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

学号：1170300418
姓名：于新蕊

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

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）。

- 用k-means聚类，测试效果；
- 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）。

实验环境

windows64，pycharm，python3.0，anaconda

三、设计思想（本程序中的用到的主要算法及数据结构）

k-means算法

算法思想

k-means算法是解决聚类问题的方法之一，它算法实现简单，效果也不错。

聚类属于无监督学习，也就是在训练集中不给出标签信息y，只有特征x。

考虑到我们最后的训练最后大概率会聚成一“团”，所以我们考虑这些这一“团”中，每个点到质心的距离不会很远。

所以考虑这样的算法：

对于训练集$(x_1, x_2, ..., x_m)$，其中每一个都是一个d-维向量，k-means算法将样本据类成k个cluster，这个k是要被初始指定的。然后初始指定k个质心$\mu_1, \mu_2, ..., \mu_k \in \mathbb{R}^n$。

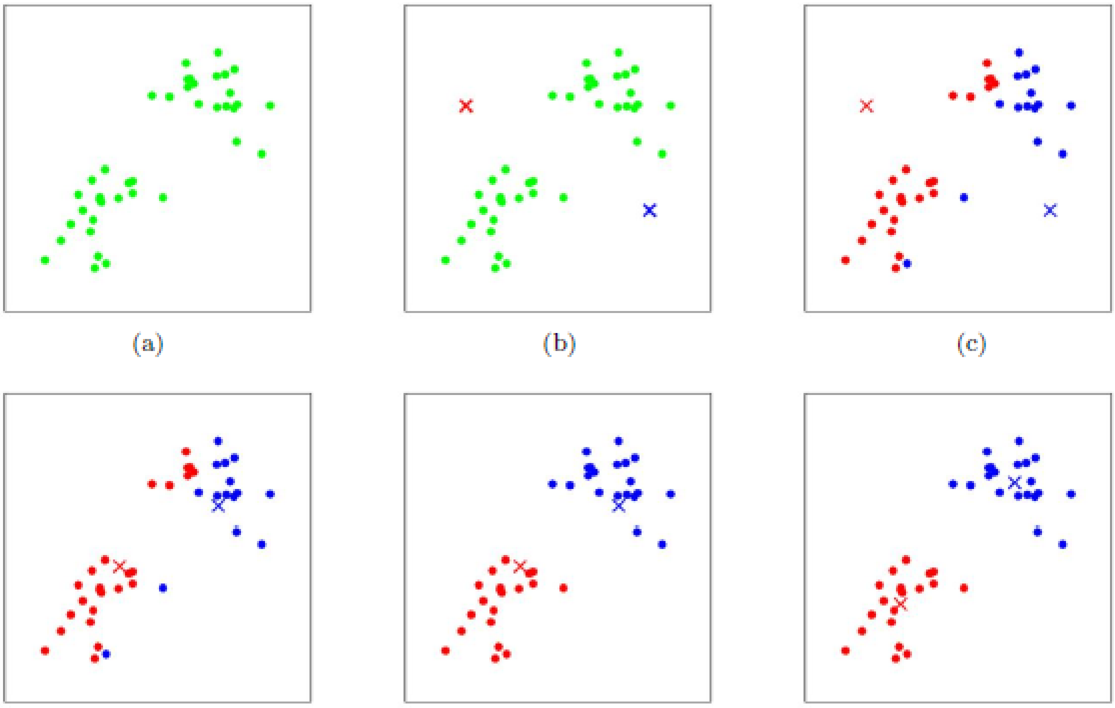
接着是迭代过程，迭代分为两步：

- 分配：将每个样本x_i分配到聚类$j(1 \le j \le k)$中，使得x_i到这类的质心μ_i距离最小。

  $c_i := \arg\min_j \|x_i - \mu_j\|^2$，其中$c_i$就是$x_i$被分配的聚类标签。
- 更新：对于上一步得到的每一个聚类，根据聚类中的每一个样本x_i计算出新的质心，即

  $\mu_i = \frac{1}{|S_i|}\sum_{x_j \in S_i} x_j$

下图展示了对n个样本点进行K-means聚类的效果，这里k取2。



质心点初始化

1. 随即划分

   把质心点都放到靠近数据集中心的地方。

2. Forgy方法

   随机地从数据集中选择*k*个观测作为初始的质心点。

高斯混合模型

混合高斯模型概念

1. 单高斯模型(GSM)

   多维变量$X=(x_1,x_2,\dots x_n)$的联合概率密度函数为：

   $$\phi(X|\theta)=\frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}}\exp\left[-\frac{1}{2}(X-u)^T\Sigma^{-1}(X-u)\right]$$

   其中$X=(x_1,x_2\dots x_n)$，

   $u=\begin{pmatrix}u_1\\u_2\\\dots\\u_n\end{pmatrix}$是各维变量的均值

   Σ是协方差矩阵

   d是变量维度

   $\theta=(u,\Sigma)$

2. 混合高斯模型(GMM)

k个GSM混合成一个GMM，每个GSM称为GMM的一个component，也就是分为k个类。

$$P(X|\theta)=\sum_{i=1}^{k}\alpha_i\phi\left(X|\theta_i\right)$$

其中，α_k是样本集合中k类被选中的概率：$\alpha_k=P(z=k|\theta)$，$z=k$指的是样本属于k类。$\sum\limits_{i=1}^{k}\alpha_i=1$。

EM算法

1. Jensen不等式

   如果f是上凸函数，X是随机变量，那么$f(E[X])\geq E[f(X)]$

   特别地，如果f是严格上凸函数，那么$E[f(X)]=f(E[X])$当且仅当$p(X=E[X])=1$，也就是说X是常量。

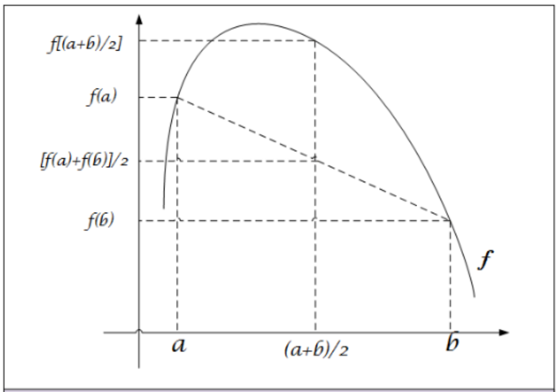


图2.1 上凸函数

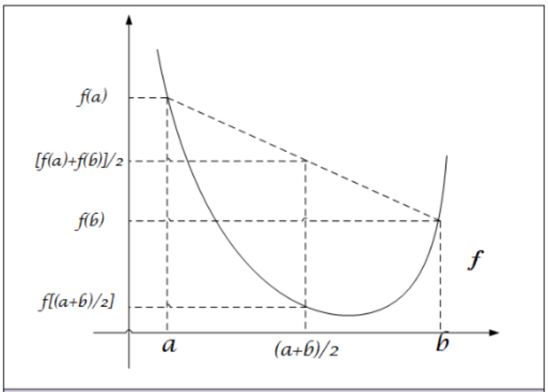


图2.2 下凸函数

2. EM模型

   考虑一个参数估计问题，现有$\{x_1,x_2,...,x_m\}$共m个训练样本，需有多个参数θ去拟合数据。

   这个log似然函数是$l(\theta)=\sum_{j=1}^{n}\log P\left(y_j|\theta\right)$。且由于某幢关系，导致求导困难，无法使用梯度下降方法求解θ，这是我们考虑EM算法。

3. EM算法

   EM算法，我们引入一个因变量z_i：

$$\begin{aligned}l(\theta)&=\sum_{j=1}^{n}\log\sum_{i}P\left(y_j,z_i|\theta\right)\\&=\sum_{j=1}^{n}\log\sum_{i}Q_j\left(z_i\right)\frac{P\left(y_j,z_i|\theta\right)}{Q_j\left(z_i\right)}\\&\geq\sum_{j=1}^{n}\sum_{i}Q_j\left(z_i\right)\log\frac{P\left(y_j,z_i|\theta\right)}{Q_j\left(z_i\right)}\end{aligned}$$

   假设$Q(z)$是z的某种分布，即$\sum_i Q_j\left(z_i\right)=1$。

   那么$\sum_i Q_j\left(z_i\right)\frac{P\left(y_j,z_i|\theta\right)}{Q_j\left(z_i\right)}$是$\frac{P\left(y_j,z_i|\theta\right)}{Q_j\left(z_i\right)}$的期望。

   第二步到第三步是Jensen不等式($f(x)=\log x$)。

   而第二步到第三步如果取等号，条件是$\frac{P\left(y_j,z_i|\theta\right)}{Q_j\left(z_i\right)}=c$。

又$\sum_i Q_j\left(z_i\right)=1$，

所以$\sum_i P\left(y_j, z_i \mid \theta\right)=c$，

故，

$$
\begin{aligned}
Q_j\left(z_i\right) & =P\left(y_j, z_i \mid \theta\right) / \sum_i P\left(y_j, z_i \mid \theta\right) \\
& =P\left(y_j, z_i \mid \theta\right) / P\left(y_j \mid \theta\right) \\
& =P\left(z_i \mid y_j, \theta\right)
\end{aligned}
$$

这样$l(\theta)=\sum_{j=1}^n \sum_i Q_j\left(z_i\right) \log \frac{P\left(y_j, z_i \mid \theta\right)}{Q_j\left(z_i\right)}$

我们就可以用梯度下降等方法求出θ，那么又可以计算新$Q_j\left(z_i\right)$的值，依次迭代，EM算法就实现了。

所以EM算法就是：

选取初始值θ_0，初始化θ

重复

- E步

  $$
  \begin{aligned}
  Q_j\left(z_i\right) & =P\left(y_j, z_i \mid \theta\right) / \sum_i P\left(y_j, z_i \mid \theta\right) \\
  & =P\left(y_j, z_i \mid \theta\right) / P\left(y_j \mid \theta\right) \\
  & =P\left(z_i \mid y_j, \theta\right)
  \end{aligned}
  $$

- M步

  $\theta=\arg \max _\theta \sum_{j=1}^n \sum_i Q_j\left(z_i\right) \log \frac{P\left(y_j, z_i \mid \theta\right)}{Q_j\left(z_i\right)}$

EM算法解决混合高斯模型问题

E步

引入隐变量$\gamma_{jk}=Q_j(z_k)$

$$
\begin{aligned}
l(\theta) & =\sum_{j=1}^n \log \sum_k^K \alpha_k \phi(x_j \mid \theta_k) \\
& =\sum_{j=1}^n \log \sum_{k=1}^K \gamma_{jk} \frac{\alpha_k \phi(x_j \mid \theta_k)}{\gamma_{jk}} \\
& \geq \sum_{j=1}^n \sum_{k=1}^K \gamma_{jk} \log \frac{\alpha_k \phi(x_j \mid \theta_k)}{\gamma_{jk}}
\end{aligned}
$$

这样$\gamma_{jk}=\frac{\alpha_k \phi(x_j \mid \theta_k)}{\sum_{k=1}^K \phi(x_j \mid \theta_k)}$

M步

设$H(\theta)=\sum_{j=1}^n \sum_{k=1}^K \gamma_{jk} \log \frac{\alpha_k \phi(x_j \mid \theta_k)}{\gamma_{jk}}$

令$L(\theta)=H(\theta)+\lambda(\sum_{k=1}^K \alpha_k-1)$

$\frac{\partial L(\theta)}{\partial \alpha_k}=0$

$\frac{\partial L(\theta)}{\partial \lambda} = 0$

得出$\alpha_k = \frac{\sum_{j=1}^{N} \gamma_{jk}}{N}$

$\frac{\partial H(\theta)}{\partial \mu_k} = \frac{\partial H(\theta)}{\partial \sigma_k} = 0$

得出$\mu_k = \frac{\sum_{j=1}^{N} \gamma_{jk} x_k}{\sum_{j=1}^{N} \gamma_{jk}}$

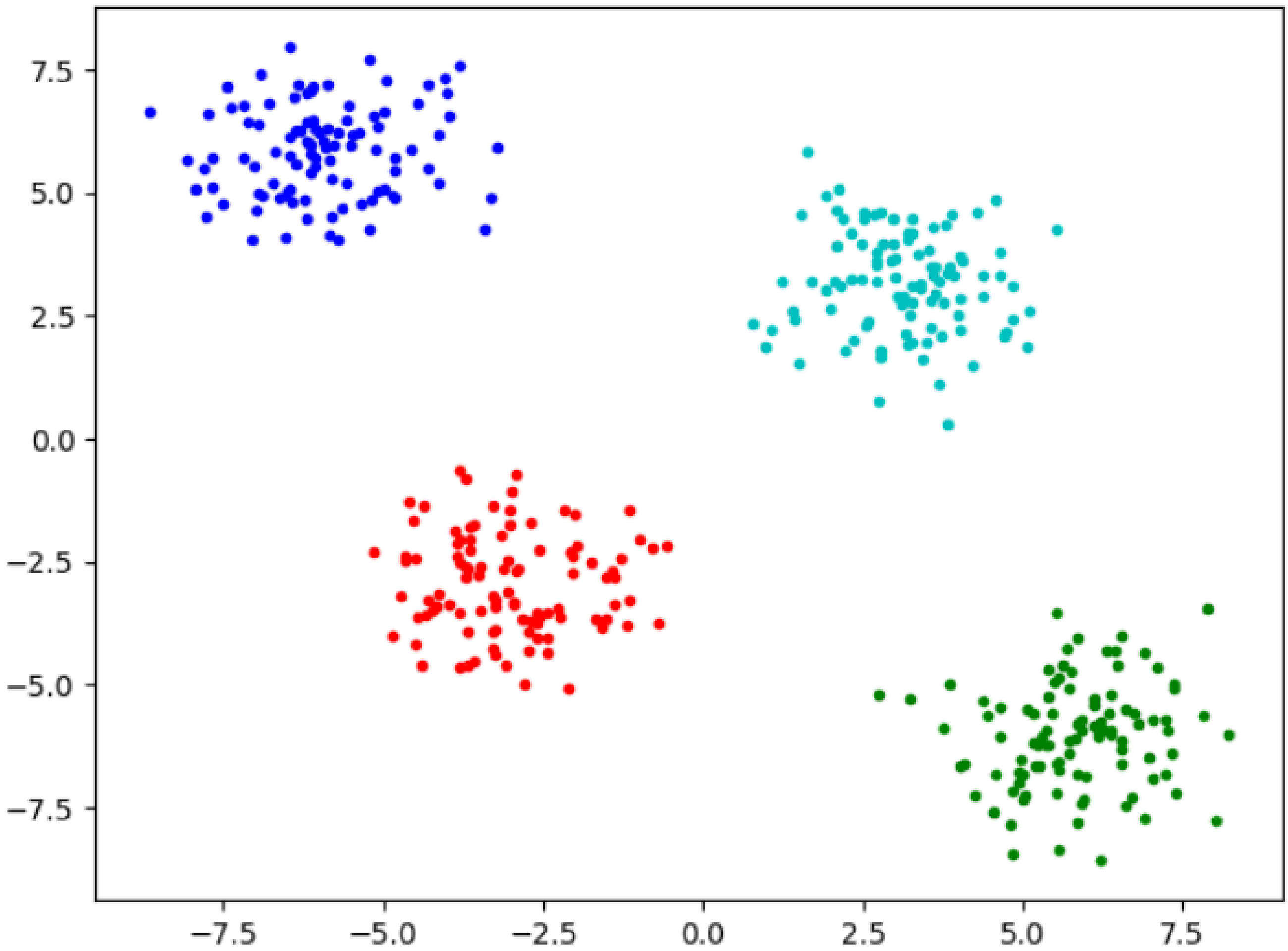
$\sigma_k^2 = \frac{\sum_{j=1} N\gamma(x_k - \mu_k)^2}{\sum_{j=1}^{N} \gamma_{jk}}$

k-means和GMM的对比

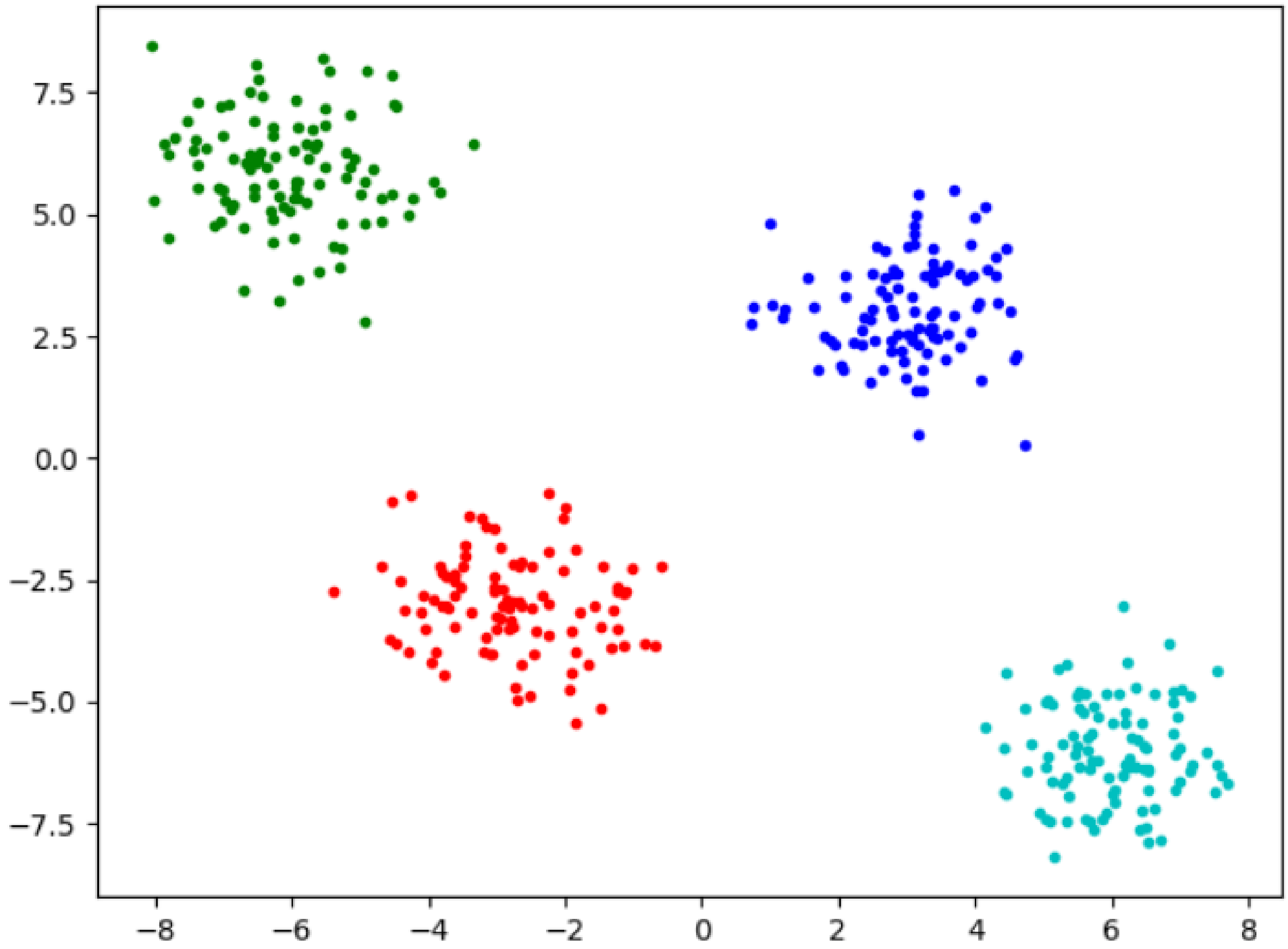
- 相同点
    1. 需要指定k
    2. 需要指定初始值
    3. EM思想的运用
- 不同点
    1. 优化的目标函数，即M步不同，k-means是最短距离函数，GMM是似然函数
    2. E步不同，k-means的指标是点到质心的距离，GMM的指标是每个样本来自某个概率分布的概率

四、实验结果与分析

k-means



GMM



UCI数据集

```
已收敛，迭代次数270
real data:
num of 0:  50
num of 1:  50
num of 2:  50
gmm result:
num of 0:  50
num of 1:  65
num of 2:  35
```

分析

k-means算法易于理解且易于实现，而GMM的计算复杂，推导繁琐。

GMM在真实数据集上表现时而好时而不好，原因是GMM找到的可能是局部最优点，而不是全局最优点。（k-means也存在这个问题）

五、结论

k-means和GMM都是无监督学习的方法，当今工业上流行的做法是用k-means分号簇后，用GMM迭代求解，效果可能会更好一些。

六、参考文献

[吴恩达机器学习(Coursera)]

七、附录：源代码（带注释）

k-means

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


color = ['r', 'g', 'b', 'c']


def generate_x(mean, var, size):
    x = np.array([])
    for i in range(len(mean)):
        tmp = np.random.normal(loc=mean[i], scale=var, size=size)
        x = np.append(x, tmp)
    x = x.reshape(-1, size).T
    return x


def generate_data():
    """
    生成数据
    :return: None
    """
    mean1 = [-6, 6]
    mean2 = [6, -6]
    mean3 = [3, 3]
    mean4 = [-3, -3]
    var = 1
    size1 = size2 = size3 = size4 = 100
    x1 = generate_x(mean1, var, size1)
    x2 = generate_x(mean2, var, size2)
    x3 = generate_x(mean3, var, size3)
    x4 = generate_x(mean4, var, size4)
    x1 = np.row_stack((x1, x2))
    x1 = np.row_stack((x1, x3))
    x1 = np.row_stack((x1, x4))
    np.random.shuffle(x1)
    return x1
```

```python
def random_center(x):
    """
    随机中心点
    :param x: 样本
    :return: 中心点
    """
    index = random.sample(list(range(len(x))), 4)
    center = np.array([x[i, :] for i in index])
    return center.reshape(4, 2)


def get_distance(x, y):
    """
    计算两个向量x y的距离
    :param x: 向量x
    :param y: 向量y
    :return: 距离
    """
    return np.sqrt(np.sum(np.square(x - y)))


def calculate_distance(x, center):
    x_size = np.shape(x)[0]
    k = np.shape(center)[0]
    dis = np.zeros((x_size, k))
    for i in range(x_size):
        for j in range(k):
            dis[i, j] = get_distance(x[i, :], center[j, :])
    return dis


def get_means(data):
    tot_x = tot_y = 0
    for i in range(len(data)):
        tot_x += data[i][0]
        tot_y += data[i][1]
    return tot_x / len(data), tot_y / len(data)


def kmeans(x):
    """
    k means算法
    :param x: 样本
    :return: 中心点
    """
    center = random_center(x)
    size = np.shape(x)[0]
    k = np.shape(center)[0]
    for i in range(1000):
        dis = calculate_distance(x, center)
        y = np.argmin(dis, axis=1)
        clusters = [[] for j in range(k)]
```

```python
        for j in range(size):
            clusters[y[j]].append(x[j, :].tolist())
        new_center = np.array([])
        for j in range(k):
            new_center = np.append(new_center, get_means(clusters[j]))
        new_center = new_center.reshape(k, -1)
        if (new_center == center).all():
            print('已收敛，迭代次数{}'.format(i + 1))
            return y
        center = new_center
    return y


def plot(x, y):
    for i in range(y.size):
        plt.plot(x[i, 0], x[i, 1], '.', color=color[y[i]])
    plt.show()


if __name__ == '__main__':
    x = generate_data()
    y = kmeans(x)
    plot(x, y)
```

GMM

```python
import numpy as np
import random
import kmeans


def rand_params(dim, k):
    """
    随机初始参数
    :param dim:
    :param k:
    :return:
    """
    mu = np.random.rand(k, dim).reshape(k, dim)
    sigma = np.array([np.eye(dim)] * k).reshape(k, dim, dim)
    alpha = (np.ones(k) * (1.0 / k)).reshape(k, 1)
    return mu, sigma, alpha


def get_probability(x, mu, sigma, threshold=1e-8):
    """
    计算概率
    :param x:
    :param mu:
    :param sigma:
    :param threshold:
    :return:
    """
```

```python
    n = mu.shape[1]
    if np.linalg.det(sigma) == 0:
        for i in range(sigma.shape[0]):
            sigma[i, i] += threshold
    p = np.exp(-0.5 * np.dot(np.dot(x - mu, np.linalg.pinv(sigma)), (x - mu).T))
    p = p / (np.power(2 * np.pi, n / 2.0) * np.sqrt(np.linalg.det(sigma)))
    return p


def gmm(x, k):
    """
    GMM
    :param x:
    :param k:
    :return:
    """
    x_size = np.shape(x)[0]
    dim = np.shape(x)[1]
    mu, sigma, alpha = rand_params(dim, k)
    gamma = np.zeros((x_size, k))
    lld = np.array([])
    last_l_theta = 1e9
    t = 0
    for times in range(1000):
        prob = np.zeros((x_size, k))
        for i in range(x_size):
            for j in range(k):
                prob[i, j] = get_probability(x[i, :].reshape(1, -1), mu[j, :].reshape(1,
-1), sigma[j])
        # E步
        for i in range(k):
            gamma[:, i] = alpha[i, 0] * prob[:, i]
        # 计算似然值
        l_theta = np.sum(np.log(np.sum(gamma, axis=1)))
        if np.abs(last_l_theta - l_theta) < 1e-10:
            t += 1
        else:
            t = 0
        if t == 10:
            print('已收敛，迭代次数{}'.format(times + 1))
            break
        last_l_theta = l_theta
        print(l_theta)
        lld = np.append(lld, l_theta)
        for i in range(x_size):
            gamma[i, :] /= np.sum(gamma[i, :])
        # M步
        alpha = (np.sum(gamma, axis=0) / x_size).reshape(k, 1)
        for i in range(k):
            nk = np.sum(gamma[:, i])
            mu[i, :] = np.dot(gamma[:, i].reshape(1, x_size), x) / nk
            tmp = np.zeros((dim, dim))
            for j in range(x_size):
```

```python
                v = (x[j, :] - mu[i, :]).reshape(-1, 1)
                tmp += (gamma[j, i] * np.dot(v, v.T))
            sigma[i, :, :] = tmp / nk
    return gamma


def get_y(gamma):
    return np.argmax(gamma, axis=1)


def uci_test():
    with open('bezdekIris.data', 'r') as f:
        lines = f.readlines()
        x = []
        y = []
        for line in lines:
            line = line.strip().split(',')
            for word in line[:-1]:
                x.append(float(word))
            if line[-1] == 'Iris-setosa':
                y.append(0)
            elif line[-1] == 'Iris-versicolor':
                y.append(1)
            else:
                y.append(2)
        x = np.array(x).reshape(len(y), -1)
        y = np.array(y).reshape(-1, 1)
        print(x.shape)
        print(y.shape)
        gmm_y = get_y(gmm(x, 3))
        print('real data:')
        for i in range(3):
            print('num of {}: {}'.format(i, np.sum(y == i)))
        print('gmm result:')
        for i in range(3):
            print('num of {}: {}'.format(i, np.sum(gmm_y == i)))


if __name__ == '__main__':
    # x = kmeans.generate_data()
    # gamma = gmm(x, 4)
    # y = get_y(gamma)
    # kmeans.plot(x, y)
    uci_test()
```