



哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

2020年春季学期
计算学部《机器学习》课程
Lab3 实验报告

姓名	许健
学号	1183710113
班号	1837101
电子邮件	941197279@qq.com
手机号码	18945062342

1 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。

2 实验要求及实验环境

2.1 实验要求

测试：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

Windows 10, Python 3.8.5, Jupyter notebook

3 实验原理

本实验分为两部分：K-means和GMM，这两部分都是属于EM算法，而EM算法主要分为两步：

- E步：求期望
- M步：求最大似然

E步是调整分布，M步是根据E步得到的分布求得当前分布下取到最大似然时的参数，然后更新参数，再次进入E步根据得到的参数调整新的分布，如此往复循环，直到参数收敛。

3.1 K-means

给定训练样本 $X = \{x_1, x_2, \dots, x_m\}$ ，和划分聚类的数量 k ，给出一个簇划分 $C = C_1, C_2, \dots, C_k$ ，使得该划分的平方误差 E 最小化，即使下式取最小值：

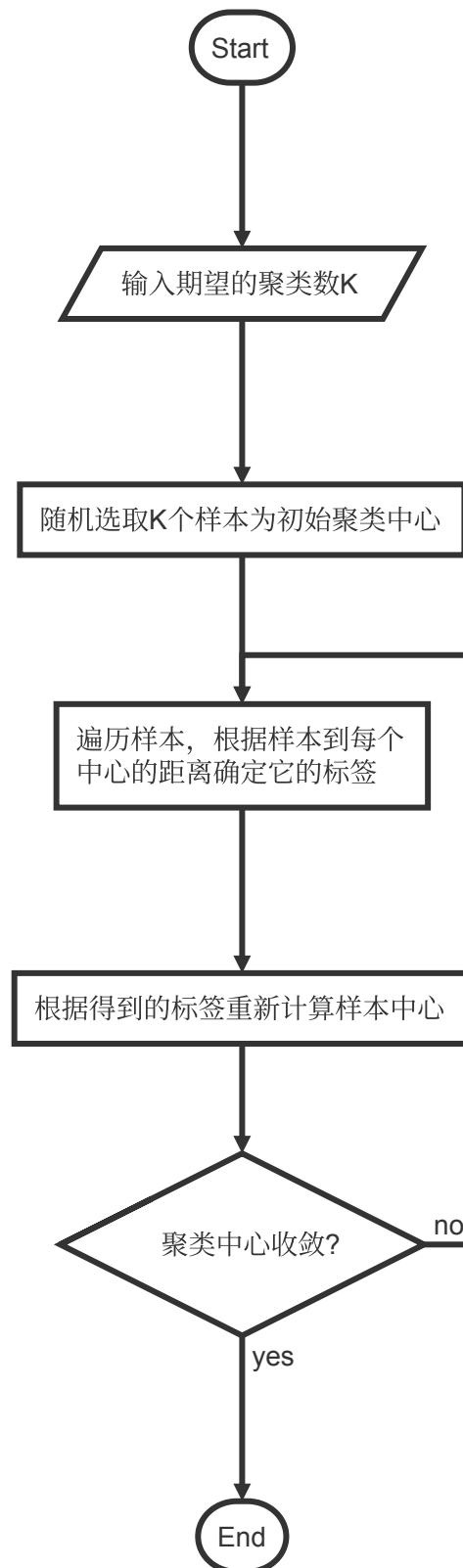
$$E = \sum_{i=1}^k \sum_{x \in C_i} \|x - \mu_i\|_2^2$$

其中， $\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x_i$ ，它是簇 C_i 的均值向量。 E 刻画了簇内样本围绕簇的均值向量的紧密程度， E 越小簇内样本的相似度越高。

具体迭代过程如下：

1. 根据输入的超参数 K 首先初始化一些向量（可以从现有的向量中挑选），作为各簇的均值向量。
2. 根据初始化的均值向量给出训练样本的一个划分，计算各个训练样本到各个均值向量的距离，找出距离最近的均值向量，并将该样本分至该均值向量所代表的簇。
3. 根据新的簇划分，重新计算每个簇的均值向量，如果新的均值向量与旧的均值向量差小于 ϵ ，则认为算法收敛；否则，更新均值向量，回到第2步重新迭代求解。

K-means算法的流程图如下：



K-means实现：

```
1 def kmeans(X, k, epsilon=1e-5):
2     """
3     K-means算法实现，算得分类结果和中心
4     """
5     center = np.zeros((k, X.shape[1]-1))
6     for i in range(k):
```

```

7         center[i,:] = X[np.random.randint(0, high=X.shape[0]),:-1]
8     while True:
9         distance = np.zeros(k)
10        # 根据中心重新给每个点贴分类标签
11        for i in range(X.shape[0]):
12            for j in range(k):
13                distance[j] = np.linalg.norm(X[i,:-1] - center[j, :])
14            X[i, -1] = np.argmin(distance)
15        # 根据每个点新的标签计算它的中心
16        new_center = np.zeros((k, X.shape[1]-1))
17        count = np.zeros(k)
18        for i in range(X.shape[0]):
19            new_center[int(X[i, -1]), :] += X[i, :-1] # 对每个类的所有点坐标求
和
20            count[int(X[i, -1])] += 1
21        for i in range(k):
22            new_center[i, :] = new_center[i, :] / count[i] # 对每个类的所有点
坐标求平均值
23        if np.linalg.norm(new_center - center) < epsilon: #用差值的二范数表示
精度
24            break
25        else:
26            center = new_center
27    return X, center

```

3.2 GMM

多元高斯分布生成的 d 维随机变量 x 的密度函数为：

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

其中 μ 为 d 维的均值向量， Σ 为 $d \times d$ 的协方差矩阵。

给定训练样本集 $X = \{x_1, x_2, \dots, x_n\}$ ，它是一个 $n \times d$ 的矩阵， n 为样本数量， d 为单个样本的维度数量。对于一个样本 x_i ，我们可以认为它是由多个对应维度的多元高斯分布所生成，所以高斯分布的线性叠加来表征数据，假设数据由 k 个高斯分布混合生成：

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

其中 μ_j 和 Σ_j 是第 j 个高斯分布的均值和协方差矩阵， π_j 为相应的混合系数，满足 $\sum_{j=1}^k \pi_j = 1$

。因此，我们也可以认为该数据的生成相当于从 k 个高斯分布中挑选出一个所生成，我们设 k 维二值变量 z ，这个变量采用了“1-of- k ”表示方法，其中一个特定的元素 z_j 等于1，其余所有的元素等于0。于是 z_j 的值满足 $z_j \in \{0, 1\}$ 且 $\sum_j z_j = 1$ ，则 π_j 加权平均概率值可以表征 z 的分布，也就是说 z 的先验分布为：

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

而在看到 x_i 的情况下 z 的后验概率为：

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

当后验概率已知时，混合高斯模型将训练样本划分成了 k 个簇 $C = C_1, C_2, \dots, C_k$ ，对于每一个样本 x_i ，其类别为 j ，满足 $j = \arg \max_j \gamma(z_j)$ ，即选择后验概率最大的类别作为标签类别。因此，当我们观测到样本集 X 时可以采用极大似然估计来求解样本的类别分布：

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

使上式最大化，对 μ_j 求导令导数为0：

$$\frac{\partial \ln p(X|\pi, \mu, \Sigma)}{\partial \mu_j} = \sum_{i=1}^n \frac{\pi_j p(x_i|\mu_j, \Sigma_j)}{\sum_{l=1}^k \pi_l p(x_i|\mu_l, \Sigma_l)} \Sigma_j^{-1} (x_i - \mu_j) = 0$$

令

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

则上式解得

$$n_j = \sum_i \gamma(z_{ij})$$

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

同理，对 Σ_j 求导令导数为0：

$$\frac{\partial \ln p(X|\pi, \mu, \Sigma)}{\partial \Sigma_j} = \sum_{i=1}^n \frac{\pi_j p(x_i|\mu_j, \Sigma_j)}{\sum_{l=1}^k \pi_l p(x_i|\mu_l, \Sigma_l)} (\Sigma_j^{-1} - \Sigma_j^{-1} (x_i - \mu_j)(x_i - \mu_j)^T \Sigma_j^{-1}) = 0$$

解得

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

对于混合系数 π_j ，还需要满足约束条件 $\sum_{j=1}^k \pi_j = 1$ 。构造拉格朗日多项式：

$$\ln p(X|\pi, \mu, \Sigma) + \lambda \left(\sum_{j=1}^k \pi_j - 1 \right)$$

对 π_j 求导，令导数为0：

$$\frac{\partial \ln p(X|\pi, \mu, \Sigma) + \lambda \left(\sum_{j=1}^k \pi_j - 1 \right)}{\partial \pi_j} = \sum_{i=1}^n \frac{p(x_i|\mu_j, \Sigma_j)}{\sum_{l=1}^k \pi_l p(x_i|\mu_l, \Sigma_l)} + \lambda = 0$$

同乘 π_j 并将 $j \in \{1, 2, \dots, k\}$ 代入相加得：

$$\sum_{j=1}^k \pi_j \frac{p(x_i | \mu_j, \Sigma_j)}{\sum_{l=1}^k \pi_l p(x_i | \mu_l, \Sigma_l)} + \lambda \sum_{j=1}^k \pi_j = 0$$

将约束条件代入：

$$\sum_{i=1}^n \left(\frac{\sum_{j=1}^k \pi_j p(x_i | \mu_j, \Sigma_j)}{\sum_{l=1}^k \pi_l p(x_i | \mu_l, \Sigma_l)} \right) + \lambda \sum_{j=1}^k \pi_j = n + \lambda = 0$$

即 $\lambda = -n$ ，代入 $\sum_{i=1}^n \frac{p(x_i | \mu_j, \Sigma_j)}{\sum_{l=1}^k \pi_l p(x_i | \mu_l, \Sigma_l)} + \lambda = 0$ 中，得

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

GMM算法过程如下：

1. 随机初始化参数 $\pi_i, \mu_i, \Sigma_i, i \in \{1, 2, \dots, k\}$
2. E步：根据式 $\gamma(z_j) = \frac{\pi_j p(x_i | \mu_j, \Sigma_j)}{\sum_{l=1}^k \pi_l p(x_i | \mu_l, \Sigma_l)}$ 计算每个样本由各个混合高斯成分生成的后验概率
3. M步：用下式更新参数 $\pi_i, \mu_i, \Sigma_i, i \in \{1, 2, \dots, k\}$

$$\begin{aligned} \pi_j &= \frac{n_j}{n} \\ \mu_j &= \frac{1}{n_j} \sum_{i=1}^n \gamma(z_{ij}) x_i \\ \Sigma_j &= \frac{\sum_{i=1}^n \gamma(z_{ij}) (x_i - \mu_j)(x_i - \mu_j)^T}{n_j} \end{aligned}$$

4. 如果参数值不再发生变化，根据 $j = \arg \max_j \gamma(z_j)$ 计算标签 j ，否则，返回第2步

E步实现：

```
1 def e_step(x, mu_list, sigma_list, pi_list):
2     """
3     e步，求每个样本由各个混合高斯成分生成的后验概率
4     """
5     k = mu_list.shape[0]
6     gamma_z = np.zeros((x.shape[0], k))
7     for i in range(x.shape[0]):
8         pi_times_pdf_sum = 0
9         pi_times_pdf = np.zeros(k)
10        for j in range(k):
11            pi_times_pdf[j] = pi_list[j] * multivariate_normal.pdf(x[i],
12                                mean=mu_list[j], cov=sigma_list[j])
13            pi_times_pdf_sum += pi_times_pdf[j]
14        for j in range(k):
15            gamma_z[i, j] = pi_times_pdf[j] / pi_times_pdf_sum
```

m步实现:

```
1 def m_step(x, mu_list, gamma_z):
2     """
3     m步, 根据公式更新参数
4     """
5     k = mu_list.shape[0]
6     n = x.shape[0]
7     dim = x.shape[1]
8     mu_list_new = np.zeros(mu_list.shape)
9     sigma_list_new = np.zeros((k, dim, dim))
10    pi_list_new = np.zeros(k)
11    for j in range(k):
12        n_j = np.sum(gamma_z[:, j])
13        pi_list_new[j] = n_j / n # 计算新的pi
14
15        gamma = gamma_z[:, j]
16        gamma = gamma.reshape(n, 1)
17        mu_list_new[j, :] = (gamma.T @ x) / n_j # 计算新的mu
18        sigma_list_new[j] = ((x - mu_list[j]).T @ np.multiply((x -
19        mu_list[j]), gamma)) / n_j # 计算新的sigma
20    return mu_list_new, sigma_list_new, pi_list_new
```

GMM主算法:

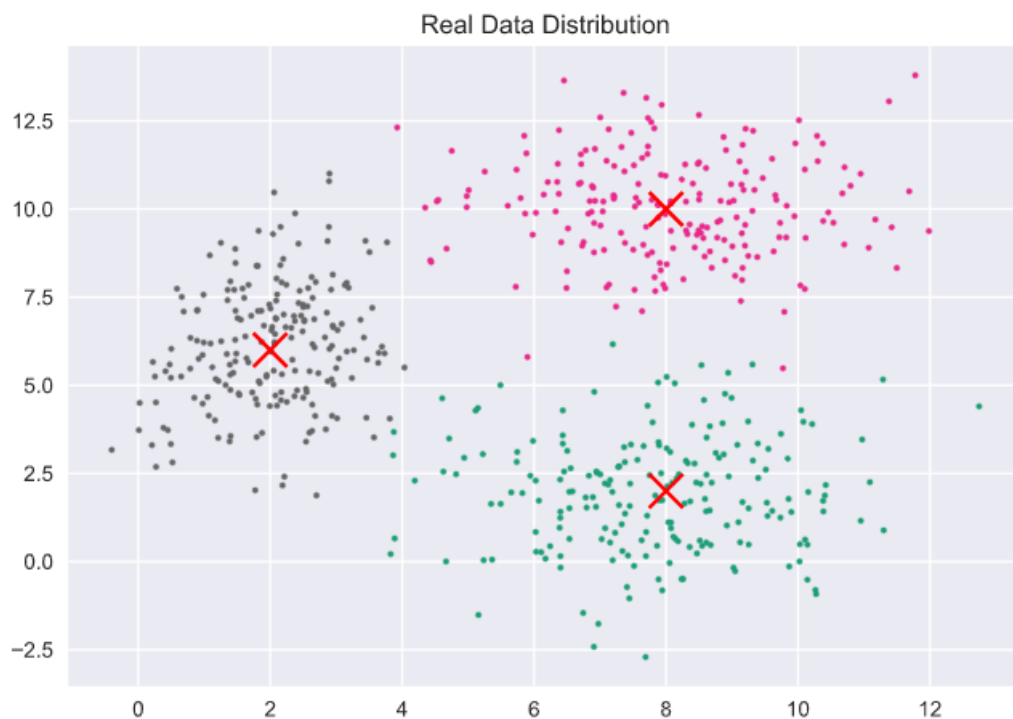
```
1 while True:
2     gamma_z = e_step(x, mu_list, sigma_list, pi_list)
3     mu_list, sigma_list, pi_list = m_step(x, mu_list, gamma_z)
4     new_log_l = log_likelihood(x, mu_list, sigma_list, pi_list)
5     if (old_log_l - new_log_l) < epsilon:
6         break
```

4 实验结果与分析

4.1 k=3, 各分布距离较远

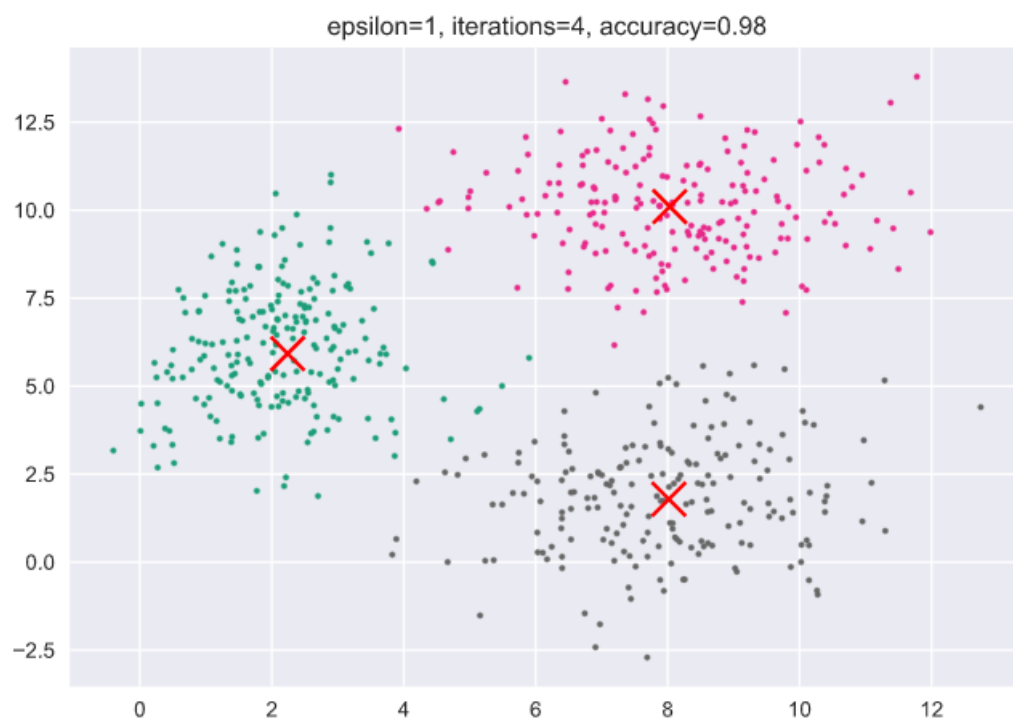
生成k=3的2维数据测试K-Means和GMM的效果，各高斯分布的均值和协方差矩阵均不同。

各高斯分布的均值: $\begin{bmatrix} 2 & 6 \end{bmatrix}$ $\begin{bmatrix} 8 & 10 \end{bmatrix}$ $\begin{bmatrix} 8 & 2 \end{bmatrix}$

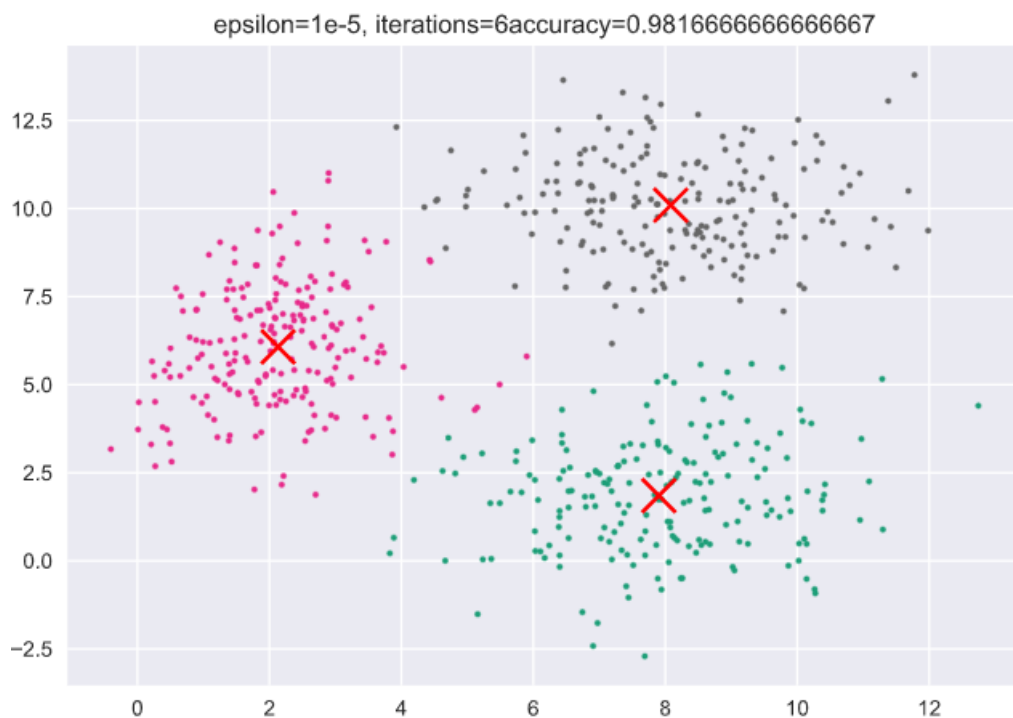


4.1.1 K-means

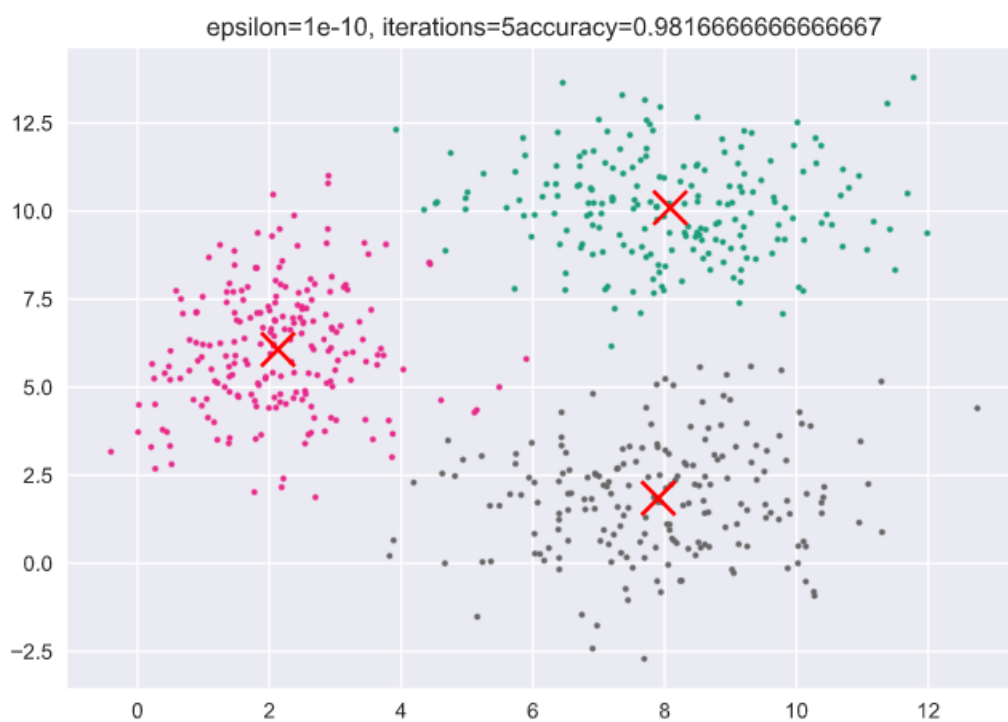
下图的各中心坐标: $\begin{bmatrix} 8.01793576 & 1.79108972 \end{bmatrix}$ $\begin{bmatrix} 8.03150573 & 10.1021806 \end{bmatrix}$ $\begin{bmatrix} 2.24065269 & 5.92428314 \end{bmatrix}$



下图的各中心坐标: $\begin{bmatrix} 8.0836514 & 10.09409582 \end{bmatrix}$ $\begin{bmatrix} 2.13241573 & 6.07466331 \end{bmatrix}$ $\begin{bmatrix} 7.90305055 & 1.85080409 \end{bmatrix}$



下图的各中心坐标: [[7.90305055 1.85080409] [2.13241573 6.07466331] [8.0836514 10.09409582]]



从图中可以看出，随着精度的增大，准确率都在0.98，变化不大，但精度从1变为1e-5时，求得的聚类中心准确度增大了。

4.1.2 GMM

Iterations		log likelihood
0	0	-18226.956449
1	10	-3689.970247
2	20	-3692.195322
3	30	-3692.195794
4	40	-3692.195794

epsilon=1, iterations=42, accuracy=0.9933333333333333

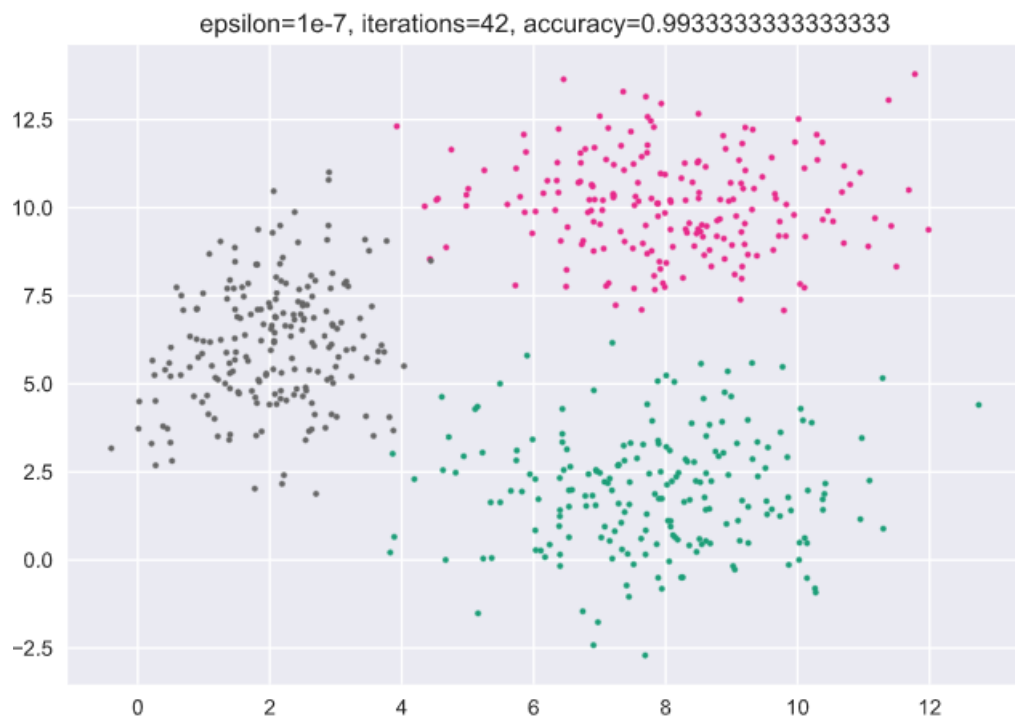


	Iterations	log likelihood
0	0	-32211.187468
1	10	-3687.083203
2	20	-3692.195100
3	30	-3692.195794

epsilon=1e-5, iterations=34, accuracy=0.9933333333333333



	Iterations	log likelihood
0	0	-18226.956449
1	10	-3689.970247
2	20	-3692.195322
3	30	-3692.195794
4	40	-3692.195794

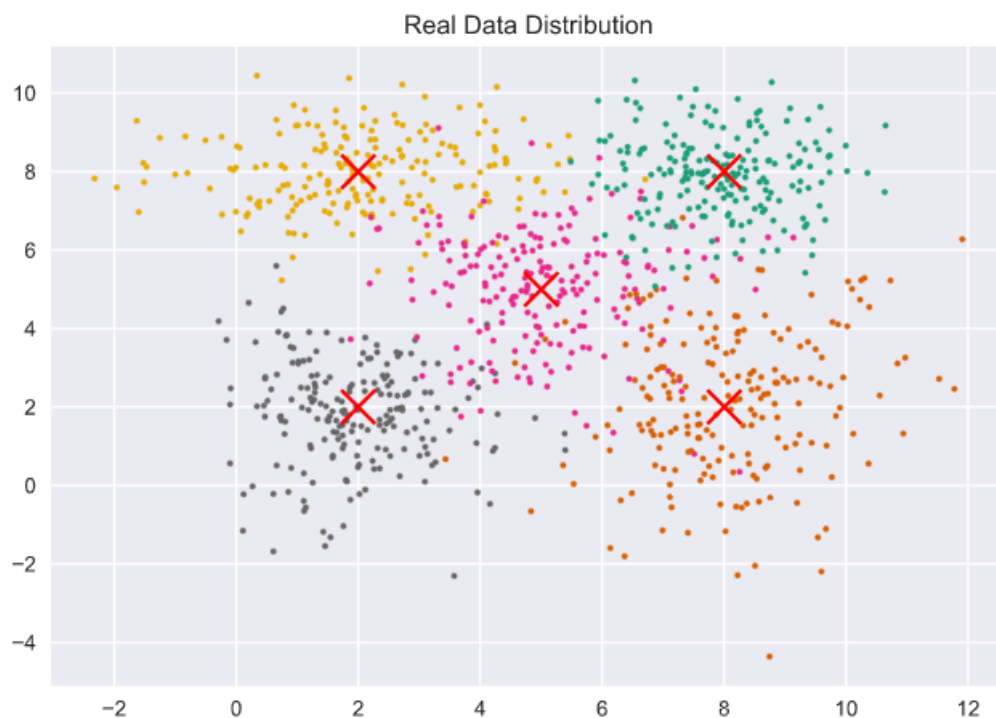


可以看出，在本条件下，GMM得到的结果要比K-means得到的结果好一些。分类结果都在0.993以上，随精度的变化不大，所以推断它已经收敛到了一定程度，甚至各参数不再发生变化，再看各次运行时的最大似然对数，发现最大似然对数确实在增大，而大约从第30次迭代开始，就不再发生变化了，这也就解释了为什么在这些精度下不随精度的增大而变化。

4.2 k=5, 各分布距离较近

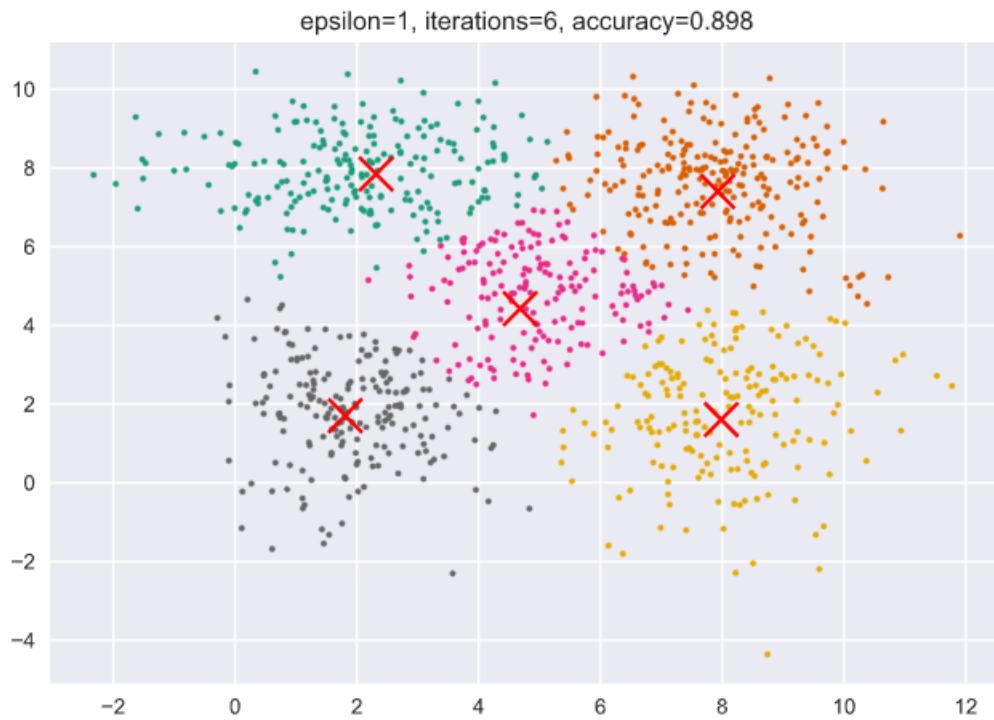
生成k=5的2维数据测试K-Means和GMM的效果，各高斯分布的均值和协方差矩阵均不同。

各高斯分布的均值：[[2 2] [2 8] [5 5] [8 2] [8 8]]

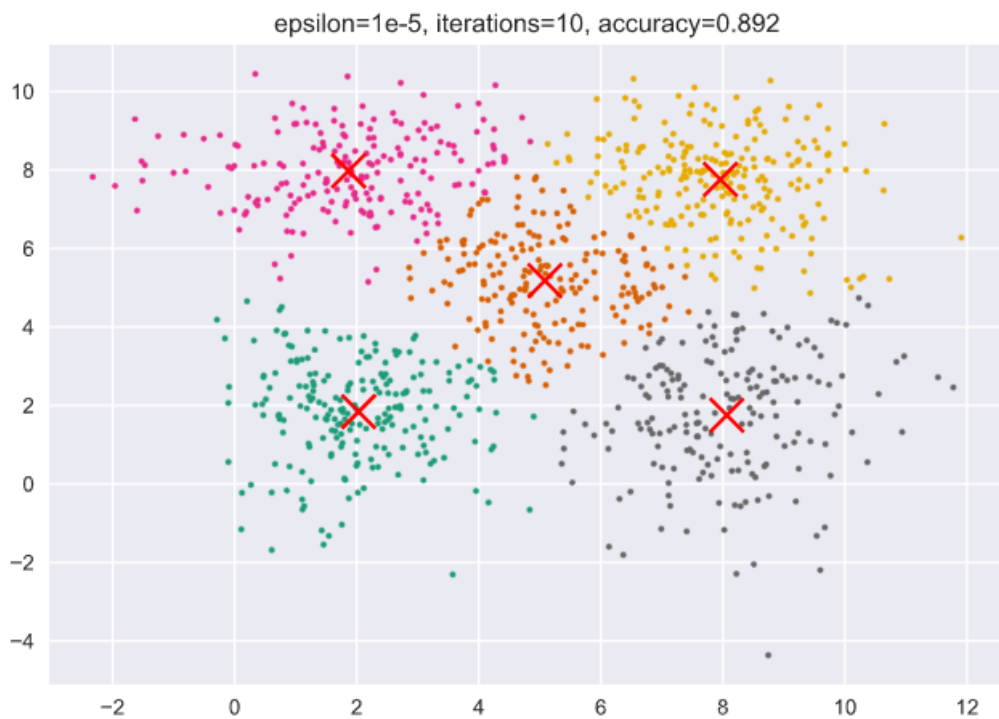


4.2.1 K-means

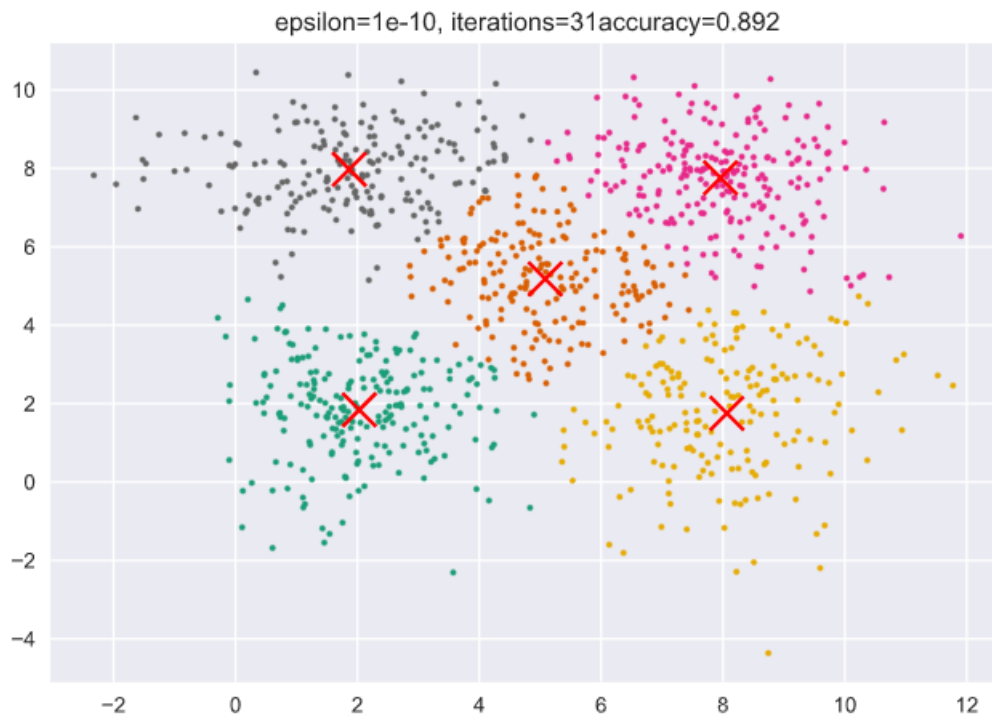
下图的各中心坐标: $[[1.81126365 \ 1.70455608] \ [7.98534324 \ 1.60414057] \ [4.68148749 \ 4.42232503] \ [7.92856289 \ 7.40125028] \ [2.31526297 \ 7.84828289]]$



下图的各中心坐标: $[[8.06352571 \ 1.74963954] \ [7.95994276 \ 7.75805996] \ [1.86714372 \ 7.97501538] \ [5.08440337 \ 5.18070716] \ [2.03251864 \ 1.8425656 \]]$

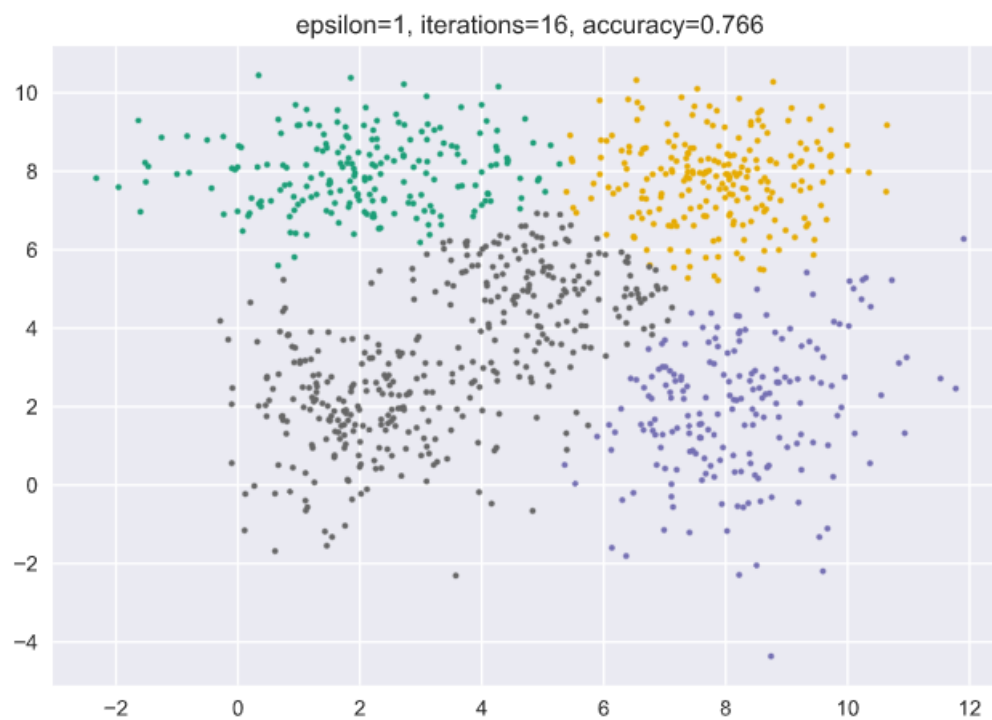


下图的各中心坐标: $[[1.86714372 \ 7.97501538] \ [8.06352571 \ 1.74963954] \ [7.95994276 \ 7.75805996] \ [5.08440337 \ 5.18070716] \ [2.03251864 \ 1.8425656 \]]$



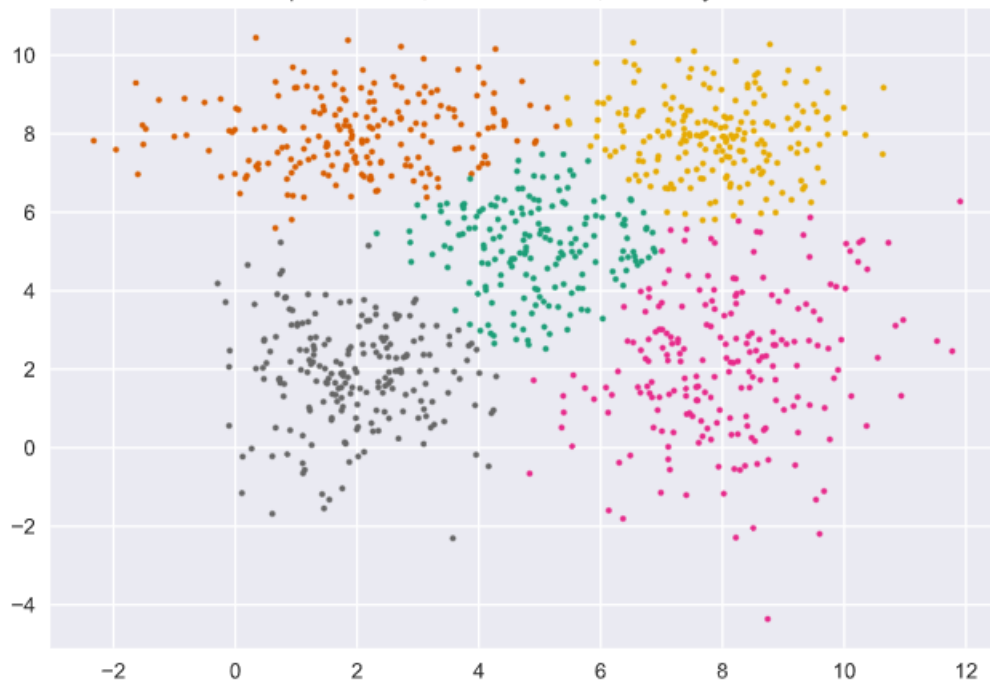
4.2.2 GMM

	Iterations	log likelihood
0	0	-7259.752107
1	10	-4765.314022

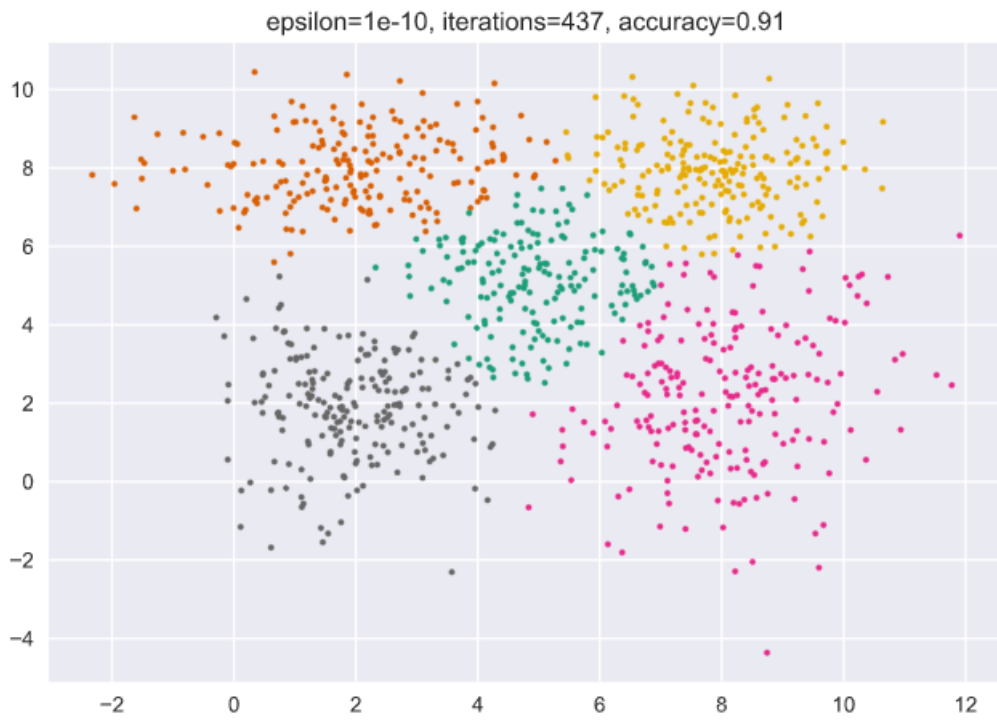


	Iterations	log likelihood
0	0	-21654.908621
1	10	-4726.982015
2	20	-4592.452950
3	30	-4476.627849
4	40	-4279.406671
5	50	-4180.283569
6	60	-4152.221810
7	70	-4142.624783
8	80	-4138.771316
9	90	-4137.080816
10	100	-4136.300408
11	110	-4135.927606
12	120	-4135.744660
13	130	-4135.652762
14	140	-4135.605617
15	150	-4135.580966
16	160	-4135.567856
17	170	-4135.560781
18	180	-4135.556914
19	190	-4135.554778
20	200	-4135.553588
21	210	-4135.552921
22	220	-4135.552544
23	230	-4135.552331

epsilon=1e-5, iterations=238, accuracy=0.91



	Iterations	log likelihood
0	0	-21654.908621
1	10	-4726.982015
2	20	-4592.452950
3	30	-4476.627849
4	40	-4279.406671
5	50	-4180.283569
6	60	-4152.221810
7	70	-4142.624783
8	80	-4138.771316
9	90	-4137.080816
10	100	-4136.300408
11	110	-4135.927606
12	120	-4135.744660
13	130	-4135.652762
14	140	-4135.605617
15	150	-4135.580966
16	160	-4135.567856
17	170	-4135.560781
18	180	-4135.556914
19	190	-4135.554778
20	200	-4135.553588
21	210	-4135.552921
22	220	-4135.552544
23	230	-4135.552331
24	240	-4135.552210
25	250	-4135.552140
26	260	-4135.552101
27	270	-4135.552078
28	280	-4135.552065
29	290	-4135.552058
30	300	-4135.552054
31	310	-4135.552051
32	320	-4135.552050
33	330	-4135.552049
34	340	-4135.552049
35	350	-4135.552048
36	360	-4135.552048
37	370	-4135.552048
38	380	-4135.552048
39	390	-4135.552048
40	400	-4135.552048
41	410	-4135.552048
42	420	-4135.552048
43	430	-4135.552048



可以看出在精度较低时，GMM的效果并不好，但提高精度后，准确率和大幅度提升，最后比K-means的分类效果要好一点。

4.3 UCI数据集

```
K-means
[[5.00784314 3.4          1.49411765 0.26078431]
 [6.84545455 3.02954545 5.53863636 1.95227273]
 [5.81636364 2.75272727 4.43454545 1.46545455]]
epsilon=1, iterations=2, accuracy=0.8733333333333333

[[5.88360656 2.74098361 4.38852459 1.43442623]
 [5.006       3.418       1.464       0.244       ]
 [6.85384615 3.07692308 5.71538462 2.05384615]]
epsilon=1e-5, iterations=5, accuracy=0.8866666666666667

[[5.88360656 2.74098361 4.38852459 1.43442623]
 [5.006       3.418       1.464       0.244       ]
 [6.85384615 3.07692308 5.71538462 2.05384615]]
epsilon=1e-10, iterations=10, accuracy=0.8866666666666667
```



```
GMM
  Iterations  log likelihood
0           0    -340.267489
1          10    180.479148
epsilon=1, iterations=11, accuracy=0.76

  Iterations  log likelihood
0           0    -340.267489
1          10    180.479148
2          20    167.034569
3          30    167.032893
4          40    167.032705
5          50    167.032697
6          60    167.032696
7          70    167.032696
8          80    167.032696
9          90    167.032696
epsilon=1e-5, iterations=95, accuracy=0.76

  Iterations  log likelihood
0           0    -340.267489
1          10    180.479148
2          20    167.034569
3          30    167.032893
4          40    167.032705
5          50    167.032697
6          60    167.032696
7          70    167.032696
8          80    167.032696
9          90    167.032696
epsilon=1e-10, iterations=95, accuracy=0.76
```

5 结论

K-Means实际上假设数据呈球状分布，假设使用的欧式距离来衡量样本与各个簇中心的相似度(假设数据的各个维度对于相似度计算的作用是相同的)，它的簇中心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簇中心值容易使之陷入局部最优解；与之相比GMM使用更加一般的数据表示即高斯分布，GMM使用EM算法进行迭代优化，因为其涉及到隐变量的问题，没有之前的完全数据，而是在不完全数据上进行。

K-Means其实就是一种特殊的高斯混合模型，我们假设每种类在样本中出现的概率相等，都为 $\frac{1}{k}$ ，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，这样我们可以直接用欧氏距离作为K-Means的协方差去衡量相似性；K-Means对响应度也做了简化，每个样本只属于一个类，即每个样本属于某个类，则响应度为1，对于不属于的类，响应度直接设为0，算是对GMM的一种简化。而在高斯混合模型中，每个类的数据出现在样本中的概率为 π_k ，用协方差矩阵替代K-Means中的欧式距离去度量点和点之间的相似度，响应度也由离散的0，1变成了需要通过全概率公式计算的值。但是由

于GMM并未增加如K-Means那么多假设条件，分类最终效果比K-Means好，但是GMM-EM算法过于细化，容易被噪声影响，所以适合作为优化算法，即适合对K-Means的分类结果进行进一步优化。