Eren Barış Bostancı
68770

ENGR 421 – HW7

Data Importing:

I imported the data and set the given class parameters, which are means, covariances, and class sizes. Also plotted the imported data.

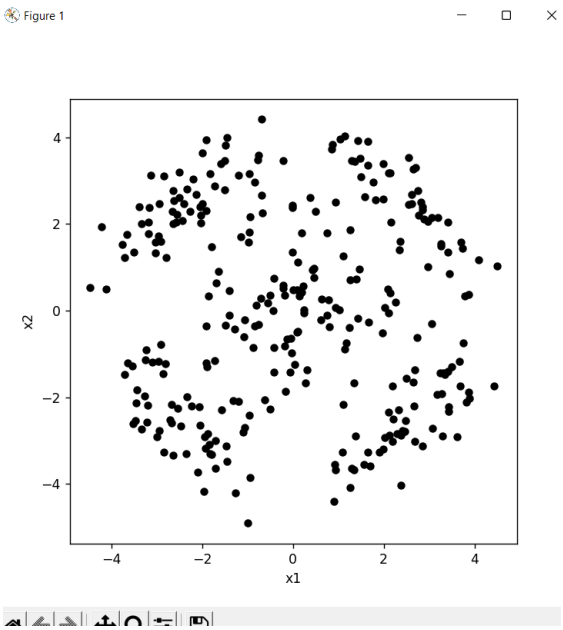
```
X = np.genfromtxt("hw07_data_set.csv", delimiter=",")

N = X.shape[0]
D = X.shape[1]
K = 5

class_means = np.array([[+2.5, +2.5],
                        [-2.5, +2.5],
                        [-2.5, -2.5],
                        [+2.5, -2.5],
                        [+0.0, +0.0]])

class_covariances = np.array([[[+0.8, -0.6],
                               [-0.6, +0.8]],
                              [[+0.8, +0.6],
                               [+0.6, +0.8]],
                              [[+0.8, -0.6],
                               [-0.6, +0.8]],
                              [[+0.8, +0.6],
                               [+0.6, +0.8]],
                              [[+1.6, +0.0],
                               [+0.0, +1.6]]])
class_sizes = np.array([50, 50, 50, 50, 100])
```

```
plt.figure(figsize=(6, 6))
plt.plot(X[:, 0], X[:, 1], "k.", markersize=10)
plt.xlabel("x1")
plt.ylabel("x2")
plt.show()
```



Also, I take the update_centroids, update_memberships, and plot_current_state functions from lab11 to use them. I modified update_centroids method to return the imported given initial centroids from csv file for initial call. Also modified the plot_current_state to not put a square for each class on the plot.

```
def update_centroids(memberships, X):
    if memberships is None:
        # initialize centroids
        centroids = np.genfromtxt("hw07_initial_centroids.csv", delimiter=",")
    else:
        # update centroids
        centroids = np.vstack([np.mean(X[memberships == k, :], axis=0) for k in range(K)])
    return centroids


def update_memberships(centroids, X):
    # calculate distances between centroids and data points
    D = spa.distance_matrix(centroids, X)
    # find the nearest centroid for each data point
    memberships = np.argmin(D, axis=0)
    return memberships
```

```
def plot_current_state(centroids, memberships, X):
    cluster_colors = np.array(["#1f78b4", "#33a02c", "#e31a1c", "#ff7f00", "#6a3d9a", "#b15928",
                               "#a6cee3", "#b2df8a", "#fb9a99", "#fdbf6f", "#cab2d6", "#ffff99"])
    if memberships is None:
        plt.plot(X[:, 0], X[:, 1], ".", markersize=10, color="black")
    else:
        for c in range(K):
            plt.plot(X[memberships == c, 0], X[memberships == c, 1], ".", markersize=10,
                     color=cluster_colors[c])

    plt.xlabel("x1")
    plt.ylabel("x2")
```

Then, I set the initial values of centroids, memberships, sample_means, sample_covariance, and priors.

```
memberships = None
centroids = update_centroids(memberships, X)
memberships = update_memberships(centroids, X)

sample_means = centroids
sample_covariances = [None, None, None, None, None]
priors = [None, None, None, None, None]

for i in range(K):
    sample_covariances[i] = np.eye(D)
    priors[i] = class_sizes[i] / K
```

Eren Barış Bostancı
68770

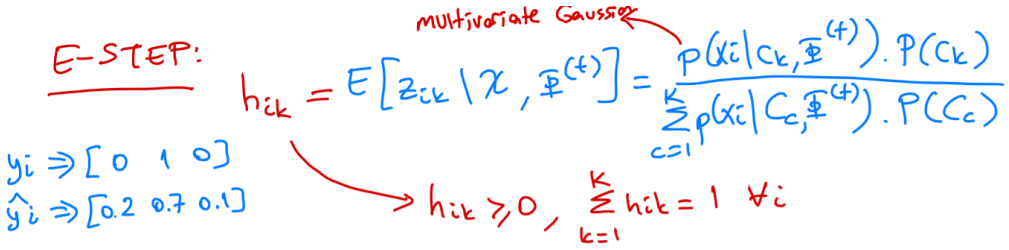
Expectation-Maximization Clustering

I defined E_Step and M_Step functions for the implementation.

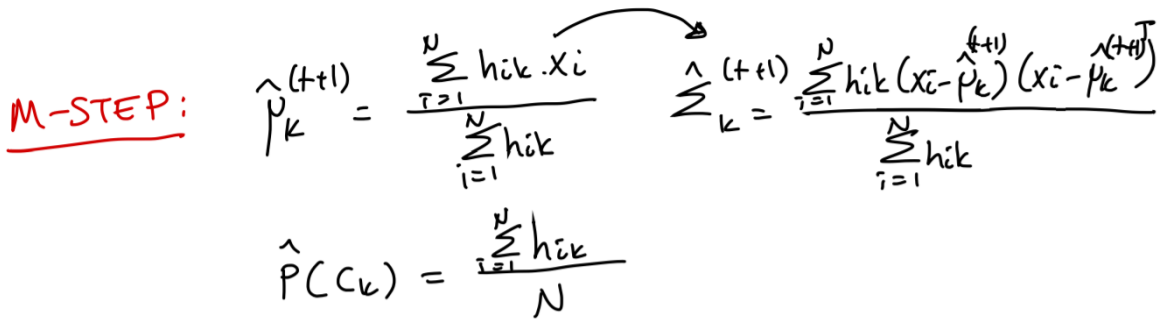
```
def E_Step(h, X, means, covariances, k):
    for n in range(N):
        h[n][k] = (stats.multivariate_normal.pdf(X[n], means[k], covariances[k]) * priors[k]) / np.sum(
            [stats.multivariate_normal.pdf(X[n], means[i], covariances[i]) * priors[i] for i in range(K)])
    return h


def M_Step(X, h, means, priors, covariances, k):
    priors[k] = np.sum(h[:, k]) / N
    means[k] = h[:, k].dot(X) / np.sum(h[:, k])
    covariances[k] = sum((h[n, k] * np.matrix((X[n] - means[k])).T.dot(np.matrix(X[n] - means[k]))) for n in range(N)) / np.sum(h[:, k])
    return means, priors, covariances
```

In E_Step function I update the given hik matrix according to the given means and covariance. For the formula I used the formula that given in lecture notes. I take the k index as an parameter and only update the n indexes k values on each call, since I iterate k value during the call of this method inside the 100 times iteration.



In M_Step, I updated the sample_means, sample_covariances, priors according to the given h matrix and sample_means, sample_covariances, priors. While updating the data I used the formula that given in lecture notes.



Then I iterate the 100 times and update the h matrix in inside iteration for each class and with E_Step function, and I updated the sample_means, priors, sample_covariances with the M_Step function. After all of them I update the memberships for the final version.

```
for i in range(100):
    print("Iteration: " + str(i + 1))
    h = np.zeros((N, K))
    for k in range(K):
        h = E_Step(h, X, sample_means, sample_covariances, k)
        sample_means, priors, sample_covariances = M_Step(X, h, sample_means, priors, sample_covariances, k)

memberships = update_memberships(sample_means, X)
print("sample_means:\n")
print(sample_means)
```

Then I printed sample_means. They are really close with the pdf's output.

```
sample_means:

[[-2.44390017 -2.54539396]
 [ 2.5035433   2.51134854]
 [ 2.56726403 -2.55477256]
 [ 0.12794703  0.15595816]
 [-2.41465309  2.4855615 ]]
```

Finally, I plotted.

```
plt.figure(figsize=(6, 6))
plot_current_state(centroids, memberships, X)
x, y = np.meshgrid(np.linspace(-6, 6, 200), np.linspace(-6, 6, 200))
coordinates = np.empty(x.shape + (2,))
coordinates[:, :, 0] = x
coordinates[:, :, 1] = y
for i in range(K):
    pdf_i = stats.multivariate_normal.pdf(coordinates, class_means[i], class_covariances[i])
    pdf_f = stats.multivariate_normal.pdf(coordinates, sample_means[i], sample_covariances[i])
    plt.contour(x, y, pdf_i, linestyles='dashed', levels=[0.05])
    plt.contour(x, y, pdf_f, levels=[0.05])
plt.show()
```