Indian Institute Of Space Science And Technology

Topic : Advance Machine Learning Lab(MA644)

Assignment 1 : Modal Based Clustering

MD SOHAIL ANSARI
SC23M008

January 21, 2024

Introduction

Modal-based clustering is a clustering technique that focuses on identifying modes, or high-density regions, within a dataset to group similar data points together. In modal-based clustering, the primary objective is to locate regions where the data distribution exhibits significant peaks, indicating the presence of clusters. Each cluster(component) is mathematically represented by a parametric distribution as for example Gaussian (continuous) or a poisson (discrete).

A Gaussian Mixture Model (GMM) is a probabilistic model widely used for clustering and density estimation tasks. Gaussian Mixture Models (GMMs) overcome k-means limitations by allowing flexible cluster shapes through Gaussian distributions, providing soft assignments with probabilities, accommodating varying cluster covariance, being less sensitive to outliers, and modeling different cluster sizes. This makes GMMs more adaptable to real-world data distributions, offering improved cluster representation and capturing intricate structures that k-means might struggle to handle.

In a GMM, the parameters include the mean, covariance, and weight for each Gaussian distribution. The mean and covariance define the shape and orientation of each Gaussian component, while the weight represents its contribution to the overall mixture. GMMs are particularly effective when dealing with datasets that exhibit complex and overlapping cluster structures.

The Expectation-Maximization (EM) algorithm is commonly employed to train GMMs. During the Expectation step, the algorithm estimates the probability of each data point belonging to each cluster, given the current parameters. In the Maximization step, the parameters are updated to maximize the likelihood of the observed data under the model. This iterative process continues until convergence is achieved.

Expectation Minimization Algorithm

E Step : In the E step, compute the probabilities ,

$$P_{ij} = w_i \cdot \frac{p(x_j|C_i)}{p(x_j)}$$

for i = 1,2,...k and j = 1,2,...N

M Step : Compute the new mean, covariance and component weights as follows:

$$\mu(i) = \frac{\sum_{j=1}^{N} P_{ij} \cdot x_j}{\sum_{j=1}^{N} P_{ij}}$$

$$\sum_i = \frac{\sum_j P_{ij} \left((x_j - \mu(i))(x_j - \mu(i))^T\right)}{\sum_{j=1}^{N} P_{ij}}$$

$$w_i = \frac{\sum_{j=1}^{N} P_{ij}}{N}$$

Pseudocode

Initialize μ_i, Σ_i, w_i for $i = 1, 2, \ldots, k$.

Iterate untill convergence:

E step

for i=1 to k do

for j=1 to N do

calculate :

$$P_{ij} = (p(x_j/C = i)w_i/(\sum_{j=1}^{N} p(x_j/C = i)w_i)$$

end for

M Step

for i=1 to k do

calculate :

$$\mu(i) = \frac{\sum_{j=1}^{N} p_{ij}x_j}{p_i}$$

caculate :

$$\sum_i = \frac{\sum_j P_{ij}\left((x_j - \mu(i))(x_j - \mu(i))^T\right)}{\sum_{j=1}^{N} P_{ij}}$$

Set : $w_i = \frac{p_i}{N}$.

end for

Question : 1
Apply model based clustering on Data 1
(a) Report the number of clusters and the technique used to find the number of clusters.
(b) Plot the clusters.
(c) Report the mean, co-variance matrix and prior probability corresponding to each cluster.
(d) Assess the quality of the clusters.

Python Code

Data Reading

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

```
data_1 = pd.read_csv("/content/drive/MyDrive/AML/ASSIGNMENT 1/Data1.csv")
data_1
```

	Unnamed: 0	0	1
0	0	1.004939	2.319887
1	1	3.412653	-1.637157
2	2	7.483318	-1.399250
3	3	0.702826	2.038150
4	4	0.287620	2.191703
...	...	...	...
1595	1595	1.475069	2.329653
1596	1596	4.277030	2.183024
1597	1597	0.814996	2.246927
1598	1598	7.999698	-1.811024
1599	1599	4.007795	0.121834

1600 rows × 3 columns

Removing First Column

```
df = data_1.iloc[:, 1:]
df
```

	0	1
0	1.004939	2.319887
1	3.412653	-1.637157
2	7.483318	-1.399250
3	0.702826	2.038150
4	0.287620	2.191703
...	...	...
1595	1.475069	2.329653
1596	4.277030	2.183024
1597	0.814996	2.246927
1598	7.999698	-1.811024
1599	4.007795	0.121834

1600 rows × 2 columns

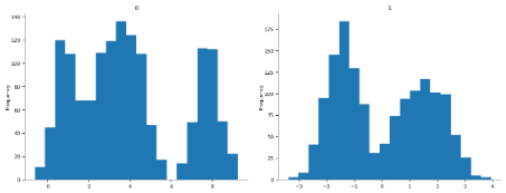
Shuffling the data

```
df_shuffled = df.sample(frac = 0.9 , random_state = 42)
df_shuffled
```

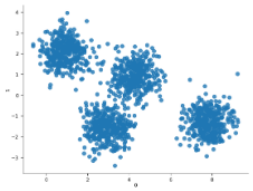
	0	1
526	-0.627080	2.363018
354	2.349838	-2.028020
168	1.423081	1.773681
135	3.724327	-1.533585
937	2.541988	-1.804040
...	...	...
524	1.814574	-2.799610
1059	4.541439	1.830060
540	0.907467	2.095763
1194	4.239404	0.939294
748	7.177305	-1.496538

1440 rows × 2 columns

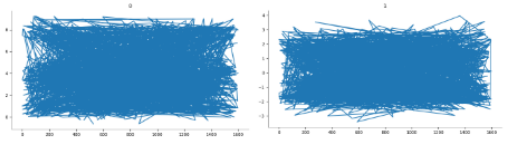
Distributions



2-d distributions



Values



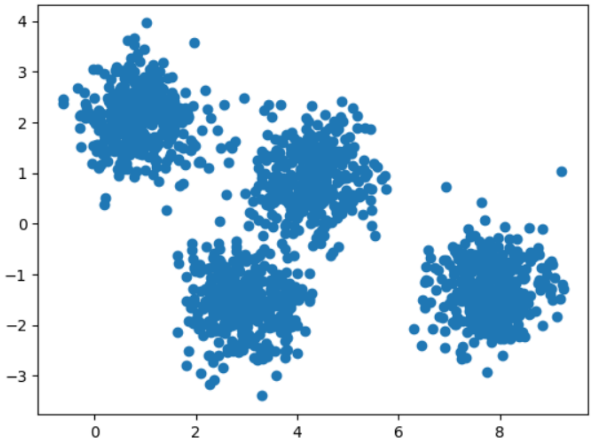
```
data = df_shuffled.values
data
```

```
array([[-0.62707962,  2.36301763],
       [ 2.34983801, -2.02801963],
       [ 1.42308107,  1.77368108],
       ...,
       [ 0.90746651,  2.09576288],
       [ 4.23940416,  0.93929424],
       [ 7.17730503, -1.49653817]])
```

Visulalizing the data

```
plt.scatter(data[:, 0], data[:, 1], cmap='viridis')
```

```
<matplotlib.collections.PathCollection at 0x7ba1706520b0>
```



Creating Function for EM Algorithm

```
def initialize_parameters(num_components, data_dim):
    means = np.random.rand(num_components, data_dim)
    covariances = np.array([np.eye(data_dim) for _ in range(num_components)])
    weights = np.ones(num_components) / num_components
    return means, covariances, weights

def e_step(data, means, covariances, weights):
    num_components = len(means)
    num_data = len(data)

    probabilities = np.zeros((num_data, num_components))

    for i in range(num_components):
        # Use multivariate_normal.pdf for n-dimensional arrays
        probabilities[:, i] = weights[i] * multivariate_normal.pdf(data, mean=means[i], cov=covariances[i])

    normalization_factor = np.sum(probabilities, axis=1, keepdims=True)
    probabilities /= normalization_factor

    return probabilities
```

```python
def m_step(data, probabilities):
    num_components = probabilities.shape[1]
    num_data = len(data)
    data_dim = data.shape[1]

    means = np.zeros((num_components, data_dim))
    covariances = np.zeros((num_components, data_dim, data_dim))
    weights = np.zeros(num_components)

    for i in range(num_components):
        means[i] = np.sum(probabilities[:, i, np.newaxis] * data, axis=0) / np.sum(probabilities[:, i])
        covariances[i] = np.dot((data - means[i]).T, (probabilities[:, i, np.newaxis] * (data - means[i]))) / np.sum(probabilities[:, i])
        weights[i] = np.mean(probabilities[:, i])

    return means, covariances, weights
```

```python
def em_algorithm(data, num_components, num_iterations=100, tol=1e-4):
    data_dim = data.shape[1]

    means, covariances, weights = initialize_parameters(num_components, data_dim)

    for iteration in range(num_iterations):
        # E Step
        probabilities = e_step(data, means, covariances, weights)

        # M Step
        new_means, new_covariances, new_weights = m_step(data, probabilities)

        # Checking for convergence
        if np.allclose(means, new_means, atol=tol) and np.allclose(covariances, new_covariances, atol=tol) and np.allclose(weights, new_weights, atol=tol):
            break

        # Updating parameters for the next iteration
        means, covariances, weights = new_means, new_covariances, new_weights

    return means, covariances, weights
```

Calling the Function

```python
num_components = 4
num_iterations = 100

result_means, result_covariances, result_weights = em_algorithm(data, num_components, num_iterations)

print("Final means:", result_means)
print("Final covariances:", result_covariances)
print("Final weights:", result_weights)

# Report prior probability for each cluster
for i, weight in enumerate(result_weights):
    print(f"Cluster {i + 1} Prior Probability (Weight): {weight}")
```

Output

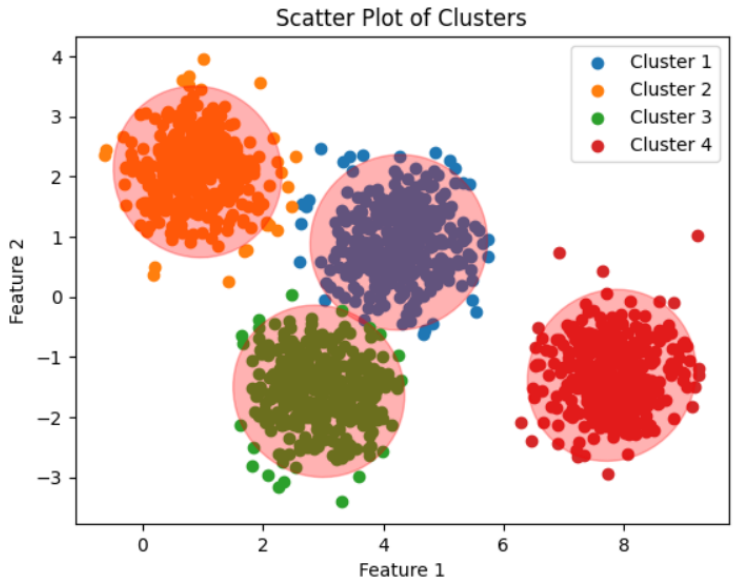
```
Final means: [[ 4.26057675  0.90602825]
 [ 0.91434393  2.07820265]
 [ 2.93190095 -1.56732569]
 [ 7.80187984 -1.30349482]]
Final covariances: [[[ 0.36216569  0.00658985]
  [ 0.00658985  0.35486411]]

 [[ 0.32684902 -0.01155883]
  [-0.01155883  0.33873281]]

 [[ 0.33908297 -0.01622286]
  [-0.01622286  0.34110245]]

 [[ 0.32790122  0.021395  ]
  [ 0.021395    0.33950129]]]
Final weights: [0.24831372 0.25383625 0.24790988 0.24994014]
Cluster 1 Prior Probability (Weight): 0.248313724139934
Cluster 2 Prior Probability (Weight): 0.2538362513754885
Cluster 3 Prior Probability (Weight): 0.24790988190841037
Cluster 4 Prior Probability (Weight): 0.24994014257616715
```

(a) Report the number of clusters and the technique used to find the number of clusters.
Ans : Total No. of Cluster obtained here is Four. To find the no. of Clusters Davies-Bouldin index is used.
Davis-Bouldin Index: 0.4555194724675402

(b) Plot the Cluster

```python
import matplotlib.pyplot as plt

def plot_clusters(data, means, covariances, probabilities):
    num_components = len(means)

    for i in range(num_components):
        cluster_points = data[probabilities[:, i] > 0.5]  # Adjust threshold as needed
        plt.scatter(cluster_points[:, 0], cluster_points[:, 1], label=f'Cluster {i + 1}')

        # Plot ellipses representing covariance matrices
        plot_ellipse(means[i], covariances[i])

    plt.title('Scatter Plot of Clusters')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.legend()
    plt.show()

def plot_ellipse(mean, covariance, color='red', label=None):
    eigvalues, eigvectors = np.linalg.eigh(covariance)
    angle = np.degrees(np.arctan2(*eigvectors[:, 0][::-1]))

    width, height = 2 * np.sqrt(5.991 * eigvalues)  # Use 95% confidence interval (chi-square value)
    ellip = plt.matplotlib.patches.Ellipse(mean, width, height, angle, color=color, alpha=0.3, label=label)
    plt.gca().add_patch(ellip)
```

```python
# Perform E-step to get cluster probabilities
probabilities = e_step(data, result_means, result_covariances, result_weights)

# Plot the clusters
plot_clusters(data, result_means, result_covariances, probabilities)
```



(c) Report the mean, co-variance matrix and prior probability
Final Means :
[[2.93192778 -1.56728869]
[0.91422164 2.07823437]
[4.26048033 0.90612579]
[7.80188278 -1.30350134]]
co-variance matrix :
[[[0.33909891 -0.01619297] [-0.01619297 0.3411371]]
[[0.32666005 -0.01151166] [-0.01151166 0.33872681]]
[[0.3623657 0.0064827] [0.0064827 0.35484151]]
[[0.32789954 0.02140085] [0.02140085 0.33948919]]]
Prior Probability :
Cluster 1 Prior Probability (Weight): 0.2479180255486487
Cluster 2 Prior Probability (Weight): 0.25381793163107746
Cluster 3 Prior Probability (Weight): 0.24832471423606858
Cluster 4 Prior Probability (Weight): 0.24993932858420528

(d) Assess the quality of the clusters.
Ans : silhouette score is a metric which is used to calculate the Quality of Clusters. If its value ranges from -1 to 1.1 then it means that clusters are well apart from each other and clearly distinguished.
Silhouette Score: 0.3412118896568931

Question:2
Apply model based clustering on ECG data [UCR Time Series Classification and Clustering website]. Describe the experimental analysis.

Reading the ECG data

```
data_2_test= pd.read_csv("/content/drive/MyDrive/AML/ASSIGNMENT 1/ECG 200/ECG200_TEST.csv", delimiter='\s+', header=None)
data_2_test
```

	0	1	2	3	4	5	6	7	8	9	...	87	88	89	90	91	92	93	94
0	1.0	0.425189	1.418599	2.668791	3.298629	2.264554	0.165179	-0.959727	0.282668	0.842850	...	-0.018196	-0.220493	-0.152557	-0.117011	-0.145160	-0.116668	-0.199262	-0.109837
1	1.0	0.653929	2.177290	3.644783	2.279203	0.978367	-0.388296	-0.911911	-0.148330	0.305439	...	0.099883	0.141395	-0.239092	0.247424	-0.085150	0.007800	0.407868	-0.194915
2	1.0	0.404953	0.553996	0.724097	1.449039	2.012616	1.624477	1.204726	1.117511	0.562099	...	1.297938	1.243332	1.463616	1.694916	1.456115	1.570372	1.934584	1.497556
3	1.0	1.088088	2.011949	2.301733	1.630199	0.636768	-0.140502	0.547954	1.075071	0.896861	...	0.125239	0.071349	0.138197	0.116399	0.131198	0.119412	0.082968	0.151348
4	-1.0	0.443621	0.947285	1.924084	2.159790	1.499447	0.964508	0.223256	-0.424946	0.121279	...	0.900511	1.278688	2.317878	2.916254	2.211947	2.034753	2.071302	1.254178
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	1.0	0.843834	2.564764	3.343127	2.591811	1.989914	0.441743	0.356408	1.283447	0.820812	...	-0.243969	-0.242760	-0.356488	-0.245672	-0.184878	-0.277344	-0.105103	-0.169774
96	1.0	1.398472	2.874771	3.453428	2.897919	1.260043	0.173332	1.289293	1.470054	1.028398	...	-0.406242	-0.525755	-0.439422	-0.398495	-0.366739	-0.399198	-0.401109	-0.371077
97	-1.0	1.113685	1.275951	1.171770	1.554551	2.280557	2.636614	2.514625	2.363170	2.290844	...	0.484357	0.474087	0.323893	0.387200	0.575901	0.426197	0.077527	0.148549
98	-1.0	2.318208	2.139772	1.794243	2.344508	1.724400	0.431501	-0.464933	-0.542414	-0.830620	...	-0.099292	-0.239948	-0.065850	-0.090294	-0.234900	-0.032727	-0.102668	-0.064539
99	1.0	2.395329	3.283697	2.918598	2.279072	1.393210	0.592044	0.044003	0.110276	0.198213	...	-0.049051	-0.099085	0.027991	0.141353	0.085650	0.005304	0.149875	0.103510

100 rows × 97 columns

```
data_2_train = pd.read_csv("/content/drive/MyDrive/AML/ASSIGNMENT 1/ECG 200/ECG200_TRAIN.txt", delimiter='\s+', header=None)
data_2_train
```

	0	1	2	3	4	5	6	7	8	9	...	87	88	89	90	91	92	93	94
0	-1.0	0.502055	0.542163	0.722383	1.428885	2.136516	2.281149	1.936274	1.468890	1.008845	...	0.931043	0.610298	0.638894	0.684679	0.583238	0.640522	0.708585	0.705011
1	1.0	0.147647	0.804668	0.367771	0.243894	0.026614	-0.274402	0.096731	-0.747731	-1.609777	...	-0.533503	-0.400228	0.176084	1.111768	2.438428	2.734889	1.736054	0.036857
2	-1.0	0.316646	0.243199	0.370471	1.063738	1.678187	1.759558	1.697717	1.612159	1.168188	...	0.764229	0.610621	0.552900	0.566786	0.604002	0.777068	0.812345	0.748848
3	-1.0	1.168874	2.075901	1.760141	1.606446	1.949046	1.302842	0.459332	0.516412	0.852180	...	0.419006	0.723888	1.323947	2.136488	1.746597	1.470220	1.893512	1.256949
4	1.0	0.648658	0.752026	2.636231	3.455716	2.118157	0.520620	-0.188627	0.780818	0.933775	...	-0.097869	-0.136787	-0.340237	-0.089441	-0.080297	-0.192584	-0.304704	-0.454556
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	1.0	0.581277	0.876188	1.042767	1.796120	2.541399	2.246653	1.500387	1.031521	0.382672	...	1.002770	0.907869	0.916457	0.923975	0.767357	0.656223	0.762357	0.501373
96	-1.0	2.689017	2.708703	2.008381	2.235800	1.516982	0.029916	-0.561346	-0.793702	-0.979371	...	-0.136610	-0.072176	-0.082738	-0.138468	-0.120396	-0.089411	-0.243141	-0.119710
97	-1.0	0.197677	0.455417	0.973110	1.935956	2.259463	1.741341	1.158296	0.418241	-0.071605	...	0.482452	0.325569	0.247991	0.184127	0.050358	0.241988	0.331451	-0.120006
98	1.0	0.179500	1.038409	1.946421	2.705141	1.670706	-0.101167	-1.578876	-0.750906	0.175310	...	0.324323	0.330489	0.111953	0.448948	0.567132	0.136757	0.444768	0.151050
99	1.0	0.073124	0.776054	2.181336	3.440325	2.168475	0.497315	-0.924284	-1.499227	-0.679328	...	-0.058935	-0.130638	-0.347235	-0.177933	-0.060332	-0.347634	-0.447443	-0.066689

100 rows × 97 columns

```
df_1 = pd.concat([data_2_test, data_2_train], ignore_index=True)
df_1
```

	0	1	2	3	4	5	6	7	8	9	...	87	88	89	90	91	92	93	94
0	1.0	0.425189	1.418599	2.668791	3.298629	2.264554	0.165179	-0.959727	0.282668	0.842850	...	-0.018196	-0.220493	-0.152557	-0.117011	-0.145160	-0.116668	-0.199262	-0.109837
1	1.0	0.653929	2.177290	3.644783	2.279203	0.978367	-0.388296	-0.911911	-0.148330	0.305439	...	0.099883	0.141395	-0.239092	0.247424	-0.085150	0.007800	0.407868	-0.194915
2	1.0	0.404953	0.553996	0.724097	1.449039	2.012616	1.624477	1.204726	1.117511	0.562099	...	1.297938	1.243332	1.463616	1.694916	1.456115	1.570372	1.934584	1.497556
3	1.0	1.088088	2.011949	2.301733	1.630199	0.636768	-0.140502	0.547954	1.075071	0.896861	...	0.125239	0.071349	0.138197	0.116399	0.131198	0.119412	0.082968	0.151348
4	-1.0	0.443621	0.947285	1.924084	2.159790	1.499447	0.964508	0.223256	-0.424946	0.121279	...	0.900511	1.278688	2.317878	2.916254	2.211947	2.034753	2.071302	1.254178
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
195	1.0	0.581277	0.876188	1.042767	1.796120	2.541399	2.246653	1.500387	1.031521	0.382672	...	1.002770	0.907869	0.916457	0.923975	0.767357	0.656223	0.762357	0.501373
196	-1.0	2.689017	2.708703	2.008381	2.235800	1.516982	0.029916	-0.561346	-0.793702	-0.979371	...	-0.136610	-0.072176	-0.082738	-0.138468	-0.120396	-0.089411	-0.243141	-0.119710
197	-1.0	0.197677	0.455417	0.973110	1.935956	2.259463	1.741341	1.158296	0.418241	-0.071605	...	0.482452	0.325569	0.247991	0.184127	0.050358	0.241988	0.331451	-0.120006
198	1.0	0.179500	1.038409	1.946421	2.705141	1.670706	-0.101167	-1.578876	-0.750906	0.175310	...	0.324323	0.330489	0.111953	0.448948	0.567132	0.136757	0.444768	0.151050
199	1.0	0.073124	0.776054	2.181336	3.440325	2.168475	0.497315	-0.924284	-1.499227	-0.679328	...	-0.058935	-0.130638	-0.347235	-0.177933	-0.060332	-0.347634	-0.447443	-0.066689

200 rows × 97 columns

```
df = df_1.iloc[:, 1:]
df
```

	1	2	3	4	5	6	7	8	9	10	...	87	88	89	90	91	92	93
0	0.425189	1.418599	2.668791	3.298629	2.264554	0.165179	-0.959727	0.282668	0.842850	0.683458	...	-0.018196	-0.220493	-0.152557	-0.117011	-0.145160	-0.116668	-0.199262
1	0.653929	2.177290	3.644783	2.279203	0.978367	-0.388296	-0.911911	-0.148330	0.305439	-0.230098	...	0.099883	0.141395	-0.239092	0.247424	-0.085150	0.007800	0.407868
2	0.404953	0.553996	0.724097	1.449039	2.012616	1.624477	1.204726	1.117511	0.562099	-0.011556	...	1.297938	1.243332	1.463616	1.694916	1.456115	1.570372	1.934584
3	1.088088	2.011949	2.301733	1.630199	0.636768	-0.140502	0.547954	1.075071	0.896861	1.137943	...	0.125239	0.071349	0.138197	0.116399	0.131198	0.119412	0.082968
4	0.443621	0.947285	1.924084	2.159790	1.499447	0.964508	0.223256	-0.424946	0.121279	0.830197	...	0.900511	1.278688	2.317878	2.916254	2.211947	2.034753	2.071302
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
195	0.581277	0.876188	1.042767	1.796120	2.541399	2.246653	1.500387	1.031521	0.382672	-0.197561	...	1.002770	0.907869	0.916457	0.923975	0.767357	0.656223	0.762357
196	2.689017	2.708703	2.008381	2.235800	1.516982	0.029916	-0.561346	-0.793702	-0.979371	-1.288162	...	-0.136610	-0.072176	-0.082738	-0.138468	-0.120396	-0.089411	-0.243141
197	0.197677	0.455417	0.973110	1.935956	2.259463	1.741341	1.158296	0.418241	-0.071605	-0.241250	...	0.482452	0.325569	0.247991	0.184127	0.050358	0.241988	0.331451
198	0.179500	1.038409	1.946421	2.705141	1.670706	-0.101167	-1.578876	-0.750906	0.175310	0.064901	...	0.324323	0.330489	0.111953	0.448948	0.567132	0.136757	0.444768
199	0.073124	0.776054	2.181336	3.440325	2.168475	0.497315	-0.924284	-1.499227	-0.679328	-0.549892	...	-0.058935	-0.130638	-0.347235	-0.177933	-0.060332	-0.347634	-0.447443

200 rows × 96 columns

```
n_attributes = 96
X = df.iloc[:, 1:n_attributes+1].values
X
```

```
array([[ 1.4185988 ,  2.6687913 ,  3.298629  , ..., -0.10983685,
        -0.14248753,  0.09875814],
       [ 2.1772899 ,  3.6447828 ,  2.2792033 , ..., -0.19491519,
         0.17110719,  0.19702731],
       [ 0.55399598,  0.72409707,  1.4490394 , ...,  1.4975555 ,
         0.81257   ,  0.39063926],
       ...,
       [ 0.45541658,  0.97310983,  1.9359564 , ..., -0.1200061 ,
         0.04242264,  0.34329344],
       [ 1.0384092 ,  1.9464212 ,  2.7051409 , ...,  0.15104997,
         0.19337751,  0.45170853],
       [ 0.77605406,  2.181336  ,  3.4403251 , ..., -0.06668896,
        -0.17844792, -0.25605159]])
```

Here PCA is used for Dimensionality Reduction

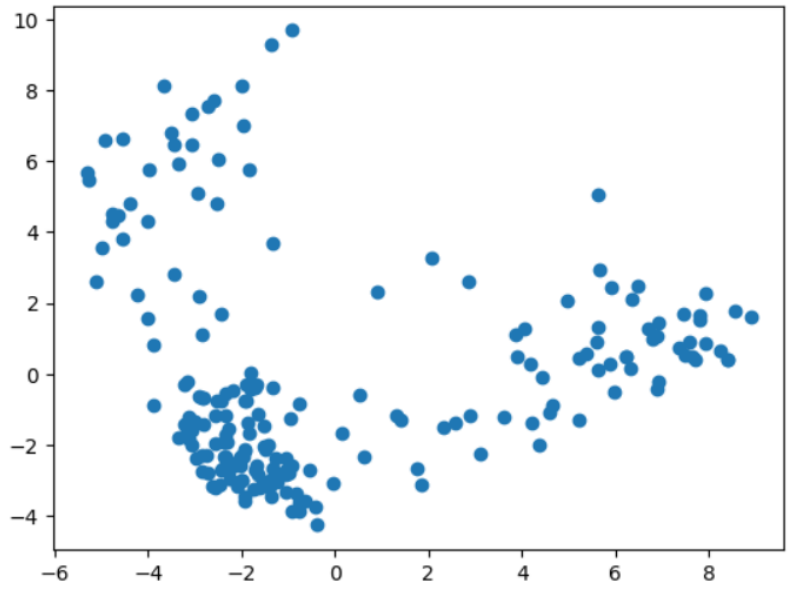
```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
X_pca
```

```
array([[-2.08864412, -3.15330513],
       [-3.21416166, -1.41627931],
       [ 6.46903499,  2.46845871],
       [-1.43942867, -2.01220175],
       [ 5.90893541,  2.42702621],
       [-2.02849598, -2.31344488],
       [ 4.0649022 ,  1.26486873],
       [-2.4277669 , -0.76454045],
       [-1.68359897, -2.57592532],
       [ 7.70921214,  0.39418866],
       [-4.01710207,  1.58668168],
       [-3.51020001,  6.81660081],
       [-1.8436604 ,  5.77513463],
       [ 6.34650254,  2.11711675],
       [ 5.62178616,  1.33581941],
       [ 4.64720688, -0.88500823],
       [-2.9911373 , -1.3412686 ],
       [-1.79823578,  0.02149346],
       [-4.77378296,  4.3187904 ],
       [-4.00377815,  4.30018531],
       [-2.00812586,  8.13872648],
       [-2.8331111 , -2.73020445],
       [-2.79530587, -0.65855417],
       [-2.02825086, -2.59775918],
```

Visulalizing the Data

```
plt.scatter(X_pca[:, 0], X_pca[:, 1], cmap='viridis')
```

```
<matplotlib.collections.PathCollection at 0x7ba16a960f70>
```



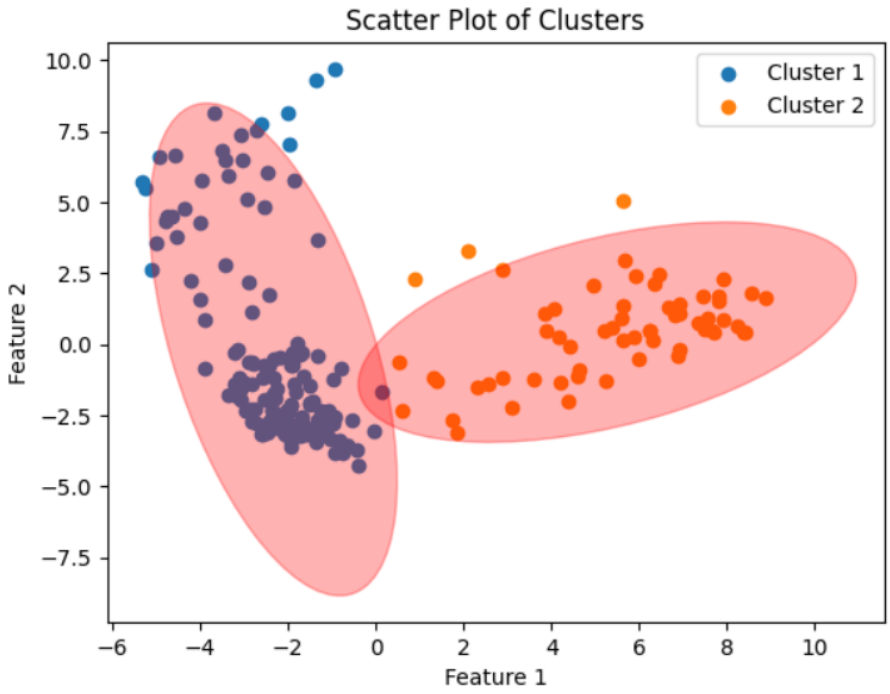
Calling the Function

```
num_components = 2
num_iterations = 100

result_means, result_covariances, result_weights = em_algorithm(X_pca, num_components, num_iterations)

print("Final means:", result_means)
print("Final covariances:", result_covariances)
print("Final weights:", result_weights)

# Report prior probability for each cluster
for i, weight in enumerate(result_weights):
    print(f"Cluster {i + 1} Prior Probability (Weight): {weight}")
```

```
Final means: [[-2.33002611 -0.18620722]
 [ 5.27393031  0.42147334]]
Final covariances: [[[ 1.32487182 -2.18273359]
  [-2.18273359 12.5311902 ]]

 [[ 5.36168006  1.73567407]
  [ 1.73567407  2.50160078]]]
Final weights: [0.69357714 0.30642286]
Cluster 1 Prior Probability (Weight): 0.6935771356280422
Cluster 2 Prior Probability (Weight): 0.3064228643719578
```

Plotting the Clusters

```
# Perform E-step to get cluster probabilities
probabilities = e_step(X_pca, result_means, result_covariances, result_weights)

# Plot the clusters
plot_clusters(X_pca, result_means, result_covariances, probabilities)
```



Experimental Analysis:
Final means:
[[1.96854274 2.15577442] [-1.9576938 -2.14389363]]
Final covariances:
[[[21.40193448 -8.0388772] [-8.0388772 8.74481797]]
[[0.6482789 -0.42285977] [-0.42285977 1.10554083]]]
Silhouette Score: 0.44578048723997227
Davies-Bouldin Index: 1.0952402810229847
Cluster 1 Prior Probability (Weight): 0.49861840500618804
Cluster 2 Prior Probability (Weight): 0.501381594993812