Machine Learning Homework 2

专业：软件工程　　　　　姓名：沈金龙　　　　　学号：18214806

1. 实验题目

- Please build a Gaussian mixture model (GMM) to model the data in file TrainingData_GMM.csv. Note that the data is composed of 4 clusters, and the model should be trained by expectation maximization (EM) algorithm.
- Based on the GMM learned above, assign each training data point into one of 4 different clusters.

2. 实验要求

- Show how the log-likelihood evolves as the training proceeds.
- The learned mathematical expression for the GMM model after training on the given dataset.
- Randomly select 500 data points from the given dataset and plot them on a 2-dimensional coordinate system. Mark the data points coming from the same cluster (using the results of Problem 2) with the same color.
- Some analyses on the impacts of initialization on the converged values of EM Algorithm.
- Some analyses on the results you obtained.

3. 实验过程及代码

- 本实验采用 python3.6 完成。

- **关于高斯混合模型 GMM**

高斯混合模型（GMM, Gaussian Mixture Model）是多个高斯模型的线性叠加，高斯混合模型的概率分布可以表示如下：

$$P(x)=\sum_{k=1}^{K}a_k\phi(x;\mu_k,\Sigma_k)$$

其中，K 表示模型的个数，a_k 是第 k 个模型的系数，表示出现该模型的概率，$\phi(x;\mu_k,\Sigma_k)$ 是第 k 个高斯模型的概率分布。

注：考虑多个随机变量的情况，即多元高斯分布，因此高斯分布中的参数不再是方差 σ_k，而是协方差矩阵 Σ_k。

我们的目标是给定一堆没有标签的样本和模型的个数 K，以此求得混合模型的参数，然后就可以用这个模型来对样本进行聚类。

- **关于期望最大化算法 EM**

期望最大算法（EM，Expectation-Maximization）是通过不断迭代来求得最佳参数的。在执行该算法之前，需要先给出一个初始化的模型参数。我们让每个模型的 μ 为随机值，Σ 为单位矩阵，a 为 $\frac{1}{K}$，即每个模型初始时都是等概率出现的。EM 算法可以分为 E 步和 M 步。

E 步:

直观理解就是我们已经知道了样本 x_i，那么它是由哪个模型产生的呢？我们这里求的就是：样本 x_i 来自于第 k 个模型的概率，我们把这个概率称为模型 k 对样本 x_i 的“责任”，也叫“响应度”，记作 γ_{ik}，计算公式如下：

$$\gamma_{ik}=\frac{a_k\phi(x_i;\mu_k,\Sigma_k)}{\sum_{k=1}^{K}a_i\phi(x_i;\mu_k,\Sigma_k)}$$

M 步:

根据样本和当前 γ 矩阵重新估计参数，注意这里 x 为列向量：

$$\mu_k = \frac{1}{N_k}\sum_{i=1}^{N}\gamma_{ik}x_i$$

$$\Sigma_k = \frac{1}{N_k}\sum_{i=1}^{N}\gamma_{ik}(x_i-\mu_k)(x_i-\mu_k)^T$$

$$a_k = \frac{N_k}{N}$$

4. 实验结果与分析

- 随机选择 500 个点并在二维坐标系中画出来，以同样的颜色标出各自的聚类：

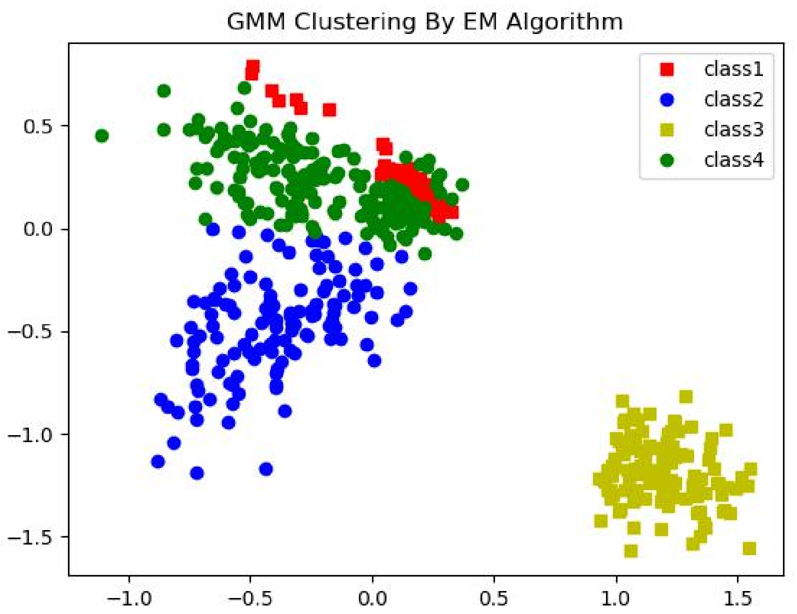
```
# 载入数据
my_matrix = np.loadtxt(open("TrainingData_GMM.csv","rb"),delimiter=",",skiprows=0)
indexs = list(range(0,5000))
indexs_slice = random.sample(indexs,500)
my_matrix_slice = []
for i in indexs_slice:
    my_matrix_slice.append(my_matrix[i])
Y = my_matrix_slice
matY = np.matrix(Y, copy=True)
```

- EM 算法的核心代码：

```
######################################################
# E 步：计算每个模型对样本的响应度
# Y 为样本矩阵，每个样本一行，只有一个特征时为列向量
# mu 为均值多维数组，每行表示一个样本各个特征的均值
# cov 为协方差矩阵的数组，alpha 为模型响应度数组
######################################################
def getExpectation(Y, mu, cov, alpha):
    # 样本数
    N = Y.shape[0]
    # 模型数
    K = alpha.shape[0]

    # 为避免使用单个高斯模型或样本，导致返回结果的类型不一致
    # 因此要求样本数和模型个数必须大于1
    assert N > 1, "There must be more than one sample!"
    assert K > 1, "There must be more than one gaussian model!"

    # 响应度矩阵，行对应样本，列对应响应度
    gamma = np.mat(np.zeros((N, K)))

    # 计算各模型中所有样本出现的概率，行对应样本，列对应模型
    prob = np.zeros((N, K))
    for k in range(K):
        prob[:, k] = phi(Y, mu[k], cov[k])
    prob = np.mat(prob)

    # 计算每个模型对每个样本的响应度
    for k in range(K):
        gamma[:, k] = alpha[k] * prob[:, k]
    for i in range(N):
        gamma[i, :] /= np.sum(gamma[i, :])
    return gamma
```

```python
##########################################################
# M 步：迭代模型参数
# Y 为样本矩阵，gamma 为响应度矩阵
##########################################################
def maximize(Y, gamma):
    # 样本数和特征数
    N, D = Y.shape
    # 模型数
    K = gamma.shape[1]

    #初始化参数值
    mu = np.zeros((K, D))
    cov = []
    alpha = np.zeros(K)

    # 更新每个模型的参数
    for k in range(K):
        # 第 k 个模型对所有样本的响应度之和
        Nk = np.sum(gamma[:, k])
        # 更新 mu
        # 对每个特征求均值
        mu[k, :] = np.sum(np.multiply(Y, gamma[:, k]), axis=0) / Nk
        # 更新 cov
        cov_k = (Y - mu[k]).T * np.multiply((Y - mu[k]), gamma[:, k]) / Nk
        cov.append(cov_k)
        # 更新 alpha
        alpha[k] = Nk / N
    cov = np.array(cov)
    return mu, cov, alpha
```

- 聚类结果：



- 基于给定数据得到的相关参数如下所示：

μ_k ：

[[0.44845318 0.77713784] [0.28080906 0.48586375]

[0.86577615 0.16207885][0.34891429 0.75986815]]

Σ_k ：

[[[0.00629149 -0.00577172][-0.00577172 0.00599824]]
 [[0.0099899   0.00733243][0.00733243 0.01512776]]
 [[0.00319738 -0.00074098][-0.00074098 0.00416401]]
 [[0.01335553 -0.00431358][-0.00431358 0.0043973]]]

a_k ：

[0.09262192 0.27225105 0.246   0.38912703]

- 关于 EM 算法的初始化的讨论：

传统的 EM 算法对初始值敏感，聚类结果随不同的初始值而波动较大。且随着迭代的次数，越往后收敛速度越慢，同时很容易 trap 在 local 解附近。总的来说，EM 算法收敛的优劣很大程度上取决于其初始参数。

5. 总结

本实验详细了解了 GMM 和 EM 的相关知识，并完成基于 python 的实现。在实验中发现，传统的 EM 算法对初始值敏感，算法收敛的优劣很大程度上取决于其初始参数。据此可能采取的优化措施为：采用一种基于网格的聚类算法来初始化 EM 算法。