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：1190202107
姓名： 姚舜宇

一、实验目的

实现一个 k-means 算法和混合高斯模型，并用 EM 算法估计模型中的参数。使用一个数据集测试模型和算法。

二、实验要求及实验环境

1. 使用高斯分布产生不同均值和方差的 k 组数据。
2. 使用 k-means 聚类算法测试效果。
3. 用混合高斯模型和 EM 算法估计模型的参数，查看每次迭代后似然值的变化情况，将最终结果与自己设定的参数进行比较，考察 EM 算法是否可以得到正确的结果。
4. 寻找一个数据集进行测试，使用实现的模型进行聚类。

环境：Windows10，python3.9，pycharm

三、设计思想（本程序中的用到的主要算法及数据结构）

1. 生成数据

   设数据为$X=\begin{bmatrix}\boldsymbol{x}_1^T\\ \boldsymbol{x}_2^T\\ \vdots\\ \boldsymbol{x}_N^T\end{bmatrix}=\begin{bmatrix}x_{11} & x_{12} & \cdots & x_{1p}\\ x_{21} & x_{22} & \cdots & x_{2p}\\ \vdots & \vdots & \ddots & \vdots\\ x_{N1} & x_{N2} & \cdots & x_{Np}\end{bmatrix}$，每一个样本$\boldsymbol{x}_i=\begin{bmatrix}x_{i1}\\ x_{i2}\\ \vdots\\ x_{ip}\end{bmatrix}$，是一个 p 维的向量。

   在接下来的实验中，自己生成的数据集采用二维高斯分布。

2. 使用 k-means 算法进行聚类

   首先描述算法流程。

   输入：N 个样本的集合 X；

   输出：样本集合的聚类。

   1) 初始化。令 t=0，随机选择 k 个样本点作为初始聚类中心。
   2) 对样本进行聚类。对固定的类中心，计算每个样本到类中心的距离，将每个样本指派到与其最近的中心的类中，构成聚类结果。
   3) 计算新的类中心。对上一步骤中的聚类结果，计算当前各个类中的样本的均值，作为新的类中心。
   4) 如果迭代收敛或符合停止条件，输出聚类结果，否则令 t=t+1，返回步骤 2)。

   下面使用数学公式来说明 k-means 算法。

   记聚类集合$Cluster=\{C_1,\cdots,C_k\}$，样本矩阵$X=\begin{bmatrix}\boldsymbol{x}_1^T\\ \boldsymbol{x}_2^T\\ \vdots\\ \boldsymbol{x}_N^T\end{bmatrix}=\begin{bmatrix}x_{11} & x_{12} & \cdots & x_{1p}\\ x_{21} & x_{22} & \cdots & x_{2p}\\ \vdots & \vdots & \ddots & \vdots\\ x_{N1} & x_{N2} & \cdots & x_{Np}\end{bmatrix}$，从 N 个样本随机取出 K 个，作为 K 个簇的初始中心。记中心为$\boldsymbol{c}_{k0}=\begin{bmatrix}x_{k1}\\ x_{k2}\\ \vdots\\ x_{kp}\end{bmatrix}$，$k=1,\cdots K$。接下来遍历所有样本，计算每个样本到各个聚类中心的距离，取最小值对应的下标。计算公式为$(X-\mathbf{1}_{N\times 1}\boldsymbol{c}_k^T)\odot(X-\mathbf{1}_{N\times 1}\boldsymbol{c}_k^T)$，按列相加算得$N\times 1$矩阵。对每一个聚类中心即每一个 k 计算上式，得到 K 个$N\times 1$矩阵，按列拼接成$N\times K$矩阵，取每行最小值对应

的下标，即得到$N\times1$矩阵，每行元素代表了每个样本对应的簇。然后对于每一个聚类，计算新的聚类中心$\boldsymbol{c}_{kt}=\begin{bmatrix}\bar{x}_{k1}\\ \bar{x}_{k2}\\ \vdots\\ \bar{x}_{kp}\end{bmatrix}$。计算损失函数的方式为每一个聚类每个样本点到聚类中心的距离之和，即$loss=\sum_{k=1}^{K}\sum_{C(i)=k}\|\boldsymbol{x}_i-\bar{\boldsymbol{x}}_k\|^2$。迭代终止条件为两次迭代的损失函数差距不超过 10 的-5 次方。

3. 使用高斯混合模型并用 EM 算法估计参数

高斯混合模型：是指具有如下形式的概率分布模型：$p(x)=\sum_{k=1}^{K}\alpha_k N(x|\mu_k,\Sigma_k)$，其中$\alpha_k$为权重，$\sum_{k=1}^{K}\alpha_k=1$，$N(x|\mu_k,\Sigma_k)=\frac{1}{(2\pi)^{\frac{2}{p}}|\Sigma_k|^{\frac{1}{2}}}\exp\left\{-\frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)\right\}$。在本实验中，令 x 为观测变量，z 为隐变量，$z_i$表示第 i 个样本属于的聚类编号，即某一个样本属于哪一个聚类。聚类用c_k表示，$k=1,\cdots K$令某一个样本x_i属于聚类c_k的概率为$p(z=c_k)=p_k$。每一个聚类的概率分布都是高斯分布，均值和协方差矩阵分别为μ_k和Σ_k。

EM 算法的核心公式为$\theta^{(t+1)}=arg\max E_{z|x,\theta^{(t)}}[\log\ p(X,Z|\theta)]$，表示第 t+1 轮的参数迭代公式。$Q(\theta,\theta^{(t)})=E_{z|x,\theta^{(t)}}[\log\ p(X,Z|\theta)]$，接下来对$Q(\theta,\theta^{(t)})$进行变换。

$$\begin{aligned}Q(\theta,\theta^{(t)})&=\sum_z\log\prod_{i=1}^{N}p(X,Z|\theta)p(Z|X,\theta^{(t)})\\&=\sum_{z_1,\cdots,z_N}\sum_{i=1}^{N}\log\ p(x_i,z_i|\theta)\prod_{i=1}^{N}p(z_i|x_i,\theta^{(t)})\\&=\sum_{z_1,\cdots,z_N}[\log\ p(x_1,z_1|\theta)+\cdots+\log\ p(x_N,z_N|\theta)]\prod_{i=1}^{N}p(z_i|x_i,\theta^{(t)})\\&=\sum_{i=1}^{N}\sum_{z_i}[\log\ p(x_i,z_i|\theta)]p(z_i|x_i,\theta^{(t)})\end{aligned}$$

由于$p(x,z)=p(z)p(x|z)=p_z\cdot N(x|\mu_z,\Sigma_z)$，$Q(\theta,\theta^{(t)})$可以继续变换：

$$
\begin{aligned}
Q(\theta,\theta^{(t)}) &= \sum_{i=1}^{N}\sum_{z_i}[\log\ p_{z_i}\cdot N(x_i|\mu_{z_i},\Sigma_{z_i})]\,p(z_i|x_i,\theta^{(t)}) \\
&= \sum_{z_i}\sum_{i=1}^{N}[\log\ p_{z_i}\cdot N(x_i|\mu_{z_i},\Sigma_{z_i})]\,p(z_i|x_i,\theta^{(t)}) \\
&= \sum_{k=1}^{K}\sum_{i=1}^{N}[\log\ p_k\cdot N(x_i|\mu_k,\Sigma_k)]\,p(z_i=c_k|x_i,\theta^{(t)}) \\
&= \sum_{k=1}^{K}\sum_{i=1}^{N}[\log\ p_k+\log\ N(x_i|\mu_k,\Sigma_k)]\,p(z_i=c_k|x_i,\theta^{(t)})
\end{aligned}
$$

其中$p(z_i=c_k|x_i,\theta^{(t)})=\dfrac{p(z_i=c_k,x_i)}{p(x_i)}=\dfrac{p_k\cdot N(x_i|\mu_k^{(t)},\Sigma_k^{(t)})}{\sum_{k=1}^{K}p_k\cdot N(x_i|\mu_k^{(t)},\Sigma_k^{(t)})}$，为一个常量。

由$\theta^{(t+1)}=arg\max_{\theta}Q(\theta,\theta^{(t)})$，求$p^{(t+1)}=[\,p_1^{(t+1)}\ \cdots\ p_K^{(t+1)}]$、$\mu^{(t+1)}=[\,\mu_1^{(t+1)}\ \cdots\ \mu_K^{(t+1)}]$、$\Sigma^{(t+1)}=[\,\Sigma_1^{(t+1)}\ \cdots\ \Sigma_K^{(t+1)}]$。

1) 求$p^{(t+1)}=[\,p_1^{(t+1)}\ \cdots\ p_K^{(t+1)}]$

根据概率的归一性，可以构造关于p的约束优化问题（省略了部分无关项）：

$$
\begin{cases}
\max\limits_{p}\sum\limits_{k=1}^{K}\sum\limits_{i=1}^{N}[\log\ p_k]\,p(z_i=c_k|x_i,\theta^{(t)}) \\
s.t.\ \sum\limits_{k=1}^{K}p_k=1
\end{cases}
$$

使用拉格朗日乘子法，构造拉格朗日函数：

$$
L(p,\lambda)=\sum_{k=1}^{K}\sum_{i=1}^{N}[\log\ p_k]\,p(z_i=c_k|x_i,\theta^{(t)})+\lambda\left(\sum_{k=1}^{K}p_k-1\right)
$$

对p_k求偏导并令其为0：$\sum\limits_{i=1}^{N}\dfrac{1}{p_k}p(z_i=c_k|x_i,\theta^{(t)})+\lambda=0$，解得

$p_k^{(t+1)}=\dfrac{1}{N}\sum\limits_{i=1}^{N}p(z_i=c_k|x_i,\theta^{(t)})$。则得到$p^{(t+1)}=[\,p_1^{(t+1)}\ \cdots\ p_K^{(t+1)}]$。

2) 求$\mu^{(t+1)}=[\,\mu_1^{(t+1)}\ \cdots\ \mu_K^{(t+1)}]$

省略无关项，定义$L=\sum\limits_{k=1}^{K}\sum\limits_{i=1}^{N}[\log\ N(x_i|\mu_k,\Sigma_k)]\,p(z_i=c_k|x_i,\theta^{(t)})$。对$\mu_k$求偏导并令导数为0。

$$\frac{\partial L}{\partial \mu_k}=\sum_{i=1}^{N} p(z_i=c_k|x_i,\theta^{(t)})\cdot\left[-\frac{1}{2}(-2\Sigma_k^{-1}x_i+2\Sigma_k^{-1}\mu_k)\right]=0$$，得到

$$\mu_k^{(t+1)}=\frac{\sum_{i=1}^{N} p(z_i=c_k|x_i,\theta^{(t)})x_i}{\sum_{i=1}^{N} p(z_i=c_k|x_i,\theta^{(t)})}$$，故$\mu^{(t+1)}=[\mu_1^{(t+1)}\ \cdots\ \mu_K^{(t+1)}]$。

3) 求$\Sigma^{(t+1)}=[\Sigma_1^{(t+1)}\ \cdots\ \Sigma_K^{(t+1)}]$

同理，对Σ_k求偏导并令导数为 0。

$$\frac{\partial L}{\partial \Sigma_k}=\sum_{i=1}^{N} p(z_i=c_k|x_i,\theta^{(t)})\left[-\frac{1}{2}(\Sigma_k^{-1}-\Sigma_k^{-1}(x_ix_i^T-x_i\mu_k^T-\mu_kx_i^T+\mu_k\mu_k^T)\Sigma_k^{-1})\right]=0$$

得到$\Sigma_k^{(t+1)}=\frac{\sum_{i=1}^{N} p(z_i=c_k|x_i,\theta^{(t)})\,(x_i-\mu_k^{(t)})\,(x_i-\mu_k^{(t)})^T}{\sum_{i=1}^{N} p(z_i=c_k|x_i,\theta^{(t)})}$

故$\Sigma^{(t+1)}=[\Sigma_1^{(t+1)}\ \cdots\ \Sigma_K^{(t+1)}]$。

四、实验结果与分析

在下面的样本中，为便于表示，选择 p=2。

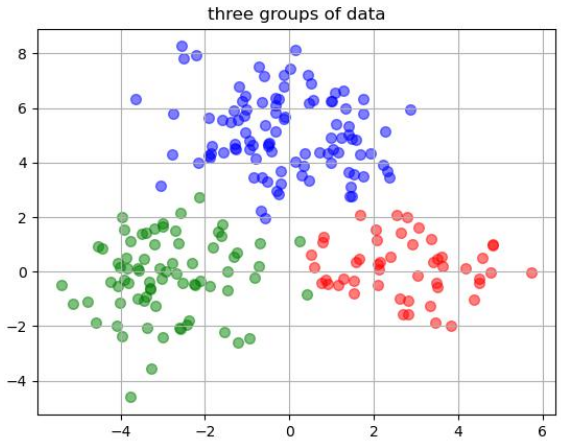
生成三组二维高斯分布的数据。三组数据的参数如下：

第一组：样本量：50；均值：[3,0]，协方差矩阵：$\begin{bmatrix}1.5 & 0\\ 0 & 1\end{bmatrix}$，标记为红色。

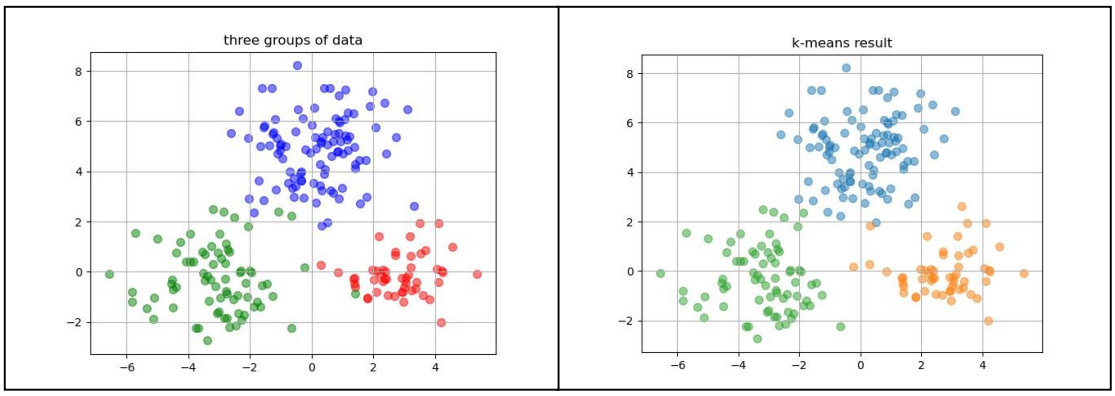
第二组：样本量：75；均值：[-3,0]，协方差矩阵：$\begin{bmatrix}1.5 & 0\\ 0 & 2\end{bmatrix}$，标记为绿色。

第三组：样本量：100；均值：[0,5]，协方差矩阵：$\begin{bmatrix}2 & 0\\ 0 & 2\end{bmatrix}$，标记为蓝色。

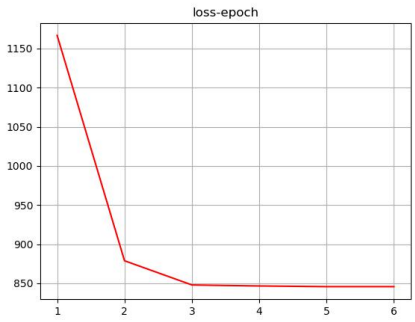
将数据进行可视化，效果如下：



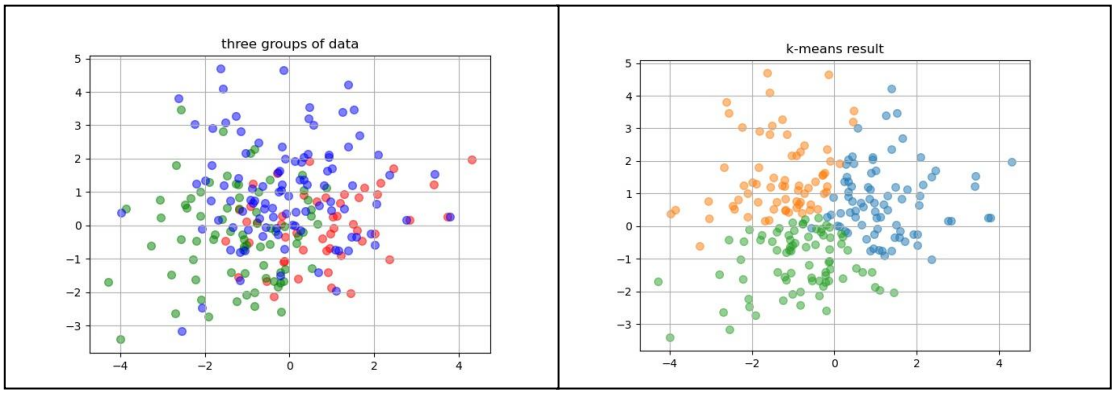
下面使用二维高斯分布的样本来测试 k-means 算法。如下图所示。



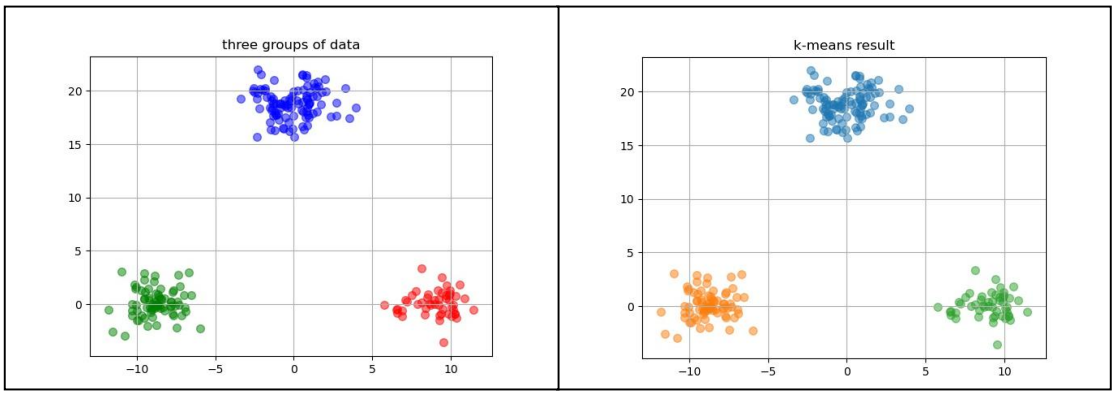
其中左图为生成的数据，右图为 k-means 算法聚类的结果。能够看出，效果较好。除了部分不同类的数据混合在一起的情况难以分开，其他都能够正确分类。多次实验，得出结果共计迭代了 4 至 9 轮不等，损失函数变化的情况如下图。



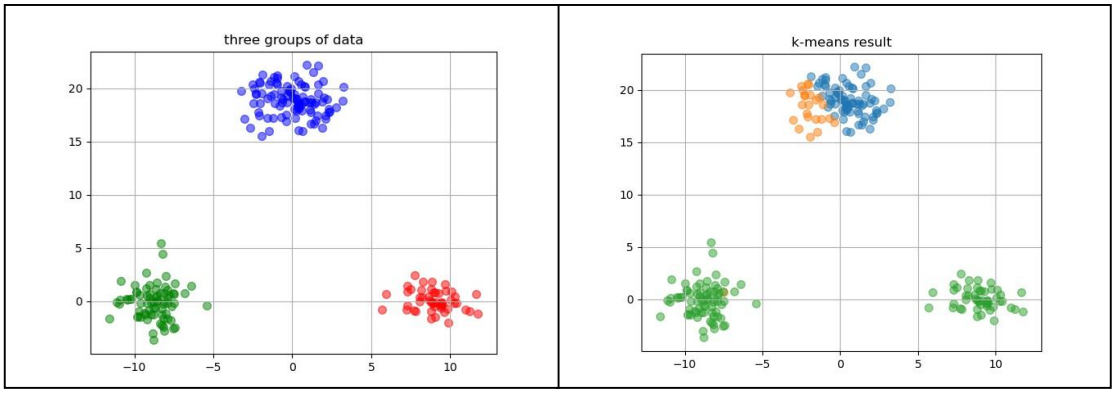
使用不同的数据分布参数进行实验。
当数据混合程度较高时，结果如下。



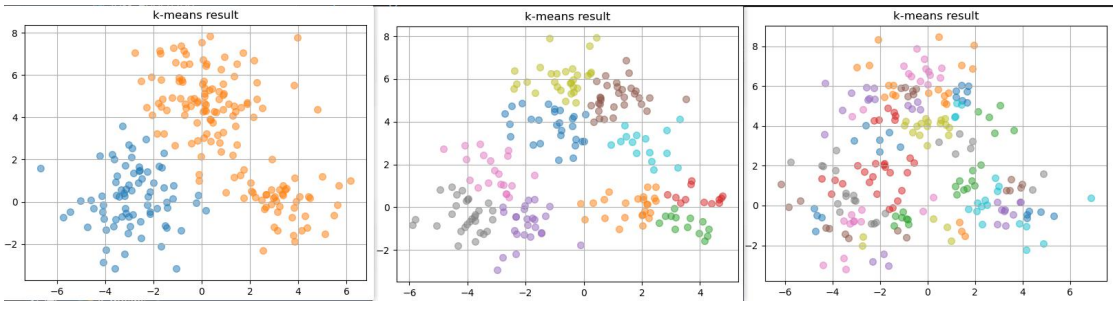
从数据图中可以显然看出，样本混合程度较高，无法准确分类混合区域的样本。K-means 算法的结果符合人的主观分类方法，符合预期。
当数据完全没有混合时，结果如下。



多次实验，发现每次情况可能不同，聚类结果也可能有较大错误。



右图中可以看出，这时 k-means 算法将下方的距离较远的两类数据分为了一类，而上方的一类数据分割成了两类。通过查阅资料，发现 k-means 算法对初始值较为敏感，每次计算出的结果可能不同，并且对离群点非常敏感，因为 k-means 算法中使用欧氏距离进行计算，所以一个离群点对损失函数的影响非常大。
再使用不同的 k，即聚类数目进行实验。分别取 2，10，50。结果分布如下图从左到右所示。



随着 k 的增加，越来越难以看出样本的分布情况。所以，k-means 算法中，k 的选择也是一个需要实验的问题。

下面使用最初生成的数据对高斯混合模型的 EM 算法进行测试。
数据的生成参数如下：

第一组：样本量：50；均值：[3,0]，协方差矩阵：$\begin{bmatrix} 1.5 & 0 \\ 0 & 1 \end{bmatrix}$。

第二组：样本量：75；均值：[-3,0]，协方差矩阵：$\begin{bmatrix}1.5 & 0\\ 0 & 2\end{bmatrix}$。

第三组：样本量：100；均值：[0,5]，协方差矩阵：$\begin{bmatrix}2 & 0\\ 0 & 2\end{bmatrix}$。

当两次迭代计算出的$Q(\theta,\theta^{(t)})$差距不超过 10 的-5 次方时，结束迭代。进行 3 次实验，最终得到的参数如下。

```
[0.4528339482941928, 0.2271635801388543, 0.32000247156695266]
[matrix([[-0.44673245,  4.86697828]]), matrix([[3.11937641, 0.04277351]]), matrix([[-2.76780167,  0.1112188 ]])]
[matrix([[1.94877179, 0.06647374],
        [0.06647374, 1.90106976]]), matrix([[ 1.42538027, -0.03250691],
        [-0.03250691,  1.18526231]]), matrix([[1.47614319, 0.23820874],
        [0.23820874, 1.99154713]])]
```

```
[0.33872679766378533, 0.22234204772400462, 0.43893115461221005]
[matrix([[-3.08546121,  0.49164902]]), matrix([[2.87170455, 0.03697125]]), matrix([[0.04371886, 5.26140034]])]
[matrix([[ 2.02213006, -0.18816309],
        [-0.18816309,  1.87800314]]), matrix([[1.58828184, 0.162497  ],
        [0.162497  , 0.60646851]]), matrix([[1.65652093, 0.05708932],
        [0.05708932, 1.66801594]])]
```

```
[0.34085428372658266, 0.44061810447203825, 0.21852761180137875]
[matrix([[-2.9102273 ,  0.08375262]]), matrix([[-0.09906691,  4.81705056]]), matrix([[ 3.20916128, -0.07424692]])]
[matrix([[1.30643593, 0.08655277],
        [0.08655277, 2.21673459]]), matrix([[1.80001288, 0.13969646],
        [0.13969646, 2.65936363]]), matrix([[ 1.31381456, -0.05073283],
        [-0.05073283,  0.70993046]])]
```

进行一定的整理，结果如下。

第一次实验，三组数据参数如下：

一：$p_1=0.227$，$\mu_1=[3.12\ \ 0.04]$，$\Sigma_1=\begin{bmatrix}1.43 & -0.03\\ -0.03 & 1.19\end{bmatrix}$。

二：$p_2=0.320$，$\mu_2=[-2.77\ \ 0.11]$，$\Sigma_2=\begin{bmatrix}1.48 & 0.23\\ 0.23 & 1.99\end{bmatrix}$。

三：$p_3=0.453$，$\mu_3=[-0.45\ \ 4.87]$，$\Sigma_3=\begin{bmatrix}1.95 & 0.06\\ 0.06 & 1.90\end{bmatrix}$。

第二次实验，三组数据参数如下：

一：$p_1=0.219$，$\mu_1=[2.87\ \ 0.04]$，$\Sigma_1=\begin{bmatrix}1.59 & 0.16\\ 0.16 & 0.61\end{bmatrix}$。

二：$p_2=0.341$，$\mu_2=[-3.09\ \ 0.49]$，$\Sigma_2=\begin{bmatrix}2.02 & -0.19\\ -0.19 & 1.88\end{bmatrix}$。

三：$p_3=0.441$，$\mu_3=[0.04\ \ 5.26]$，$\Sigma_3=\begin{bmatrix}1.65 & 0.06\\ 0.06 & 1.67\end{bmatrix}$。

第三次实验，三组数据参数如下：

一：$p_1=0.227$，$\mu_1=[3.21\ \ -0.07]$，$\Sigma_1=\begin{bmatrix}1.31 & -0.05\\ -0.05 & 0.71\end{bmatrix}$。

二：$p_2=0.320$，$\mu_2=[-2.91\ \ 0.08]$，$\Sigma_2=\begin{bmatrix}1.31 & 0.09\\ 0.09 & 2.22\end{bmatrix}$。

三：$p_3=0.453$，$\mu_3=[-0.10\ \ 4.82]$，$\Sigma_3=\begin{bmatrix}1.80 & 0.14\\ 0.14 & 2.66\end{bmatrix}$。

和真实数据参数相比，有一定的差距，但多次实验也可以获得参数的大致信息。

最后找一个简单的问题数据，用前面实现的模型进行聚类。

使用鸢尾花数据集。该数据集中每个样本有 4 个属性，共分为 3 类，每类样本数均为 50。分别使用 k-means 算法和 GMM 模型进行求解。

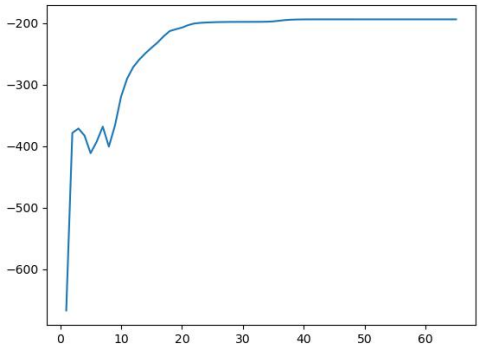
使用 k-means 算法。令 K=3，即将数据分为三类。多次实验，记录每类的个数。发现结果大部分都在 40，50，60 附近浮动。损失函数最终收敛到 79 附近。如下所示。

```
epoch: 8   loss: 78.94084142614601
50
38
62
```

```
epoch: 4   loss: 78.94506582597731
50
61
39
```

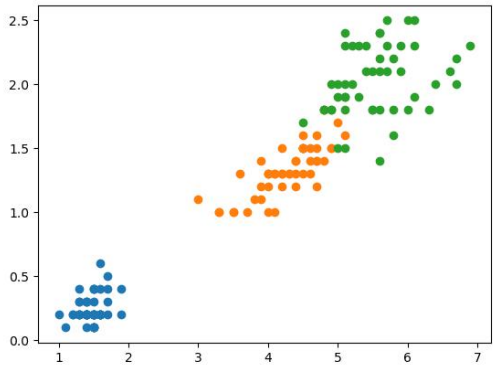
使用对高斯混合模型的 EM 算法进行实验。较好的一次结果和 Q 函数的值随迭代轮数的变化曲线情况如下图所示。

```
[0.33333333333333326, 0.36747339397048967, 0.29919327269617707]
[matrix([[5.006, 3.418, 1.464, 0.244]]), matrix([[6.54454874, 2.94866118, 5.47955362, 1.98460507]]), matrix([[5.91496966, 2.77784365, 4.20155337, 1.29696
[matrix([[0.121764, 0.098292, 0.015816, 0.010336],
        [0.098292, 0.142276, 0.011448, 0.011208],
        [0.015816, 0.011448, 0.029504, 0.005584],
        [0.010336, 0.011208, 0.005584, 0.011264]]), matrix([[0.3870443 , 0.09220792, 0.3028117 , 0.06165101],
        [0.09220792, 0.1103377 , 0.08428756, 0.05601149],
        [0.3028117 , 0.08428756, 0.32779726, 0.07452996],
        [0.06165101, 0.05601149, 0.07452996, 0.08579769]]), matrix([[0.27531878, 0.09694137, 0.18466241, 0.05439075],
        [0.09694137, 0.09264604, 0.09114317, 0.04299735],
        [0.18466241, 0.09114317, 0.20063047, 0.06097849],
        [0.05439075, 0.04299735, 0.06097849, 0.03199697]])]
```



计算得到的每一类的概率都在 1/3 附近，符合每类样本量都是 50 的数据集。多次实验，结果表明 EM 算法对初值极为敏感，每次实验的结果可能差距较大。EM 算法对初值敏感是一种可能的原因，其次还有数据集的每一类可能并不完全地服从高斯分布，这也是影响结论的另一个原因。

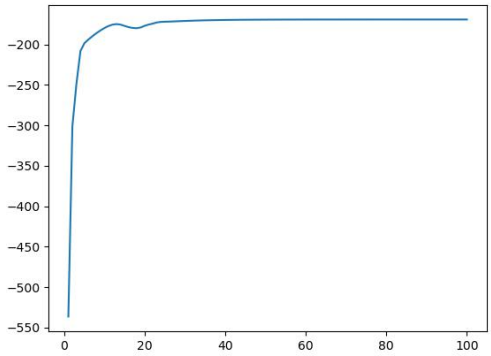
取鸢尾花后两个特征 petal_length、petal_width 进行测试，将三类样本绘制出散点图如下：



发现具有较好的可分性。使用这两个特征进行两种算法的测试。

k-means 算法：分为三类时，多次实验，三类样本个数约为 52，48，50，和真实值很接近。

使用对高斯混合模型的 EM 算法：迭代 100 次后，Q 函数值变化趋势如下图所示。




```
[0.33333294463819624, 0.3257218635398144, 0.34094519182198935]
[matrix([[1.46399961, 0.24399976]]), matrix([[5.55313244, 2.03276493]]), matrix([[4.28775873, 1.33516335]])]
[matrix([[0.02950388, 0.00558394],
        [0.00558394, 0.01126395]]), matrix([[0.30927474, 0.05041299],
        [0.05041299, 0.07331387]]), matrix([[0.24164031, 0.07948212],
        [0.07948212, 0.04146584]])]
```

输出结果与真实值很接近。

所以对于多维度的特征时，有时可能需要进行特征选取或降维处理。

考虑到随机初始化的极大不确定性，下面首先使用 k-means 算法对参数进行初始化，然后作为 EM 算法的初始解进行迭代，测试最终计算结果。

```
[0.3333329440343897, 0.3256693086897516, 0.3409977472758587]
[matrix([[1.46399961, 0.24399976]]), matrix([[5.55324241, 2.03281734]]), matrix([[4.28784871, 1.33522081]])]
[matrix([[0.02950388, 0.00558394],
        [0.00558394, 0.01126395]]), matrix([[0.30923398, 0.05038128],
        [0.05038128, 0.07330461]]), matrix([[0.24167061, 0.07950735],
        [0.07950735, 0.04148475]])]
```

对于鸢尾花数据集，发现具有良好的结果。多次实验，发现每次结果相差较小，具有较好的稳定性。

五、结论

k-means 算法是一种简单的无监督学习聚类算法。对于 k-means 算法，首先需要注意的是 K 值的选择，一般来说，需要根据对数据的先验进行选取一个合适的 K 值，确定 K 之后，需要随机初始化 K 个聚类中心。这就对算法引入了一定的不确定性。如果聚类中心选择不合适，会影响聚类的结果。一般来说，中心选择不要太靠近。在每次迭代中，需要计算每个样本到每个聚类中心的距离，在本次实验中使用欧氏距离进行计算。对不同的数据集，衡量距离的方式也可能对结果有一定影响。另外，k-means 算法求解的只是局部最优，所以要得到最佳结果，需要进行多次实验计算。

高斯混合模型可以看作是由 K 个单高斯模型组合而成的模型，这 K 个子模型是混合模型的隐变量。高斯分布是确定数据均值和方差情况下的最大熵分布，所以一般情况下，高斯分布对一组数据的表示能力较强，这也让高斯混合模型对于具有隐变量的概率模型有着较强的拟合能力。要估计每一个子模型的高斯分布参数，可以使用 EM 算法进行计算。

EM 算法是一种用于含有隐变量的概率参数模型的最大似然估计或极大后验概率估计。分为 E 步和 M 步。E 步主要通过观察数据和现有模型来估计参数，然后用这个估计的参数值来计算似然函数的期望值；而 M 步是寻找似然函数最大化时对应的参数。由于算法会保证在每次迭代之后似然函数都会增加，所以函数最终会收敛。由于初始化的不确定性且 EM 算法对初值较为敏感，所以聚类结果随不同的初始值的波动较大。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。为了缓解这种情况，可以首先使用 k-means 算法计算出一个初始解，然后再使用 EM 算法进行迭代。实验表明，这种方法具有较强的稳定性。另外，EM 算法可以保证收敛到一个稳定点，但是却不能保证收敛到全局的极大值点，因此它

是局部最优的算法，当然，如果优化目标函数是凸函数，则 EM 算法可以保证收敛到全局最大值。

六、参考文献

《统计学习方法》李航

七、附录：源代码（带注释）

1.GenerateData.py

```
import numpy as np
import matplotlib.pyplot as plt


def Generate_data(draw=True):
    """
    generate 3 groups of gaussian data
    :param draw: draw the 2D scatter picture
    :return: data
    """
    # 第一组高斯分布数据的参数
    num1 = 50
    mean1 = [3, 0]
    cov1 = [[1.5, 0], [0, 1]]
    data1 = np.random.multivariate_normal(mean1, cov1, num1)
    # 第二组高斯分布数据的参数
    num2 = 75
    mean2 = [-3, 0]
    cov2 = [[1.5, 0], [0, 2]]
    data2 = np.random.multivariate_normal(mean2, cov2, num2)
    # 第三组高斯分布数据的参数
    num3 = 100
    mean3 = [0, 5]
    cov3 = [[2, 0], [0, 2]]
    data3 = np.random.multivariate_normal(mean3, cov3, num3)
    if draw:
        plt.figure(1)
        # 画出散点图
        plt.scatter(data1[:, 0], data1[:, 1], s=49, c='r', alpha=0.5)
        plt.scatter(data2[:, 0], data2[:, 1], s=49, c='g', alpha=0.5)
        plt.scatter(data3[:, 0], data3[:, 1], s=49, c='b', alpha=0.5)
        plt.grid()
        plt.title('three groups of data')
        plt.savefig('data.jpg')
```

```
    X = np.vstack((np.vstack((np.mat(data1), np.mat(data2))), np.mat(data3)))
    permutation = np.random.permutation(X.shape[0])
    shuffled_X = X[permutation, :]  # 将数据顺序打乱
    return shuffled_X


if __name__ == '__main__':
    plt.ion()
    X = Generate_data()
    plt.ioff()
    plt.show()
```

2.GMM_EM.py

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


def cal_Pz(X, p, mu, sigma):
    """
    计算 p(zi=ck|xi,theta(t))的 N*K 矩阵
    :param X: 样本
    :param p: 各个隐变量的概率
    :param mu: 各个高斯分布的均值
    :param sigma: 各个高斯分布的协方差矩阵
    :return: p(zi=ck|xi,theta(t))的 N*K 矩阵
    """
    K = len(p)
    N = X.shape[0]
    P = np.mat(np.zeros(shape=(N, K)))
    for i in range(N):
        Denominator = 0
        for k in range(K):
            Denominator += p[k] * multivariate_normal.pdf(X[i], np.array(mu[k])[0], sigma[k], allow_singular=True)
        for k in range(K):
            P[i, k] = p[k] * multivariate_normal.pdf(X[i], np.array(mu[k])[0], sigma[k], allow_singular=True) / Denominator
    return P

```

```
28.
29. def cal_Q(X, new_p, new_mu, new_sigma, p, mu, sigma):
30.     """
31.     计算 Q(theta,theta(t))函数的值
32.     :param X: 数据
33.     :param new_p: 新的各个隐变量的概率
34.     :param new_mu: 新的各个高斯分布的均值
35.     :param new_sigma: 新的各个高斯分布的协方差矩阵
36.     :param p: 各个隐变量的概率
37.     :param mu: 各个高斯分布的均值
38.     :param sigma: 各个高斯分布的协方差矩阵
39.     :return: Q 函数的值
40.     """
41.     K = len(new_p)
42.     N = X.shape[0]
43.     Q = 0
44.     P = cal_Pz(X, p, mu, sigma)
45.     for k in range(K):
46.         for i in range(N):
47.             Q += (np.log(new_p[k] + 1e-5) +
48.                   np.log(multivariate_normal.pdf(X[i], np.array(new_mu[k])[0
    ], new_sigma[k], allow_singular=True) + 1e-5)) * P[i, k]
49.     return Q
50.
51.
52. def initialization(K, X):
53.     """
54.     随机初始化各个参数
55.     :param K: 聚类数目
56.     :param X: 样本
57.     :return: 随机初始化的参数
58.     """
59.     d = X.shape[1]
60.     p = np.random.random(K)
61.     psum = np.sum(p)
62.     p /= psum
63.     mu = []
64.     sigma = []
65.     for k in range(K):
66.         mu.append(np.mat(np.random.randn(d)))
67.         r = np.random.random(d)
68.         sigma.append(np.mat(np.diag(r)))
69.     return p, mu, sigma
70.
```

```
71.
72. def initialization_k_means(K, X):
73.     """
74.     使用 k_means 算法初始化参数
75.     :param K: 聚类数目
76.     :param X: 样本
77.     :return: 初始化的参数
78.     """
79.     X1 = k_means(X, K)
80.     N = X.shape[0]
81.     p = []
82.     mu = []
83.     sigma = []
84.     for i in range(K):
85.         p.append(X1[i].shape[0] / N)
86.         mu.append(np.mean(X1[i], axis=0))
87.         sigma.append(np.cov(X1[i].T))
88.     return p, mu, sigma
89.
90.
91. def GMM_EM(X, p, mu, sigma, max_epoch=100):
92.     """
93.     使用 GM 算法计算高斯混合模型的各个子高斯分布的参数
94.     :param max_epoch: 最大迭代轮数
95.     :param X: 数据
96.     :param p: 各个隐变量的概率初值
97.     :param mu: 各个高斯分布的均值初值
98.     :param sigma: 各个高斯分布的协方差初值
99.     :return: 高斯混合模型的各个子高斯分布的参数
100.     """
101.     Q = 0
102.     new_Q = 1
103.     N = X.shape[0]
104.     K = len(p)
105.     epoch = 0
106.     epoch_list = []
107.     Q_list = []
108.     new_p = []
109.     new_mu = []
110.     new_sigma = []
111.     while abs(new_Q - Q) > 1e-5 and epoch < max_epoch:  # 当两次迭代的 Q 值差
    距大于 1e-5 时继续迭代
112.         epoch += 1
113.         Q = new_Q
```

```
        new_p = []
        new_mu = []
        new_sigma = []
        P = cal_Pz(X, p, mu, sigma)
        for k in range(K):  # 根据各个参数的迭代公式计算下一轮的参数值
            temp = P.sum(axis=0)[0, k]
            new_p.append(temp / N)
            new_mu.append(np.multiply(1 / temp, P[:, k].T @ X))
            Numerator = 0
            for i in range(N):
                Numerator += np.multiply(P[i, k], (X[i] - mu[k]).T @ (X[i] - mu[k]))
            new_sigma.append(np.multiply(1 / temp, Numerator))
        new_Q = cal_Q(X, new_p, new_mu, new_sigma, p, mu, sigma)  # 计算新的Q值
        print(new_Q)
        epoch_list.append(epoch)
        Q_list.append(new_Q)
        p = new_p
        mu = new_mu
        sigma = new_sigma
    plt.plot(epoch_list, Q_list)  # 画出迭代轮数-Q函数值的变化曲线图
    return new_p, new_mu, new_sigma


if __name__ == '__main__':
    X = Generate_data()
    K = 3
    # p, mu, sigma = initialization(K, X)  # 初始化参数
    p, mu, sigma = initialization_k_means(K, X)  # 使用k-means算法初始化参数
    p, mu, sigma = GMM_EM(X, p, mu, sigma)  # 调用EM算法计算参数
    print(p)
    print(mu)
    print(sigma)
    plt.show()
```

3.k_means.py

```
import matplotlib.pyplot as plt
import numpy as np
import random
from GenerateData import Generate_data

```

```
6.
7.  # 簇的类
8.  class cluster:
9.      def __init__(self, x):
10.         self.center = x  # 簇的中心
11.         self.X = 0
12.
13.     def add(self, x):
14.         self.X = x
15.         self.center = self.X.mean(axis=0)
16.
17.
18. def cal_loss(Cluster_set):
19.     """
20.     计算损失函数的值
21.     :param Cluster_set: 簇的集合
22.     :return: 损失函数的值
23.     """
24.     l = 0
25.     for i in range(len(Cluster_set)):
26.         center = Cluster_set[i].center
27.         temp = Cluster_set[i].X - center
28.         temp = np.multiply(temp, temp)
29.         l += np.sum(temp)
30.     return l
31.
32.
33. def k_means(X, k):
34.     """
35.     k-means algorithm
36.     :param X: data
37.     :param k: the number of cluster
38.     :return: 每一个聚类里的数据
39.     """
40.     Cluster_set = []
41.     l = random.sample(range(X.shape[0]), k)
42.     for i in range(len(l)):
43.         Cluster_set.append(cluster(X[l[i]]))  # 随机选择 k 个聚类中心
44.     loss_list = []
45.     loss = 100
46.     new_loss = 200
47.     epoch_list = []
48.     epoch = 0
```

```
    while abs(new_loss - loss) > 1e-5:  # 当两次迭代的损失函数差距大于 1e-5 则继续循环
        epoch += 1
        loss = new_loss
        dis = 0
        for i in range(k):  # 计算每个样本到每个聚类中心的距离
            center = Cluster_set[i].center
            temp = X - center
            temp = np.multiply(temp, temp)
            temp = np.sum(temp, axis=1)
            if i == 0:
                dis = temp
            else:
                dis = np.hstack((dis, temp))
        classification = dis.argmin(axis=1)
        cluster_X = [0]*k
        flag_cluster_X = [0]*k
        for i in range(X.shape[0]):  # 将每个样本加入距离最短的聚类内并更新每个聚类中心
            cl = classification[i][0, 0]
            if flag_cluster_X[cl] == 0:
                cluster_X[cl] = X[i]
                flag_cluster_X[cl] = 1
            else:
                cluster_X[cl] = np.vstack((cluster_X[cl], X[i]))
        for i in range(k):
            Cluster_set[i].add(cluster_X[i])
        new_loss = cal_loss(Cluster_set)
        print('epoch: '+str(epoch)+'   loss: '+str(new_loss))
        loss_list.append(new_loss)
        epoch_list.append(epoch)
    plt.figure(2)
    plt.plot(epoch_list, loss_list, c='r')  # 画出迭代轮数-损失函数变化曲线
    plt.title('loss-epoch')
    plt.grid()
    plt.savefig('loss_epoch.jpg')
    X = []
    for i in range(len(Cluster_set)):
        X.append(Cluster_set[i].X)
    return X


if __name__ == '__main__':
    X = Generate_data()
```

```
    k = 50
    res = k_means(X, k)
    plt.ion()
    plt.figure(3)
    for i in range(k):
        plt.scatter(np.array(res[i])[:, 0], np.array(res[i])[:, 1], s=49, al
pha=0.5)  # 画出 k-means 算法结果的散点图
    plt.grid()
    plt.title('k-means result')
    plt.savefig('k_means_result.jpg')
     plt.ioff()
     plt.show()
```

4.iris_clustering.py

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from k_means import k_means
from GMM_EM import GMM_EM, initialization, initialization_k_means


def use_k_means(X, K):
    """
    使用 k-means 算法聚类鸢尾花数据集
    :param X: 数据
    :param K: 聚类数目
    :return: 无
    """
    ret_X = k_means(X, K)
    for i in range(len(ret_X)):
        print(ret_X[i].shape[0])


def use_GMM_EM(X, K):
    """
    调用 EM 算法计算鸢尾花数据集高斯混合模型的各个子模型的参数
    :param X: 数据
    :param K: 聚类数目
    :return: 无
    """
    p, mu, sigma = initialization_k_means(K, X)  # 初始化参数
    p, mu, sigma = GMM_EM(X, p, mu, sigma)  # 使用 EM 算法计算参数
    print(p)
```

```
    print(mu)
    print(sigma)
    plt.show()


if __name__ == '__main__':
    iris = pd.read_csv('iris.csv')
    iris = iris.drop(['class'], axis=1)
    iris = iris.drop(['sepal_length'], axis=1)
    iris = iris.drop(['sepal_width'], axis=1)
    X = np.mat(np.array(iris))
    permutation = np.random.permutation(X.shape[0])
    X = X[permutation, :]
    K = 3
    # use_k_means(X, K)
    use_GMM_EM(X, K)
```