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：1190301610
姓名： 王家琪

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

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。
环境要求：
Windows，pycharm，python3.7

三、设计思想（本程序中的用到的主要算法及数据结构）

3.1 实验原理

本实验应用了两种算法：k-means 算法，EM 算法，来解决聚类问题。K-means 算法原理比较简单，主要是通过计算距离，选择最小的距离来进行分类。EM 算法比较复杂，需要比较深厚的概率论和线性代数的功底。

EM 算法分为两部：

- E 步：求期望
- M 步：求最大似然

E 步是根据参数计算期望，M 步是通过代入期望，求最大似然更新参数，E、M 两步反复循环，只至到达停止条件。

3.2 k-means 算法

k-means 算法是最常见的基于欧式距离的聚类算法，其认为两个目标的距离越近，相似度越大。

给定训练样本$X=\{x_1,x_2,\ldots\ldots,x_n\}$，和划分的聚类的种类 k，给出一个簇划分$C=\{c_1,c_2,\ldots\ldots,c_k\}$，使得该划分的平方误差 E 最小，即：

$$E=\sum_{i=1}^{k}\sum_{x\in C_i}\|x-\mu_i\|_2^2 \text{ ,其中}\mu_i=\frac{1}{C_i}\sum_{x\in C_i}x_i$$

μ_i是簇C_i的均值向量，E 刻画了簇内样本围绕均值向量的紧密程度，E 越小样本的相似度越高。

k-means 算法的流程如下：

K-MEANS 算法

输入：期望的簇类数 K，训练样本
输出：簇划分
1、随机选取 K 个聚类中心

2、遍历样本，根据样本到每个中心的距离确定标签（归类到距离最小的簇）。
3、根据得到的簇重新计算每个簇的聚类中心（均值）
4、判断聚类是否收敛，如果不收敛，回到 3
5、结束，输出均值向量和标签

k-means 算法的实现：

```
def kMeans(data,cluster=4,max_iter=30,start=0,end=10):
    l=data.shape[1]
    num=data.shape[0]
    dim=l-1 #原始数据多一个标志位
    # cluster x dim 每行为一个聚点的坐标，一共有cluster列
    re=np.random.uniform(start,end,(cluster,dim))
    re=np.array(re)
    #终止条件为迭代次数
    Data=data.iloc[:, 0:l - 1]
    Data=np.array(Data) # 数量 x 维度
    for i in range(max_iter):
        #这个创建方式很重要
        result=[[]*num for num in range(cluster)]
        #对各个点分类
        for j in range(num):
            count=100000
            flag=0
            dot=Data[j]
            for m in range(cluster):
                clu=re[m]
```

```
        #计算新的聚点坐标
        for numclu in range(cluster):
            cluData=result[numclu]
            l=len(cluData)
            if l==0:
                break
            for numdim in range(dim):
                sum=0.0
                for datanum in range(l):
                    sum +=cluData[datanum][numdim]
                sum=sum/l
                re[numclu][numdim]=sum

    return result,re
```

3.3 GMM 模型

1) 混合模型（MM）：

混合模型是一个可以用来表示在总体分布中有 k 个子分步的概率模型。也就是说，这个模型的概率分布不是单一的，而是由多个分布组合而成的。混合模型观测数据的时候，不知道这个数据是属于哪个子分布的。

2) 高斯模型（GM）：

设样本数据为多维数据，高斯分布的概率密度函数如下：

$$P(x|\theta)=\frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}}\exp\left(-\frac{(x-\mu)^T\Sigma^{-1}(x-\mu)}{2}\right)$$

其中μ为数据均值（期望），Σ为协方差，D为数据维度

3) 高斯混合模型（GMM）：

高斯混合模型可以看作将多个单高斯模型混合到一起得到的模型。这 k 个子模型是混合模型店的隐变量。一般来说，混合模型可以用任何概率分布，但是高斯混合模型具有更好的数学性质。

定义如下变量：

x_i表示第i个观测数据，$i=1，2，\ldots\ldots，n$

K是GMM中单高斯模型的数目

α_k是观测数据属于第k个高斯子模型的概率：$\alpha_k\geq 0$，$\sum_{k=1}^{K}\alpha_k=1$

$\phi(x|\theta_k)$是第k个子模型的高斯分布函数，$\theta_k=(\mu_k,\sigma_k^2)$。

γ_{jk}表示第j个观测数据属于第k个子模型的概率

高斯混合模型的概率分布为：$P(x|\theta)=\sum_{k=1}^{K}\alpha_k\phi(x|\theta_k)$

本模型的参数为$\theta=(\alpha,\mu,\sigma^2)$。

4) 模型参数估计：

对于GMM模型来说，仍然可以用最大似然法来估计参数。即$\theta=arg\max_{\theta}L(\theta)$，其中最大对数似然函数为：

$$L(\theta)=\sum_{j=1}^{n}\log P(x_j|\theta)=\sum_{j=1}^{n}\log\left(\sum_{k=1}^{K}\alpha_k\phi(x|\theta_k)\right)$$

对于这样复杂的对数里面还有求和的函数，我们无法像实验一多项式拟合和实验二逻辑回归那样直接求导来估计参数，只能通过迭代来求解，也就是我们下面介绍的EM算法。

3.4 EM算法

EM算法是一种迭代算法，应用于隐马尔可夫模型、GMM等含有隐变量的概率模型参数的最大似然估计。

每次迭代包含两个步骤：1、E步——求期望。2、M步——求方差

EM算法的数学推导和收敛性验证都十分复杂，在本报告中我们只针对GMM模型进行特殊化介绍。首先需要明确隐变量函数。在GMM模型里面，隐变量是γ_{jk}，定义如下：

$$\gamma_{jk}=\begin{cases}1，第j个观测来自第k个子模型\\0，其他\end{cases}\quad j=1,2,.....n，k=1，2.....，K$$ 加上隐变量之后，得到的完全数据为$(y_j,\gamma_{j1},\gamma_{j2},.....，\gamma_{jK})\quad j=1,2,.....，n$

则完全数据的似然函数为：

$$P(y,\gamma|\theta)=\prod_{j=1}^{n}P(y_j,\gamma_{j1},\gamma_{j2},.....，\gamma_{jK}|\theta)$$
$$=\prod_{j=1}^{n}\prod_{k=1}^{K}[\alpha_k\phi(y_j|\theta_k)]^{\gamma_{jk}}$$

EM算法流程如下：

EM算法

输入：观测数据y_1，y_2，$.....y_n$，高斯混合模型

输出：高斯混合模型参数

（1） 取高斯混合模型参数的初始值

（2） E 步，依据当前模型的参数，计算第 K 个子高斯模型对观测数据y_j的相应程度，也就是隐变量。即：$\hat{\gamma}_{jk}=\frac{\alpha_k\phi(y_j|\theta_k)}{\sum_{k=1}^{K}\alpha_k\phi(y_j|\theta_k)}$，$j=1,2.....,n$

（3） M 步，计算新一轮迭代的高斯模型参数：

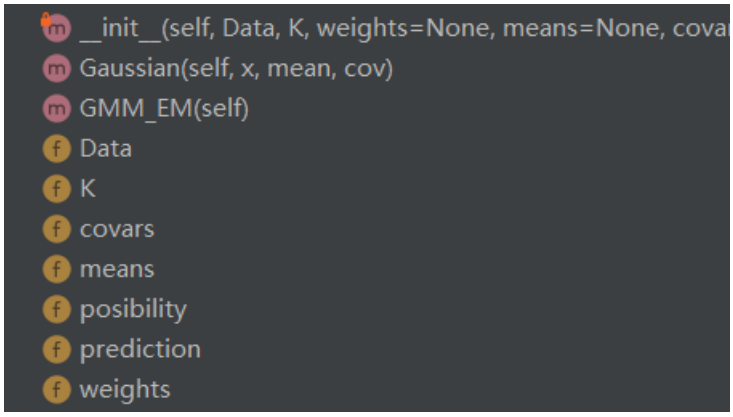
$$\hat{\mu}_k=\frac{\sum_{j=1}^{n}\hat{\gamma}_{jk}y_j}{\sum_{j=1}^{n}\hat{\gamma}_{jk}},\quad \hat{\sigma}_k{}^2=\frac{\sum_{j=1}^{n}\hat{\gamma}_{jk}(y_j-\mu_k)^2}{\sum_{j=1}^{n}\hat{\gamma}_{jk}},\quad \hat{\alpha}_k=\frac{\sum_{j=1}^{n}\hat{\gamma}_{jk}}{n}$$

（4） 重复（2）（3）直到收敛

EM 算法的实现：
为了系统的实现 EM 算法，我们定义了一个类：

```
class GMM:
    def __init__(self,Data,K,weights = None,means = None,covars = None):
        """
        这是GMM（高斯混合模型）类的构造函数
        :param Data: 训练数据
        :param K: 高斯分布的个数
        :param weigths: 每个高斯分布的初始概率（权重）
        :param means: 高斯分布的均值向量
        :param covars: 高斯分布的协方差矩阵集合
        """
```

这个类含有这些参数和方法：



其中，Gaussion 函数的功能是计算单高斯概率密度。GMM_EM 函数的功能是进行 EM 算法。
Gaussian 函数的核心代码为：

```
prob = 1.0/(np.power(np.power(2*np.pi,dim)*np.abs(covdet),0.5))*\
       np.exp(-0.5*xdiff.dot(covinv).dot(xdiff.T))[0][0]
return prob
```

GMM_EM 算法：
1、初始化隐变量为 0

```
# gamma表示第n个样本属于第k个混合高斯的概率
gammas = [np.zeros(self.K) for i in range(len)]
```

2、确定终止条件，最大似然函数的值变化精度小于 0.00000001

```
np.abs(loglikelyhood-oldloglikelyhood) > 0.00000001:
```

3、E 步：

```
4、  for n in range(len):
5、                 # respons 是 GMM 的 EM 算法中的权重 w，即后验概率
6、                 respons = [self.weights[k] * self.Gaussian(self.Data
   [n], self.means[k], self.covars[k]) for k in range(self.K)]
7、                 respons = np.array(respons)
8、                 sum_respons = np.sum(respons)
9、                 gammas[n] = respons/sum_respons
```

4、M 步：

```
1.   for k in range(self.K):
2.                 nk = np.sum([gammas[n][k] for n in range(len)])
3.                 self.weights[k] = 1.0 * nk / len
4.                 self.means[k] = (1.0/nk) * np.sum([gammas[n][k] *
    self.Data[n] for n in range(len)], axis=0)
5.                 xdiffs = self.Data - self.means[k]
6.                 self.covars[k] = (1.0/nk)*np.sum([gammas[n][k]*xd
   iffs[n].reshape((dim,1)).dot(xdiffs[n].reshape((1,dim))) for n in
    range(len)],axis=0)
```

四、实验结果与分析

4.1 k-means 算法

如图所示，我们将三组二维高斯分布数据混合到一起，各选取 100 组样本。第一组数据的均值为[4,3],协方差为[[1,0],[0,1]]。第二组数据的均值为[8,7]，协方差为[[2,0],[0,2]]。第三组数据的均值为[10,2]，协方差为[[1,0],[0,1]]。

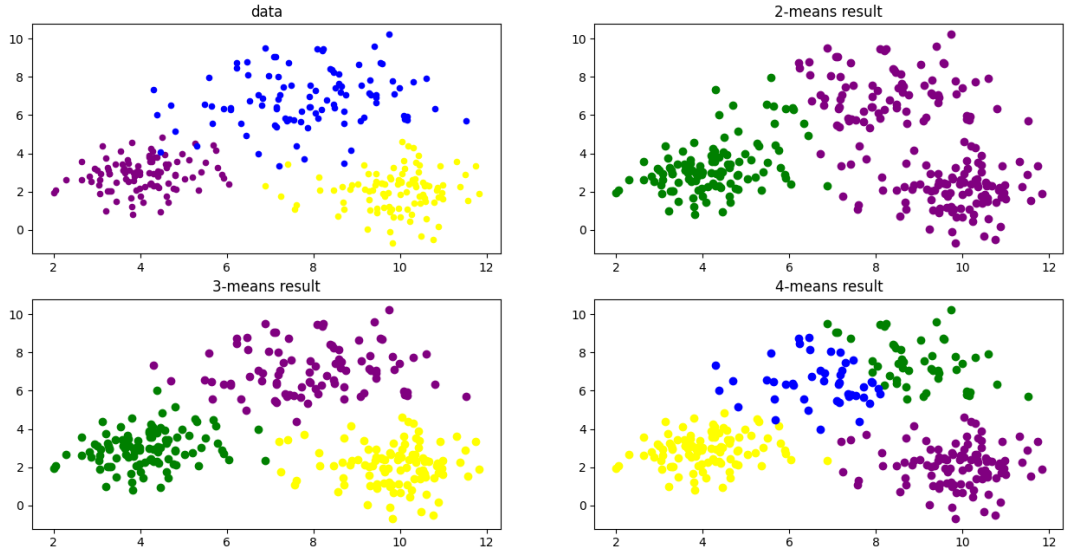
我们讲 300 组样本混合到一起，可视化处理，得到的是下图的 data 数据。其中三种颜色代表三组高斯分类。

然后我们用 k-means 算法进行聚类分析，分别选取 k=2，3，4 进行对比。初始化方式选择随机初始化，设置随机初始化的范围为[0,10],可以更好的提高拟合效果。终止条件是迭代 30 次。得到了下图结果。

2-聚类得到的聚类向量为[[9.11406402 4.34443321]，[4.17799704 3.19200746]]

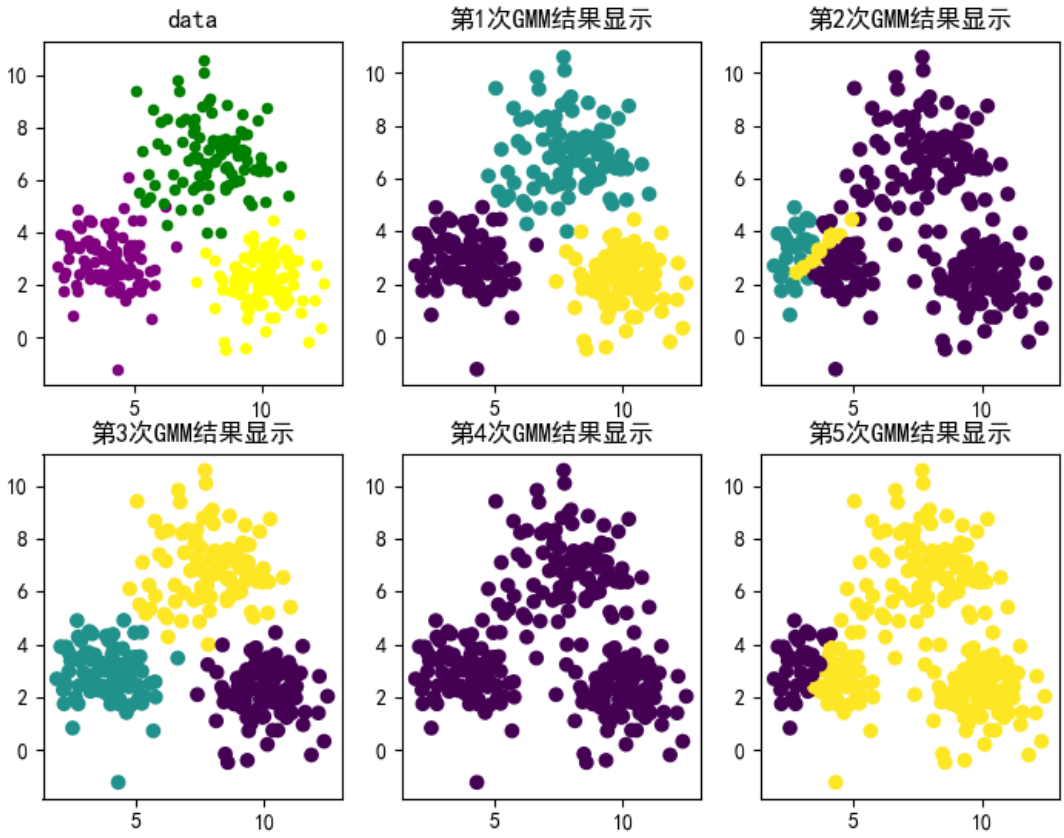
3-聚类得到的聚类向量为[[4.12603077 2.96922115]，[8.00213702 7.089965]，[9.86238808 2.08091284]]

4-聚类得到的聚类向量为[[4.09864184 2.92977842] ，[9.86238808 2.08091284]，[9.19467771 7.41118139]，[6.83120736 6.71704734]]



4.2 EM 算法

本算法所用的数据与 4.1 k-menas 算法所用的数据完全相同，可视化结果如下图的 data 所示。我们进行 5 次不同的初始化来允许 EM 算法。可以发现，GMM 算法比 K-means 算法更依赖于初始化一点，第 1、3 次 GMM 结果显示的效果非常好，可以很好的区分三类不同的高斯分布；但是第 2、4、5 次区分效果比较弱。



五、结论

K-Means 实际上假设数据式呈球状分布，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，假设每种类在样本中出现的概率相等均为 ，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响应度为 1，对于不属于的类响应度设为 0，算是对 GMM 的一种简化。

在高斯混合模型中，每个类的数据出现在样本中的概率不同，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。

由于 GMM 不像 K-means 做了很多假设，所以分类最终效果比 K-Means 好，但是 GMM-EM 算法过于细化，容易被噪声影响，所以适合对 K-Means 的分类结果进行进一步优化。

六、参考文献

https://blog.csdn.net/weixin_39946029/article/details/113637946
https://zhuanlan.zhihu.com/p/30483076
https://zhuanlan.zhihu.com/p/40991784
《统计学习方法》李航
《机器学习》周志华
老师上课的 PPT 和参考资料

七、附录：源代码（带注释）

```
1.  import numpy as np
2. import matplotlib.pyplot as plt
3. import pandas as pd
4. def generateData(path,k=3):
5.     f=open(path,'w')
6.     mean=[4,3]
7.     cov=np.mat([[1,0],[0,1]])
8.     x=np.random.multivariate_normal(mean,cov,100)
9.     for i in range(len(x)):
10.         line=[]
11.         line.append(x[i][0])
12.         line.append(x[i][1])
```

```
13.         line.append(1)
14.         line=','.join(str(i) for i in line)
15.         line=line+'\n'
16.         f.write(line)
17.     mean1=[8,7]
18.     cov1=np.mat([[2,0],[0,2]])
19.     x1=np.random.multivariate_normal(mean1,cov1,100)
20.     for i in range(len(x1)):
21.         line=[]
22.         line.append(x1[i][0])
23.         line.append(x1[i][1])
24.         line.append(2)
25.         line=','.join(str(i) for i in line)
26.         line=line+'\n'
27.         f.write(line)
28.     mean2=[10,2]
29.     conv2=np.mat([[1,0],[0,1]])
30.     x2=np.random.multivariate_normal(mean2,conv2,100)
31.     for i in range(len(x1)):
32.         line=[]
33.         line.append(x2[i][0])
34.         line.append(x2[i][1])
35.         line.append(3)
36.         line=','.join(str(i) for i in line)
37.         line=line+'\n'
38.         f.write(line)
39.     f.close()
40.
41.def datadeal(path,datadim=3):
42.     data=pd.read_csv(path,header=None,names=['data x','data y','n
    um'])
43.     data.head()
44.     plt.subplot(221)
45.     #经过分类的点可视化
46.     data1 = data[data['num'].isin([1])]
47.     data2 = data[data['num'].isin([2])]
48.     data3 = data[data['num'].isin([3])]
49.     plt.scatter(data1['data x'], data1['data y'], s=20, color='pu
    rple', marker='o',label='data1')
50.     plt.scatter(data2['data x'], data2['data y'], s=20, color='bl
    ue', marker='o',label='data2')
51.     plt.scatter(data3['data x'], data3['data y'], s=20, color='ye
    llow', marker='o',label='data3')
52.     plt.title('data')
```

```
    #ax.legend()
    #未进行分类的点可视化
    # fig, ax = plt.subplots()
    # ax.scatter(data1['data x'], data1['data y'], s=20, color='purple', marker='o')

    #plt.show()
    return data

def kMeans(data,cluster=4,max_iter=30,start=0,end=10):
    l=data.shape[1]
    num=data.shape[0]
    dim=l-1 #原始数据多一个标志位
    # cluster x dim 每行为一个聚点的坐标，一共有cluster列
    re=np.random.uniform(start,end,(cluster,dim))
    re=np.array(re)
    #终止条件为迭代次数
    Data=data.iloc[:, 0:l - 1]
    Data=np.array(Data) # 数量 x 维度
    for i in range(max_iter):
        #这个创建方式很重要
        result=[[]*num for num in range(cluster)]
        #对各个点分类
        for j in range(num):
            count=100000
            flag=0
            dot=Data[j]
            for m in range(cluster):
                clu=re[m]
                countNow=0
                for n in range(dim):
                    countNow+=(clu[n]-dot[n])**2
                if countNow<count:
                    count=countNow
                    flag=m
            dot=dot.tolist()
            result[flag].append(dot)
        #计算新的聚点坐标
        for numclu in range(cluster):
            cluData=result[numclu]
            l=len(cluData)
            if l==0:
                break
            for numdim in range(dim):
```

```
                sum=0.0
                for datanum in range(l):
                    sum +=cluData[datanum][numdim]
                sum=sum/l
                 re[numclu][numdim]=sum

    return result,re

def drawcluster(data,resultdata,resultCluster,cluster,picture):
    plt.subplot(2,2,picture)
    drawcolor=['green','purple','yellow','blue','pink']
    print(resultCluster)
    for cl in range(cluster):
        plt.scatter([i[0]for i in resultdata[cl]],
                    [i[1]for i in resultdata[cl]],
                    marker='o',c=drawcolor[cl])
    plt.title(str(cluster)+'-means result')

def kMeansMain():
    path='exp3.txt'
    #generateData(path)
    data=datadeal(path)
    for i in range(2,5):
        resultdata,resultCluster=kMeans(data,cluster=i)
        print(i,resultCluster)
        drawcluster(data,resultdata,resultCluster,cluster=i,pict
ure=i)
    plt.show()
kMeansMain()
```

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.preprocessing import Normalizer
from sklearn.metrics import accuracy_score

def generateData(path,k=3):
    f=open(path,'w')
    mean=[4,3]
    cov=np.mat([[1,0],[0,1]])
```

```
    x=np.random.multivariate_normal(mean,cov,100)
    for i in range(len(x)):
        line=[]
        line.append(x[i][0])
        line.append(x[i][1])
        line.append(0)
        line=','.join(str(i) for i in line)
        line=line+'\n'
        f.write(line)
    mean1=[8,7]
    cov1=np.mat([[2,0],[0,2]])
    x1=np.random.multivariate_normal(mean1,cov1,100)
    for i in range(len(x1)):
        line=[]
        line.append(x1[i][0])
        line.append(x1[i][1])
        line.append(1)
        line=','.join(str(i) for i in line)
        line=line+'\n'
        f.write(line)
    mean2=[10,2]
    conv2=np.mat([[1,0],[0,1]])
    x2=np.random.multivariate_normal(mean2,conv2,100)
    for i in range(len(x1)):
        line=[]
        line.append(x2[i][0])
        line.append(x2[i][1])
        line.append(2)
        line=','.join(str(i) for i in line)
        line=line+'\n'
        f.write(line)
    f.close()
def datadeal(path,datadim=3):
    data=pd.read_csv(path,header=None,names=['data x','data y','num'])
    data.head()
    plt.subplot(2,3,1)
    #经过分类的点可视化
    data1 = data[data['num'].isin([0])]
    data2 = data[data['num'].isin([1])]
    data3 = data[data['num'].isin([2])]
    plt.scatter(data1['data x'], data1['data y'], s=20, color='purple', marker='o',label='data0')
```

```
    plt.scatter(data2['data x'], data2['data y'], s=20, color='green', marker='o',label='data1')
    plt.scatter(data3['data x'], data3['data y'], s=20, color='yellow', marker='o',label='data2')
    plt.title('data')
    #ax.legend()
    #未进行分类的点可视化
    # fig, ax = plt.subplots()
    # ax.scatter(data1['data x'], data1['data y'], s=20, color='purple', marker='o')

    #plt.show()
    return data

class GMM:
    def __init__(self,Data,K,max_iterator,weights = None,means = None,covars = None):
        """
        这是 GMM（高斯混合模型）类的构造函数
        :param Data: 训练数据
        :param K: 高斯分布的个数
        :param weigths: 每个高斯分布的初始概率（权重）
        :param means: 高斯分布的均值向量
        :param covars: 高斯分布的协方差矩阵集合
        """
        self.Data = Data
        self.K = K
        self.max_iterator=max_iterator
        if weights is not None:
            self.weights = weights
        else:
            self.weights  = np.random.rand(self.K)
            self.weights /= np.sum(self.weights)        # 归一化
        col = np.shape(self.Data)[1]
        if means is not None:
            self.means = means
        else:
            # self.means = []
            means = np.random.uniform(0, 10, (self.K, col))
            self.means = np.array(means)
            # for i in range(self.K):
            #     mean = np.random.rand(col)
            #     #mean = mean / np.sum(mean)        # 归一化
            #     self.means.append(mean)
```

```
        if covars is not None:
            self.covars = covars
        else:
            self.covars  = []
            for i in range(self.K):
                cov = np.random.rand(col,col)
                 #cov = cov / np.sum(cov)                    # 归一化
                 self.covars.append(cov)                    # cov 是 np.array,但是 self.covars 是 list

    def Gaussian(self,x,mean,cov):
        """
        这是自定义的高斯分布概率密度函数
        :param x: 输入数据
        :param mean: 均值数组
        :param cov: 协方差矩阵
        :return: x 的概率
        """
        dim = np.shape(cov)[0]
        # cov 的行列式为零时的措施
        #det 计算数组的行列式
        #inv 计算矩阵（乘法）的逆矩阵
        covdet = np.linalg.det(cov + np.eye(dim) * 0.001)
        covinv = np.linalg.inv(cov + np.eye(dim) * 0.001)
        xdiff = (x - mean).reshape((1,dim))
        # 概率密度
        #多元正态分布的概率密度计算
        prob = 1.0/(np.power(np.power(2*np.pi,dim)*np.abs(covdet),0.5))*\
                np.exp(-0.5*xdiff.dot(covinv).dot(xdiff.T))[0][0]
        return prob

    def GMM_EM(self):
        """
        这是利用 EM 算法进行优化 GMM 参数的函数
        :return: 返回各组数据的属于每个分类的概率
        """
        loglikelyhood = 0 #对数最大似然估计
        oldloglikelyhood = 1
        len,dim = np.shape(self.Data)
        # gamma 表示第 n 个样本属于第 k 个混合高斯的概率
        gammas = [np.zeros(self.K) for i in range(len)]
        #迭代精度是 0.000001
```

```
135.         while abs(loglikelyhood-oldloglikelyhood)>0.00000001
136.             oldloglikelyhood = loglikelyhood
137.             # E-step
138.             for n in range(len):
139.                 # respons 是 GMM 的 EM 算法中的权重 w，即后验概率
140.                 respons = [self.weights[k] * self.Gaussian(self.
    Data[n], self.means[k], self.covars[k]) for k in range(self.K)]
141.                 respons = np.array(respons)
142.                 sum_respons = np.sum(respons)
143.                 gammas[n] = respons/sum_respons
144.             # M-step
145.             for k in range(self.K):
146.                 #nk 表示 N 个样本中有多少属于第 k 个高斯
147.                 nk = np.sum([gammas[n][k] for n in range(len)])
148.                 # 更新每个高斯分布的概率
149.                 self.weights[k] = 1.0 * nk / len
150.                 # 更新高斯分布的均值
151.                 self.means[k] = (1.0/nk) * np.sum([gammas[n][k]
    * self.Data[n] for n in range(len)], axis=0)
152.                 xdiffs = self.Data - self.means[k]
153.                 # 更新高斯分布的协方差矩阵
154.                 self.covars[k] = (1.0/nk)*np.sum([gammas[n][k]*x
    diffs[n].reshape((dim,1)).dot(xdiffs[n].reshape((1,dim))) for n i
    n range(len)],axis=0)
155.             loglikelyhood = []
156.             for n in range(len):
157.                 tmp = [np.sum(self.weights[k]*self.Gaussian(self
    .Data[n],self.means[k],self.covars[k])) for k in range(self.K)]
158.                 tmp = np.log(np.array(tmp))
159.                 loglikelyhood.append(list(tmp))
160.             loglikelyhood = np.sum(loglikelyhood)
161.         for i in range(len):
162.             gammas[i] = gammas[i]/np.sum(gammas[i])
163.         self.posibility = gammas
164.         self.prediction = [np.argmax(gammas[i]) for i in range(l
    en)]
165.
166. def run_main():
167.     """
168.         这是主函数
169.     """
170.     path='exp3Gauss.txt'
171.     #generateData(path,k=3)
172.     dataSet=datadeal(path)
```

```
    data=dataSet.iloc[:, 0:-1]
    data=np.array(data)
    # print(np.shape(data))
    label=dataSet.iloc[:,-1]
    label=np.array(label)
    # 导入 Iris 数据集
    # iris = load_iris()
    # label = np.array(iris.target)
    # print(label)
    # data = np.array(iris.data)
    # print(data)
    print("数据集的标签：\n",label)

    # 对数据进行预处理
    #data = Normalizer().fit_transform(data)

    # 解决画图是的中文乱码问题
    mpl.rcParams['font.sans-serif'] = [u'simHei']
    mpl.rcParams['axes.unicode_minus'] = False

    # 数据可视化
    # plt.subplot(121)
    # plt.scatter(data[:,0],data[:,1],c = label)
    # plt.title("Iris 数据集显示")
    # GMM 模型
    K = 3
    for i in range(5):
        gmm = GMM(data,K,max_iterator=100)
        gmm.GMM_EM()
        y_pre = gmm.prediction
        print("GMM 预测结果：\n",y_pre)
        print("GMM 正确率为：\n",accuracy_score(label,y_pre))
        plt.subplot(2,3,i+2)
        plt.scatter(dataSet.iloc[:, 0], dataSet.iloc[:, 1], c=y_
pre)
        plt.title('第'+str(i+1)+"次 GMM 结果显示")
    plt.show()


if __name__ == '__main__':
    run_main()

```