Homework 07 – Expectation-Maximization Clustering

Importing Data Set

- After importing NumPy, Pandas, and Matplotlib libraries, and spatial and multivariate_normal functions from the SciPy library, I've imported the data set and initial centroids using Pandas' read_csv() function.

Algorithm Steps

- First, I have defined three functions: initial_memberships, initial_covariance_matrices, and initial_prior_probabilities.

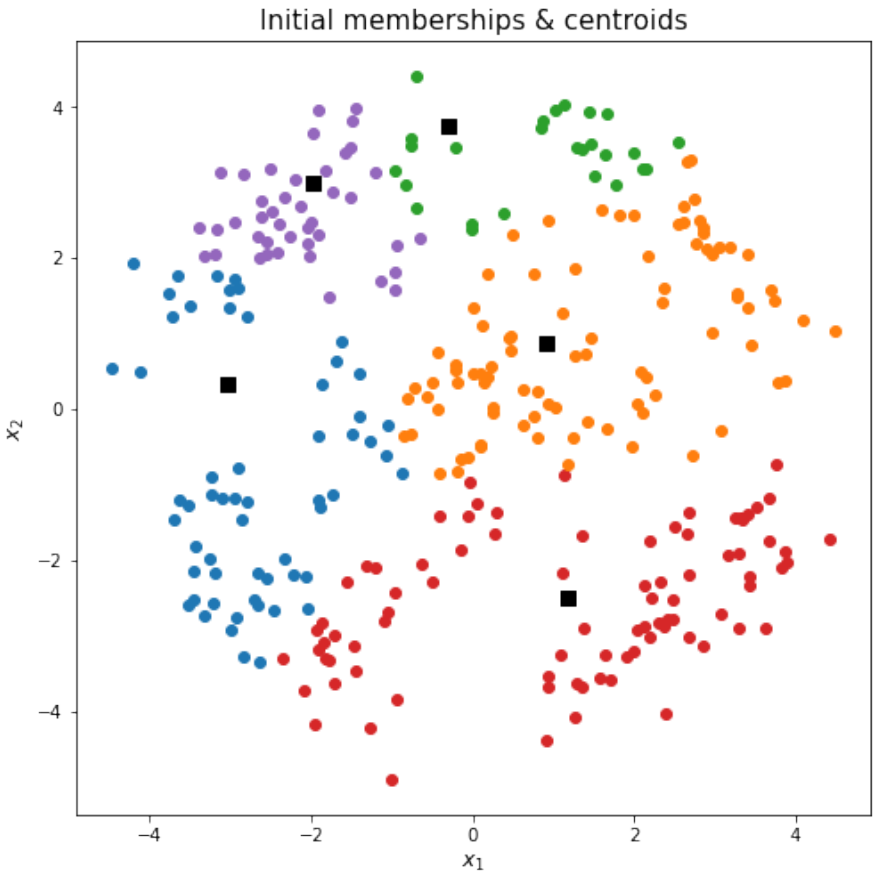
```
def initial_memberships(centroids, X):
    D = spa.distance_matrix(centroids, X)
    memberships = np.argmin(D, axis = 0)
    return(memberships)

def initial_covariance_matrices(memberships, centroids, X, K):
    covariance_matrices = [np.sum([(X[memberships == k][i] - centroids[k]).reshape(2, 1) @ (X[memberships == k][i] - centroids[k]
    return(covariance_matrices)

def initial_prior_probabilities(memberships, X, N):
    prior_probabilities = np.array(pd.Series(memberships).value_counts().sort_index()/N)
    return(prior_probabilities)
```

- Using these functions, I found the initial memberships by assigning the data points to the nearest centroid and estimated initial covariance matrices and prior probabilities.

```
memberships = initial_memberships(centroids, X)
covariance_matrices = initial_covariance_matrices(memberships, centroids, X, K)
prior_probabilities = initial_prior_probabilities(memberships, X, N)
```



Expectation-Maximization (EM) Algorithm

- For the EM algorithm, I have used the following formulas from the lecture notes:

E-STEP:

$$h_{ik} = E\left[z_{ik} \mid X, \Phi^{(t)}\right] = \frac{p(x_i|C_k,\Phi^{(t)})P(C_k)}{\sum_{c=1}^{K} p(x_i|C_c,\Phi^{(t)})P(C_c)}$$

M-STEP:

$$\hat{\mu}_k^{(t+1)} = \frac{\sum_{i=1}^{N} h_{ik} x_i}{\sum_{i=1}^{N} h_{ik}}$$

$$\hat{\sigma}_k^{(t+1)} = \frac{\sum_{i=1}^{N} h_{ik}(x_i - \hat{\mu}_k^{(t+1)})(x_i - \hat{\mu}_k^{(t+1)})^T}{\sum_{i=1}^{N} h_{ik}}$$

$$\hat{P}(C_K) = \frac{\sum_{i=1}^{N} h_{ik}}{N}$$

- I have defined the function for the calculation of h_{ik} in three steps; first one calculates the $p(x_i|C_k, \varphi^{(t)})$, second one calculates the numerator of h_{ik}, which is the mixture density, and the last one finally calculates h_{ik}.

```
def multivariate_gaussian(x, mean, covariance):
    return (1. / (np.sqrt((2 * np.pi)**2 * np.linalg.det(covariance))) *  np.exp(-(np.linalg.solve(covariance, x - mean).T.dot(x
def mixture_density(x, mean, covariance, prior_probability):
    return multivariate_gaussian(x, mean, covariance) * prior_probability
def h_ik(x, centroids, covariance_matrices, prior_probabilities):
    return [mixture_density(x, centroids[k], covariance_matrices[k], prior_probabilities[k]) / np.sum([mixture_density(x, centroi
```

Expectation-Maximization (EM) Algorithm: E-Step

- For the E-step, I have defined a function called find_memberships, which uses h_ik function, and then assigns the data point to the cluster with the highest likelihood.

```
def find_memberships(X, centroids, covariance_matrices, prior_probabilities):
    memberships = []

    for m in range(300):
        posterior_probabilities = h_ik(X[m], centroids, covariance_matrices, prior_probabilities)
        max_value = max(posterior_probabilities)
        max_index = posterior_probabilities.index(max_value)
        memberships.append(max_index)
    return np.array((memberships))
```

Expectation-Maximization (EM) Algorithm: M-Step

- For the M-step, again, I have defined three functions that updates centroids, covariance matrices and prior probabilities:

```
def update_centroids(memberships, X):
    return np.vstack([np.sum([h_ik(X[memberships == k][i], centroids, covariance_matrices, prior_probabilities)[k] * X[membershi

def update_covariance_matrices(memberships, centroids, X, K):
    return [np.sum([h_ik(X[memberships == k][i], centroids, covariance_matrices, prior_probabilities)[k] * (X[memberships == k][i

def update_prior_probabilities(memberships, X, N):
    return np.array(pd.Series(memberships).value_counts().sort_index()/N)
```

Running EM Algorithm

- After defining the necessary functions, I ran the EM algorithm for 100 times, each time updating the memberships, centroids, covariance matrices and prior probabilities. Also, I have plotted the memberships and centroids at each iteration.

```
for iteration in range(100):
    memberships = find_memberships(X, centroids, covariance_matrices, prior_probabilities)
    centroids = update_centroids(memberships, X)
    covariance_matrices = update_covariance_matrices(memberships, centroids, X, K)
    prior_probabilities = update_prior_probabilities(memberships, X, N)


    print(f"Iteration#:{iteration+1}")

    for i in range(5):
        plt.scatter(X[memberships == i][:, 0], X[memberships == i][:, 1], color=colors[i])
        plt.scatter(centroids[i][0], centroids[i][1], color="k", marker="s", s=50)
    plt.show()
```

- The mean vectors my EM algorithm have found after 100 iterations are:

```
Mean vectors after 100 iterations:

[[-2.25320393 -2.40773298]
 [ 2.42091488  2.40690293]
 [ 2.51541881 -2.5153351 ]
 [ 0.26442888 -0.00693325]
 [-2.07993059  2.38274832]]
```

Visualization

- Lastly, I have visualized the clustering results, along with the original Gaussian densities with dashed lines and the Gaussian densities EM algorithm found as normal lines.