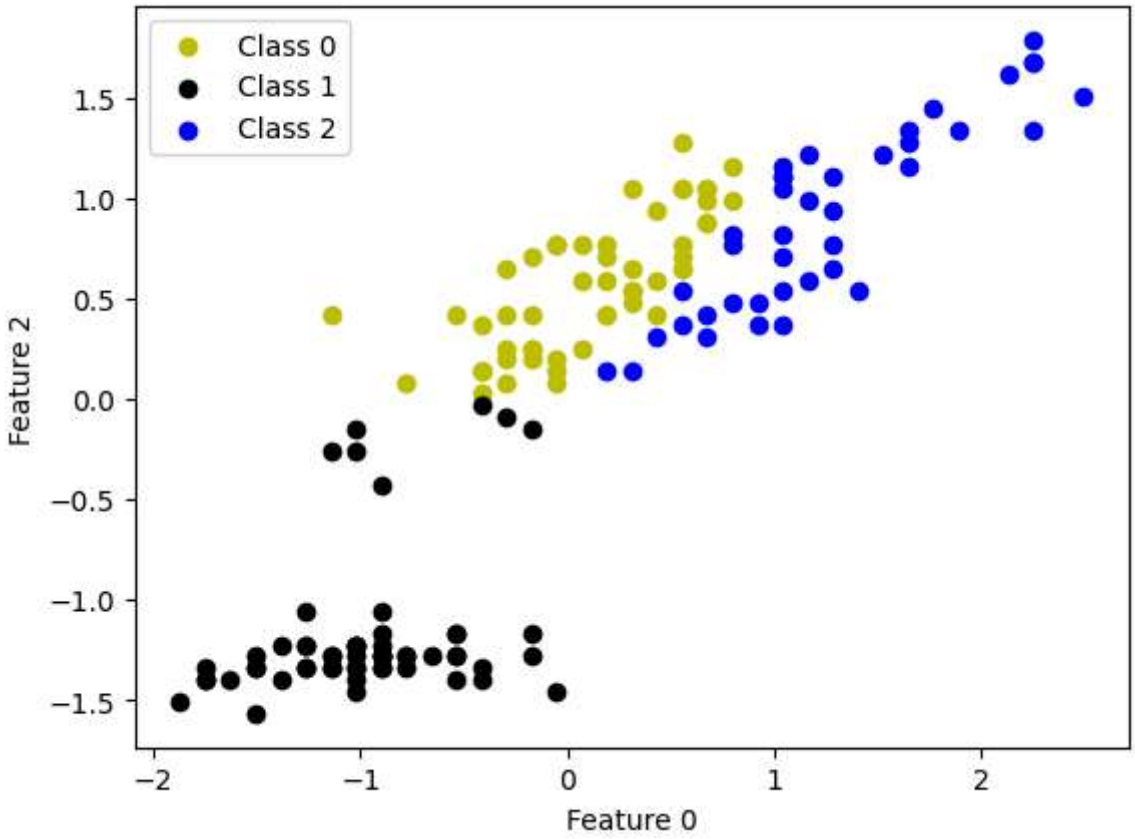



```
Davies Bouldin Score: 0.7658
Dunn Index: 0.0208
Mutual Information Score: 0.4935
Rand Index: 0.3974
Purity Score: 0.6667
The clustering for iteration: 10
```

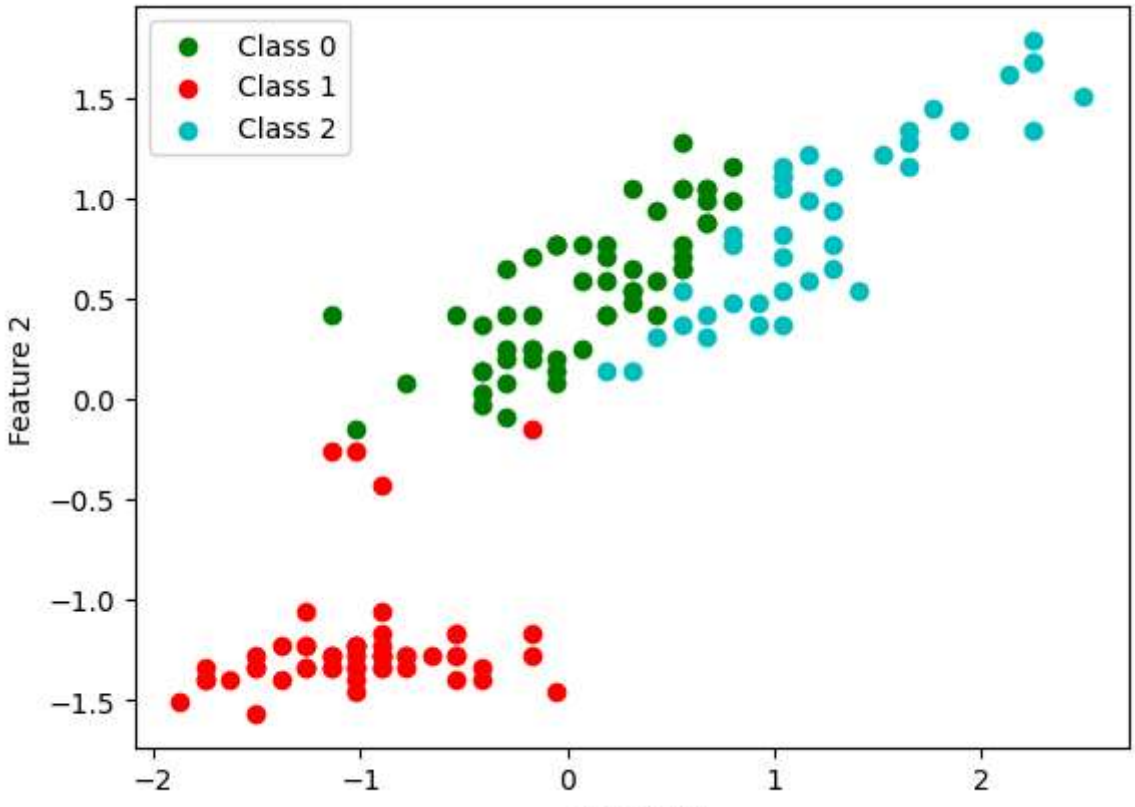


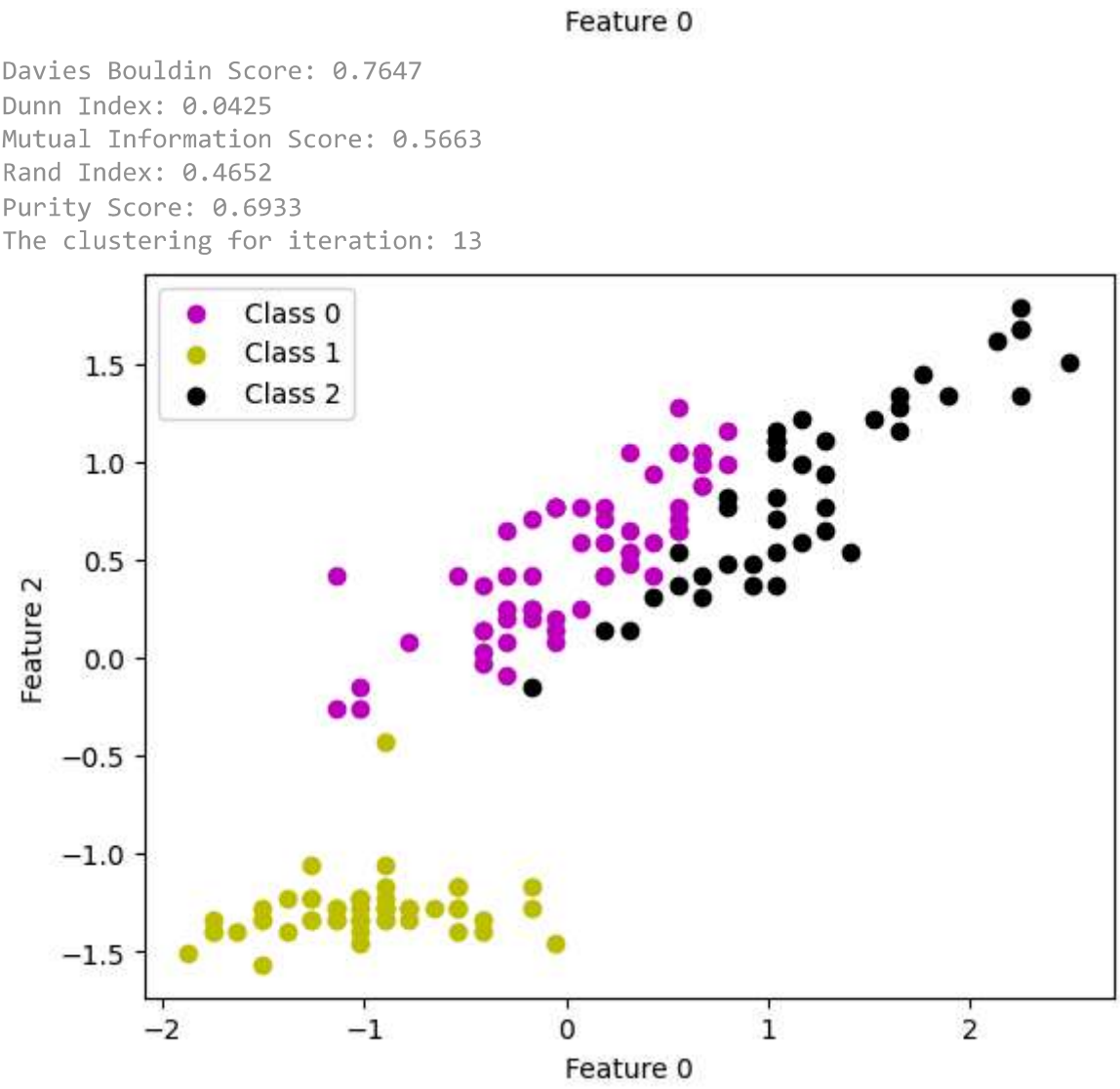

```
Davies Bouldin Score: 0.7922
Dunn Index: 0.0212
Mutual Information Score: 0.5155
Rand Index: 0.4204
```

```
Purity Score: 0.6667
The clustering for iteration: 11
```



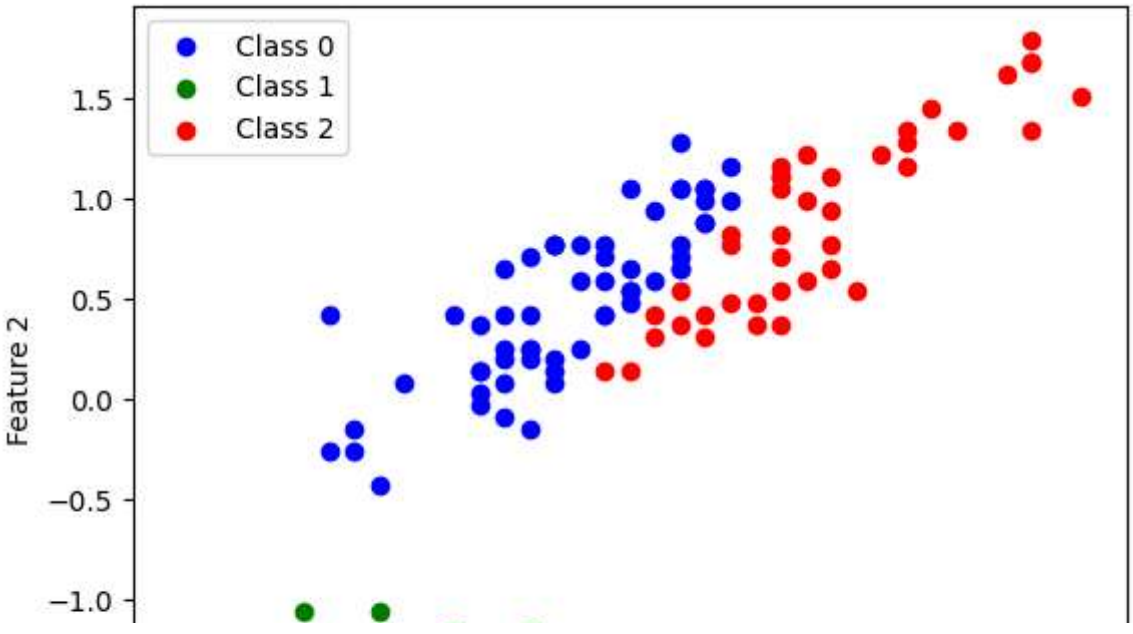

```
Davies Bouldin Score: 0.7787
Dunn Index: 0.0212
Mutual Information Score: 0.5328
Rand Index: 0.4369
Purity Score: 0.6733
The clustering for iteration: 12
```

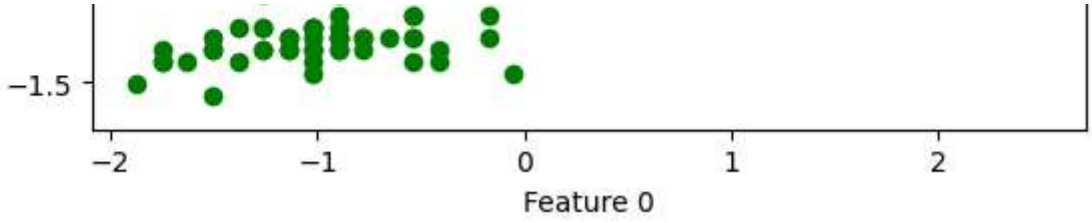




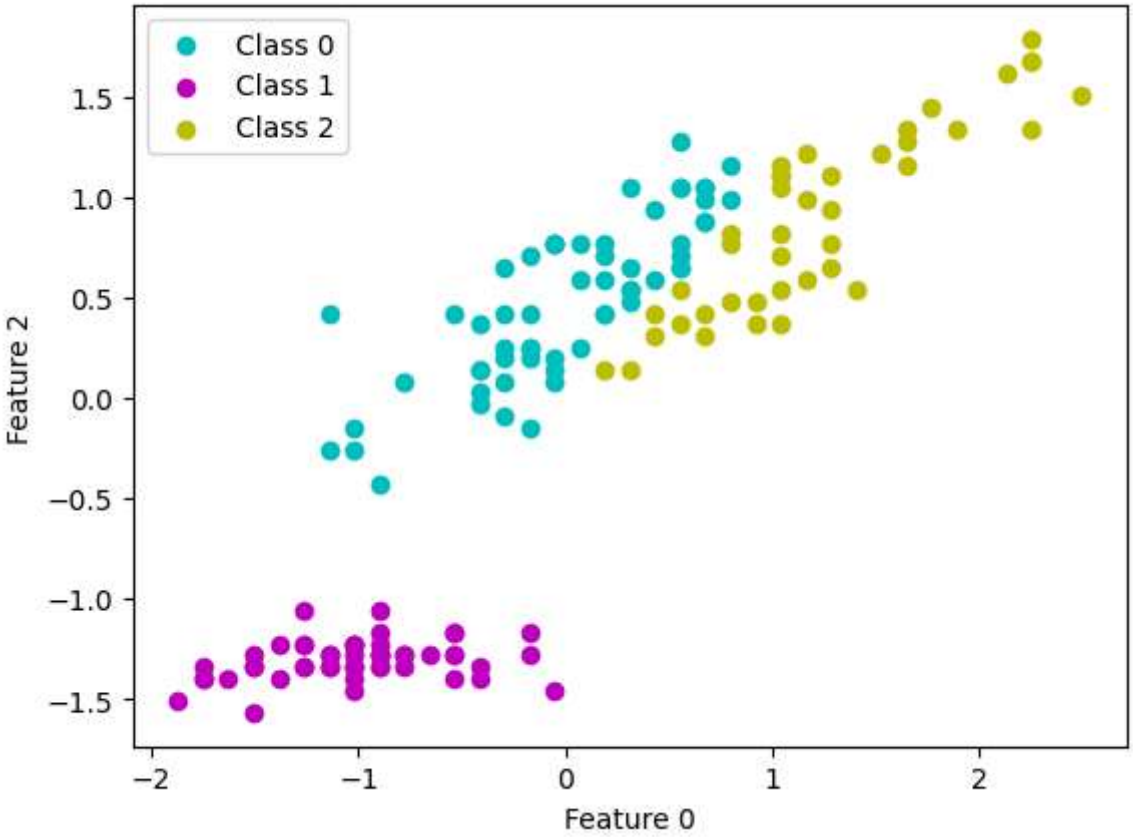

```
Davies Bouldin Score: 0.7647
Dunn Index: 0.0425
Mutual Information Score: 0.5663
Rand Index: 0.4652
Purity Score: 0.6933
The clustering for iteration: 13
```

```
Davies Bouldin Score: 0.7895
Dunn Index: 0.0363
Mutual Information Score: 0.6144
Rand Index: 0.4943
Purity Score: 0.7067
The clustering for iteration: 14
```

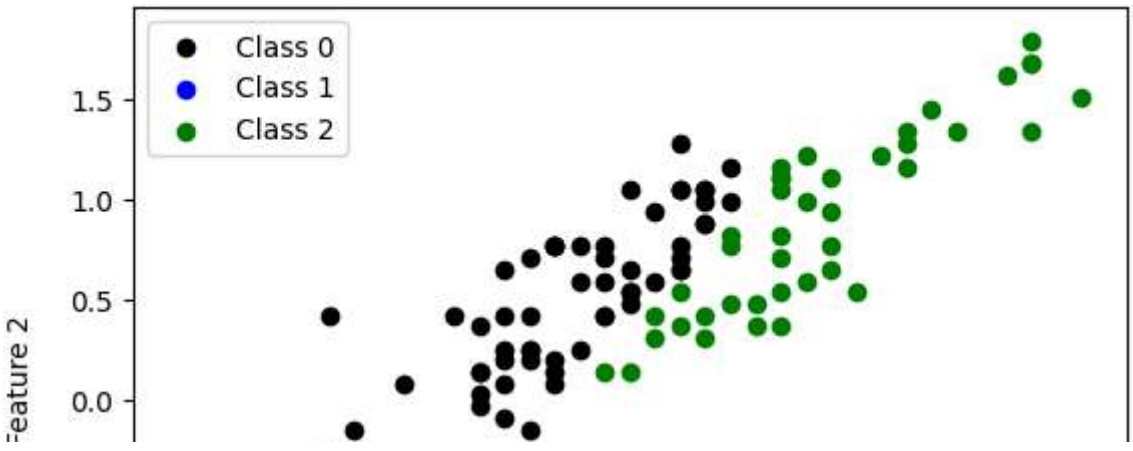


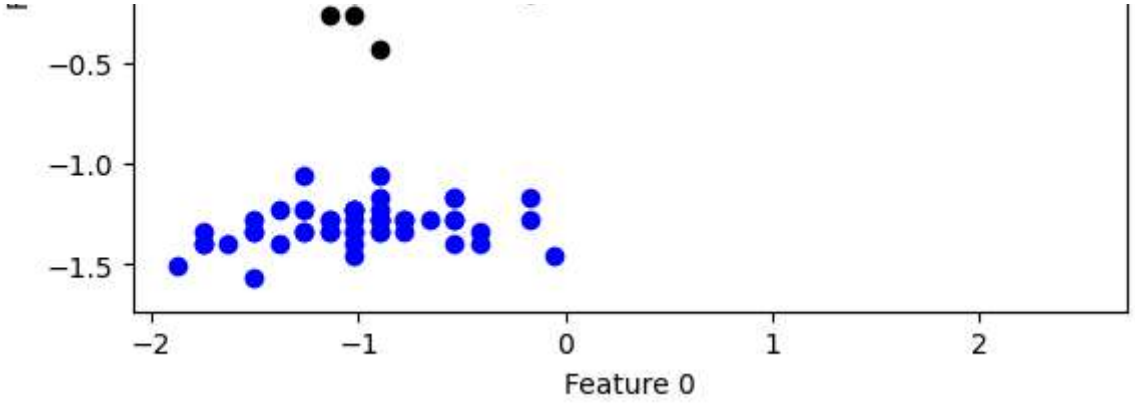



```
Davies Bouldin Score: 0.7671
Dunn Index: 0.0425
Mutual Information Score: 0.6433
Rand Index: 0.5059
Purity Score: 0.7133
The clustering for iteration: 15
```

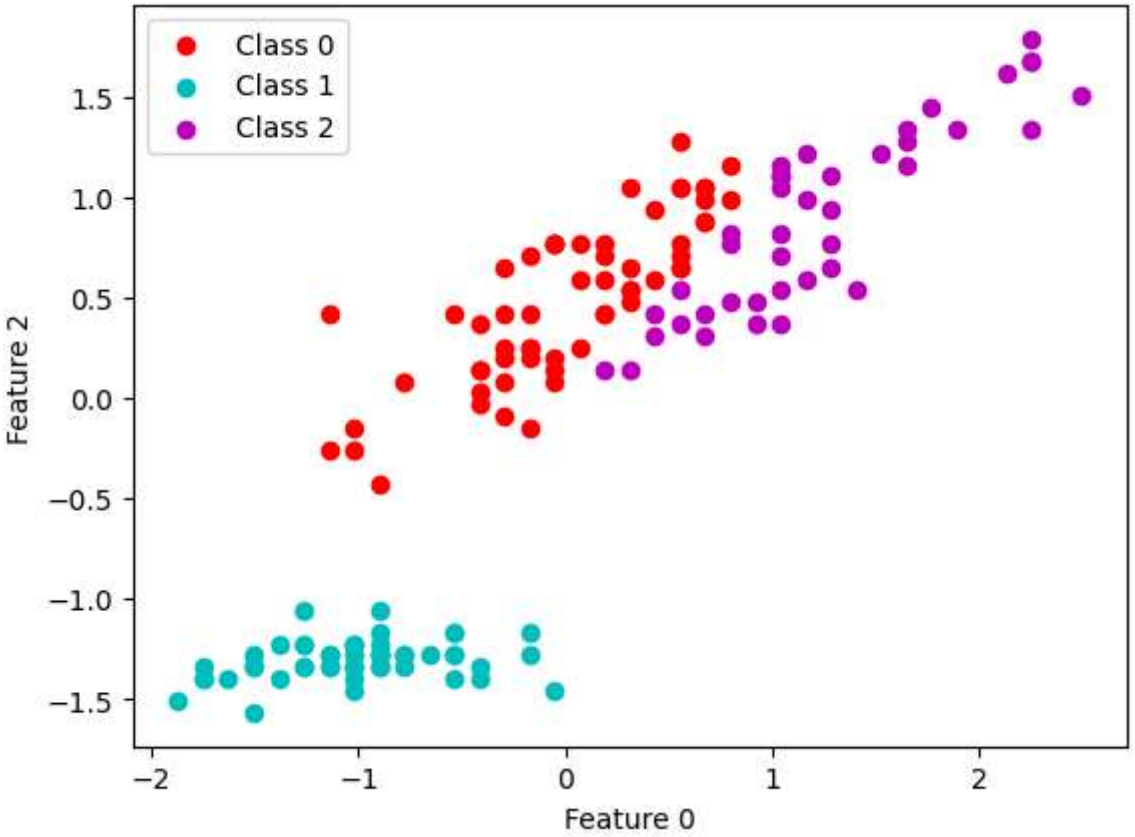



```
Davies Bouldin Score: 0.7671
Dunn Index: 0.0425
Mutual Information Score: 0.6433
Rand Index: 0.5059
Purity Score: 0.7133
The clustering for iteration: 16
```

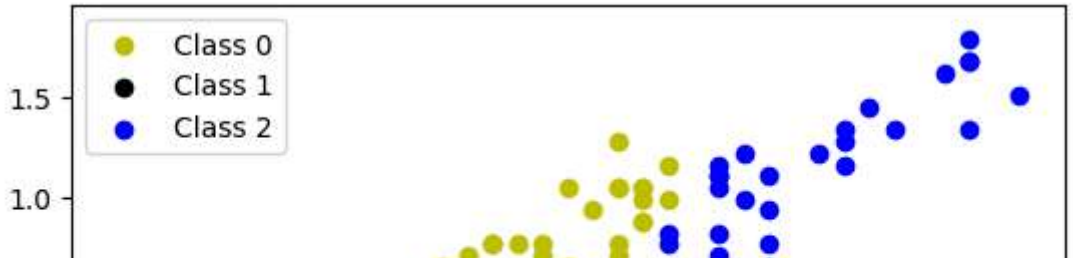


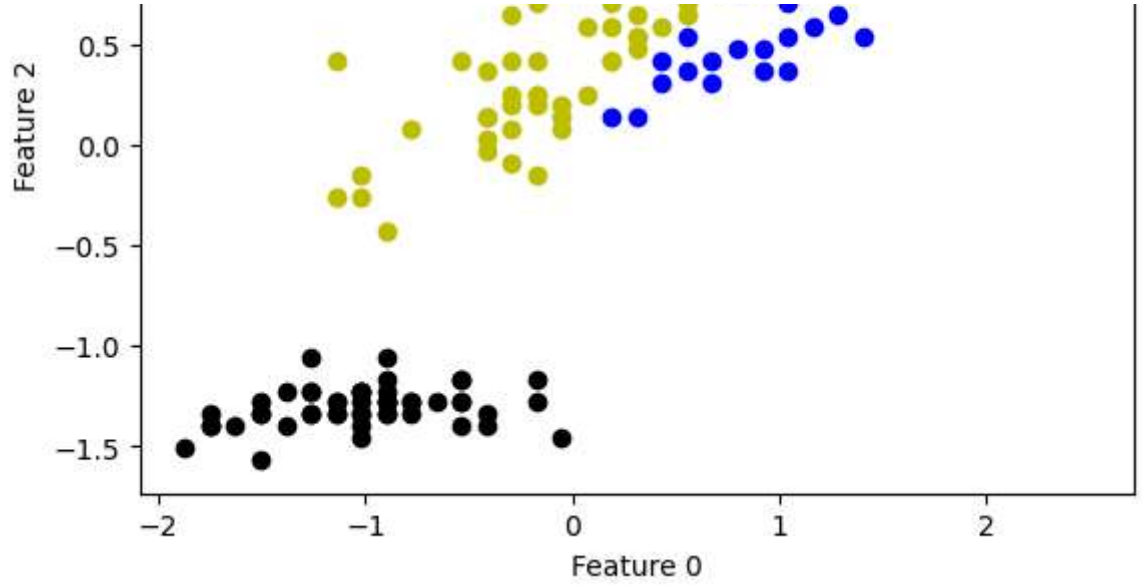



```
Davies Bouldin Score: 0.7671
Dunn Index: 0.0425
Mutual Information Score: 0.6433
Rand Index: 0.5059
Purity Score: 0.7133
The clustering for iteration: 17
```

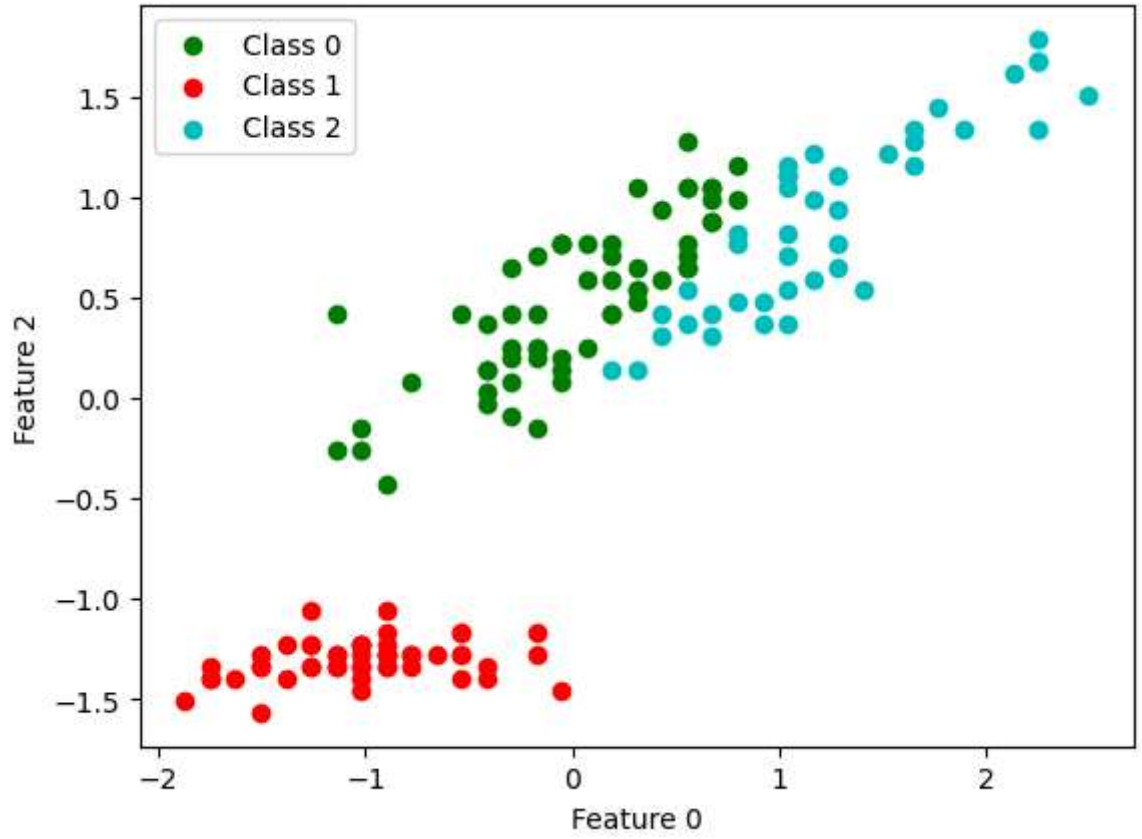



```
Davies Bouldin Score: 0.7671
Dunn Index: 0.0425
Mutual Information Score: 0.6433
Rand Index: 0.5059
Purity Score: 0.7133
The clustering for iteration: 18
```

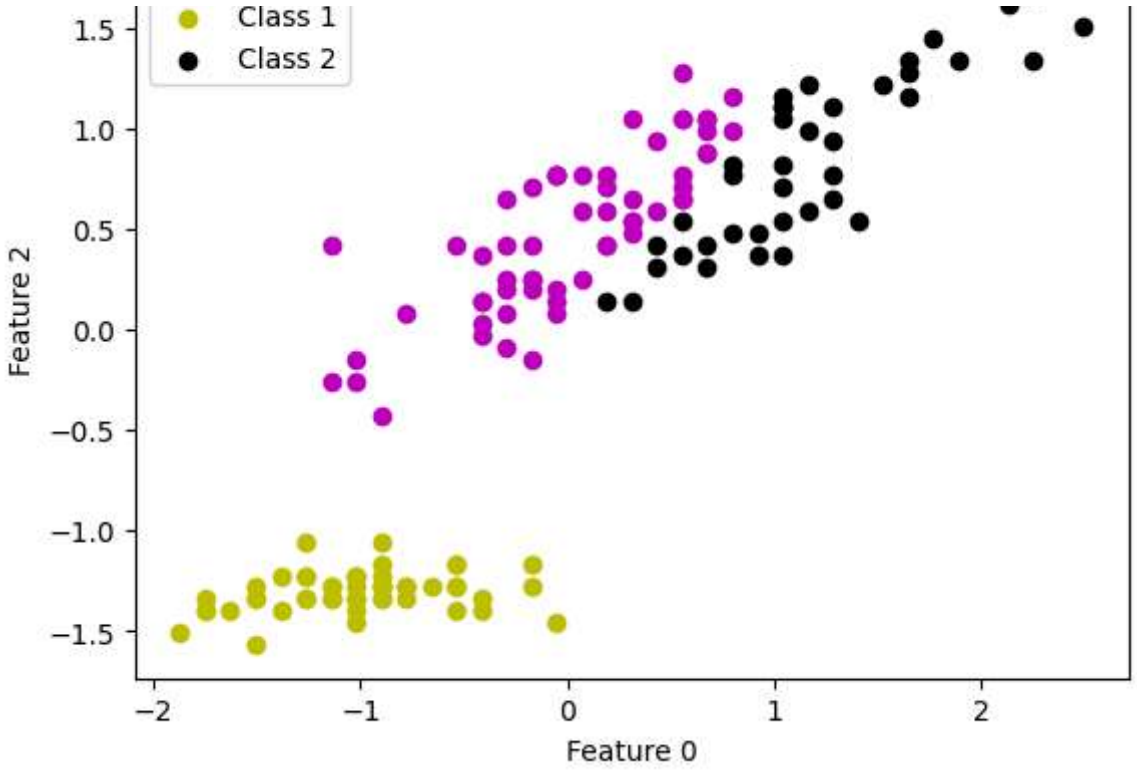





```
Davies Bouldin Score: 0.7671
Dunn Index: 0.0425
Mutual Information Score: 0.6433
Rand Index: 0.5059
Purity Score: 0.7133
The clustering for iteration: 19
```




```
Davies Bouldin Score: 0.7671
Dunn Index: 0.0425
Mutual Information Score: 0.6433
Rand Index: 0.5059
Purity Score: 0.7133
Final Clustering Result
```




```
Davies Bouldin Score: 0.7671
Dunn Index: 0.0425
Mutual Information Score: 0.6433
Rand Index: 0.5059
Purity Score: 0.7133
```

Problem 5: Implementing Image Segmentation via Unsupervised Clustering on Kaggle Competition (25pts)

After designing your very own classes implementing the popular K-means and GMM algorithms for clustering, we are now going to test them out on image data for segmentation of different structures therein. The first part of this question is designed to guide you in a step-by-step process to convert a simple, gray-scale image in a form that can be processed by the clustering classes you designed above before converting the result back in a spatiotemporal form for visualization of the segmented structures.

(a)

Run the code cell below to load an image from `sklearn`'s `digits` dataset representing images of numbers $0-9$. Complete the function template in the cell to reshape the features in the form of an 8×8 image that can be displayed using `matplotlib`'s `imshow` function. (*Hint: You may find it helpful to use* `np.reshape` *function*.)

(b)

Execute the cell below that takes a digit example as input to each of the clustering classes (K-means and GMM) you designed above to output a segmentation result. Remember that in this case, each pixel in the image is going to be a 'training example' from the perspective of the clustering algorithm, with the 'feature' being the gray scale value itself. Fill in the function template reshaping the digit example into the form needed for your custering classes.

(c)

For a simple problem like above with only gray-scale images, you learnt to process images in a form they could be used to train clustering algorithms (with each individual pixel being a 'training example'). For an RGB image, each training example would have at least 3 features (the Red, Green, and Blue values for the pixel). In addition, one could add the spatial positions of the pixel as another set of features. We are now going test what you have learnt by means of a Kaggle competition wherein you are asked to segment two RGB images. Your results will be submitted to a Kaggle leaderboard to be graded accordingly. Try different feature combinations, image processing techniques to get the best looking results

We provided you with two images: img_1 is a simple image consisting of geometric shapes and img_2 contains more complicated objects, both in RGB. The images can be found in homework folder. After uploading images in the current directory, you can then load the two images as shown below:

```
import imageio
```

```
im1 = imageio.imread('img1.png')
im2 = imageio.imread('img2.png')
```

Since this is an open-ended question, you should try to design your own features to achieve better classification results. By submitting your result to Kaggle, you will see your multi-class classification accuracy and ranking on the leaderborad. 10 pts will depend on your ranking (top 10% get 10pts, top 11%-20% get 9pts, etc.), and another 5 pts depend on your method explanation and clear, well-labeled plots and images.

[Note:]

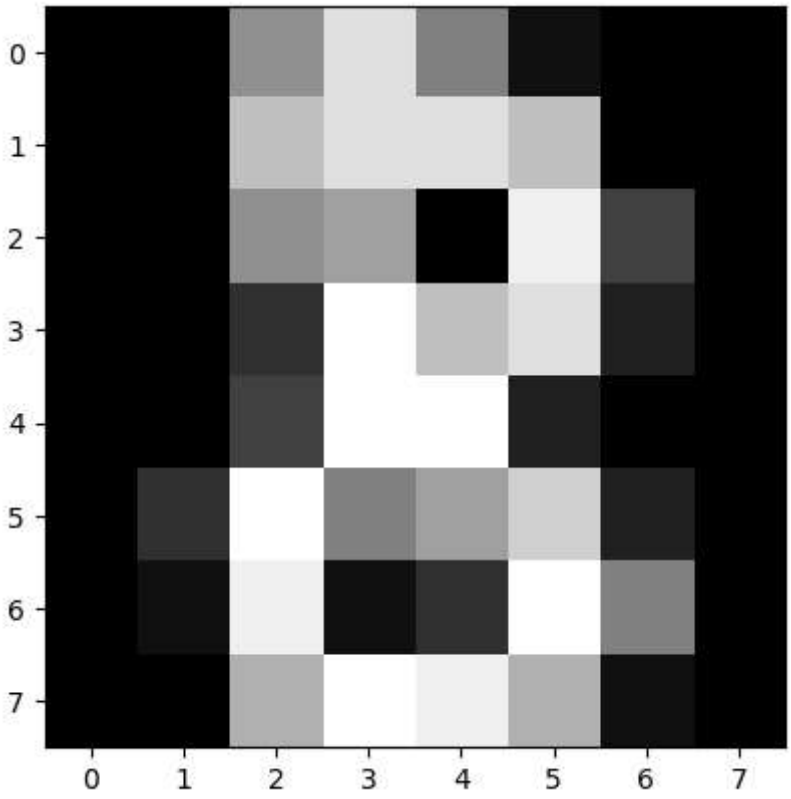
- After having your ideal result, please save your result in `result_img1` and `result_img2` in the cell below.
- We have defined the label for each class in each image. Make sure you use the same settings as us and feel free to use the provided swapping label code to swap labels if needed.
  - for `img1.png`, use k=3, background labels as 0, rectangle labels as 1 and triangle labels as 2
  - for `img2.png`, use k=2, background labels as 0, building labels as 1
- After your `result_img1` and `result_img2` are ready, run the cell below to create a `submission.csv`. Please download it from this notebook and submit it in our Kaggle competition.
- Please remember to note your Kaggle competition nickname in this notebook. We will use your ranking to grade.
- You have 10 submission quota each day to submit your result and get your multi-class classification accuracy and ranking.
- We calculate the multi-class classification accuracy with ground-turth hand-crafted labels in both images by

$$\text{Accuracy} = \frac{\#\text{ of correctly labeled pixels}}{\text{total }\#\text{ of pixels}}$$

Problem 5 (a) Solution

```
## for problem 5 (a)

import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits

# function to reshape
```

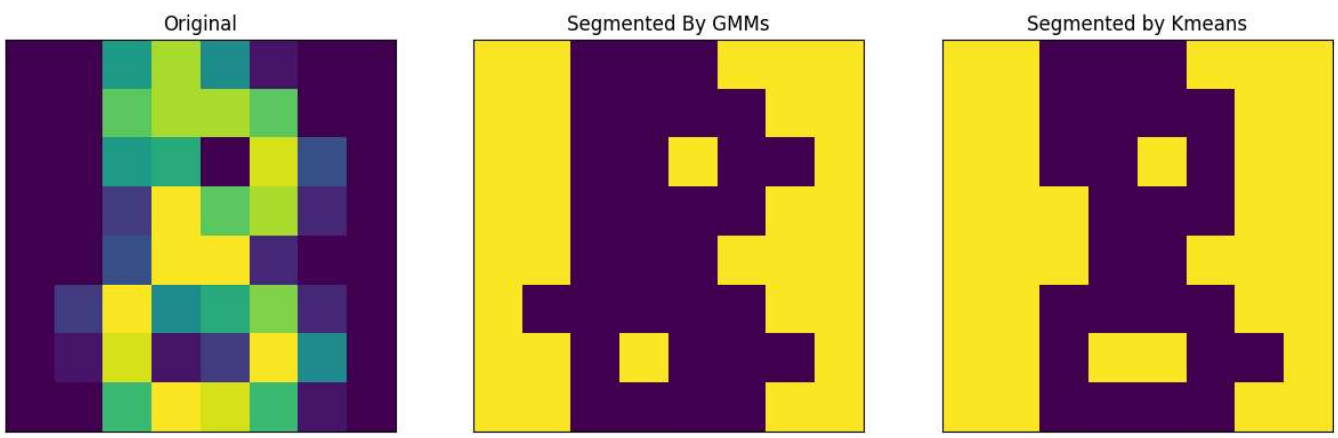
```
def image_reshape(x):
  """Function reshapes a training example from the Digits dataset into an image

  Parameters
  ----------
  x: ndarray of shape (1, num_of_features)
    flattened image example from the digits dataset

  Returns
  -------
  img: ndarray of shape (8,8)
    ndarray containing reshaped image for visualization
  """

  return x.reshape(8,8)

#-----------------Don't change anything below-----------------------#

# load data and extract a digit example
X, _ = load_digits(return_X_y=True)
digit = X[8].reshape(1, -1)

# reshape image
reshaped_img = image_reshape(digit)

# visualize
plt.imshow(reshaped_img, cmap='gray')
plt.show()
```



Problem 5 (b) Solution

```
## for problem 5 (b)

import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits

def feature_reshape(digit):
  """function reshapes a digit training example into a form acceptable for use with
  clustering classes

  Parameters:
  -----------
  digit: ndarray of shape (1, num_of_features)

  Returns:
  --------
  reshaped_digit: ndarray of shape (num_of_features, 1)
  """

  return digit.reshape(-1, 1)


#----------------Don't change anything below----------------------#
```

```
# load data and extract a digit example
X, _ = load_digits(return_X_y=True)
digit = X[8].reshape(1, -1)

# number of mixtures/clusters
k = 2

reshaped_digit = feature_reshape(digit)
reshaped_digit = (reshaped_digit - reshaped_digit.mean(axis=0)) / reshaped_digit.std(axis=0)

# cluster with model of choice
model_1 = MyGMMs(reshaped_digit, k=k, num_iter=20)
model_2 = MyKMeans(reshaped_digit, k=k, num_iter=20)
y_pred_1 = model_1.fit_predict()
y_pred_2 = model_2.fit_predict()

# visualize results
segmented_im1_1 = y_pred_1.reshape(8, 8)
segmented_im1_2 = y_pred_2.reshape(8, 8)

fig, (ax1, ax2, ax3) = plt.subplots(1,3, figsize=(15,9))
ax1.imshow(image_reshape(digit))
ax1.set_title('Original')
ax1.set_xticks([])
ax1.set_yticks([])
ax2.imshow(segmented_im1_1)
ax2.set_title('Segmented By GMMs')
ax2.set_xticks([])
ax2.set_yticks([])
ax3.imshow(segmented_im1_2)
ax3.set_title('Segmented by Kmeans')
ax3.set_xticks([])
ax3.set_yticks([])
plt.show()
```



Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit