# Homework 4

201900800302 赵淇涛

## 1. K-means

### Overview

Clustering is one of unsupervised machine learning problems.

In this part, we would implement the **k-means** algorithm to classify three sets of data points centered distinctively obeying 2-dimensional Gaussian distribution, where k indicates the number of the clusters we would like to sort data points into. Here k is set to 3 because we have already known the underlying distribution of data points and hopefully data points would be classified into 3 classes.

Our goal is to assign all data points to one of 3 cluster centers respectively and try to reduce the sum of distance between data points and their assigned center.

First, 3 cluster centers are randomly chosen from all points. Then, all data points are assigned to their nearest center. To optimize the position of center within its cluster, each center would be set to the average of all points belonging to this cluster. Since the position of centers have changed, we should do assignment again and further update the position of centers... We keep doing so until the total distance converges.

Note that each step is local optimal and therefore leads to a local optimal solution. The performance of the clustering is affected by the orginal distribution (sometimes data points intertwine and cannot be perfectly classified into 3 sets), the choice of first 3 cluster centers.

## 2. EM algorithm

### Overview

A **Gaussian mixture model** is introduced this part to solve the clustering problem, which is a linear combination of several Gaussian distributions.

Each Gaussian distribution has its own mean, covariance and is assgined a weight. In the training process, these parameters would be updated to maximize a log likelihood that each point is generated at its position given parameters of Gaussian distributions.

Optimizing such a log likelihood directly is a hard problem. Therefore, **EM** algorithm is introduced to split the optimization process into two steps: E (expectation) and M (maximization). In the E step we use given parameters to evaluate the posterior probabilities that the data points belong to each Gaussian distribution provided their positions and re-estimate the parameters of each Gaussian distribution in the M step.

To compare the performance of the k-means with that of the EM algorithm, initial centers of each cluster (means of Gaussian distributions) are the same. And the covairance of each Gaussian distribution is set to the identity matrix.

## Algorithm in detail

The posterior probability of **z** given **x** is shown below, where **z** is a label vector indicating the index of Gaussian distribution to which data point **x** belongs, $\pi_k$ is the prior probability of $z_k = 1$, i.e., the weight, which is set to 1/k initially.

$$\gamma(z_k) = p(z_k = 1|\mathbf{x}) = \frac{p(z_k = 1)p(\mathbf{x}|z_k = 1)}{\sum\limits_{j=1}^{K} p(z_j = 1)p(\mathbf{x}|z_j = 1)}$$
$$= \frac{\pi_k N(\mathbf{x}|\mu_k, \Sigma_k)}{\sum\limits_{j=1}^{K} \pi_j N(\mathbf{x}|\mu_j, \Sigma_j)}$$

The optimization objective (log likelihood function) reads

$$\ln p(\mathbf{X}|\pi, \mu, \Sigma) = \sum_{n=1}^{N} \ln\{\sum_{k=1}^{K} \pi_k N(\mathbf{x}_n|\mu_k, \Sigma_k)\}$$

**E step.** Evaluate the posterior probabilities (responsibilities) using the current parameter values

$$\gamma(z_{nk}) = \frac{\pi_k N(\mathbf{x}_n|\mu_k, \Sigma_k)}{\sum\limits_{j=1}^{K} \pi_j N(\mathbf{x}_n|\mu_j, \Sigma_j)}.$$

**M step**. Re-estimate the parameters using the current responsibilities.

$$\mu_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})\mathbf{x}_n$$
$$\Sigma_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(\mathbf{x}_n - \mu_k^{new})(\mathbf{x}_n - \mu_k^{new})^T$$
$$\pi_k^{new} = \frac{N_k}{N}$$

where

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk}).$$

We would do **E step** and **M step** alternatively until the log likelihood is maximized.

## Code and process

0. Set up

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

1. Function *generate_data* to produce data points

```python
def generate_data():
    X_A = np.random.multivariate_normal((0,0), [[1,0],[0,1]], 30)
    X_B = np.random.multivariate_normal((1,2), [[2,0],[0,1]], 20)
    X_C = np.random.multivariate_normal((2,0), [[1,0.3],[0.3,1]], 20)
    X = np.concatenate((X_A, X_B, X_C), axis=0)
    y = np.ones(30 + 20 + 20)
    y[30:] += 1
    y[50:] += -2
    return X, y
```

2. Function *k_means*

```python
def k_means(k, X, epoch):
    N, D = X.shape
    rng = np.random.default_rng(2050)
    centers = rng.choice(X, size=k, replace=False, p=None, axis=0)
    y = np.zeros(N)
    loss_history = []
    print("K-means:")

    for i in range(epoch):
        distance_matrix = np.zeros((N, k, D))
        distance_matrix = (distance_matrix.transpose(1,0,2) + X).transpose(1,0,2)
        distance_matrix -= centers
        distance_matrix = np.sqrt((distance_matrix ** 2).sum(axis=2))
        y = np.argmin(distance_matrix, axis=1)
        loss = distance_matrix[np.arange(N), y].mean()

        if len(loss_history) > 0:
            if abs(loss-loss_history[-1]) < 1e-3:
                break

        loss_history.append(loss)

        for j in range(k):
            mask = np.where(y==j)[0]
            centers[j, :] = np.mean(X[mask, :], axis=0)

        if i % 1 == 0:
            print(" Epoch: %d, loss: %.4f" % (i, loss))

    return y, centers
```

3. Function *EM*

```python
def EM(k, X, epoch):
    N, D = X.shape
    rng = np.random.default_rng(2050)
    means = rng.choice(X, size=k, replace=False, p=None, axis=0)
    covars = np.identity(D)
    mixing_coeffs = np.zeros(k) + 1/k # start uniformly
    mul_normals = []
    log_likelihoods = []
    print("EM algorithm:")

    temp = np.zeros((N, k))
    for i in range(k):
        mul_normals.append(multivariate_normal(means[i], covars))
        temp[:, i] = mul_normals[i].pdf(X)

    for i in range(epoch):

        # E step
        temp *= mixing_coeffs
        log_likelihood = (np.log((temp).sum(axis=1))).sum()
        log_likelihoods.append(log_likelihood)
        gamma = (temp.T / temp.sum(axis=1)).T # (N, k)
        N_k = gamma.sum(axis=0) # (k, )

        # M step
        means = np.ones((N, k, D))
        means = (means.transpose(1,0,2) * X).transpose(2,1,0) * gamma / N_k # (D, N, k)
        means = means.transpose(2,0,1).sum(axis=2)

        covars = np.zeros((k, D, D))
        for j in range(k):
            covars[j, :, :] = ((X-means[j]).T * gamma[:, j]).dot(X-means[j]) / N_k[j]

        mixing_coeffs = N_k / N

        # Evaluate the log likelihood
        for j in range(k):
            mul_normals[j] = multivariate_normal(means[j], covars[j])
            temp[:, j] = mul_normals[j].pdf(X)

        if i % 1 == 0:
            print(" Epoch: %d, log_likelihood: %.4f" % (i, log_likelihood))

        if i > 0:
            if abs(log_likelihoods[-1] - log_likelihoods[-2]) < 1e-3:
                break

    return gamma
```

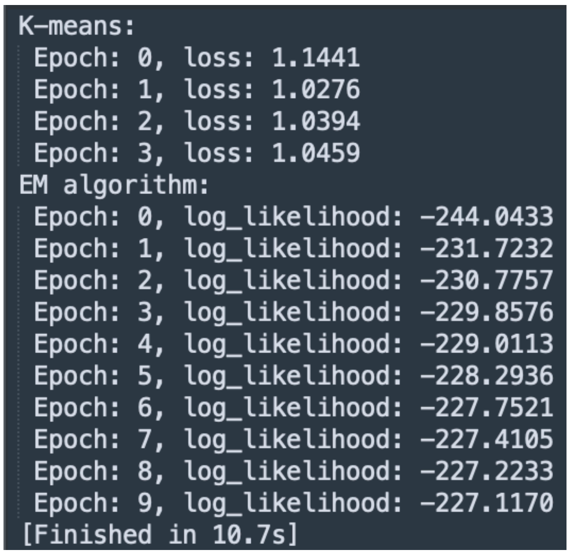
4. Main part

```
np.random.seed(2050) # 2050
```

```
X, y = generate_data()
y_predicted_1, _ = k_means(3, X, 10)
log_likelihood = EM(3, X, 10)
y_predicted_2 = np.argmax(log_likelihood, axis=1)

plt.figure(figsize=(7, 9))

plt.subplot(3, 1, 1)
plt.scatter(X[np.where(y==0)[0], 0], X[np.where(y==0)[0], 1], c='r', label="A")
plt.scatter(X[np.where(y==1)[0], 0], X[np.where(y==1)[0], 1], c='g', label="B")
plt.scatter(X[np.where(y==2)[0], 0], X[np.where(y==2)[0], 1], c='b', label="C")
plt.title("Original distribution")
plt.legend(loc="best", ncol=4)

plt.subplot(3, 1, 2)
plt.scatter(X[np.where(y_predicted_1==0)[0], 0], X[np.where(y_predicted_1==0)[0], 1],
c='r', label="A")
plt.scatter(X[np.where(y_predicted_1==1)[0], 0], X[np.where(y_predicted_1==1)[0], 1],
c='g', label="B")
plt.scatter(X[np.where(y_predicted_1==2)[0], 0], X[np.where(y_predicted_1==2)[0], 1],
c='b', label="C")
plt.title("K-means, accuracy: %.2f%%" % (np.sum(y_predicted_1==y)/70*100))
plt.legend(loc="best", ncol=4)

plt.subplot(3, 1, 3)
plt.scatter(X[np.where(y_predicted_2==0)[0], 0], X[np.where(y_predicted_2==0)[0], 1],
c='r', label="A")
plt.scatter(X[np.where(y_predicted_2==1)[0], 0], X[np.where(y_predicted_2==1)[0], 1],
c='g', label="B")
plt.scatter(X[np.where(y_predicted_2==2)[0], 0], X[np.where(y_predicted_2==2)[0], 1],
c='b', label="C")
plt.title("EM algorithm, accuracy: %.2f%%" % (np.sum(y_predicted_2==y)/70*100))
plt.legend(loc="best", ncol=4)

plt.subplots_adjust(hspace=0.25, top=0.95, bottom=0.05)
plt.show()
```

## Result and discussion

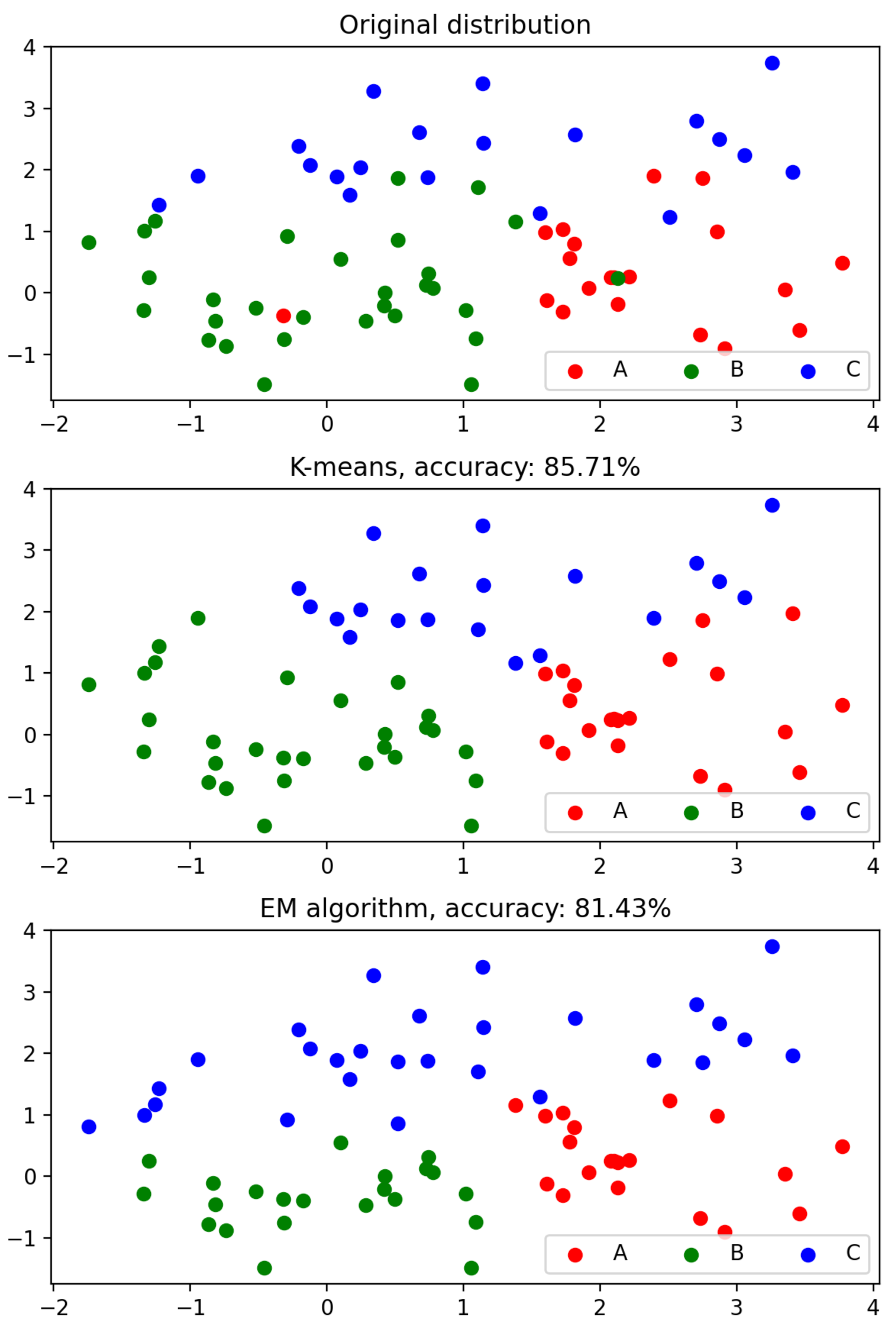
1. Training process



```
K-means:
 Epoch: 0, loss: 1.1441
 Epoch: 1, loss: 1.0276
 Epoch: 2, loss: 1.0394
 Epoch: 3, loss: 1.0459
EM algorithm:
 Epoch: 0, log_likelihood: -244.0433
 Epoch: 1, log_likelihood: -231.7232
 Epoch: 2, log_likelihood: -230.7757
 Epoch: 3, log_likelihood: -229.8576
 Epoch: 4, log_likelihood: -229.0113
 Epoch: 5, log_likelihood: -228.2936
 Epoch: 6, log_likelihood: -227.7521
 Epoch: 7, log_likelihood: -227.4105
 Epoch: 8, log_likelihood: -227.2233
 Epoch: 9, log_likelihood: -227.1170
[Finished in 10.7s]
```

Note that the k-means algorithm converges much faster than EM algorithm.

2. Result plot



The first plot shows the original distribution of test samples and the following two show the predicted results generated by k-means and EM algorithm respectively.

K-means and the EM algorithm yield similar results and the k-means performs slightly better. Since the performance of the clustering is affected by several factors, many random seeds have been tried and one that shows a good result is fixed.

In practice, we would like to first run k-means algorithm to get a reasonable initialization for means of Gaussian distributions and then do EM algotirhm to attain a final solution.