《模式识别与机器学习A》实验报告

实验题目：实现k-means聚类方法和混合高斯模型

班级： 2203601

学号： 2022113416

姓名： 刘子康

**实验报告内容**

**一、实验目的**

·实现一个k-means聚类算法和混合高斯模型，并且用EM算法估计模型中的参数；

·手动生成k组高斯分布的数据，或利用UCI上的数据集，对聚类模型加以验证。

**二、实验内容**

手动生成k组不同均值和方差高斯分布的数据，参数自行设定：

（1）用k-means聚类，测试算法效果；

（2）用混合高斯模型和实现的EM算法估计参数，查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（与设定的结果比较）。

应用：在UCI上找一个简单问题数据，用实现的GMM进行聚类。

**三、实验环境**

·操作系统：Windows 11

·编程语言：Python 3.10

·第三方库：Numpy 1.23.4，Scipy 1.10.0，Pandas 2.1.4，Matplotlib 3.8.2

·IDE：Pycharm 2022 社区版

**四、实验过程、结果及分析**

**4.1实验原理**

K-means是一种基于距离的硬聚类方法，目标是将数据分成K个簇，每个簇内的点尽可能接近聚类中心，而簇间尽可能分开。其基本原理为维护K个聚类中心，将每个样本点分配到距离最近的聚类中心所在簇，并计算每个簇中所有样本点均值来更新聚类中心，重复此过程，直至聚类中心不再移动。

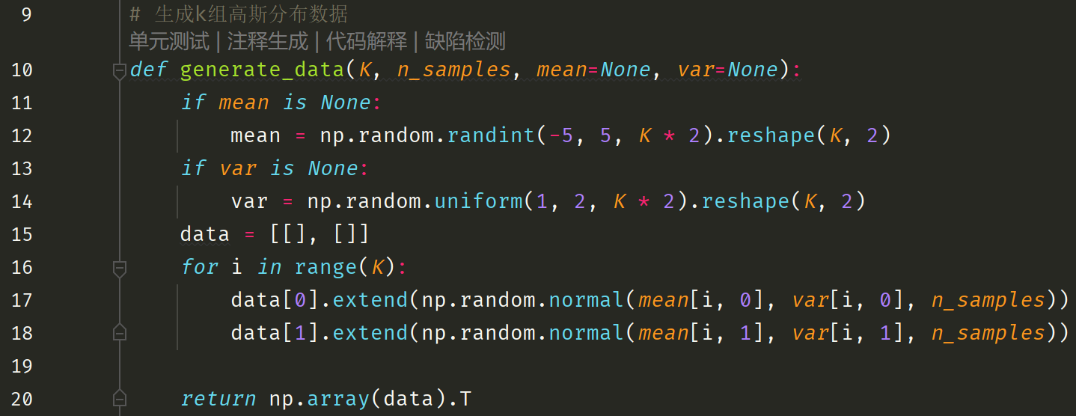
高斯混合模型（Gaussian Mixture Model，GMM） 是一种基于概率模型的软聚类方法，可以看作是对K-means的一种扩展。GMM认为数据是由多个高斯分布混合而成的，每个簇对应一个高斯分布，每个样本点对于各个簇有一个概率值，表示它属于该簇的可能性。GMM 使用期望最大化（Expectation-Maximization，EM）算法进行参数估计，EM算法是一种迭代算法，用于含有隐藏变量的概率模型参数的极大似然估计，主要包括两个步骤：E步求期望，M步极大化。

聚类评价指标选择兰德指数（RI），它是一种常用的聚类评估指标，通过计算同类且同聚类或异类且异聚类的比例，比较聚类结果与真实标签之间的相似性。

**4.2实验过程**

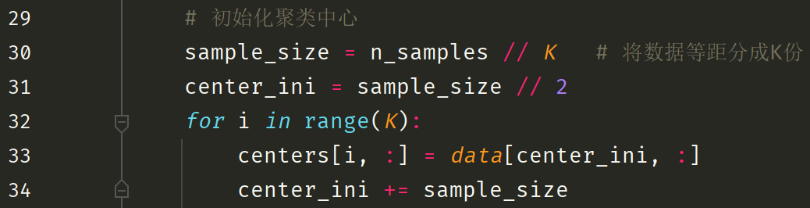
**4.2.1生成K组高斯分布数据**

使用Numpy库的random模块随机生成K组不同的均值和方差，并使用normal()函数每组随机生成n\_samples个符合高斯分布的样本点（为便于绘图，特征数设置为2），拼接后返回生成的数据集。

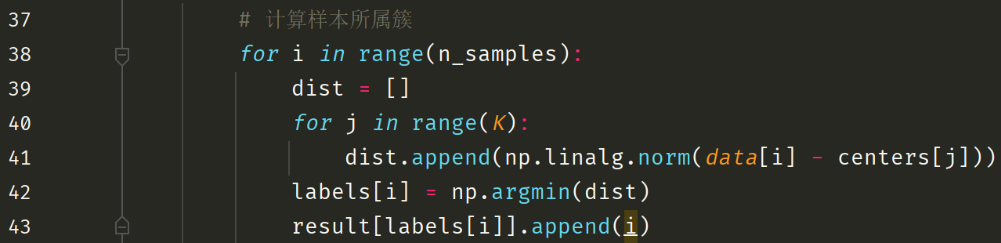


**4.2.2 K-means聚类算法实现**

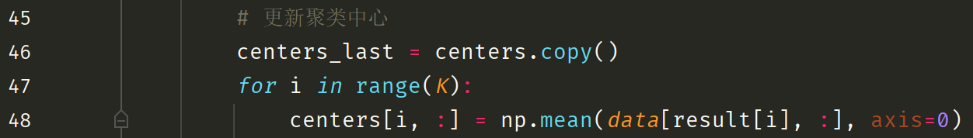
（1）初始化聚类中心。随机或线性选取K个样本点作为初始聚类中心；



（2）分配样本点。对于数据集中的每个样本点，计算它到每个聚类中心的距离（通常使用欧氏距离），将其分配给距离最近的簇；



（3）更新每个簇的中心。对于每个簇，计算其所有成员点的均值，即为新的聚类中心；

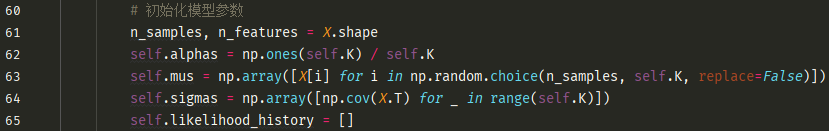


（4）重复（2）和（3），直至聚类中心不再变化。记录前一次的聚类中心，每次循环更新完聚类中心后与之比较，若相等则结束迭代。

（5）输出聚类结果，计算RI，绘制图像，不同颜色代表不同簇的聚类结果。

**4.2.3 GMM模型构建**

（1）初始化每个簇的高斯分布参数：均值（μ）、协方差矩阵（σ）和权重（α）。随机选择K个样本点作为初始均值，计算数据集转置后的协方差矩阵作为初始协方差矩阵，初始权重设置为平均权重。



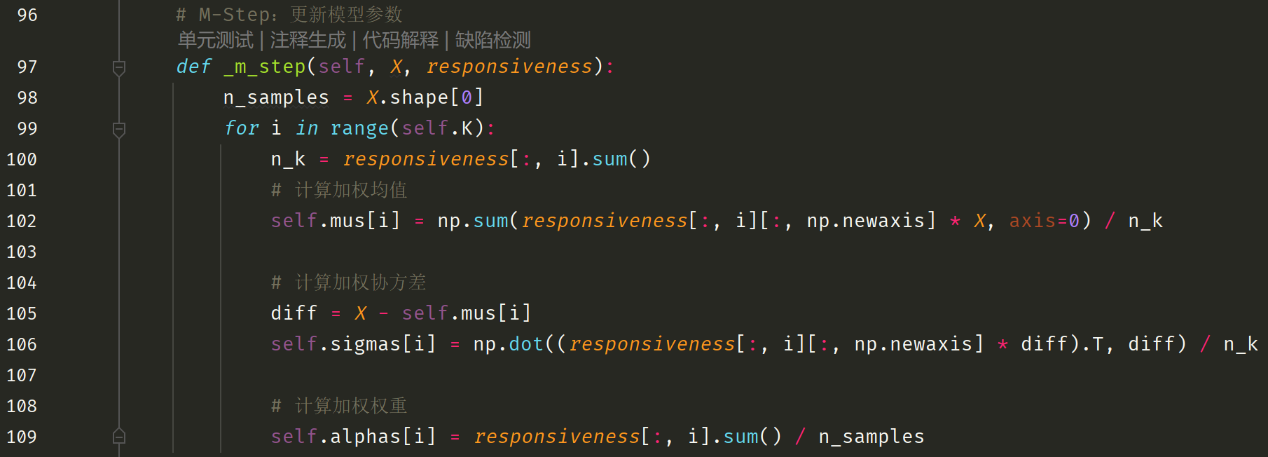
（2）E-Step，计算分模型k对观测数据yj的响应度。

根据公式，计算当前模型参数下第j个观测数据来自第k个分模型（簇）的概率。使用Scipy库stats模块的multivariate\_normal函数，计算观测值在参数下的概率密度函数值，并乘上相应权重，最后除以各分模型响应度求和。



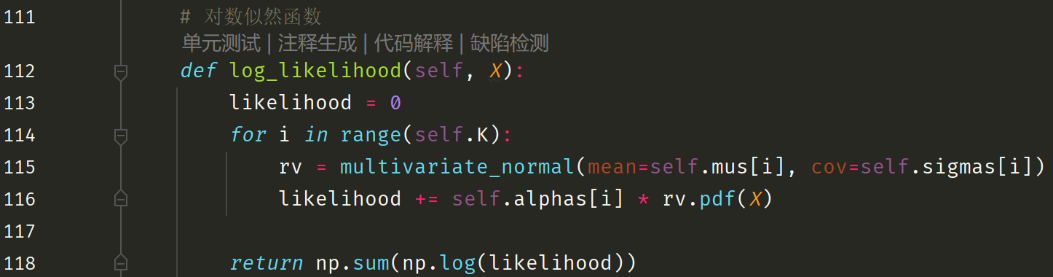
（3）M-Step，更新新一轮迭代的模型参数。

用、和表示的各参数，更新模型参数。求、只需将分别对、求偏导并令其为0，即可得到，公式为，；求是在条件下求偏导并令其为0得到的，公式为。

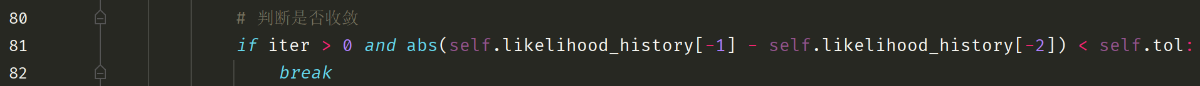


（4）重复（2）和（3），直至收敛。

明确隐变量，构造完全数据的对数似然函数。



维护一个历史对数似然函数值列表，每次迭代后与上一次的似然函数值比较，若几乎没有变化则结束迭代（tol默认值1e-6）。



（5）输出结果并绘图。每个样本点通过对属于各个簇的概率值取最大值对应簇，作为它的类别，输出聚类结果，计算RI，绘制图像，不同颜色代表不同簇的聚类结果。

**4.2.4 UCI鸢尾花数据集测试**

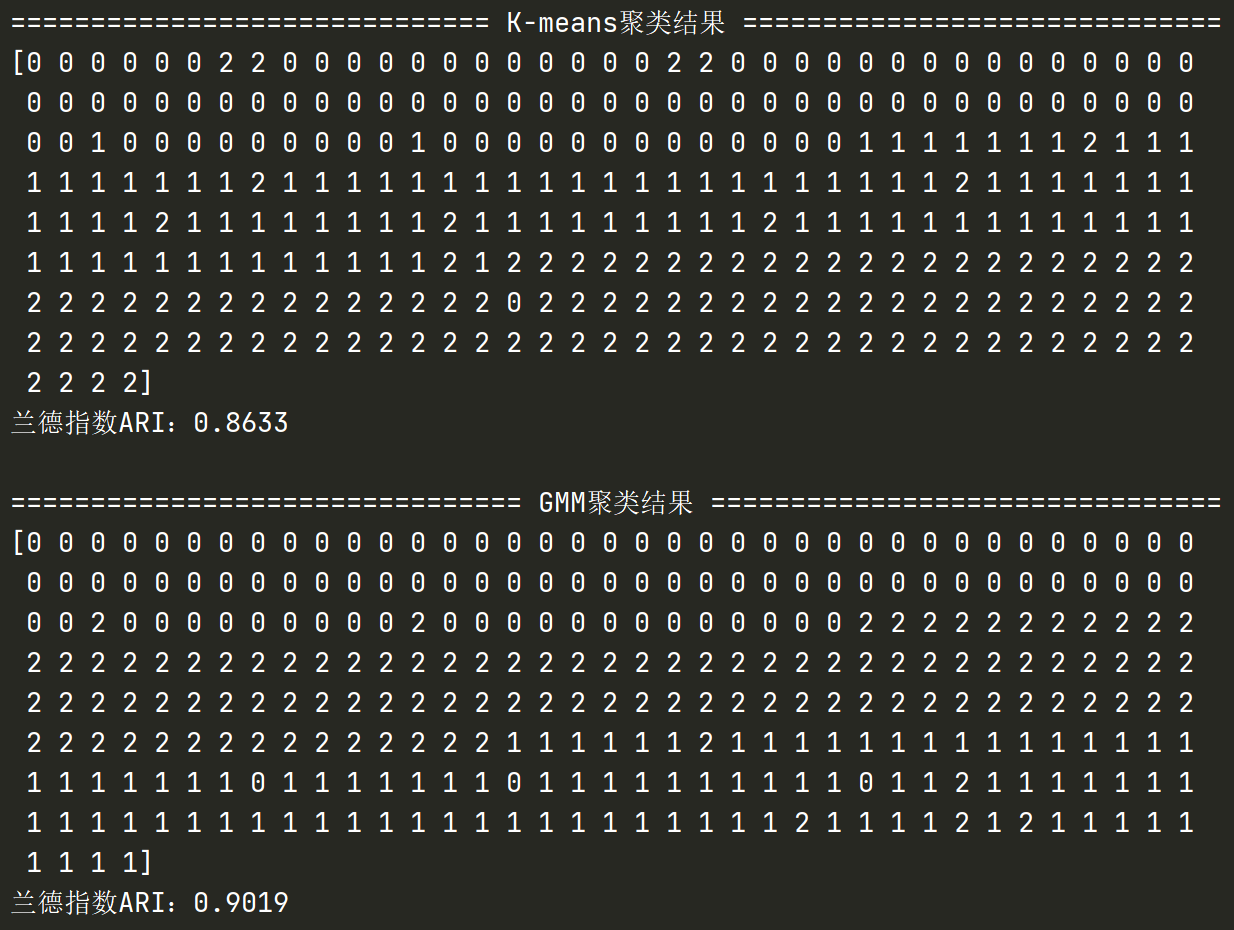
选择UCI数据库中的鸢尾花数据集进行测试验证，该数据集有4个特征、3个类别，可用于聚类算法。使用Pandas库加载本地的鸢尾花数据集，并转换为Numpy数组格式，使用GMM模型进行测试，输出聚类结果和RI，并选择前两个特征进行绘图。



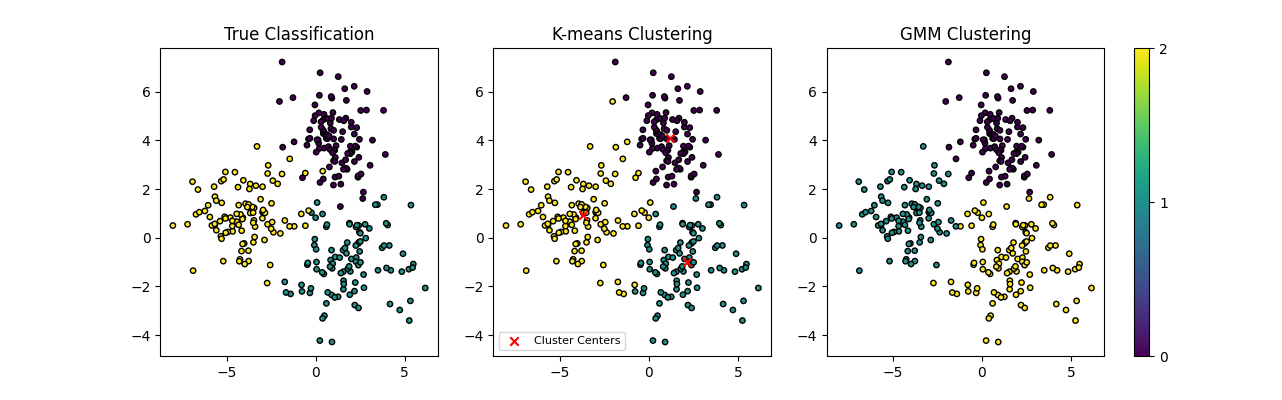
**4.3实验结果及分析**

**4.3.1手动生成的K组高斯分布数据**

聚类结果及RI指标如下所示，其中K-means聚类结果RI为0.8633，GMM聚类结果RI为0.9019。

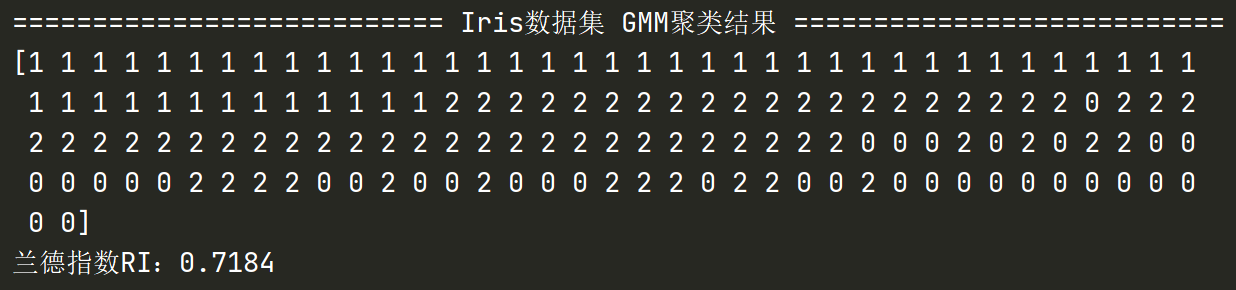


绘制样本点图像如下所示，注：聚类结果中的不同颜色仅用于区分不同簇，并不表示类别标签，与真实类别的颜色无一一对应关系。

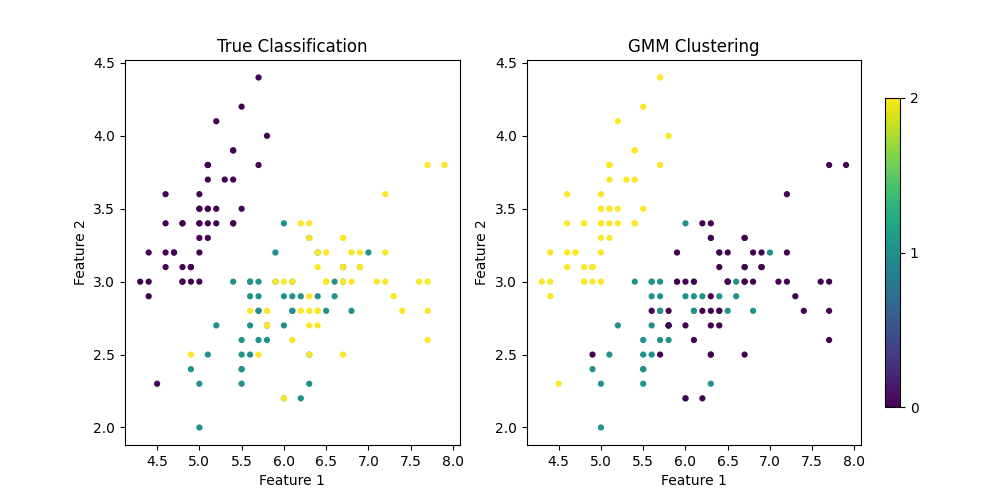


**4.3.2 UCI鸢尾花数据集**

聚类结果及RI指标如下所示，聚类结果RI为0.7184。



绘制样本点图像（前两个特征）如下所示，注：聚类结果中的不同颜色仅用于区分不同簇，并不表示类别标签，与真实类别的颜色无一一对应关系。



**五、实验总体结论**

K-means算法和高斯混合模型均可用于数据集聚类，K-means通过计算样本点和聚类中心之间的距离将样本点归类，而高斯混合模型输出每个样本点属于各个簇的概率值，取最大值对应簇进行归类。

此次实验使用K-means算法和基于E-M算法估计参数的高斯混合模型，对手动生成的K组高斯分布数据和UCI数据库中的鸢尾花数据集进行测试验证，使用兰德指数作为评价指标，达到了较好的聚类效果。

**六、完整实验代码**

1. import numpy as np
2. import pandas as pd
3. from scipy.stats import multivariate\_normal
4. from sklearn.metrics import adjusted\_rand\_score
5. import matplotlib.pyplot as plt
6. np.random.seed(69)
7. *# 生成k组高斯分布数据*
8. def generate\_data(K, n\_samples, mean=None, var=None):
9. if mean is None:
10. mean = np.random.randint(-5, 5, K \* 2).reshape(K, 2)
11. if var is None:
12. var = np.random.uniform(1, 2, K \* 2).reshape(K, 2)
13. data = [[], []]
14. for i in range(K):
15. data[0].extend(np.random.normal(mean[i, 0], var[i, 0], n\_samples))
16. data[1].extend(np.random.normal(mean[i, 1], var[i, 1], n\_samples))
17. return np.array(data).T
18. def Kmeans(K, data):
19. n\_samples, n\_features = data.shape
20. result = [[] for \_ in range(K)] *# 聚类结果*
21. labels = np.zeros(n\_samples, dtype=int)     *# 各样本聚类标签*
22. centers = np.zeros((K, n\_features))         *# 当前聚类中心*
23. centers\_last = np.zeros((K, n\_features))    *# 上一次的聚类中心*
24. *# 初始化聚类中心*
25. sample\_size = n\_samples // K   *# 将数据等距分成K份*
26. center\_ini = sample\_size // 2
27. for i in range(K):
28. centers[i, :] = data[center\_ini, :]
29. center\_ini += sample\_size
30. while centers.all() != centers\_last.all():
31. *# 计算样本所属簇*
32. for i in range(n\_samples):
33. dist = []
34. for j in range(K):
35. dist.append(np.linalg.norm(data[i] - centers[j]))
36. labels[i] = np.argmin(dist)
37. result[labels[i]].append(i)
38. *# 更新聚类中心*
39. centers\_last = centers.copy()
40. for i in range(K):
41. centers[i, :] = np.mean(data[result[i], :], axis=0)
42. return labels, centers
43. class GMM\_EM:
44. def \_\_init\_\_(self, K, max\_iter=100, tol=1e-6):
45. self.K = K
46. self.max\_iter = max\_iter
47. self.tol = tol
48. *# 模型迭代*
49. def fit(self, X):
50. *# 初始化模型参数*
51. n\_samples, n\_features = X.shape
52. self.alphas = np.ones(self.K) / self.K
53. self.mus = np.array([X[i] for i in np.random.choice(n\_samples, self.K, replace=False)])
54. self.sigmas = np.array([np.cov(X.T) for \_ in range(self.K)])
55. self.likelihood\_history = []
56. *# 迭代过程*
57. for iter in range(self.max\_iter):
58. *# E-step: 计算每个数据点属于每个高斯分布的后验概率*
59. responsiveness = self.\_e\_step(X)
60. *# M-step: 更新模型参数*
61. self.\_m\_step(X, responsiveness)
62. *# 计算对数似然函数*
63. likelihood = self.log\_likelihood(X)
64. self.likelihood\_history.append(likelihood)
65. *# print(f"Iter {iter + 1}, Likelihood: {likelihood}")*
66. *# 判断是否收敛*
67. if iter > 0 and abs(self.likelihood\_history[-1] - self.likelihood\_history[-2]) < self.tol:
68. break
69. return self
70. *# E-Step：计算每个数据点属于每个高斯分布的后验概率*
71. def \_e\_step(self, X):
72. responsiveness = np.zeros((X.shape[0], self.K))
73. for i in range(self.K):
74. rv = multivariate\_normal(mean=self.mus[i], cov=self.sigmas[i])
75. responsiveness[:, i] = self.alphas[i] \* rv.pdf(X)
76. responsiveness /= responsiveness.sum(axis=1, keepdims=True)
77. return responsiveness
78. *# M-Step：更新模型参数*
79. def \_m\_step(self, X, responsiveness):
80. n\_samples = X.shape[0]
81. for i in range(self.K):
82. n\_k = responsiveness[:, i].sum()
83. *# 计算加权均值*
84. self.mus[i] = np.sum(responsiveness[:, i][:, np.newaxis] \* X, axis=0) / n\_k
85. *# 计算加权协方差*
86. diff = X - self.mus[i]
87. self.sigmas[i] = np.dot((responsiveness[:, i][:, np.newaxis] \* diff).T, diff) / n\_k
88. *# 计算加权权重*
89. self.alphas[i] = responsiveness[:, i].sum() / n\_samples
90. *# 对数似然函数*
91. def log\_likelihood(self, X):
92. likelihood = 0
93. for i in range(self.K):
94. rv = multivariate\_normal(mean=self.mus[i], cov=self.sigmas[i])
95. likelihood += self.alphas[i] \* rv.pdf(X)
96. return np.sum(np.log(likelihood))
97. if \_\_name\_\_ == '\_\_main\_\_':
98. K = 3
99. n\_samples = 100
100. *# 生成k组高斯分布数据*
101. data = generate\_data(K, n\_samples)
102. true\_labels = [0] \* n\_samples + [1] \* n\_samples + [2] \* n\_samples
103. *# print(data.shape)*
104. *# K-means聚类算法*
105. kmeans\_labels, cluster\_centers = Kmeans(K, data)
106. RI = adjusted\_rand\_score(true\_labels, kmeans\_labels)
107. print("=" \* 30 + " K-means聚类结果 " + "=" \* 30)
108. print(kmeans\_labels)
109. print(f"兰德指数ARI：{RI:.4f}\n")
110. *# 高斯混合模型+EM算法*
111. gmm\_em = GMM\_EM(K)
112. gmm\_em.fit(data)
113. gmm\_labels = np.argmax(gmm\_em.\_e\_step(data), axis=1)    *# 获取每个点的聚类标签*
114. RI = adjusted\_rand\_score(true\_labels, gmm\_labels)
115. print("=" \* 32 + " GMM聚类结果 " + "=" \* 32)
116. print(gmm\_labels)
117. print(f"兰德指数ARI：{RI:.4f}\n")
118. *# 绘图*
119. cmap = plt.get\_cmap("viridis")
120. fig1, axs = plt.subplots(1, 3, figsize=(20, 4))
121. scatter1 = axs[0].scatter(data[:, 0], data[:, 1], c=true\_labels, s=15, edgecolor='black')
122. axs[0].set\_title("True Classification")
123. scatter2 = axs[1].scatter(data[:, 0], data[:, 1], c=kmeans\_labels, s=15, edgecolor='black')
124. axs[1].scatter(cluster\_centers[:, 0], cluster\_centers[:, 1], c='red', marker='x', label='Cluster Centers')
125. axs[1].legend(fontsize=8, loc='lower left')
126. axs[1].set\_title("K-means Clustering")
127. scatter3 = axs[2].scatter(data[:, 0], data[:, 1], c=gmm\_labels, s=15, edgecolor='black')
128. axs[2].set\_title("GMM Clustering")
129. fig1.colorbar(scatter1, ax=axs, fraction=0.02, pad=0.03, ticks=[0, 1, 2, 3])
130. *# UCI鸢尾花数据集*
131. iris = pd.read\_csv('./iris.csv').drop(['Id', 'Species'], axis=1).to\_numpy()
132. true\_labels = [0] \* 50 + [1] \* 50 + [2] \* 50
133. gmm\_em = GMM\_EM(K=3)
134. gmm\_em.fit(iris)
135. gmm\_labels = np.argmax(gmm\_em.\_e\_step(iris), axis=1)    *# 获取每个点的聚类标签*
136. RI = adjusted\_rand\_score(true\_labels, gmm\_labels)
137. print("=" \* 27 + " Iris数据集 GMM聚类结果 " + "=" \* 27)
138. print(gmm\_labels)
139. print(f"兰德指数RI：{RI:.4f}")
140. *# 绘图*
141. fig2, axs = plt.subplots(1, 2, figsize=(10, 5))
142. scatter4 = axs[0].scatter(iris[:, 0], iris[:, 1], c=true\_labels, s=12)
143. axs[0].set\_title("True Classification")
144. axs[0].set\_xlabel("Feature 1")
145. axs[0].set\_ylabel("Feature 2")
146. scatter5 = axs[1].scatter(iris[:, 0], iris[:, 1], c=gmm\_labels, s=12)
147. axs[1].set\_title("GMM Clustering")
148. axs[1].set\_xlabel("Feature 1")
149. axs[1].set\_ylabel("Feature 2")
150. fig2.colorbar(scatter4, ax=axs, fraction=0.02, pad=0.03, ticks=[0, 1, 2, 3])
151. plt.show()

**七、参考文献**

[1] 刘远超. 深度学习基础: 高等教育出版社, 2023.