哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称： 机器学习

课程类型： 选修

实验题目： k-means 聚类方法和 GMM

学号：1190201421
姓名： 张瑞

一、实验目的

实现一个 k-means 算法和混合高斯模型，并且用 EM 算法估计模型中的参数。

二、实验要求及实验环境

（一）实验要求

1. 用高斯分布产生 k 个高斯分布的数据（不同均值和方差）。
2. 用 k-means 聚类，测试效果。
3. 用混合高斯模型和实现的 EM 算法估计参数，看看每次迭代后似然值变化情况，考察 EM 算法是否可以获得正确的结果（与设定的结果比较）。
4. 在 UCI 上找一个简单问题数据，用实现的 GMM 进行聚类。

（二）实验环境

Windows 10; PyCharm Community Edition 2021.2;Python 3.6

三、设计思想

（一）EM 算法

无论是 k-means 算法还是 GMM 模型，其背后的思想都是 EM 算法，而 EM（Expectation-Maximization）算法正如他的名字一样，分为两个步骤：

第一步（E 步）：求期望值（完全数据的对数似然函数对于未观测数据的条件概率分布的期望，也就是 Q 函数）；

第二步（M 步）：求使得期望值表达式最大化的参数。

不断重复上面这两步骤直到参数收敛。

（二）k-means 算法

给定样本$X = \{x_1, x_2, \dots, x_n\}$和划分聚类的数量$k$，给出一个类划分$C = \{C_1, C_2, \dots, C_k\}$使得该划分的误差$E$最小化，$E$如下所示：

$$E = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - \mu_i||_2^2$$

其中，$\mu_i = \frac{1}{|C_i|} \sum_{x_i \in C_i} x_i$ ，它是类C_i的中心。E刻画了类中样本围绕类的中心的紧密程度，E越小类中样本的相似度越高。

具体迭代过程如下：

1. 根据输入的类总数k先初始化一些类中心（本实验从现有的数据中挑选）。
2. 根据初始化的中心给出所有样本的一个划分：计算各个样本到各个类中心的距离，将该样本分至距离最近的类中心所代表的类。
3. 根据新的类划分，重新计算每个类的类中心，并进行样本的划分，如果新的划分结果与旧的划分结果相同，则认为聚类中心收敛；否则回到第 2 步继续迭代求解。

（三）GMM

GMM 即混合高斯模型，是一种用来描述样本分布的模型，可以很好地描述一组由多个高斯模型产生的样本。

GMM 算法和 k-means 不同的是，在 E 步时并没有使用“最近距离”来给每个样本赋予类别（硬赋值），而是增加了变量γ。γ为$n \times k$的矩阵（其中 n 为样本个数，k 为聚类的类数），γ_{nk}代表第 n 个样本属于第 k 类的概率，γ具有归一性。

给定样本$X = \{x_1, x_2, \dots, x_n\}$，对于一个样本$x_i$，可以认为它是由多个对应维度的多元高斯分布所生成，所以用这些高斯分布的线性叠加来表征该样本，假设数据由 k 个高斯分布混合生成，则有：

$$p(x_i) = \sum_{j=1}^{k} \pi_j p(x_i|\mu_j, \Sigma_j)$$

其中μ_j和Σ_j分别为第 j 个高斯分布的均值和协方差矩阵，π_j为样本属于第 j 类的概率，满足:

$$\sum_{j=1}^{k} \pi_j = 1$$

也可以认为该样本的生成相当于从 k 个高斯分布中挑选出一个所生成，设 k 维二值变量 z，该变量采用“1-of-k”表示方法，其中一个元素z_j为 1，其余元素均为 0。于是z_j满足:

$$z_j \in \{0,1\}$$

$$\sum_{j} z_j = 1$$

于是 z 的先验分布为：

$$p(z) = \prod_{j=1}^{k} \pi_j^{z_j}$$

在已知x_i的情况下 z 的后验概率为：

$$\gamma(z_j) \equiv p(z_j = 1|x_i) = \frac{p(z_j = 1)p(x_i|z_j = 1)}{p(x_i)} = \frac{\pi_j p(x_i|\mu_j, \Sigma_j)}{\sum_{l=1}^{k} \pi_l p(x_i|\mu_l, \Sigma_l)}$$

对于样本x_i，若其类别为 j，则应满足$j = \arg\max_j \gamma(z_j)$，即选择后验概率最大的类别作为样本的标签类别。

当观测到样本集 X 时，用极大似然估计求解样本的类别分布，对数似然函数如下：

$$\ln p(X|\pi, \mu, \Sigma) = \ln \prod_{i=1}^{n} p(x_i) = \sum_{i=1}^{n} \ln \sum_{j=1}^{k} \pi_j p(x_i|\mu_j, \Sigma_j)$$

对μ_j，Σ_j和π_j分别求导，并令导数为 0，得：

$$\mu_j = \frac{1}{n_j}\sum_{i=1}^{n}\gamma_{ij}x_i$$

$$\Sigma_j = \frac{\sum_{i=1}^{n}\gamma_{ij}(x_i-\mu_j)^2}{n_j}$$

$$\pi_j = \frac{n_j}{n}$$

其中，$n_j = \sum_i \gamma_{ij}$，$\gamma_{ij} = \frac{p(z_j=1|x_i)}{\sum_{j=1}^{k} p(z_j=1|x_i)} = \frac{\pi_j p(x_i|\mu_j,\Sigma_j)}{\sum_{l=1}^{k}\pi_l p(x_i|\mu_l,\Sigma_l)}$。

具体迭代过程如下：

1. 根据输入的类总数k和经过 k-means 初步处理得到的各个类中心，先初始化参数μ_j，Σ_j和π_j，$j \in \{1,2,\dots,k\}$。

2. 根据$\gamma_{ij} = \frac{p(z_j=1|x_i)}{\sum_{j=1}^{k} p(z_j=1|x_i)} = \frac{\pi_j p(x_i|\mu_j,\Sigma_j)}{\sum_{l=1}^{k}\pi_l p(x_i|\mu_l,\Sigma_l)}$一式，计算出各个样本属于各类的概率大小。

3. 用下式更新参数μ_j，Σ_j和π_j，$j \in \{1,2,\dots,k\}$：

$$\mu_j = \frac{1}{n_j}\sum_{i=1}^{n}\gamma_{ij}x_i$$

$$\Sigma_j = \frac{\sum_{i=1}^{n}\gamma_{ij}(x_i-\mu_j)^2}{n_j}$$

$$\pi_j = \frac{n_j}{n}$$

4. 计算对数似然函数，当函数值收敛时即可终止迭代；否则回到第 2 步继续迭代求解模型参数。

四、实验结果与分析

（一）k-means 算法

我们可以通过调整各类数据的高斯分布参数，来探究各类数据分布对于分类结果的影响。

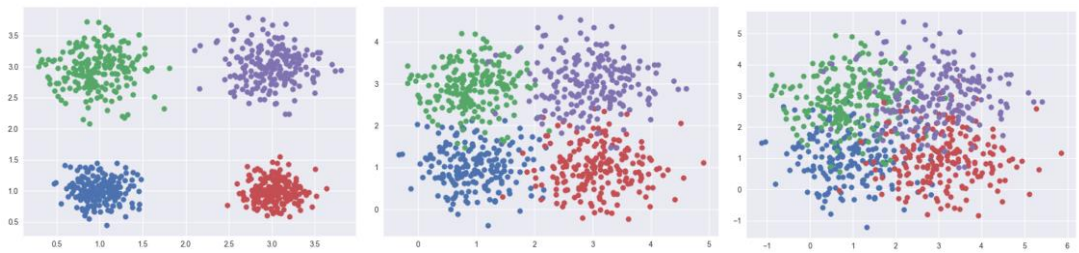
各类数据高斯分布参数的选取：

```
mus = [[1, 1], [1, 3], [3, 1], [3, 3]]
sigmas = [[0.2, 0.2], [0.3, 0.3], [0.2, 0.2], [0.3, 0.3]]
```

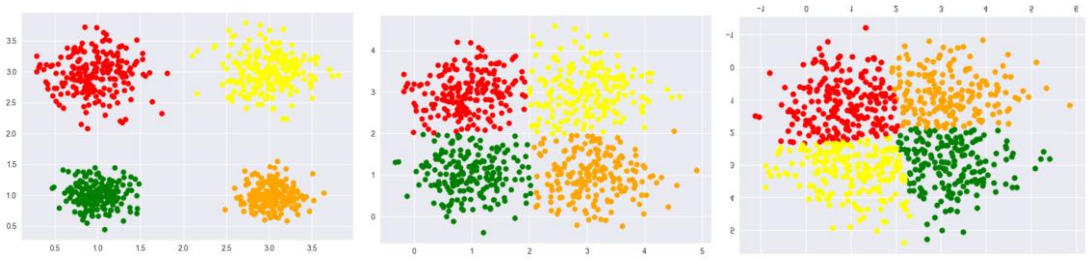
```
mus = [[1, 1], [1, 3], [3, 1], [3, 3]]
sigmas = [[0.5, 0.5], [0.5, 0.5], [0.6, 0.6], [0.6, 0.6]]
```

```
mus = [[1, 1], [1, 3], [3, 1], [3, 3]]
sigmas = [[0.8, 0.8], [0.8, 0.8], [0.9, 0.9], [0.9, 0.9]]
```

原始数据及类别情况：



分类结果：



可以看到，当不同类数据分布较远不出现交叉的时候，分类结果很理想；当不同类数据分布较近的时候，各类数据交叉重叠，分类结果将会是“硬性”的边界划分，无法准确划分叠加部分，这是由于数据是按“最小距离”分类的。

（二）GMM

同样的，我们也可以通过调整各类数据的高斯分布参数，来探究各类数据分布对于分类结果的影响。

各类数据高斯分布参数的选取：

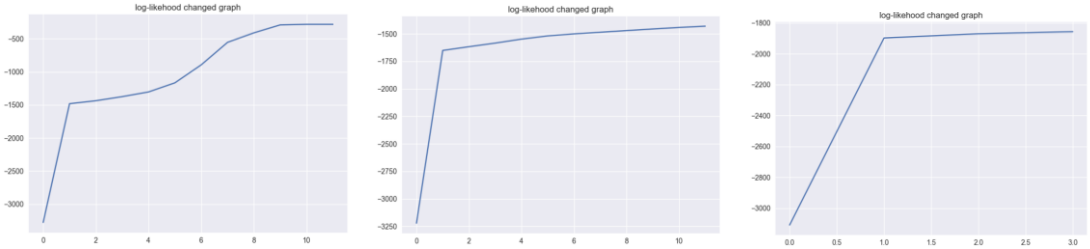
```
mus = [[1, 1], [2, 3], [3, 2]]
sigmas = [[0.3, 0.3], [0.2, 0.2], [0.1, 0.1]]
```

```
mus = [[1, 1], [2, 3], [3, 2]]
sigmas = [[0.6, 0.6], [0.5, 0.5], [0.4, 0.4]]
```

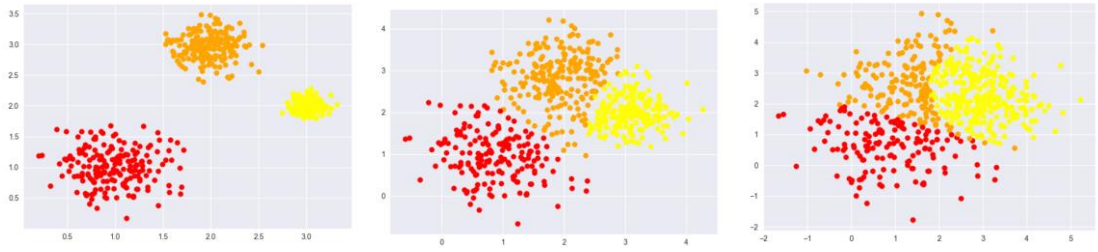
```
 mus = [[1, 1], [2, 3], [3, 2]]
 sigmas = [[1, 1], [0.8, 0.8], [0.7, 0.7]]
```

原始数据及类别情况：

对数似然函数值变化情况：



分类结果：



可以看到，当不同类数据分布较远不出现交叉的时候，分类结果很理想；当不同类数据分布较近的时候，各类数据交叉重叠，分类结果同样也无法准确划分叠加部分。

（三）UCI 数据集

选取 iris.data 数据集进行测试，将数据集分为三类后打乱，用上述实现的 GMM 模型进行聚类。值得注意的是，由于分类的过程属于无监督学习，是无法学习到类别标签的，需要尝试各种分类标签的对应情况，最终在该对应情况下得到真正的分类准确率。

```
real labels: [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 1 1 1 2 0 2 0 0 1 2 2 2 2 1 2 1 1 2 2 2 2 1 2 1 0 2 1 1 1 1 2 0 0 2 1 0 0
 1 0 2 1 0 1 2 1 0 2 2 2 2 0 0 2 2 0 2 0 2 2 0 0 2 0 0 0 1 2 2 0 0 0 1 1 0
 0 1 0 2 1 2 1 0 2 0 2 0 0 2 0 2 1 1 1 2 2 1 1 0 1 2 2 0 1 1 1 1 0 0 0 2 1
 2 0]
prediction labels: [0. 2. 1. 0. 1. 0. 1. 2. 2. 2. 2. 2. 2. 2. 2. 1. 2. 2. 1. 1. 2. 2. 1. 1.
 2. 1. 1. 2. 2. 1. 0. 2. 1. 2. 0. 2. 1. 2. 2. 2. 0. 1. 0. 1. 1. 2. 0. 0.
 2. 0. 2. 0. 2. 2. 2. 2. 2. 2. 2. 0. 2. 1. 2. 2. 2. 2. 2. 0. 1. 1. 0. 2.
 1. 1. 2. 1. 2. 2. 1. 2. 0. 2. 1. 0. 0. 0. 0. 1. 1. 0. 0. 1. 0. 1. 2. 0.
 1. 1. 0. 1. 1. 1. 2. 0. 0. 1. 1. 1. 2. 2. 1. 1. 2. 1. 0. 2. 0. 2. 1. 2.
 1. 0. 1. 1. 0. 1. 0. 2. 2. 2. 0. 0. 2. 0. 1. 2. 0. 2. 1. 2. 2. 2. 2. 1.
 1. 1. 0. 2. 0. 1.]
cluster centers: [[ 1.20779667  0.06550201  1.1367984   1.21227134]
 [-1.01458087  0.84230692 -1.30487854 -1.25512883]
 [ 0.11337026 -0.69481436  0.38006931  0.29876473]]
accuracy: 0.9
```

可以看到，测试准确率达到 90%，说明 GMM 模型在该数据集上的表现还是很理想的。

五、结论

K-means 和 GMM 是 EM 算法的两种表现形式，都按 E 步和 M 步进行迭代优化。但 k-means 假设了所有聚类对总的贡献是相等的，而不是概率的；假设一个样本由某一个聚类产生的概率是 1，其他的就都是 0。而 GMM 假设多个高斯模型对总模型的贡献是有权重的，是概率的；且样本属于某一聚类也是有概率的。两者都能较好地解决简单分类问题，但也都存在着只取到局部最优的问题。

K-Means 假设数据呈球状分布，使用欧式距离来衡量样本与各个聚类中心的相似度(假设数据的各个维度对于相似度计算的作用是相同的)，聚类中心的初始化对于最终的结果有很大的影响，如果选择不好初始的聚类中心容易陷入局部最优解。

GMM 不像 K-means 的假设强，可以用于对 K-Means 的分类结果进行进一步优化，得到效果更好的分类结果。

六、参考文献

周志华.机器学习[M].北京：清华大学出版社，2016

七、附录：源代码（带注释）

1. main.py

```
import numpy as np

import mytool as mt
import kmeans
import gmm

if __name__ == '__main__':

    # 用 K_means 对数据集分类
    np.random.seed(0)
    times = 10
    mus = [[1, 1], [1, 3], [3, 1], [3, 3]]
    sigmas = [[0.8, 0.8], [0.8, 0.8], [0.9, 0.9], [0.9, 0.9]]
    nums = [200, 200, 200, 200]
    data = mt.generate_data(mus, sigmas, nums, 2)
    labels_pre = kmeans.k_means(data, 4, times)
    mt.color_data(np.concatenate([data, labels_pre], axis=1))

    # 用 GMM 对数据集分类（生成的数据）
    np.random.seed(0)
    times = 10
    mus = [[1, 1], [2, 3], [3, 2]]
    sigmas = [[1, 1], [0.8, 0.8], [0.7, 0.7]]
    nums = [200, 200, 200]
    data = mt.generate_data(mus, sigmas, nums, 2)
    labels_pre = gmm.gmm(3, data, times)
    mt.color_data(np.concatenate([data, labels_pre], axis=1))

    # 用 GMM 对数据集分类（UCI 上的数据）
    np.random.seed(0)
    times = 10
    data, labels_real = mt.load_data("iris.data")
    labels_pre, mu = gmm.gmm(3, data, times, show=True,
get_cluster_centers=True)
    print("real labels:", labels_real)
    print("prediction labels:", labels_pre)
    print("cluster centers:", mu)
    print("accuracy:", mt.cal_accuracy(labels_pre, labels_real))
```

2. mytool.py

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.utils import shuffle

```

```
plt.style.use('seaborn')


def generate_data(mus, sigmas, nums, dim, show=True):
    """
    生成 num 个高斯分布的数据
    :param mus: 各类高斯分布的均值
    :param sigmas: 各类高斯分布的标准差
    :param nums: 高斯分布的个数
    :param dim: 数据的维度
    :param show: 是否展示最终生成数据集和类别情况图
    :return: 生成的数据集
    """
    datasets = []
    for mu, sigma, num in zip(mus, sigmas, nums):
        # 临时存储一个高斯分布的数据集
        dataset = np.random.randn(num, dim)
        for i, v in enumerate(sigma):
            dataset[:, i] *= sigma[i]
        for i, m in enumerate(mu):
            dataset[:, i] += mu[i]
        datasets.append(dataset)
    if show:
        for dataset in datasets:
            # 画出每一个数据集
            plt.scatter(dataset.T[0], dataset.T[1])
        plt.show()
    datasets = np.concatenate(datasets)
    return np.array(datasets, dtype=float)


def load_data(path):
    """
    使用从 UCI 获取的数据集
    :param path: 数据集所在路径
    :return: 获取的数据集与标签，各类的均值与标准差
    """
    data = []
    file = open(path, encoding='utf-8')
    for line in file:
        data.append(line.strip('\n').split(sep=','))
    # 将数据分为 0 类、1 类和 2 类
    all_data = np.array(data)
    all_data = np.where(all_data == 'Iris-setosa', 0, all_data)
```

```
    all_data = np.where(all_data == 'Iris-versicolor', 1, all_data)
    all_data = np.where(all_data == 'Iris-virginica', 2, all_data)
    # 将数据与类别标签分开
    x = all_data[:, :len(all_data[0]) - 1]
    y = all_data[:, -1]
    x = np.array(x, dtype=np.float32)
    y = np.array(y, dtype=int)
    # 把数据集进行归一化处理
    x = (x - np.mean(x, axis=0))/(np.std(x, axis=0))
    # 将数据集打乱
    x, y = shuffle(x, y)
    return x, y


def color_data(data):
    """
    将数据集分类情况画图展示出
    :param data: 数据集及其标签
    :return: 分类情况图
    """
    if data.shape[1] != 3:
        print("only 2D pictures are supported")
        return
    # 计数，分离数据与类标签
    num = data.shape[0]
    label = data[:, -1]
    x = data[:, :-1]

    color = ['r', 'orange', 'yellow', 'g', 'b', 'purple', 'pink']
    # 按类标签选色画图
    for i in range(num):
        plt.scatter(x[i, :].T[0], x[i, :].T[1],
color=color[int(label[i])])
    plt.show()


def cal_accuracy(label_pre, label_real):
    """
    计算分类的准确度
    :param label_pre: 预测的分类标签
    :param label_real: 实际的分类标签
    :return: 分类准确度
    """
    count = 0
```

```
    num = len(label_real)
    for i in range(num):
        # 无监督学习需要考虑标签名不对应的情况，以下为试验后的匹配情况
        if label_real[i] == 0:
            if label_pre[i] == 1:
                count += 1
        elif label_real[i] == 1:
            if label_pre[i] == 2:
                count += 1
        else:
            if label_pre[i] == 0:
                count += 1
    return count / num
```

3. kmeans.py

```
import numpy as np
import math


def k_means(data, cluster_num, times):
    """
    用 K_means 求解数据集分类问题
    :param data: 待分类的数据集
    :param cluster_num: 类的数量
    :param times: 迭代次数
    :return: 对各数据的分类标签
    """

    # 从样本中随机选择初始时的类中心
    center = np.zeros((cluster_num, data.shape[1]))
    for num in range(cluster_num):
        center[num, :] = data[np.random.randint(0, data.shape[0]), :]

    new_center = np.zeros((center.shape[0], center.shape[1]))
    label = np.zeros(data.shape[0], )

    for i in range(times):
        distance = np.zeros(center.shape[0], )
        # 计算每个点到各个类中心的距离，并按最小距离进行分类
        for j in range(data.shape[0]):
            for k in range(center.shape[0]):
                d = data[j, :]
                c = center[k, :]
                distance[k] = math.sqrt(sum((d - c) ** 2))
```

```
            label[j] = np.argmin(distance)
        # 重新计算每一类的中心
        for k in range(center.shape[0]):
            cluster = []
            for j in range(data.shape[0]):
                if label[j] == k:
                    cluster.append(data[j, :])
            if not cluster:
                continue
            new_center[k, :] = np.average(cluster, axis=0)
        center = new_center

    label = label.reshape(-1, 1)
    return label
```

4.gmm.py

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

import kmeans


def plot_clusters(data, mu, sigma, mu_true=None, sigma_true=None):
    """
    画出各类的实际边界和预测边界
    :param data: 待分类数据集
    :param mu: 预测的各类的均值
    :param sigma: 预测的各类的标准差
    :param mu_true: 实际的各类的均值
    :param sigma_true: 实际的各类的标准差
    :return: 数据集、实际边界和预测边界的图
    """
    cluster_num = len(mu)
    # 画出数据集
    plt.scatter(data[:, 0], data[:, 1])
    ax = plt.gca()
    # 画出预测边界
    for i in range(cluster_num):
        plot_args = {'fc': 'None', 'lw': 2, 'edgecolor': 'g', 'ls':
':'}
        ellipse = Ellipse(mu[i], 3 * math.sqrt(sigma[i][0]), 3 *
```

```
math.sqrt(sigma[i][1]), **plot_args)
        ax.add_patch(ellipse)
    # 画出实际边界
    if (mu_true is not None) and (sigma_true is not None):
        for i in range(cluster_num):
            plot_args = {'fc': 'None', 'lw': 2, 'edgecolor': 'b'}
            ellipse = Ellipse(mu_true[i], 3 *
math.sqrt(sigma_true[i][0]), 3 * math.sqrt(sigma_true[i][1]),
**plot_args)
            ax.add_patch(ellipse)
    plt.show()


def gmm(cluster_num, data, times, show=True,
get_cluster_centers=False):
    """
    用GMM对数据集分类
    :param cluster_num: 类的数量
    :param data: 待分类数据集
    :param times: 迭代次数
    :param show: 是否展示对数似然函数随迭代次数的变化情况图
    :param get_cluster_centers: 是否返回最终各类中心
    :return: 对各数据的分类标签（及最终各类中心）
    """
    dim, num = data.shape[1], data.shape[0]

    # 用 kmeans 结果作为初始化均值与标准差
    label = kmeans.k_means(data, cluster_num, 10)
    mus = np.zeros((cluster_num, dim))
    sigmas = np.zeros((cluster_num, dim))
    cluster = []
    for i in range(cluster_num):
        for j in range(num):
            if label[j] == i:
                cluster.append(data[j])
        mus[i, :] = np.average(cluster, axis=0)
        cluster = np.array(cluster)
        sigmas[i, :] = np.average((cluster - mus[i])**2, axis=0)
        cluster = []

    # 选定最初参数
    mu = mus + 2 * np.random.randn(cluster_num, dim)
    sigma = sigmas + abs(2 * np.random.randn(cluster_num, dim))
    gama_matrix = np.ones((num, cluster_num)) / cluster_num
```

```
    pi = gama_matrix.sum(axis=0) / gama_matrix.sum()

    log_lh = []
    for i in range(times):
        # 展示每次迭代的效果图
        # plot_clusters(data, mu, sigma, mus, sigmas)
        # 计算对数似然函数值，并更新参数
        log_lh.append(cal_log_lh(data, pi, mu, sigma))
        gama_matrix = cal_matrix(data, mu, sigma, pi)
        pi = cal_pi(gama_matrix)
        mu = cal_mu(data, gama_matrix)
        sigma = cal_sigma(data, mu, gama_matrix)
        print('log-likehood:%.5f' % log_lh[-1])

    if show:
        plt.plot(log_lh)
        plt.title("log-likehood changed graph")
        plt.show()

    # 对数据集进行分类
    label = np.zeros(num)
    for xi in range(num):
        probability = np.zeros(cluster_num)
        for i in range(cluster_num):
            probability[i] = multivariate_normal.pdf(data[xi, :],
mu[i], np.diag(sigma[i]))
        label[xi] = np.argmax(probability)

    if not get_cluster_centers:
        label = label.reshape(-1, 1)
        return label
    else:
        return label, mu


def cal_log_lh(data, pi, mu, sigma):
    """
    计算对数似然函数值
    :param data: 待分类数据集
    :param pi: 公式中 pi 值
    :param mu: 公式中 mu 值
    :param sigma: 公式中 sigma 值
    :return: 对数似然函数值
    """
```

```
    num, cluster_num = len(data), len(pi)
    pdfs = np.zeros((num, cluster_num))
    for i in range(cluster_num):
        pdfs[:, i] = pi[i] * multivariate_normal.pdf(data, mu[i],
np.diag(sigma[i]))
    return np.sum(np.log(pdfs.sum(axis=1)))


def cal_matrix(data, mu, sigma, pi):
    """
    计算公式中的gama矩阵
    :param data: 待分类数据集
    :param mu: 公式中mu值
    :param sigma: 公式中sigma值
    :param pi: 公式中pi值
    :return: 公式中的gama矩阵
    """
    num, cluster_num = len(data), len(pi)
    pdfs = np.zeros((num, cluster_num))
    for i in range(cluster_num):
        pdfs[:, i] = pi[i] * multivariate_normal.pdf(data, mu[i],
np.diag(sigma[i]))
    gama_matrix = pdfs / pdfs.sum(axis=1, keepdims=True)
    return gama_matrix


def cal_pi(gama_matrix):
    """
    计算公式中的pi
    :param gama_matrix: 公式中gama矩阵
    :return: 公式中的pi
    """
    n = gama_matrix.shape[0]
    pi = gama_matrix.sum(axis=0) / n
    return pi


def cal_mu(data, gama_matrix):
    """
    计算公式中的mu
    :param data: 待分类数据集
    :param gama_matrix: 公式中gama矩阵
    :return: 公式中的mu
    """
```

```
    dim, cluster_num = data.shape[1], gama_matrix.shape[1]
    mu = np.zeros((cluster_num, dim))
    for i in range(cluster_num):
        mu[i, :] = np.average(data, axis=0, weights=gama_matrix[:, i])
    return mu


def cal_sigma(data, mu, gama_matrix):
    """
    计算公式中的sigma
    :param data: 待分类数据集
    :param mu: 公式中mu
    :param gama_matrix: 公式中gama矩阵
    :return: 公式中的sigma
    """
    dim, cluster_num = data.shape[1], gama_matrix.shape[1]
    sigma = np.zeros((cluster_num, dim))
    for i in range(cluster_num):
        sigma[i, :] = np.average((data - mu[i]) ** 2, axis=0,
weights=gama_matrix[:, i])
    return sigma
```