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

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。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K0

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Si（其中，i ∈ [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Kt（其中t代表第t次迭代）

4．if Kt ≠ Kt-1重复2，3步，else return Kt，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；

           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。

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：

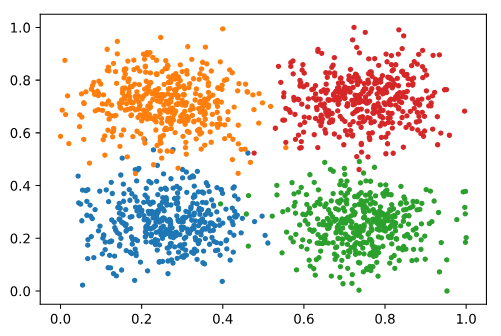


图 1样本点集

生成数据所需函数的python代码如下

1. # k个分类
2. # M为每个分布的样本个数
3. # mean为平均值矩阵组成的一个list
4. # cov为协方差矩阵组成的list
5. **def** genGMM(k, mean, cov, M):
6. X = np.random.multivariate\_normal(mean[0], cov[0], M).T
7. Y = []
8. Y.append(0 \* np.ones((1, M)))
9. **for** i **in** range(1, k):
10. X = np.c\_[X, np.random.multivariate\_normal(mean[i], cov[i], M).T]
11. Y.append(i \* np.ones((1, M)))
13. # 归一化
14. theMax = np.max(X, axis=1).reshape((2, -1))
15. theMin = np.min(X, axis=1).reshape((2, -1))
16. X = X - theMin
17. X = X / (theMax - theMin)
18. **return** X, Y

生成数据的代码如下：

1. k = 4
2. M = 400
3. mean = [[0, 0], [0, 1], [1, 0], [1, 1]]
4. cov = []
5. cov.append(0.04 \* np.eye(2))
6. cov.append(0.04 \* np.eye(2))
7. cov.append(0.04 \* np.eye(2))
8. cov.append(0.04 \* np.eye(2))
10. X, Y = genGMM(k, mean, cov, M)
11. points = genPoints(k, X)
12. # points = np.array(np.meshgrid(np.linspace(0.25,0.75,2),np.linspace(0.25,0.75,2))).reshape((2,-1))
13. # points = np.r\_[points,np.array([0,1,2,3]).reshape((1,-1))]
15. 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

1. # 随机选取k个点
2. # points中
3. # 第一行是横坐标
4. # 第二行是纵坐标
5. # 第三行代表第i个分类
6. **def** genPoints(k):
7. points = np.array([random.random(), random.random(), 0]).reshape((-1, 1))
8. **for** i **in** range(1, k):
9. temp = np.array([random.random(), random.random(), i]).reshape((-1, 1))
10. points = np.c\_[points, temp]
11. **return** points

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

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)
38. #     a = X[0].reshape(1,-1)
39. #     print(a.shape)
40. #     print(a[label==i])
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
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

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

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

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

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

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

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]

148. #     print(mu)
149. #     print(mu.shape)
150. **return** mu

153. **def** getNewSigma(gamma, X, mu, k, M):
154. flag = True
155. sigma = np.ones((2, 2 \* k))
156. **for** i **in** range(k):
157. down = np.sum(gamma[i])
158. up = np.zeros((2, 2))
159. **for** j **in** range(k \* M):
160. temp = (X[:, j] - mu[:, i]).reshape((2, 1))
161. up += np.dot(temp, temp.T) \* gamma[i][j]
162. **if** flag:
163. sigma = up / down
164. flag = False
165. **else**:
166. sigma = np.c\_[sigma, up / down]
167. **return** sigma

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

176. **def** EStep(alpha, sigma, mu, k, M, X):
177. phi = getPhi(X, mu, sigma, k, M)
178. gamma = getGamma(phi, alpha, k, M)
179. **return** gamma

182. **def** MStep(gamma, mu, k, M, X):
183. newSigma = getNewSigma(gamma, X, mu, k, M)
184. newMu = getNewMu(gamma, X, k)
185. newAlpha = getNewAlpha(gamma, k, M)
186. **return** newAlpha, newMu, newSigma

189. **def** EM(alpha, sigma, mu, k, M, X):
190. newGamma = EStep(alpha, sigma, mu, k, M, X)
191. newAlpha, newMu, newSigma = MStep(newGamma, mu, k, M, X)
192. **return** newAlpha, newMu, newSigma

195. **def** control(X, mu, sigma, alpha, k, M):
196. stop = False
197. phi = getPhi(X, mu, sigma, k, M)
198. a = alpha.T
199. mul = phi \* a
200. Sum = np.sum(mul, axis=1) / (k \* M)
201. #     Mul = Sum.prod()
202. Mul = np.log(Sum)
203. Mul = np.sum(Sum)
204. **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：

1. varSum = float('inf')
2. res = points.copy()
3. **for** i **in** range(100):
4. flag = True
5. count = 0
6. **while** (**not** (newPoints == points).all()) **or** flag:
7. count = count + 1
8. flag = False
9. points = newPoints.copy()
10. newPoints, label, newVarSum = kmeans(k, M, points)
11. **if** newVarSum <= varSum:
12. resLabel = label
13. varSum = newVarSum
14. res = newPoints
16. newPoints = res
17. label = resLabel
19. **print**("最终的", k, "个样本中心点为:")
20. **print**(res)
21. **print**(varSum)
23. alpha = getAlpha(X, label, k, M)
24. sigma = getSigma(X, label)
25. mu = getMu(X, res, label, k)
27. newAlpha = alpha
28. newMu = mu
29. newSigma = sigma
30. oldLoss = control(X, mu, sigma, alpha, k, M)
31. newLoss = oldLoss
33. **for** i **in** range(100):
34. newAlpha, newMu, newSigma = EM(alpha, sigma, mu, k, M, X)
35. alpha = newAlpha
36. mu = newMu
37. sigma = newSigma
38. oldLoss = newLoss
39. newLoss = control(X, mu, sigma, alpha, k, M)
40. **print**('newLoss =', newLoss)
41. **print**('t =', abs(newLoss - oldLoss))
42. **if** abs(newLoss - oldLoss) < 1e-6:
43. **print**(i)
44. **print**(abs(newLoss - oldLoss))
45. **break**
46. **print**(newAlpha)
47. **print**(newMu)
48. **print**(newSigma)
50. x = X[0].reshape((1, -1))
51. y = X[1].reshape((1, -1))
52. pointX = mu[0]
53. pointY = mu[1]
55. phi = getPhi(X, mu, sigma, k, M)
56. label = np.argmax(phi, axis=0).reshape((1, -1))
57. maxLabel = np.max(phi, axis=0).reshape((1, -1))
58. 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：

1. **def** readFile(filename):
2. f = open(filename, 'r', encoding='utf-8')
3. **return** f.readlines()

6. # M为训练集样本点个数
7. # O为维度数
8. **def** genDataFromFile(filename):
9. context = readFile(filename)
11. **for** i **in** range(len(context)):
12. context[i] = context[i].replace('\n', '')
13. context[i] = context[i].replace(' ', '')
14. context[i] = (context[i]).split(',')
15. O = len(context[0]) - 1
16. M = len(context)
18. **for** i **in** range(M):
19. **for** j **in** range(O + 1):
20. **if** j != O:
21. context[i][j] = float(context[i][j])
22. **else**:
23. context[i][j] = int(context[i][j])
25. data = np.array(context).reshape((M, O + 1))
26. data = np.random.permutation(data)
28. # 归一化
29. dataMax = np.max(data, axis=0)
30. dataMin = np.min(data, axis=0)
31. temp = dataMax - dataMin
32. data = data - dataMin
33. data = data / temp
35. # data, dataT
36. M\_ = M
37. M = int(M \* 0.8)
38. MT = M\_ - M
39. **print**("M =", M)
40. **print**("MT =", MT)
41. dataT = data.copy()[M:, :]
42. data = data.copy()[0:M, :]
44. # data1, data2
45. flag1 = False
46. flag2 = False
47. **for** i **in** range(M):
48. **if** data[i][O] == 0:
49. **if** flag1:
50. data1 = np.c\_[data1, data[:][i].reshape((O + 1, 1))]
51. **else**:
52. data1 = data[:][i].reshape((O + 1, 1))
53. flag1 = True
54. **if** data[i][O] == 1:
55. **if** flag2:
56. data2 = np.c\_[data2, data[:][i].reshape((O + 1, 1))]
57. **else**:
58. data2 = data[:][i].reshape((O + 1, 1))
59. flag2 = True
61. # dataT1, dataT2
62. flag1 = False
63. flag2 = False
64. **for** i **in** range(MT):
65. **if** dataT[i][O] == 0:
66. **if** flag1:
67. dataT1 = np.c\_[dataT1, dataT[:][i].reshape((O + 1, 1))]
68. **else**:
69. dataT1 = dataT[:][i].reshape((O + 1, 1))
70. flag1 = True
71. **if** dataT[i][O] == 1:
72. **if** flag2:
73. dataT2 = np.c\_[dataT2, dataT[:][i].reshape((O + 1, 1))]
74. **else**:
75. dataT2 = dataT[:][i].reshape((O + 1, 1))
76. flag2 = True
77. data = data.T
78. dataT=dataT.T
80. # X, Y, XT, YT
81. X = data[0:O][:].copy()
82. Y = data[O][:].copy().reshape((1, M))
83. XT = dataT[0:O][:].copy()
84. YT = dataT[O][:].copy().reshape((1, MT))
86. **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]

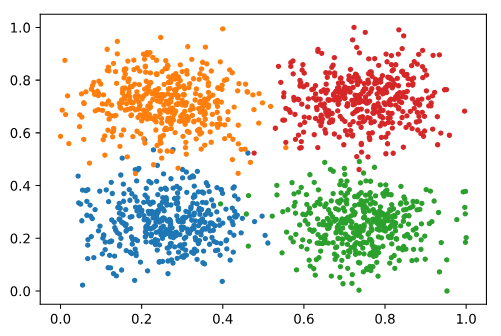
[0.26551264 0.71346611 0.25127398 0.72707481]]

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）

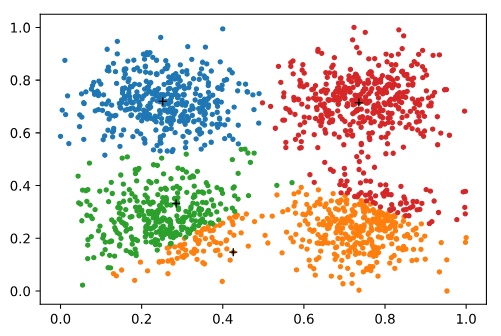
[[0.00924654 0.00021624 0.01058996 -0.00080631 0.00898275 -0.00046889 0.0092174 0.00067074]

[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]

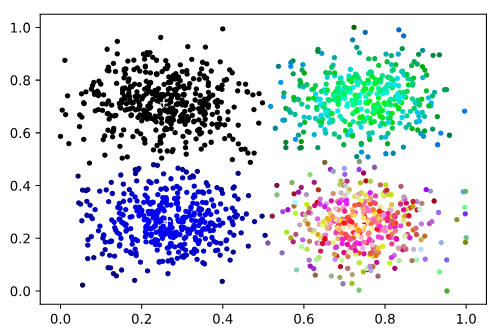
[0.71362326 0.25096370 0.26544462 0.72693546]]

其协方差矩阵为

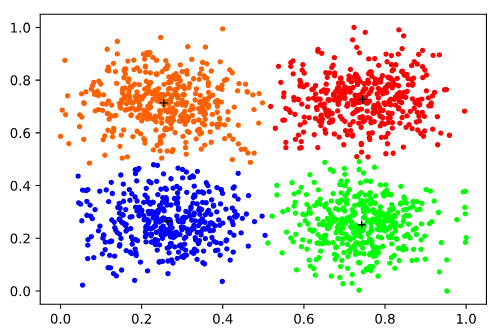
[[ 0.01031368 -0.0006512 0.00832545 -0.00024494 0.00969452 0.00043982 0.00956405 0.00074137]

[-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）

1. **import** numpy as np
2. **import** math
3. **import** matplotlib.pyplot as plt
4. **import** random
6. colorMaps1 = {
7. 0: '#000000',
8. 1: '#ff0000',
9. 2: '#0000ff',
10. 3: '#00ff00',
11. 7: '#00ff00',
12. 4: '#ff6100',
13. 5: '#0000ff',
14. 6: '#ff0000'
15. }

18. **def** strToHexWithRatio(string, ratio):
19. string = string.replace('#', '')
20. myHex = int(string, 16)
21. myHex = int(ratio \* myHex)
22. myStr = '{:0>6x}'.format(myHex)
23. myStr = '#' + myStr
24. **return** myStr
26. # k个分类
27. # M为每个分布的样本个数
28. # mean为平均值矩阵组成的一个list
29. # cov为协方差矩阵组成的list
30. **def** genGMM(k, mean, cov, M):
31. X = np.random.multivariate\_normal(mean[0], cov[0], M).T
32. Y = []
33. Y.append(0 \* np.ones((1, M)))
34. **for** i **in** range(1, k):
35. X = np.c\_[X, np.random.multivariate\_normal(mean[i], cov[i], M).T]
36. Y.append(i \* np.ones((1, M)))
38. # 归一化
39. theMax = np.max(X, axis=1).reshape((2, -1))
40. theMin = np.min(X, axis=1).reshape((2, -1))
41. X = X - theMin
42. X = X / (theMax - theMin)
43. **return** X, Y

46. # 随机选取k个点
47. # points中
48. # 第一行是横坐标
49. # 第二行是纵坐标
50. # 第三行代表第i个分类
51. **def** genPoints(k):
52. points = np.array([random.random(), random.random(), 0]).reshape((-1, 1))
53. **for** i **in** range(1, k):
54. temp = np.array([random.random(), random.random(), i]).reshape((-1, 1))
55. points = np.c\_[points, temp]
56. **return** points

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

69. # k-means算法的一次迭代
70. **def** kmeans(k, M, points):
71. theMax = 10 \* np.ones((1, k \* M))
72. label = -1 \* np.ones((1, k \* M))
73. **for** i **in** range(k):
74. temp = np.sum((X - points[0:2, i].reshape(2, -1))\*\*2, axis=0)
75. temp = temp.reshape((1, -1))
76. label[temp < theMax] = i
77. theMax[temp < theMax] = temp[temp < theMax]
78. newPoints = points.copy()
79. label.reshape((1, -1))
80. #     print(X.shape)
81. #     print((label==i).shape)
83. #     a = X[0].reshape(1,-1)
84. #     print(a.shape)
85. #     print(a[label==i])
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
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

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

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

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

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

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

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]

193. #     print(mu)
194. #     print(mu.shape)
195. **return** mu

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

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

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

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

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

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
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))
257. cov.append(0.04 \* np.eye(2))
258. cov.append(0.04 \* np.eye(2))
260. X, Y = genGMM(k, mean, cov, M)
261. points = genPoints(k, X)
262. # points = np.array(np.meshgrid(np.linspace(0.25,0.75,2),np.linspace(0.25,0.75,2))).reshape((2,-1))
263. # points = np.r\_[points,np.array([0,1,2,3]).reshape((1,-1))]
265. newPoints = points.copy()
266. label = np.array(Y)
267. label = label.reshape((1, -1))
268. **print**(getMu(X, points, label, k))
269. **print**((getSigma(X, label)))
271. # 画出样本点
272. plt.figure()
273. x = X[0, :].reshape((1, -1))
274. y = X[1, :].reshape((1, -1))
275. # plt.plot(x, y, '.', color='blue')
276. **for** i **in** range(k):
277. plt.plot(x[label == i], y[label == i], '.')
278. plt.xlim((-0.05, 1.05))
279. plt.ylim((-0.05, 1.05))
280. plt.show()
282. varSum = float('inf')
283. res = points.copy()
284. **for** i **in** range(100):
285. flag = True
286. count = 0
287. **while** (**not** (newPoints == points).all()) **or** flag:
288. count = count + 1
289. flag = False
290. points = newPoints.copy()
291. newPoints, label, newVarSum = kmeans(k, M, points)
292. **if** newVarSum <= varSum:
293. resLabel = label
294. varSum = newVarSum
295. res = newPoints
297. newPoints = res
298. label = resLabel
300. **print**("最终的", k, "个样本中心点为:")
301. **print**(res)
302. **print**(varSum)
304. # 画出结果
305. plt.figure()
306. x = X[0].reshape((1, -1))
307. y = X[1].reshape((1, -1))
308. pointX = newPoints[0]
309. pointY = newPoints[1]
310. **for** i **in** range(k):
311. plt.plot(x[label == i], y[label == i], '.')
312. plt.plot(pointX, pointY, '+', color='black')
313. plt.xlim((-0.05, 1.05))
314. plt.ylim((-0.05, 1.05))
315. plt.show()
317. alpha = getAlpha(X, label, k, M)
318. sigma = getSigma(X, label)
319. mu = getMu(X, res, label, k)
321. newAlpha = alpha
322. newMu = mu
323. newSigma = sigma
324. oldLoss = control(X, mu, sigma, alpha, k, M)
325. newLoss = oldLoss
327. **for** i **in** range(100):
328. newAlpha, newMu, newSigma = EM(alpha, sigma, mu, k, M, X)
329. alpha = newAlpha
330. mu = newMu
331. sigma = newSigma
332. oldLoss = newLoss
333. newLoss = control(X, mu, sigma, alpha, k, M)
334. **print**('newLoss =', newLoss)
335. **print**('t =', abs(newLoss - oldLoss))
336. **if** abs(newLoss - oldLoss) < 1e-6:
337. **print**(i)
338. **print**(abs(newLoss - oldLoss))
339. **break**
340. **print**(newAlpha)
341. **print**(newMu)
342. **print**(newSigma)
344. x = X[0].reshape((1, -1))
345. y = X[1].reshape((1, -1))
346. pointX = mu[0]
347. pointY = mu[1]
349. phi = getPhi(X, mu, sigma, k, M)
350. label = np.argmax(phi, axis=0).reshape((1, -1))
351. maxLabel = np.max(phi, axis=0).reshape((1, -1))
352. maxLabel = maxLabel / np.max(maxLabel)
354. # 画出结果
355. plt.figure()
356. # for i in range(k):
357. #     plt.plot(x[label == i], y[label == i], '.')
358. **for** i **in** range(k \* M):
359. plt.plot(x[0, i],
360. y[0, i],
361. '.',
362. color=strToHexWithRatio(colorMaps1[label[0, i]],
363. maxLabel[0, i]\*\*0.2))
365. plt.plot(pointX, pointY, '+', color='white')
366. plt.xlim((-0.05, 1.05))
367. plt.ylim((-0.05, 1.05))
368. plt.show()
370. # 画出结果
371. plt.figure()
372. # for i in range(k):
373. #     plt.plot(x[label == i], y[label == i], '.')
374. newLabel = label + 4 \* np.ones\_like(label)
375. **for** i **in** range(k \* M):
376. plt.plot(x[0, i],
377. y[0, i],
378. '.',
379. color=strToHexWithRatio(colorMaps1[newLabel[0, i]], 1))
381. plt.plot(pointX, pointY, '+', color='black')
382. plt.xlim((-0.05, 1.05))
383. plt.ylim((-0.05, 1.05))
384. plt.show()