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： GMM 模型

学号：

姓名：

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

2.1 实验要求

用高斯分布产生 k 个高斯分布的数据（不同均值和方差）（其中参数自己设定）。

（1）用 k-means 聚类，测试效果；

（2）用混合高斯模型和你实现的 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（与你设定的结果比较）。

应用：可以 UCI 上找一个简单问题数据，用你实现的 GMM 进行聚类。

2.2 实验环境

Windows10+python3.7+PyCharm+Jupyter notebook

三、设计思想（本程序中的用到的主要算法及数据结构）

这一部分主要涉及 K-means 算法和 GMM 算法的实现，这两个算法的优化过程使用的都是 EM 算法。EM 算法是一种迭代优化策略，由于它的计算方法中每一次迭代都分两步，其中一个为期望步（E 步），另一个为极大步（M 步）。在 E 步中调整分布，M 步在 E 步调整的情况下求优化目标的最值时的参数值，这个过程循环往复，直到参数收敛表示训练结束。

3.1 K-means

K-means 算法主要解决的是无监督情况下的聚类问题，其解决问题的数学描述如下：假设给定训练样本集合为$X = \{x_1, x_2, .., x_n\}$，其中每一个训练样本都是一个 d 维的向量，向量的每一维表示样本的一个特征。假设需要将这一组样本集合聚类生成 k 个簇，也就是说给出 k 个标签，$C_1, C_2, .. C_k$，给样本集中每一个样本打标签，达到的结果是使得聚类结果中每一个样本到样本中心的距离之和最小，因此我们的优化目标可以定为：

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

其中μ_i表示每一个簇的中心向量，表示方式为：$\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x_i$，其实也就是计算这一簇中所有向量的均值向量。则迭代优化过程如下：

1.根据簇的数量 k 随机初始化 k 个向量作为每一个簇的初始中心向量，在本实验过程中随机选择样本集中的 k 个样本作为 k 个簇的中心样本。

2.对于其余 n-k 个向量分别计算与这 k 个中心的距离，对于每一个向量都选择与其最近的一个中心，表示归于这个中心向量表示的簇中。(E 步)

3.根据新的划分计算 k 个簇的中心向量，如果新的均值向量和旧的均值向量之间的差距已经达到精度要求，则表示迭代已经收敛，返回结果；如果还未收敛，返回第二步继续迭代。（M 步）

3.2 GMM

GMM 算法的思想就是对于任何一个样本集都可以视为是 k 个不同的多元高斯分布生成的，也就是说我们可以使用 k 个不同的多元高斯分布，则样本集中的每一个样本都可以视为是这 k 个高斯分布中的某一个生成的，因此事实上 GMM 算法解决的也是聚类问题。

假设我们考虑多元高斯分布生成 d 维随机变量 x 的概率密度函数为：

$$N(x|\mu,\Sigma)=\frac{1}{(\sqrt{2\pi})^{d}|\Sigma|^{\frac{1}{2}}}exp(-\frac{1}{2}(x-\mu)^{T}\Sigma^{-1}(x-\mu))$$

其中μ表示 d 维均值向量，Σ表示 d*d 维的协方差矩阵。

给定训练样本集合为$X=\{x_1,x_2,..,x_n\}$，其中每一个训练样本都是一个 d 维的向量，则每一个样本集都可以视为是由我们给定的 K 个高斯分布混合生成的，则可以有如下式子表示：

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

其中我们使用π_j表示该样本是由第 j 个高斯分布生成的先验混合系数，因此需要满足条件$\sum_j\pi_j=1$。不妨假设一个变量 z，其中$z_j\in\{0,1\},\sum_j z_j=1$，这个约束就表示只有一个 z 可以等于 1，其余 z 都是 0。我们用这个变量表示某一个样本是由第 j 个高斯分布生成的表示形式。因此，我们可以得到如下表示：

$$p(z_j=1)=\pi_j,0\le\pi_j\le1,\sum_j\pi_j=1$$

因此我们可以得到z的先验概率密度函数为：

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

而当我们已知x_i的情况下，可以得到 z 的后验概率分布为：

$$\gamma(z_j)=p(z_j=1|x_i)=\frac{p(z_j=1)p(x_i|z_j=1)}{p(x_i)}=\frac{p(z_j=1)p(x_i|z_j=1)}{\sum_{k=1}^{K}\pi_k\,p(x_i|\mu_k,\Sigma_k)}$$

当这个后验概率已知的时候，对于每一个样本的聚类，我们将其分至第 j 类，这个 j 满足$j=argmax\gamma(z_j)$，也就是完成了对于样本集的聚类过程。接下来是对于这个聚类过程的优化过程。在优化过程中我们可以使用极大似然估计来求解这一过程，极大似然函数为：

$$lnp(X|\pi,\mu,\Sigma)=\sum_{i=1}^{N}ln\sum_{j=1}^{K}\pi_j\,N(x_i|\mu_j,\Sigma_j)$$

我们希望上式能取得最大值，我们需要优化的参数有μ_j,Σ_j,π_j。首先对μ_j求导，并令导数等于 0 得到如下结果：

$$\frac{\partial lnp(X|\pi,\mu,\Sigma)}{\partial \mu_j}=\sum_{i=1}^{N}\frac{\pi_j N(x_i|\mu_j,\Sigma_j)}{\sum_{k=1}^{K}\pi_k\, p(x_i|\mu_k,\Sigma_k)}\Sigma_j^{-1}(x_i-\mu_j)=0$$

不妨令$\gamma(z_{ij})=\frac{\pi_j N(x_i|\mu_j,\Sigma_j)}{\sum_{k=1}^{K}\pi_k p(x_i|\mu_k,\Sigma_k)}$，则根据上式可以解得：

$$\begin{aligned} n_j &= \sum_{i=1}^{N}\gamma\,(z_{ij}) \\ \mu_j &= \frac{1}{n_j}\sum_{i=1}^{N}\gamma\,(z_{ij})x_i \end{aligned}$$

对Σ_j求导，并令导数等于 0，则有：

$$\frac{\partial lnp(X|\pi,\mu,\Sigma)}{\partial \Sigma_j}=\sum_{i=1}^{N}\frac{\pi_j N(x_i|\mu_j,\Sigma_j)}{\sum_{k=1}^{K}\pi_k\, p(x_i|\mu_k,\Sigma_k)}(\Sigma_j^{-1}-\Sigma_j^{-1}(x_i-\mu_j)(x_i-\mu_j)^T\Sigma_j^{-1})=0$$

解得：

$$\Sigma_j=\frac{\sum_{i=1}^{N}\gamma\,(z_{ij})(x_i-\mu_j)(x_i-\mu_j)^T}{n_j}$$

对于π_j的优化由于其存在约束，因此需要构造拉格朗日多项式如下：

$$\sum_{i=1}^{N} l\,n\sum_{j=1}^{K}\pi_j\,N(x_i|\mu_j,\Sigma_j)+\lambda(\sum_{j=1}^{K}\pi_j-1)$$

对π_j求导，并令导数等于 0，得到如下式子：

$$\frac{\partial\sum_{i=1}^{N} l\,n\sum_{j=1}^{K}\pi_j\,N(x_i|\mu_j,\Sigma_j)+\lambda(\sum_{j=1}^{K}\pi_j-1)}{\partial\pi_j}=\sum_{i=1}^{N}\frac{N(x_i|\mu_j,\Sigma_j)}{\sum_{k=1}^{K}\pi_k\,p(x_i|\mu_k,\Sigma_k)}+\lambda=0$$

不妨将上式两边同乘π_j并做一个累加：

$$\sum_{j=1}^{K}\pi_j\sum_{i=1}^{N}\frac{N(x_i|\mu_j,\Sigma_j)}{\sum_{k=1}^{K}\pi_k\,p(x_i|\mu_k,\Sigma_k)}+\lambda\sum_{j=1}^{K}\pi_j=N+\lambda=0$$

将$\lambda=-N$带入$\sum_{i=1}^{N}\frac{N(x_i|\mu_j,\Sigma_j)}{\sum_{k=1}^{K}\pi_k p(x_i|\mu_k,\Sigma_k)}+\lambda=0$则有：

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

因此总的来说，GMM 的迭代计算过程可以用如下过程概述：

- 随机初始化$\mu_j,\Sigma_j,\pi_j,j\in\{1,2,..,k\}$
- E 步：使用$\gamma(z_{ij})=\frac{\pi_j N(x_i|\mu_j,\Sigma_j)}{\sum_{k=1}^{K}\pi_k p(x_i|\mu_k,\Sigma_k)}$更新每个样本是由各个高斯分布生成的概率
- M 步：更新$\mu_j,\Sigma_j,\pi_j,j\in\{1,2,..,k\}$，更新方式如下：

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

$$n_j = \sum_{i=1}^{N} \gamma(z_{ij})$$

$$\mu_j = \frac{1}{n_j} \sum_{i=1}^{N} \gamma(z_{ij}) x_i$$

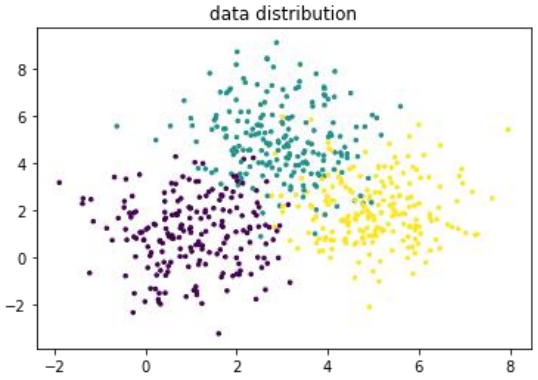
$$\Sigma_j = \frac{\sum_{i=1}^{N} \gamma(z_{ij})(x_i - \mu_j)(x_i - \mu_j)^T}{n_j}$$

- 如果参数变化已经在精度要求之内，那么选择 j，使得 j 满足$j = argmax\gamma(z_j)$；如果参数还未收敛，返回 E 步继续优化。

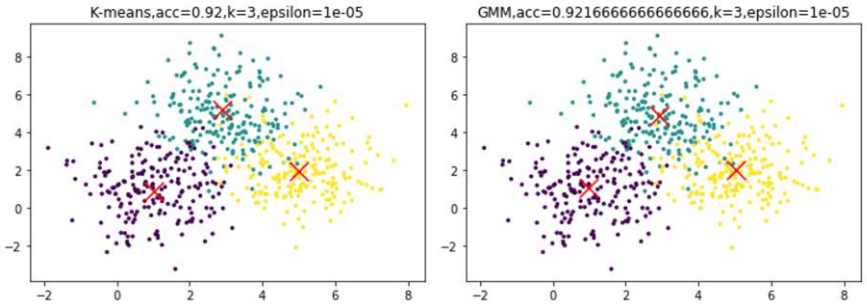
四．实验结果与分析

4.1 手工数据集

手工生成的数据集是 3 个二元高斯分布形成的，三个高斯分布两个维度上的的均值分别为：[1,1], [3,5], [5,2]，两个维度上的协方差矩阵分别为： [[1,0],[0,2]]、[[1,0],[0,3]]、[[1,0],[0,2]]，在不做特定说明的时候数据集不变。生成的数据集 的可视化表示如下图所示：



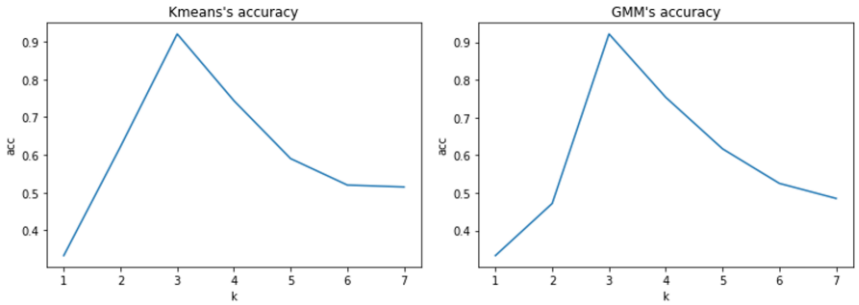
先对于 k-means 和 GMM 算法的性能进行简单考察。使用的精度均为10^{-5}，聚类簇数量均设置为 3，使用上述数据集进行测试的结果如下图所示：



可以发现在上述情况下聚类的时候 K-means 和 GMM 算法的性能相仿，接下来详细考量各个参数对于两种算法的聚类效果的影响。

4.1.1 聚类簇数

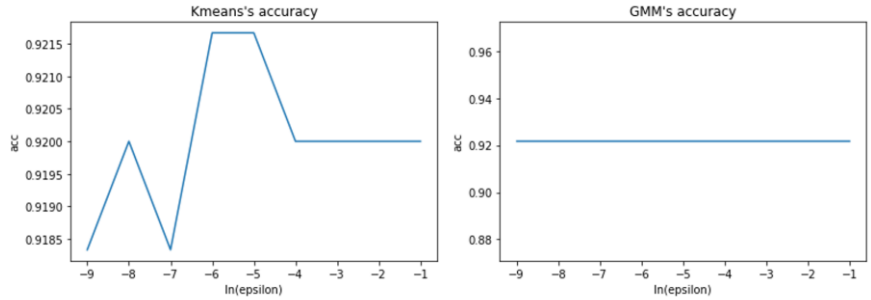
考虑到上述简单测试的时候已知正确分簇结果而进行的测试，而在真实聚类过程中并不知道实际应该聚成多少类。因此这一小节测试使用不同聚类簇数的时候两种算法的性能，如下图所示：



可以发现无论是 K-means 还是 GMM 算法，当数据集呈现出比较明显的是由多个簇形成的时候通过遍历簇数都可以比较敏锐地发现最佳的分类簇数是多少，也就是图中的最高点。这说明在明显可以可分的数据集中 K-means 和 GMM 对于分成多少簇能达到最优分类效果都可以得出结果。对于难以分出簇数的结果在 4.1.4 小节讨论。

4.1.2 精度

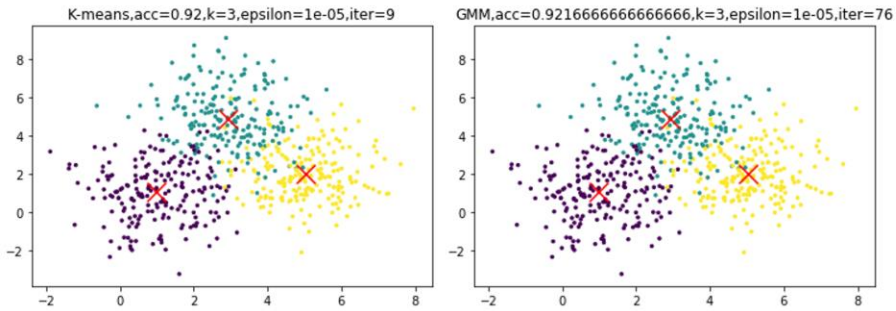
由于实验中默认设置的精度均为10^{-5}，对于 K-means 来说意味着如果两次迭代中心的差值的二范数小于这个精度我们就认为其改变量极小，视为参数收敛；对于 GMM 来说，如果两次迭代结果计算的似然函数值之差小于精度设置量，视为参数收敛；接下来在聚类簇数为 3 的时候考察多个精度设置情况下两种算法聚类结果的精度：



经过多次实验发现精度改变对于两种算法的聚类性能改变并不大。猜测是由于每次对于 k-means 和 GMM 来说，初始化都是会影响结果的，而每次实验过程中的初始化值都是随机生成的，因此对于结果造成了一定的影响；同时对于 GMM 算法来说，由于收敛的比较慢，因此存在达到最大迭代次数时还未到达精度要求的情况。

4.1.3 迭代轮数

对于两种算法都设置了最大迭代轮数，当达到迭代轮数上限的时候即使参数未收敛也视为迭代结束，默认设置的最大迭代轮数都是 100，接下来考察在聚类簇数为 3 的时候两种算法的迭代轮数：

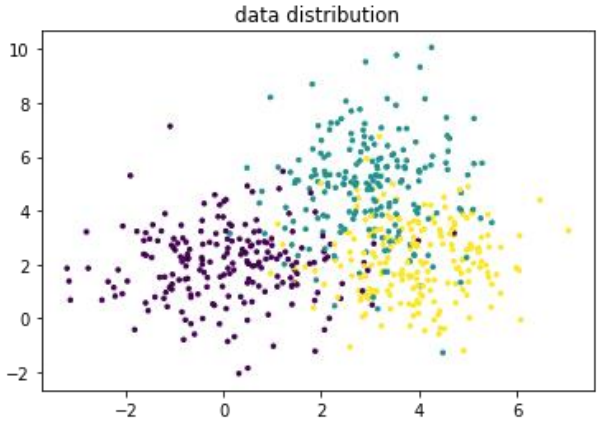


可以发现相比起 K-means，GMM 的迭代次数要多很多，因此 GMM 在每次计算参数的过程中也比 K-means 要多花很多时间。因此 GMM 的缺点就是相对来说模型复杂，运算略慢。

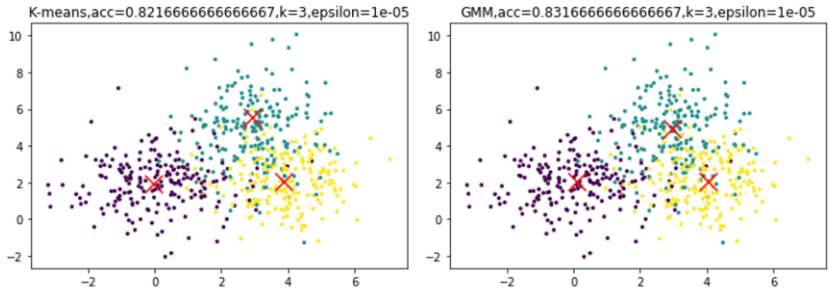
4.1.4 难聚类手工生成数据对比

上面的测试集中每一个簇间隔都比较远，也就是聚类相对来说是比较容易的，接下来测试一个相对来说每一簇之间的距离都相对较小，样本之间混合程度比较高的样本集中两种算法的聚类情况。注：下述样本都由 3 个二元高斯分布混合生成的。

测试样本集：均值[[0,2], [3,5], [4,2]]，协方差矩阵：[[[2,0],[0,2]], [[1,0],[0,3]], [[1,0],[0,2]]]，可视化结果如下：



两种算法运行结果如下：



可以发现 GMM 算法效果比 K-means 略好。

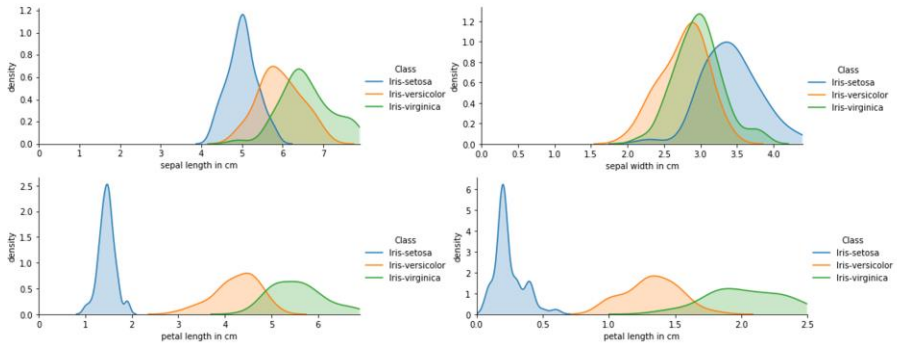
4.2 UCI 数据集

本次实验中选用的数据集是 UCI 中的 iris 数据集，首先简单介绍该数据集：该数据集中的样本一共分为三类，每一个样本都有四种特征可用于聚类使用。接下来可以查看数据集前五行，观察数据集基本组成。

Out[47]:

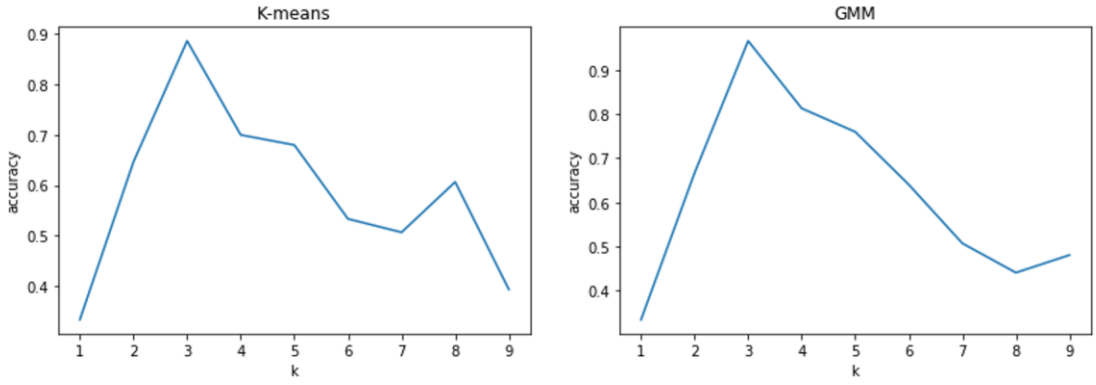
	sepal length in cm	sepal width in cm	petal length in cm	petal width in cm	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

观察每个样本四种特征分别的概率密度函数：



根据四个特征的概率密度函数，可以直观发现不同类别之间的差别相对来说比较大，因此可以猜测聚类过程是可以顺利进行的。以下是两种算法的聚类结果：

首先通过遍历找到两种算法对应的最适合的分类簇数，可以发现都是分为 3 类最优：



由于对应四维特征，因此无法做出分类图示，给出分类精确度结果：K-means：0.88，GMM：0.96。可以发现在这一数据集上两个聚类算法的聚类精度都相对较高，达到了较好的效果。

五．结论

1.K-means 和 GMM 性能对比：在根据手工生成的数据集和 UCI 数据集测试结果发现，在大部分情况下 GMM 聚类精度比 K-means 略高，但是相对来说由于 GMM 模型比较复杂，因此使用 GMM 做聚类的过程中需要更多计算量，也就需要较多时间。

2.K-means 和 GMM 性能原因分析：事实上 K-means 就是 GMM 的一种极特殊情况假设每种类在样本中出现的概率相等均为 $\frac{1}{k}$，而且假设高斯模型中的每个变量之间是独立的，即变量间的协方差矩阵是对角阵，这样就可以使用欧式距离之和代替最大似然函数来优化。而且对于 K-means 来说，将 GMM 简化为每一个样本点完全属于一个高斯分布，在更新的过程中不考虑多个高斯分布混合生成的情况。而且 K-means 比较依赖于初始化，初始化的差异可能导致比较大的结果差异。由于做了这么多假设，而真实数据大概率上并不会满足这些假设，因此大多数情况下 K-means 的效果比 GMM 略差。

六．参考文献

[1]周志华,《机器学习》,清华大学出版社,2016

七、附录：源代码（带注释）

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from itertools import permutations
import scipy.stats
def generate_data(k,n,d,mu_list,sigma_list):
    """
    生成训练数据集
    :param k: k 个高斯分布
    :param n: 每个高斯分布 n 个样本
    :param d:每个样本的特征维度
    :param mu_list:每个高斯分布的均值的 list，格式为长度为 k 的向量，每个元素都是 d 维
    :param sigma_list:每个高斯分布的方差的 list，格式为长度为 k 的向量，每个元素都是 d*d 维
    :return:带标签的训练集，形如[n*k,d]
             标签集合，形如[n*k,1]
    """
    X=np.zeros((k*n,d+1))
    for i in range(k):
        X[i*n:(i+1)*n,:d]=np.random.multivariate_normal(mu_list[i], sigma_list[i], size=n)
        X[i * n:(i+1)*n, d:d+1] = i
    #乱序输出训练数据集
    np.random.shuffle(X)
    return X[:,:d],X[:,d:d+1]
def kmeans(X,k,epsilon=1e-5):
    """
    k-means 算法实现，计算分类结果和中心
    :param X: 训练样本，[n,d]
    :param k:预计要分成多少个簇
    :param epsilon:精度
    :return:预测的标签 label，格式为[n*1],与训练样本一一对应
```

```
            center：每一类的中心，格式为[k*d]
    """
    n=X.shape[0]
    dimension=X.shape[1]
    center=np.zeros((k,dimension))
    label=np.zeros((n,1))
    #随机初始化 center,从 X 中选取 k 个作为中心
    random_index=np.random.choice(range(0,n),k,replace=False)
    for i in range(k):
        center[i,:]=X[random_index[i],:]
    iter=0
    while True:
        iter+=1
        distance=np.zeros(k)#标记每个 X 中的样本和 k 个中心点距离
        #重新计算每个样本的标签
        for i in range(n):
            for j in range(k):
                distance[j]=np.linalg.norm(X[i,:]-center[j,:])#默认使用二范数
            label[i,0]=np.argmin(distance)
        label=label.astype(np.int16)
        #计算样本的中心点
        new_center=np.zeros((k,dimension))
        num=np.zeros(k)#记录每个簇现在有多少个元素
        for i in range(n):
            new_center[label[i,0],:]=new_center[label[i,0],:]+X[i,:]
            num[label[i,0]]+=1
        for i in range(k):
            new_center[i,:]=new_center[i,:]/num[i]
        if np.linalg.norm(new_center - center,ord=2) < epsilon:  # 二范数计算精度，一定范围内表示不再变化
            break
        else:
            center = new_center
    print(iter)
    return label,center
def E(X,pi_list,mu_list,sigma_list):
    """
    EM 算法中的 E 步,计算样本由每个高斯分布生成的后验概率
    :param X:训练数据集，格式为 n*d
    :param pi_list:混合系数矩阵，格式为长度为 k 的向量
    :param mu_list:每个高斯分布的均值的 list，格式为长度为 k 的向量，每个元素都是 d 维
    :param sigma_list:每个高斯分布的方差的 list，格式为长度为 k 的向量，每个元素都是 d*d 维
    :return: 样本由各个混合高斯成分生成的后验概率，格式为 n*k
    """
    n=X.shape[0]
    d=X.shape[1]
    k=pi_list.shape[0]
    gamma_z=np.zeros((n,k))
    for i in range(n):
        pdf_sum=0
        pdf=np.zeros(k)
        for j in range(k):
            pdf[j]=scipy.stats.multivariate_normal.pdf(X[i],mean=mu_list[j],cov=sigma_list[j])#计算 pdf
            pdf_sum+=pdf[j]
        for j in range(k):
            gamma_z[i,j]=pdf[j]/pdf_sum
    return gamma_z
```

```
def M(X, gamma_z, mu_list):
    """
    EM 算法中的 M 步
    :param X:训练集，格式为 n*d
    :param gamma_z:后验概率，格式为 n*k
    :param mu_list:均值 list，格式为长度为 k 的向量，每个元素都是 d 维
    :return:mu_list，sigmma_list，pi_list 的新估计
    """
    n=X.shape[0]
    d=X.shape[1]
    k=gamma_z.shape[1]
    new_mu_list=np.zeros((k,d))
    new_sigmma_list=np.zeros((k,d,d))
            new_pi_list=np.zeros(k)
            for j in range(k):
                n_j=np.sum(gamma_z[0:,j])
                new_pi_list[j]=n_j/n
                gamma=gamma_z[:,j].reshape(1,n) #1*n
                new_mu_list[j,:]=np.matmul(gamma,X)/n_j
                temp=X-mu_list[j] #n*d
                new_sigmma_list[j]=np.matmul(temp.T,np.multiply(temp,gamma.resha
pe(n,1)))/n_j
            return new_mu_list,new_sigmma_list,new_pi_list
        def cal_likelihood(X, mu_list, sigma_list, pi_list):
            """
            计算最大似然函数（也就是 GMM 想要优化的内容）
            :param X: 训练集，格式为 n*d
            :param mu_list: 均值，格式为 k*d
            :param sigma_list: 方差，格式为 k*d*d
            :param pi_list: 混合成分，长度为 k 的向量
            :return: 最大似然函数的值
            """
            ll=0
            n=X.shape[0]
            k=mu_list.shape[0]
            for i in range(n):
                pdf = np.zeros(k)
                temp_sum=0
                for j in range(k):
                    pdf[j] = scipy.stats.multivariate_normal.pdf(X[j], mean=mu_l
ist[j], cov=sigma_list[j])
                    temp_sum+=pi_list[j]*pdf[j]
                ll+=np.log(temp_sum)
            return ll
        def GMM(X,k,max_iter=100,epsilon=1e-5):
            """
            GMM 算法实现
            :param X: 训练集，格式为 n*d
            :param k: 预计有多少个簇
            :param max_iter: 最大迭代次数
            :param epsilon: 精度
            :return: 预测的标签 label，格式为[n*1],与训练样本一一对应
                     center：每一类的中心，格式为[k*d]
            """
            ##初始化
            n=X.shape[0]
            pi_list=np.ones(k)/k
            sigma_list=np.array([0.1 * np.eye(X.shape[1])] * k)
            ##mu_list 初始化用 kmeans 来做
```

```
            label,mu_list=kmeans(X,k,epsilon)
            old_ll=cal_likelihood(X, mu_list, sigma_list, pi_list)
            gamma_z=np.zeros((n,k))
            iter=0
            for i in range(max_iter):
                iter+=1
                gamma_z=E(X,pi_list,mu_list,sigma_list)
                mu_list,sigma_list,pi_list=M(X,gamma_z,mu_list)

                new_ll=cal_likelihood(X, mu_list, sigma_list, pi_list)
                if old_ll < new_ll and new_ll - old_ll < epsilon:
                    break
                old_ll=new_ll
            for i in range(n):
                label[i]=np.argmax(gamma_z[i,:])
            print(iter)
            return label,mu_list

        def acc(real_label,pred_label,k):
            """
            计算聚类精度
            :param real_label:真实标签，n*1
            :param pred_label: 预测标签，n*1
            :param k: 预测标签中分了多少类
            :return: 聚类精度
            """
            ##计算聚类精度的时候采用 Purity 计算
            ##需要注意的是找所有可能中对应的正确率最大的那一个
            assert real_label.shape[0]==pred_label.shape[0]
            n=real_label.shape[0]
            permu=list(permutations(range(k), k))#生成预测标签的全排列
            count=np.zeros(len(permu))
            for i in range(len(permu)):
                for j in range(n):
                    if real_label[j]==permu[i][pred_label[j][0]]:
                        count[i]+=1
            return np.max(count)/n
        def uci_read(path):
            '''''
            :param path:数据集
            :return: 训练集和训练集标签
            '''
            column_names = ['sepal length in cm', 'sepal width in cm',
                            ' petal length in cm', 'petal width in cm',
                            'Class']
            data = pd.read_csv(path,names=column_names)
            data = data.replace(to_replace='?', value=np.nan)  # 非法字符的替代
            data = data.dropna(how='any')  # 去掉空值，any：出现空值行则删除
            x=data[column_names[0:4]].values.copy() ## 转换为 array 格式
            ##将分类信息变为数字信息
            y=data[column_names[4:]]
            y.loc[y['Class']=='Iris-setosa']=0
            y.loc[y['Class'] == 'Iris-versicolor'] = 1
            y.loc[y['Class'] == 'Iris-virginica'] = 2
            y=y.values.copy()
            #乱序输出
            train_data = np.column_stack((x, y))
            np.random.shuffle(train_data)
            x=train_data[:,:-1]
```

```
            y=train_data[:,-1:]
            x.astype(np.float64)
            y.astype(np.float64)
            return x,y
        def uci_run(path,epsilon=1e-5):
            X,label=uci_read(path)
            acc_kmeans_list=[]
            acc_GMM_list=[]
            for i in range(1,10):
                pred_label_kmeans,center_kmeans = kmeans(X,i,epsilon)
                pred_label_GMM,center_GMM = GMM(X,i,epsilon=epsilon)
                acc_kmeans_i=acc(label,pred_label_kmeans,i)
                acc_GMM_i=acc(label,pred_label_GMM,i)
                acc_kmeans_list.append(acc_kmeans_i)
                acc_GMM_list.append(acc_GMM_i)

            kmeans_best_k=np.argmin(np.array(acc_kmeans_list))+1
            GMM_best_k = np.argmin(np.array(acc_GMM_list)) + 1

            pred_label_kmeans, center_kmeans = kmeans(X, kmeans_best_k, epsilon)

            pred_label_GMM, center_GMM = GMM(X, GMM_best_k, epsilon=epsilon)

            title_kmeans_i="kmeans,acc="+str(acc_kmeans_list[kmeans_best_k])+",k
    ="+str(kmeans_best_k)+",epsilon="+str(epsilon)
            title_GMM_i = "GMM,acc=" + str(acc_GMM_list[GMM_best_k]) + ",k=" + s
    tr(GMM_best_k) + ",epsilon=" + str(epsilon)
            show(X,pred_label_kmeans,center_kmeans,title_kmeans_i)
            show(X, pred_label_GMM, center_GMM, title_GMM_i)

        def show(X,label,center=None,title=None):
            """
            画图函数
            :param X:数据集，本实验中格式为[k*n,d]
            :param label:标签集，格式为[k*n,1]，和数据集中数据一一对应
            :param center: 每个簇的中心点坐标，本实验中格式为[k,d]
            :param title:图名
            :return:
            """
            plt.scatter(X[:,0], X[:,1], c=label[:,0], marker='.', s=25)
            if not center is None:
                plt.scatter(center[:, 0], center[:, 1], c='r', marker='x', s=250
    )
            if not title is None:
                plt.title(title)
            plt.show()

        if __name__=="__main__":
            k=3
            n=200
            d=2
            mu_list=[ [1,3], [2,5], [5,2] ]
            sigma_list=[ [[1,0],[0,2]], [[2,0],[0,3]], [[3,0],[0,4]] ]
            X,label=generate_data(k,n,d,mu_list,sigma_list)

            # pred_label,center=kmeans(X,3)
            # print(acc(label,pred_label,3))
            # show(X, label,center)
            # pred_label,center=GMM(X,k)
            # print(acc(label, pred_label, 3))
```

```
259.            # show(X, label, center)
260.            path='./iris.data'
261.            uci_run(path)
```