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： 实现 k-means 聚类方法和混合高斯模型

学号：1173710204
姓名： 陈东鑫

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。

实验环境：

操作系统：Windows10
语言：python3
编程环境：jupyter notebook

三、设计思想（本程序中的用到的主要算法及数据结构）

1．算法原理

GMM 模型
混合高斯模型，即多个单高斯模型的叠加

涉及到了 k-means 算法和 EM 算法
k-means 算法：
1．从样本点中随机选择 k 个点作为初始的样本中心点。点集记为 K_0
2．对于所有的样本点，在 k 个样本中心点中找出与其欧式距离最短的样本中心点，并将该点加入到对应的集合 S_i（其中，$i \in [0, k)$）中。
3．对于每个点集 S，求出其中心点作为新的样本中心点。记为 K_t（其中 t 代表第 t 次迭代）
4．if $K_t \neq K_{t-1}$ 重复 2，3 步，else return K_t，S
为使 k-means 算法有一个比较好的结果，对 k-means 算法进行了改进。对每一次运行完 k-means 算法之后的结果，即样本中心点集和 k 个分类后的样本点集。对于每个分类集合，计算集合中所有点与集合的中心点的平均欧氏距离，将这 k 个平均欧氏距离再求其平均值，记为 E，作为评判结果好坏的标准。多次（如，100 次）运行 k-means 算法，找到 E 的值最小的那一次，作为 k-means 算法的最终结果。

EM 算法（Expectation-maximization algorithm，即最大期望算法）：
是在概率模型中寻找参数最大似然估计或者最大后验估计的算法，其中概率模型

依赖于无法观测的隐性变量。
最大期望算法经过两个步骤交替进行计算：

第一步是计算期望（E），利用对隐藏变量的现有估计值，计算其最大似然估计值；

$$\gamma(z_{nk}) = \mathrm{p}(z_{nk} = 1|\mathrm{x}) = \frac{\pi_k N(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j N(x_n|\mu_j, \Sigma_j)}$$

第二步是最大化（M），最大化在 E 步上求得的最大似然值来计算参数的值。M 步上找到的参数估计值被用于下一个 E 步计算中，这个过程不断交替进行。

$$\mu_k^{new} = \frac{1}{N_k} \sum_n \gamma(z_{nk}) x_n , N_k = \sum_n \gamma(z_{nk})$$

$$\Sigma_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(y_{nk})(x_n - \mu_k)(x_n - \mu_k)^T$$

$$\pi_k^{new} = \frac{N_k}{N}$$

2. 算法的实现

1）生成数据

使用 np.random.multivariate_normal(mean, cov, size)函数生成二维高斯分布数据，多次使用则可得到混合高斯分数数据集。mean 代表均值，cov 代表协方差矩阵，size 代表样本点个数。其中包含 k 个二维高斯分布的数据，在演示中，k=4。其均值的矩阵为[[0, 0], [0, 1], [1, 0], [1, 1]]，其协方差矩阵为

```
[[[0.04, 0.  ],
  [0.  , 0.04]],
 [[0.04, 0.  ],
  [0.  , 0.04]],
 [[0.04, 0.  ],
  [0.  , 0.04]],
 [[0.04, 0.  ],
  [0.  , 0.04]]]
```

每个高斯分布有 400 个样本点，即总共有 1600 个样本点
将所得到的数据进行归一化处理。得到样本点的分布如下图所示：

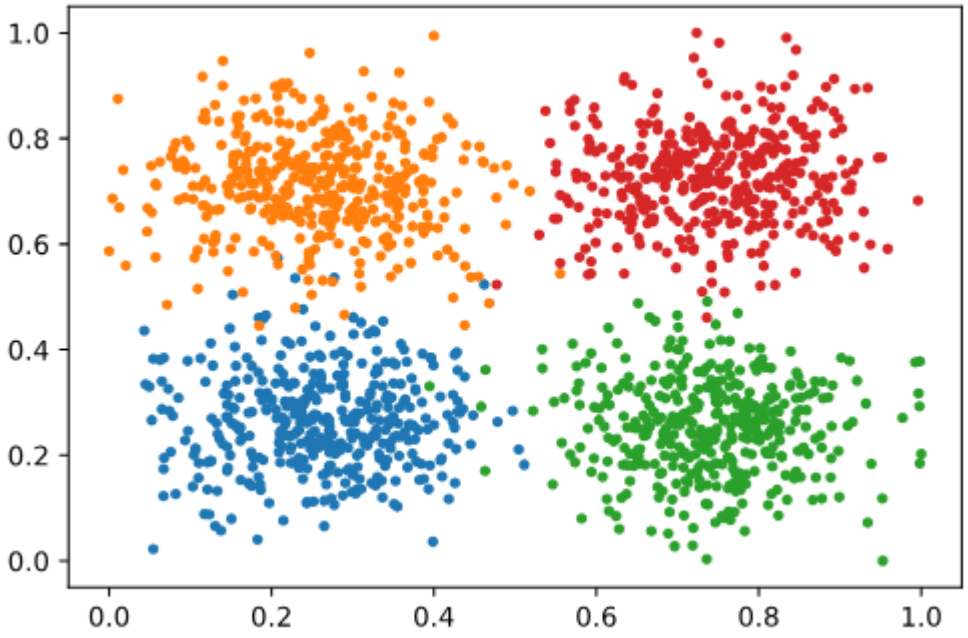


图 1 样本点集

生成数据所需函数的 python 代码如下

```
# k 个分类
# M 为每个分布的样本个数
# mean 为平均值矩阵组成的一个 list
# cov 为协方差矩阵组成的 list
def genGMM(k, mean, cov, M):
    X = np.random.multivariate_normal(mean[0], cov[0], M).T
    Y = []
    Y.append(0 * np.ones((1, M)))
    for i in range(1, k):
        X = np.c_[X, np.random.multivariate_normal(mean[i], cov[i], M).T]
        Y.append(i * np.ones((1, M)))

    # 归一化
    theMax = np.max(X, axis=1).reshape((2, -1))
    theMin = np.min(X, axis=1).reshape((2, -1))
    X = X - theMin
    X = X / (theMax - theMin)
    return X, Y
```

生成数据的代码如下：

```
k = 4
M = 400
```

```
mean = [[0, 0], [0, 1], [1, 0], [1, 1]]
cov = []
cov.append(0.04 * np.eye(2))
cov.append(0.04 * np.eye(2))
cov.append(0.04 * np.eye(2))
cov.append(0.04 * np.eye(2))

X, Y = genGMM(k, mean, cov, M)
points = genPoints(k, X)
# points = np.array(np.meshgrid(np.linspace(0.25,0.75,2),np.linspace(0.25,0.75,2))).reshape((2,-1))
# points = np.r_[points,np.array([0,1,2,3]).reshape((1,-1))]

newPoints = points.copy()
```

2）训练模型

由算法原理处的论述，算法实现所需的函数代码如下

```
# 随机选取 k 个点
# points 中
# 第一行是横坐标
# 第二行是纵坐标
# 第三行代表第 i 个分类
def genPoints(k):
    points = np.array([random.random(), random.random(), 0]).reshape((-1, 1))
    for i in range(1, k):
        temp = np.array([random.random(), random.random(), i]).reshape((-1, 1))
        points = np.c_[points, temp]
    return points


# 从 X 中随机选取 k 个点
def genPoints(k, X):
    temp = X.copy().T
    np.random.shuffle(temp)
    temp = temp.T
    index = random.randint(0, k * M - k - 1)
    points = temp[:, index:index + k].reshape(2, -1)
    return points

```

```
23.
24. # k-means 算法的一次迭代
25. def kmeans(k, M, points):
26.     theMax = 10 * np.ones((1, k * M))
27.     label = -1 * np.ones((1, k * M))
28.     for i in range(k):
29.         temp = np.sum((X - points[0:2, i].reshape(2, -1))**2, axis=0)
30.         temp = temp.reshape((1, -1))
31.         label[temp < theMax] = i
32.         theMax[temp < theMax] = temp[temp < theMax]
33.     newPoints = points.copy()
34.     label.reshape((1, -1))
35.     #     print(X.shape)
36.     #     print((label==i).shape)
37.
38.     #     a = X[0].reshape(1,-1)
39.     #     print(a.shape)
40.     #     print(a[label==i])
41.
42.     for i in range(k):
43.         if label.all() != i:
44.             continue
45.         a = X[0].reshape(1, -1)
46.         xSum = np.sum(a[label == i]) / np.sum(label == i)
47.         b = X[1].reshape(1, -1)
48.         ySum = np.sum(b[label == i]) / np.sum(label == i)
49.         newPoints[0][i] = xSum
50.         newPoints[1][i] = ySum
51.
52.     varSum = 0
53.     for i in range(label.size):
54.         x1 = (X[0, i] - points[0, label[0, i].astype('int')])**2
55.         x2 = (X[1, i] - points[1, label[0, i].astype('int')])**2
56.         varSum += (x1 + x2)**0.5
57.     varSum /= k * M
58.     return newPoints, label, varSum
59.
60.
61. # alpha.shape = (1, k)
62. def getAlpha(X, label, k, M):
63.     alpha = np.ones((1, k))
64.     for i in range(k):
65.         alpha[0, i] = np.sum(label == i) / (k * M)
66.     return alpha
```

```
67.
68.
69. # sigma.shape = (2, 2*k)
70. def getSigma(X, label):
71.     sigma = np.ones((2, 2 * k))
72.     tempX = X[0].reshape((1, -1))
73.     tempY = X[1].reshape((1, -1))
74.     flag = True
75.     for i in range(k):
76.         temp1 = tempX[label == i].reshape((1, -1))
77.         temp2 = tempY[label == i].reshape((1, -1))
78.         temp = np.r_[temp1, temp2]
79.         if flag:
80.             sigma = np.cov(temp)
81.             flag = False
82.         else:
83.             sigma = np.c_[sigma, np.cov(temp)]
84.     #     print(sigma)
85.     return sigma
86.
87.
88. # mu.shape = (2, k)
89. def getMu(X, points, label, k):
90.     mu = np.ones((2, k))
91.     for i in range(k):
92.         A = X[0].reshape((1, -1))
93.         B = X[1].reshape((1, -1))
94.         muA = np.sum(A[label == i]) / np.sum(label == i)
95.         muB = np.sum(B[label == i]) / np.sum(label == i)
96.         mu[0, i] = muA
97.         mu[1, i] = muB
98. #     print(mu)
99.     return mu
100.
101.
102. # 概率密度
103. # phi.shape = (k, k*M)
104. def getPhi(X, mu, sigma, k, M):
105.     phi = np.ones((k, k * M))
106.     for i in range(k):
107.         mySigma = sigma[:, i * 2:i * 2 + 2].reshape((2, -1))
108.         for j in range(k * M):
109.             sub = X[:, j] - mu[:, i]
110.             inv = np.linalg.inv(mySigma)
```

```
111.             dot = np.dot(sub.T, inv)
112.             dot = np.dot(dot, sub)
113.             exp = math.exp(dot / -2)
114.             det = np.linalg.det(mySigma)
115.             coef = 1 / (2 * math.pi * det)
116.             phi[i, j] = coef * exp
117.     return phi
118.
119.
120. # 响应度
121. # gamma.shape = (k, k*M)
122. def getGamma(phi, alpha, k, M):
123.     gamma = np.ones((k, k * M))
124.     for i in range(k):
125.         for j in range(k * M):
126.             up = phi[i, j] * alpha[0, i]
127.             down = 0
128.             for index in range(k):
129.                 down += alpha[0, index] * phi[index, j]
130.             gamma[i, j] = up / down
131.     return gamma
132.
133.
134. # 估计新的均值
135. def getNewMu(gamma, X, k):
136.     flag = True
137.     mu = np.ones((2, k))
138.     for i in range(k):
139.         up = np.sum(X * gamma[i].reshape((1, -
    1)), axis=1).reshape((2, 1))
140.         down = np.sum(gamma[i])
141.         if flag:
142.             mu = up / down
143.             flag = False
144.         else:
145.             mu = np.c_[mu, up / down]
146.
147.
148. #     print(mu)
149. #     print(mu.shape)
150.     return mu
151.
152.
153. def getNewSigma(gamma, X, mu, k, M):
```

```
    flag = True
    sigma = np.ones((2, 2 * k))
    for i in range(k):
        down = np.sum(gamma[i])
        up = np.zeros((2, 2))
        for j in range(k * M):
            temp = (X[:, j] - mu[:, i]).reshape((2, 1))
            up += np.dot(temp, temp.T) * gamma[i][j]
        if flag:
            sigma = up / down
            flag = False
        else:
            sigma = np.c_[sigma, up / down]
    return sigma


def getNewAlpha(gamma, k, M):
    alpha = np.sum(gamma, axis=1) / (k * M)
    alpha = alpha.reshape((1, -1))
    return alpha


def EStep(alpha, sigma, mu, k, M, X):
    phi = getPhi(X, mu, sigma, k, M)
    gamma = getGamma(phi, alpha, k, M)
    return gamma


def MStep(gamma, mu, k, M, X):
    newSigma = getNewSigma(gamma, X, mu, k, M)
    newMu = getNewMu(gamma, X, k)
    newAlpha = getNewAlpha(gamma, k, M)
    return newAlpha, newMu, newSigma


def EM(alpha, sigma, mu, k, M, X):
    newGamma = EStep(alpha, sigma, mu, k, M, X)
    newAlpha, newMu, newSigma = MStep(newGamma, mu, k, M, X)
    return newAlpha, newMu, newSigma


def control(X, mu, sigma, alpha, k, M):
    stop = False
    phi = getPhi(X, mu, sigma, k, M)
```

```
    a = alpha.T
    mul = phi * a
    Sum = np.sum(mul, axis=1) / (k * M)
    #     Mul = Sum.prod()
    Mul = np.log(Sum)
    Mul = np.sum(Sum)
    return Mul
```

使用这些所给函数，根据算法的实现原理进行调用，即可实现 k-means 算法和 EM 算法训练使用的代码如下：

```
varSum = float('inf')
res = points.copy()
for i in range(100):
    flag = True
    count = 0
    while (not (newPoints == points).all()) or flag:
        count = count + 1
        flag = False
        points = newPoints.copy()
        newPoints, label, newVarSum = kmeans(k, M, points)
    if newVarSum <= varSum:
        resLabel = label
        varSum = newVarSum
        res = newPoints

newPoints = res
label = resLabel

print("最终的", k, "个样本中心点为:")
print(res)
print(varSum)

alpha = getAlpha(X, label, k, M)
sigma = getSigma(X, label)
mu = getMu(X, res, label, k)

newAlpha = alpha
newMu = mu
newSigma = sigma
oldLoss = control(X, mu, sigma, alpha, k, M)
newLoss = oldLoss

for i in range(100):
```

```
    newAlpha, newMu, newSigma = EM(alpha, sigma, mu, k, M, X)
    alpha = newAlpha
    mu = newMu
    sigma = newSigma
    oldLoss = newLoss
    newLoss = control(X, mu, sigma, alpha, k, M)
    print('newLoss =', newLoss)
    print('t =', abs(newLoss - oldLoss))
    if abs(newLoss - oldLoss) < 1e-6:
        print(i)
        print(abs(newLoss - oldLoss))
        break
print(newAlpha)
print(newMu)
print(newSigma)

x = X[0].reshape((1, -1))
y = X[1].reshape((1, -1))
pointX = mu[0]
pointY = mu[1]

phi = getPhi(X, mu, sigma, k, M)
label = np.argmax(phi, axis=0).reshape((1, -1))
maxLabel = np.max(phi, axis=0).reshape((1, -1))
maxLabel = maxLabel / np.max(maxLabel)
```

为了读取 UCI 数据，生成数据函数重新写了一个，可从 txt 文件中读取数据，生成 80%训练集和 20%测试集：

```
def readFile(filename):
    f = open(filename, 'r', encoding='utf-8')
    return f.readlines()


# M 为训练集样本点个数
# O 为维度数
def genDataFromFile(filename):
    context = readFile(filename)

    for i in range(len(context)):
        context[i] = context[i].replace('\n', '')
        context[i] = context[i].replace(' ', '')
        context[i] = (context[i]).split(',')
```

```
    O = len(context[0]) - 1
    M = len(context)

    for i in range(M):
        for j in range(O + 1):
            if j != O:
                context[i][j] = float(context[i][j])
            else:
                context[i][j] = int(context[i][j])

    data = np.array(context).reshape((M, O + 1))
    data = np.random.permutation(data)

    # 归一化
    dataMax = np.max(data, axis=0)
    dataMin = np.min(data, axis=0)
    temp = dataMax - dataMin
    data = data - dataMin
    data = data / temp

    # data, dataT
    M_ = M
    M = int(M * 0.8)
    MT = M_ - M
    print("M =", M)
    print("MT =", MT)
    dataT = data.copy()[M:, :]
    data = data.copy()[0:M, :]

    # data1, data2
    flag1 = False
    flag2 = False
    for i in range(M):
        if data[i][O] == 0:
            if flag1:
                data1 = np.c_[data1, data[:][i].reshape((O + 1, 1))]
            else:
                data1 = data[:][i].reshape((O + 1, 1))
                flag1 = True
        if data[i][O] == 1:
            if flag2:
                data2 = np.c_[data2, data[:][i].reshape((O + 1, 1))]
            else:
                data2 = data[:][i].reshape((O + 1, 1))
```

```
                flag2 = True

    # dataT1, dataT2
    flag1 = False
    flag2 = False
    for i in range(MT):
        if dataT[i][O] == 0:
            if flag1:
                dataT1 = np.c_[dataT1, dataT[:][i].reshape((O + 1, 1))]
            else:
                dataT1 = dataT[:][i].reshape((O + 1, 1))
                flag1 = True
        if dataT[i][O] == 1:
            if flag2:
                dataT2 = np.c_[dataT2, dataT[:][i].reshape((O + 1, 1))]
            else:
                dataT2 = dataT[:][i].reshape((O + 1, 1))
                flag2 = True
    data = data.T
    dataT=dataT.T

    # X, Y, XT, YT
    X = data[0:O][:].copy()
    Y = data[O][:].copy().reshape((1, M))
    XT = dataT[0:O][:].copy()
    YT = dataT[O][:].copy().reshape((1, MT))

    return M, O, data, data1, data2, X, Y, MT, dataT, dataT1, dataT2, XT, YT
```

四、实验结果与分析

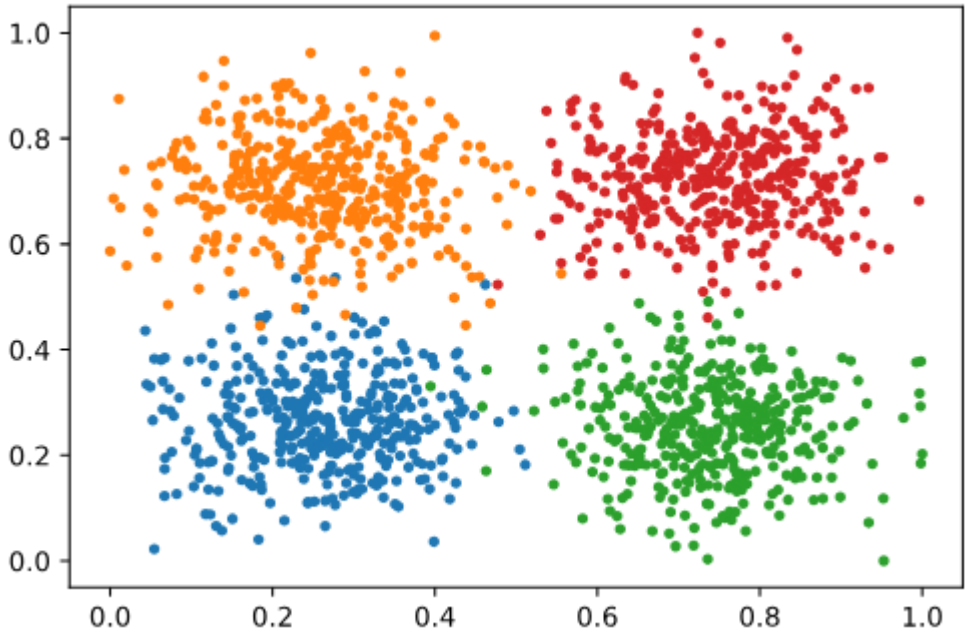
实验生成的原始数据集，其均值矩阵为（列数代表第 k 个分类，第 0 行代表 x-mean，第 1 行代表 y-mean）

```
[[0.26079499 0.25597464 0.73977219 0.74647384]
 [0.26551264 0.71346611 0.25127398 0.72707481]]
```

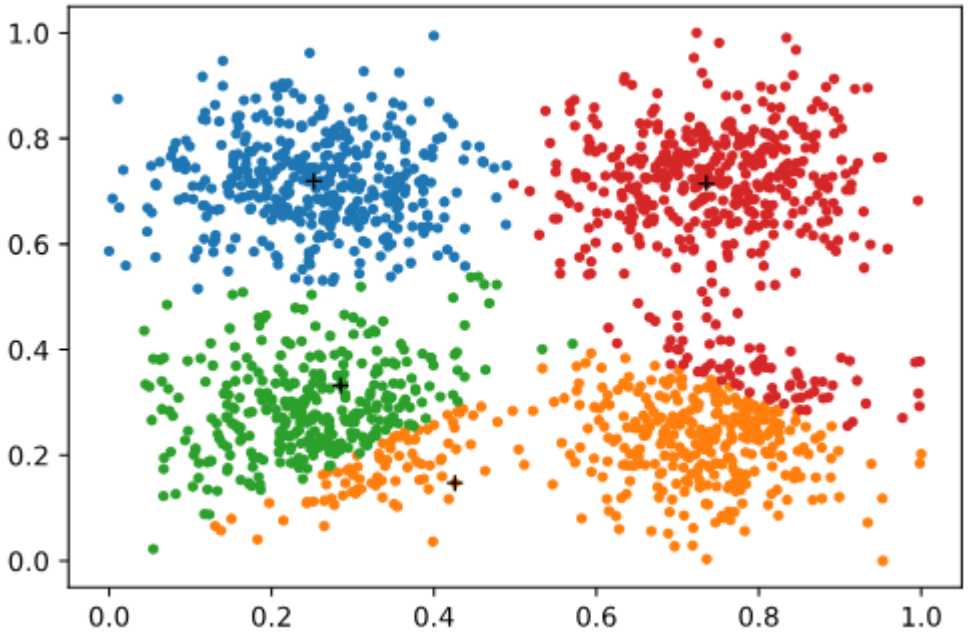
方差矩阵为（2 行 2*k 列的矩阵，其中的每个 2*2 矩阵代表第 k 个分类的协方差矩阵）

```
[[0.00924654  0.00021624  0.01058996  -0.00080631  0.00898275  -0.00046889
0.0092174 0.00067074]
 [0.00021624  0.00837669  -0.00080631  0.00901465  -0.00046889  0.00832645
0.00067074 0.00816772]]
```

数据集的分布如下图所示



k-means 算法的结果如下图所示：



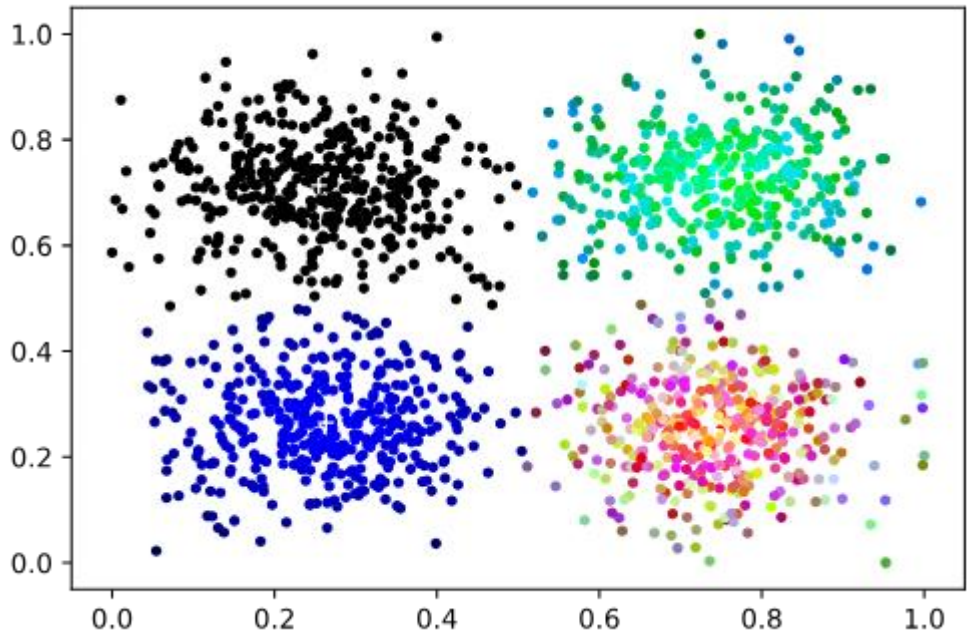
训练结果,其中 k 个分类的顺序可能会被打乱，
其均值矩阵为

```
[[0.25455214 0.74246177 0.26319183 0.74485539]
 [0.71362326 0.25096370 0.26544462 0.72693546]]
```

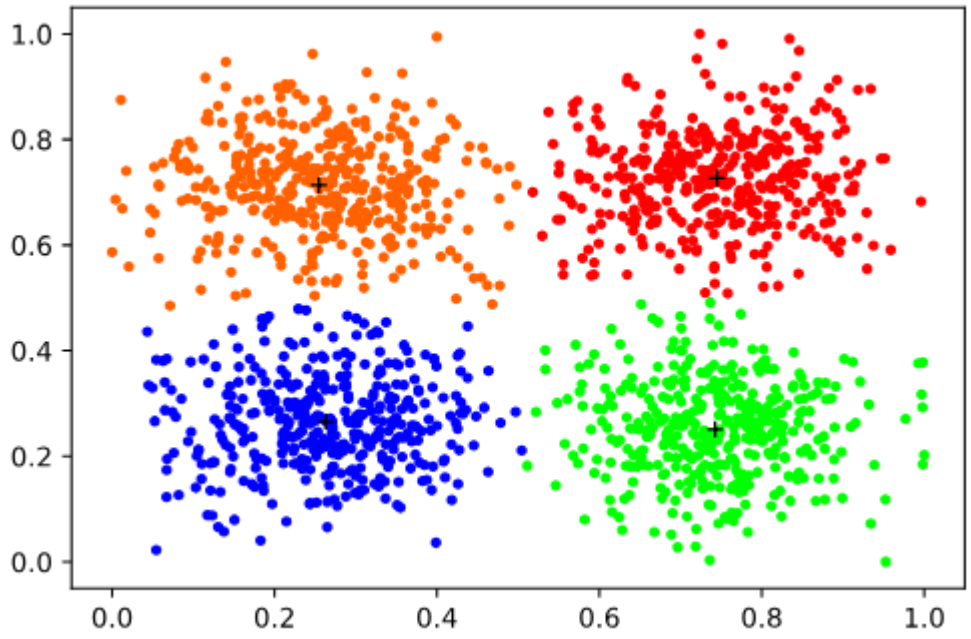
其协方差矩阵为

```
[[ 0.01031368 -0.0006512  0.00832545 -0.00024494  0.00969452  0.00043982
0.00956405 0.00074137]
 [-0.0006512  0.00895626 -0.00024494  0.00843329  0.00043982  0.00811162
0.00074137 0.00815385]]
```

训练结果的分布如下图所示（考虑到概率的情况时，对颜色的深浅做一下处理，产生该图）



对 EM 算法的结果采用硬分类的方法画图如下图所示



五、结论

k-means 算法和 EM 算法，其结果受其初始选点的随机性影响较大，容易陷入局部最优的情况。如上述图中的 k-means 算法的情况就是陷入了局部最优解。在实验时，采取两种算法结合使用的方式来使得结果较好。具体是从样本点集中随机选取 k 个点作为初始点，使用 k-means 算法进行预处理，k-means 算法得到的结果又作为 EM 算法的初始点来进一步计算，这样得到的结果往往都较为理想。

六、参考文献

七、附录：源代码（带注释）

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

colorMaps1 = {
    0: '#000000',
    1: '#ff0000',
    2: '#0000ff',
    3: '#00ff00',
    7: '#00ff00',
    4: '#ff6100',
    5: '#0000ff',
    6: '#ff0000'
}


def strToHexWithRatio(string, ratio):
    string = string.replace('#', '')
    myHex = int(string, 16)
    myHex = int(ratio * myHex)
    myStr = '{:0>6x}'.format(myHex)
    myStr = '#' + myStr
    return myStr

# k 个分类
# M 为每个分布的样本个数
# mean 为平均值矩阵组成的一个 list
# cov 为协方差矩阵组成的 list
def genGMM(k, mean, cov, M):
    X = np.random.multivariate_normal(mean[0], cov[0], M).T
    Y = []
    Y.append(0 * np.ones((1, M)))
    for i in range(1, k):
        X = np.c_[X, np.random.multivariate_normal(mean[i], cov[i], M).T]
        Y.append(i * np.ones((1, M)))

    # 归一化
    theMax = np.max(X, axis=1).reshape((2, -1))
```

```
    theMin = np.min(X, axis=1).reshape((2, -1))
    X = X - theMin
    X = X / (theMax - theMin)
    return X, Y


# 随机选取 k 个点
# points 中
# 第一行是横坐标
# 第二行是纵坐标
# 第三行代表第 i 个分类
def genPoints(k):
    points = np.array([random.random(), random.random(), 0]).reshape((-1, 1))
    for i in range(1, k):
        temp = np.array([random.random(), random.random(), i]).reshape((-1, 1))
        points = np.c_[points, temp]
    return points


# 从 X 中随机选取 k 个点
def genPoints(k, X):
    temp = X.copy().T
    np.random.shuffle(temp)
    temp = temp.T
    index = random.randint(0, k * M - k - 1)
    points = temp[:, index:index + k].reshape(2, -1)
    return points


# k-means 算法的一次迭代
def kmeans(k, M, points):
    theMax = 10 * np.ones((1, k * M))
    label = -1 * np.ones((1, k * M))
    for i in range(k):
        temp = np.sum((X - points[0:2, i].reshape(2, -1))**2, axis=0)
        temp = temp.reshape((1, -1))
        label[temp < theMax] = i
        theMax[temp < theMax] = temp[temp < theMax]
    newPoints = points.copy()
    label.reshape((1, -1))
    #     print(X.shape)
    #     print((label==i).shape)
```

```
82.
83.     #     a = X[0].reshape(1,-1)
84.     #     print(a.shape)
85.     #     print(a[label==i])
86.
87.     for i in range(k):
88.         if label.all() != i:
89.             continue
90.         a = X[0].reshape(1, -1)
91.         xSum = np.sum(a[label == i]) / np.sum(label == i)
92.         b = X[1].reshape(1, -1)
93.         ySum = np.sum(b[label == i]) / np.sum(label == i)
94.         newPoints[0][i] = xSum
95.         newPoints[1][i] = ySum
96.
97.     varSum = 0
98.     for i in range(label.size):
99.         x1 = (X[0, i] - points[0, label[0, i].astype('int')])**2
100.         x2 = (X[1, i] - points[1, label[0, i].astype('int')])**2
101.         varSum += (x1 + x2)**0.5
102.     varSum /= k * M
103.     return newPoints, label, varSum
104.
105.
106. # alpha.shape = (1, k)
107. def getAlpha(X, label, k, M):
108.     alpha = np.ones((1, k))
109.     for i in range(k):
110.         alpha[0, i] = np.sum(label == i) / (k * M)
111.     return alpha
112.
113.
114. # sigma.shape = (2, 2*k)
115. def getSigma(X, label):
116.     sigma = np.ones((2, 2 * k))
117.     tempX = X[0].reshape((1, -1))
118.     tempY = X[1].reshape((1, -1))
119.     flag = True
120.     for i in range(k):
121.         temp1 = tempX[label == i].reshape((1, -1))
122.         temp2 = tempY[label == i].reshape((1, -1))
123.         temp = np.r_[temp1, temp2]
124.         if flag:
125.             sigma = np.cov(temp)
```

```
126.             flag = False
127.         else:
128.             sigma = np.c_[sigma, np.cov(temp)]
129.     #     print(sigma)
130.     return sigma
131.
132.
133. # mu.shape = (2, k)
134. def getMu(X, points, label, k):
135.     mu = np.ones((2, k))
136.     for i in range(k):
137.         A = X[0].reshape((1, -1))
138.         B = X[1].reshape((1, -1))
139.         muA = np.sum(A[label == i]) / np.sum(label == i)
140.         muB = np.sum(B[label == i]) / np.sum(label == i)
141.         mu[0, i] = muA
142.         mu[1, i] = muB
143. #     print(mu)
144.     return mu
145.
146.
147. # 概率密度
148. # phi.shape = (k, k*M)
149. def getPhi(X, mu, sigma, k, M):
150.     phi = np.ones((k, k * M))
151.     for i in range(k):
152.         mySigma = sigma[:, i * 2:i * 2 + 2].reshape((2, -1))
153.         for j in range(k * M):
154.             sub = X[:, j] - mu[:, i]
155.             inv = np.linalg.inv(mySigma)
156.             dot = np.dot(sub.T, inv)
157.             dot = np.dot(dot, sub)
158.             exp = math.exp(dot / -2)
159.             det = np.linalg.det(mySigma)
160.             coef = 1 / (2 * math.pi * det)
161.             phi[i, j] = coef * exp
162.     return phi
163.
164.
165. # 响应度
166. # gamma.shape = (k, k*M)
167. def getGamma(phi, alpha, k, M):
168.     gamma = np.ones((k, k * M))
169.     for i in range(k):
```

```
170.         for j in range(k * M):
171.             up = phi[i, j] * alpha[0, i]
172.             down = 0
173.             for index in range(k):
174.                 down += alpha[0, index] * phi[index, j]
175.             gamma[i, j] = up / down
176.     return gamma
177.
178.
179. # 估计新的均值
180. def getNewMu(gamma, X, k):
181.     flag = True
182.     mu = np.ones((2, k))
183.     for i in range(k):
184.         up = np.sum(X * gamma[i].reshape((1, -
    1)), axis=1).reshape((2, 1))
185.         down = np.sum(gamma[i])
186.         if flag:
187.             mu = up / down
188.             flag = False
189.         else:
190.             mu = np.c_[mu, up / down]
191.
192.
193. #     print(mu)
194. #     print(mu.shape)
195.     return mu
196.
197.
198. def getNewSigma(gamma, X, mu, k, M):
199.     flag = True
200.     sigma = np.ones((2, 2 * k))
201.     for i in range(k):
202.         down = np.sum(gamma[i])
203.         up = np.zeros((2, 2))
204.         for j in range(k * M):
205.             temp = (X[:, j] - mu[:, i]).reshape((2, 1))
206.             up += np.dot(temp, temp.T) * gamma[i][j]
207.         if flag:
208.             sigma = up / down
209.             flag = False
210.         else:
211.             sigma = np.c_[sigma, up / down]
212.     return sigma
```

```
213.
214.
215. def getNewAlpha(gamma, k, M):
216.     alpha = np.sum(gamma, axis=1) / (k * M)
217.     alpha = alpha.reshape((1, -1))
218.     return alpha
219.
220.
221. def EStep(alpha, sigma, mu, k, M, X):
222.     phi = getPhi(X, mu, sigma, k, M)
223.     gamma = getGamma(phi, alpha, k, M)
224.     return gamma
225.
226.
227. def MStep(gamma, mu, k, M, X):
228.     newSigma = getNewSigma(gamma, X, mu, k, M)
229.     newMu = getNewMu(gamma, X, k)
230.     newAlpha = getNewAlpha(gamma, k, M)
231.     return newAlpha, newMu, newSigma
232.
233.
234. def EM(alpha, sigma, mu, k, M, X):
235.     newGamma = EStep(alpha, sigma, mu, k, M, X)
236.     newAlpha, newMu, newSigma = MStep(newGamma, mu, k, M, X)
237.     return newAlpha, newMu, newSigma
238.
239.
240. def control(X, mu, sigma, alpha, k, M):
241.     stop = False
242.     phi = getPhi(X, mu, sigma, k, M)
243.     a = alpha.T
244.     mul = phi * a
245.     Sum = np.sum(mul, axis=1) / (k * M)
246.     #     Mul = Sum.prod()
247.     Mul = np.log(Sum)
248.     Mul = np.sum(Sum)
249.     return Mul
250.
251. k = 4
252. M = 400
253. mean = [[0, 0], [0, 1], [1, 0], [1, 1]]
254. cov = []
255. cov.append(0.04 * np.eye(2))
256. cov.append(0.04 * np.eye(2))
```

```
cov.append(0.04 * np.eye(2))
cov.append(0.04 * np.eye(2))

X, Y = genGMM(k, mean, cov, M)
points = genPoints(k, X)
# points = np.array(np.meshgrid(np.linspace(0.25,0.75,2),np.linspace(0.25,0.75,2))).reshape((2,-1))
# points = np.r_[points,np.array([0,1,2,3]).reshape((1,-1))]

newPoints = points.copy()
label = np.array(Y)
label = label.reshape((1, -1))
print(getMu(X, points, label, k))
print((getSigma(X, label)))

# 画出样本点
plt.figure()
x = X[0, :].reshape((1, -1))
y = X[1, :].reshape((1, -1))
# plt.plot(x, y, '.', color='blue')
for i in range(k):
    plt.plot(x[label == i], y[label == i], '.')
plt.xlim((-0.05, 1.05))
plt.ylim((-0.05, 1.05))
plt.show()

varSum = float('inf')
res = points.copy()
for i in range(100):
    flag = True
    count = 0
    while (not (newPoints == points).all()) or flag:
        count = count + 1
        flag = False
        points = newPoints.copy()
        newPoints, label, newVarSum = kmeans(k, M, points)
    if newVarSum <= varSum:
        resLabel = label
        varSum = newVarSum
        res = newPoints

newPoints = res
label = resLabel

```

```
print("最终的", k, "个样本中心点为:")
print(res)
print(varSum)

# 画出结果
plt.figure()
x = X[0].reshape((1, -1))
y = X[1].reshape((1, -1))
pointX = newPoints[0]
pointY = newPoints[1]
for i in range(k):
    plt.plot(x[label == i], y[label == i], '.')
plt.plot(pointX, pointY, '+', color='black')
plt.xlim((-0.05, 1.05))
plt.ylim((-0.05, 1.05))
plt.show()

alpha = getAlpha(X, label, k, M)
sigma = getSigma(X, label)
mu = getMu(X, res, label, k)

newAlpha = alpha
newMu = mu
newSigma = sigma
oldLoss = control(X, mu, sigma, alpha, k, M)
newLoss = oldLoss

for i in range(100):
    newAlpha, newMu, newSigma = EM(alpha, sigma, mu, k, M, X)
    alpha = newAlpha
    mu = newMu
    sigma = newSigma
    oldLoss = newLoss
    newLoss = control(X, mu, sigma, alpha, k, M)
    print('newLoss =', newLoss)
    print('t =', abs(newLoss - oldLoss))
    if abs(newLoss - oldLoss) < 1e-6:
        print(i)
        print(abs(newLoss - oldLoss))
        break
print(newAlpha)
print(newMu)
print(newSigma)

```

```
x = X[0].reshape((1, -1))
y = X[1].reshape((1, -1))
pointX = mu[0]
pointY = mu[1]

phi = getPhi(X, mu, sigma, k, M)
label = np.argmax(phi, axis=0).reshape((1, -1))
maxLabel = np.max(phi, axis=0).reshape((1, -1))
maxLabel = maxLabel / np.max(maxLabel)

# 画出结果
plt.figure()
# for i in range(k):
#     plt.plot(x[label == i], y[label == i], '.')
for i in range(k * M):
    plt.plot(x[0, i],
             y[0, i],
             '.',
             color=strToHexWithRatio(colorMaps1[label[0, i]],
                                     maxLabel[0, i]**0.2))

plt.plot(pointX, pointY, '+', color='white')
plt.xlim((-0.05, 1.05))
plt.ylim((-0.05, 1.05))
plt.show()

# 画出结果
plt.figure()
# for i in range(k):
#     plt.plot(x[label == i], y[label == i], '.')
newLabel = label + 4 * np.ones_like(label)
for i in range(k * M):
    plt.plot(x[0, i],
             y[0, i],
             '.',
             color=strToHexWithRatio(colorMaps1[newLabel[0, i]], 1))

plt.plot(pointX, pointY, '+', color='black')
plt.xlim((-0.05, 1.05))
plt.ylim((-0.05, 1.05))
plt.show()
```