《模式识别与机器学习A》实验报告

实验题目： k-means聚类方法和混合高斯模型

学 号： 2021112845

姓 名： 张智雄

**实验报告内容**

1. **实验目的**

实现一个K-means算法和混合高斯模型，并用EM算法估计模型中的参数。

1. **实验内容**

用高斯分布产生个高斯分布的数据（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）。

可以UCI上找一个简单问题数据，用你实现的GMM进行聚类。

1. **实验环境**

Windows11; Anaconda+python3.11; VS Code

1. **实验过程、结果及分析（包括代码截图、运行结果截图及必要的理论支撑等）**

**4.1 算法理论支撑**

4.1.1 K-means聚类算法

K-means聚类算法的核心思想为假定聚类内部点之间的距离应该小于数据点与聚类外部的点之间的距离。即使得**每个数据点和与它最近的中心之间距离的平方和最小**。

假设数据集为，我们的目标是将数据集划分为个类别。令表示各类别的中心。聚类问题等价于求概率分布：

K-means聚类相当于假设服从多元高斯分布（**特征之间相互独立，协方差矩阵**），且为等概率均匀分布。而为已知数据分布，从似然的角度看，极大化即等价于极大化，即最小化各数据点到其类别的均值。

引入二值指示变量表示数据点的分类情况，则可定义目标函数为

因此最优化过程可以划分为两步：

1. 固定，优化。由于关于是线性的，因此可以对每个分别进行最小化，即对根据与聚类中心的距离进行最优化：
2. 固定，优化。由于是的二次函数，对其求导等于零得

即等于类别k的所有数据点的均值。

4.1.2 混合高斯模型

图片包含 形状

描述已自动生成任意连续概率密度都能用多个高斯分布的线性组合叠加的高斯混合概率分布来描述。引入编码的二值随机变量，满足且。

由右图模型定义联合概率分布，的边缘**先验**概率分布设为，也可写作。

那么，的条件概率分布为：

于是可以给出，同时，的条件后验概率由贝叶斯定理得（已知为，类别为的概率）：

于是此聚类过程可以看做将概率分布解耦成个高斯分布，对应个类别。对于数据集，对应隐变量表示为。

则对数似然函数为

将此似然函数关于求导（假设非奇异），令为能被分配到聚类的有效数量，可以得到：

由此式可视为当前所有点数据为第类的概率加权平均。

同样地，将此函数关于求导等于0可以得到：

最后使用拉格朗日乘子法关于优化 （需要满足和为1的条件）得到：

使用EM算法优化即可总结为以下步骤：

**ALGORITHM 1** EM for Gaussian Mixture Models

1：**input** 数据集，类别数目，迭代次数；

2：初始化均值、协方差和混合系数

3：计算对数似然

3：**while** **do**

4： ； （E步）

5： ；

6: ； （M步）

7： ；

8：**end while**

9：**return** //返回最优参数；

**4.2 实验设计**

4.2.1 随机数据生成

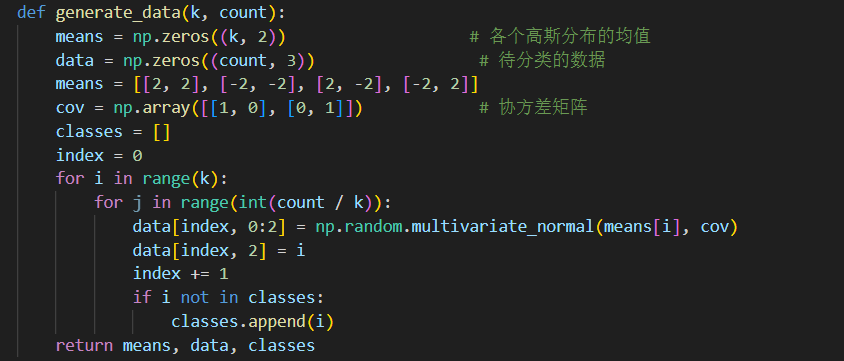


图1 随机数据生成代码截图

如上图代码，使用np.random.multivariate\_normal方法按给定的协方差矩阵和均值按多元高斯分布初始化个类别的数据点。

4.2.2 K-means聚类

首先初始化个类的中心，这里采取的是从数据集中随机选取个样本作为初始的类中心。

文本

描述已自动生成

图2 K-means初始化代码截图

而后是更新中心的算法，主要是分为两步：

1. 通过计算每个样本到个中心的距离（欧式距离），然后选取最小的距离对应的那个聚类中心作为样本标签，将该样本划分到这个类中，
2. 根据更新后的类别计算类内均值，作为新的中心。

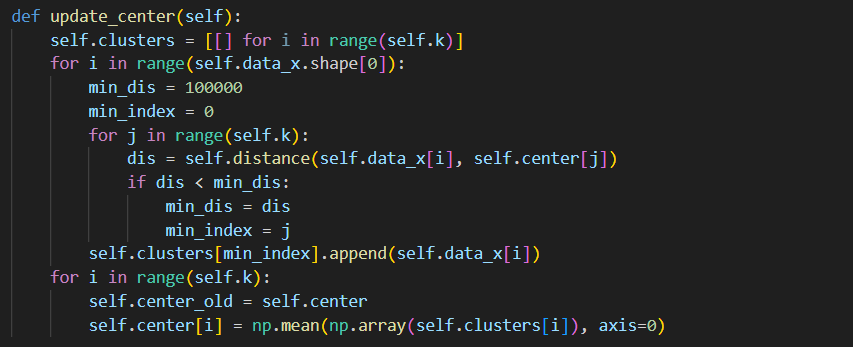


图3 更新中心代码截图

重复上述过程，直至中心更新距离较之上次变化较小时退出迭代。

文本

描述已自动生成

图4 迭代代码截图

4.2.3 混合高斯模型GMM

首先初始化个类的均值、协方差和混合系数，可以有随机生成和采用K-means聚类的结果两种方式进行初始化。

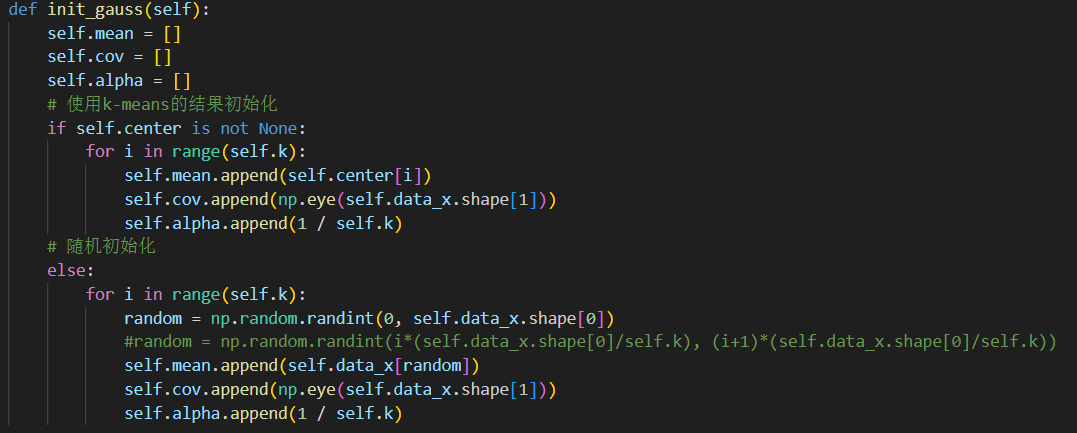


图5 GMM初始化代码截图

根据对数似然计算公式以及 关于、和的导数进行EM更新：

1. E步：计算，即各数据点的类别概率；
2. M步：计算新的均值

电脑萤幕画面

描述已自动生成

图6 EM更新代码截图

重复上述过程，直至似然函数较之上次变化较小时退出迭代。

文本

描述已自动生成

图7 迭代代码截图

**4.3 实验结果及分析**

4.3.1 自己生成数据结果比较

设置均值为，协方差矩阵均为，生成1000组数据，而后先后使用K-means和GMM进行聚类，聚类结果如下：

K-means正确率为96.7%，GMM正确率为96.9%（在使用K-means的结果作为GMM初始化的条件下）。

图表, 散点图

描述已自动生成

图8 聚类结果可视化（）

同时，经过反复实验发现，K-means和GMM的聚类结果严重依赖于初始化的结果，当初始化中心（或均值）选取区分度不高时，结果往往会比较差。

而GMM以K-means的结果初始化，能够一定程度上优化K-means的聚类结果（特别是当分类结果不好时）。

图表, 散点图

描述已自动生成

图 9 GMM在K-means基础上的优化（）

而特征之间不相互独立时，GMM聚类结果明显优于K-means。这是由于K-means假设数据呈球状分布，不能够区分开具有不相互独立的特征。

下图为均值为，协方差矩阵均为，生成1000组数据，先后使用K-means和GMM进行聚类的聚类结果如下：

此时K-means正确率为91.8%，GMM正确率为97.6%。

图表, 散点图

描述已自动生成

图10 特征不独立时的聚类结果（）

4.3.2 UCI-Iris数据集结果比较

而后使用Iris数据集进行聚类，K-means分类正确率为88%，数据如下：

表格1 K-means分类结果

|  |  |  |  |
| --- | --- | --- | --- |
|  | 类别1 | 类别2 | 类别3 |
| 均值 |  |  |  |

GMM的分类正确率为96.7%，各聚类的矩阵、协方差、混合系数数据如下：

表格2 GMM分类结果

|  |  |  |  |
| --- | --- | --- | --- |
|  | 类别1 | 类别2 | 类别3 |
| 均值 |  |  |  |
| 协方差矩阵 |  |  |  |
| 混合  系数 |  |  |  |

将上述聚类结果按第一个特征和第二个特征进行可视化结果如下：

图表, 散点图

描述已自动生成

图11 Iris聚类结果

1. **实验结论**

K-means和GMM都是EM算法的体现。两者共同之处都有隐变量，遵循EM算法的E步和M步的迭代优化。

K-means其实就是一种特殊的高斯混合模型，提出了强假设，假设每一类在样本数据中出现的概率相等均为，每个特征间相互独立，且满足变量间的协方差矩阵为单位对角阵。在此条件下，可以直接用欧氏距离作为K-means的协方差去衡量相似性。

而GMM用混合高斯模型来描述聚类结果，且假设多个高斯模型对总模型的贡献是有权重的，且样本属于某一类也是由概率的，从而相似度也由离散的变成了需要通过全概率公式计算的值，分类效果一般好于K-means。

实验发现，初值的选取对K-means和GMM的效果影响较大，初始化较差可能会导致陷入局部最优解而得不到较好的聚类结果。可以使用GMM对K-means的分类结果进行进一步优化。

1. **完整实验代码**

GMM.py

|  |
| --- |
| 1. import numpy as np 2. import matplotlib.pyplot as plt 3. import pandas as pd 4. import itertools 5. import random 6. def random\_seed(seed): 7. np.random.seed(seed) 8. random.seed(seed) 9. def iris(): 10. data = pd.read\_csv('./Iris.data') 11. data["Species"] = data["Species"].map({"Iris-setosa": 1, "Iris-versicolor": 0, "Iris-virginica": 2}) 12. data = np.array(data[:].values) 13. return data 14. def generate\_data(k, count): 15. means = np.zeros((k, 2))                 *# 各个高斯分布的均值* 16. data = np.zeros((count, 3))              *# 待分类的数据* 17. means = [[2, 2], [-2, -2], [2, -2], [-2, 2]] 18. cov = np.array([[1, 0.36], [0.36, 1]])   *# 协方差矩阵* 19. classes = [] 20. index = 0 21. for i in range(k): 22. for j in range(int(count / k)): 23. data[index, 0:2] = np.random.multivariate\_normal(means[i], cov) 24. data[index, 2] = i 25. index += 1 26. if i not in classes: 27. classes.append(i) 28. return means, data, classes 29. class K\_means: 30. def \_\_init\_\_(self, data, k): 31. self.k = k 32. self.data\_x = data[:, 0:-1] 33. self.data\_y = data[:, -1] 35. *# 初始化中心点* 36. def init\_center(self): 37. self.center = np.zeros((self.k, self.data\_x.shape[1])) 38. for i in range(self.k): 39. random = np.random.randint(0, self.data\_x.shape[0]) 40. *#random = np.random.randint(i\*(self.data\_x.shape[0]/self.k), (i+1)\*(self.data\_x.shape[0]/self.k))* 41. self.center[i] = (self.data\_x[random]) 43. def distance(self, x, y): 44. return np.sqrt(np.sum(np.square(x - y))) 46. def update\_center(self): 47. self.clusters = [[] for i in range(self.k)] 48. for i in range(self.data\_x.shape[0]): 49. min\_dis = 100000 50. min\_index = 0 51. for j in range(self.k): 52. dis = self.distance(self.data\_x[i], self.center[j]) 53. if dis < min\_dis: 54. min\_dis = dis 55. min\_index = j 56. self.clusters[min\_index].append(self.data\_x[i]) 57. for i in range(self.k): 58. self.center\_old = self.center 59. self.center[i] = np.mean(np.array(self.clusters[i]), axis=0) 61. def train(self, limit=100000): 62. self.init\_center() 63. self.update\_center() 64. for i in range(limit): 65. self.update\_center() 66. if np.sum(np.abs(self.center - self.center\_old)) < 0.1: 67. break 68. self.center\_old = self.center 69. return self.clusters, self.center 71. def accuary(self, classes): 72. classes = list(itertools.permutations(classes, len(classes))) 73. acc = [] 74. for cls in classes: 75. count = 0 76. for i in range(self.data\_x.shape[0]): 77. min\_dis = 100000 78. min\_index = 0 79. for j in range(self.k): 80. dis = self.distance(self.data\_x[i], self.center[j]) 81. if dis < min\_dis: 82. min\_dis = dis 83. min\_index = j 84. if cls[int(self.data\_y[i])] == min\_index: 85. count += 1 86. acc.append(count / self.data\_x.shape[0]) 87. *#print(acc)* 88. return max(acc) 89. def draw(self): 90. plt.figure() 91. for i in range(self.k): 92. plt.scatter(np.array(self.clusters[i])[:, 0], np.array(self.clusters[i])[:, 1]) 93. plt.show() 94. class GMM: 95. def \_\_init\_\_(self, data, k, center=None): 96. self.k = k 97. self.data\_x = data[:, 0:-1] 98. self.data\_y = data[:, -1] 99. self.center = center 100. def init\_gauss(self): 101. self.mean = [] 102. self.cov = [] 103. self.alpha = [] 104. *# 使用k-means的结果初始化* 105. if self.center is not None: 106. for i in range(self.k): 107. self.mean.append(self.center[i]) 108. self.cov.append(np.eye(self.data\_x.shape[1])) 109. self.alpha.append(1 / self.k) 110. *# 随机初始化* 111. else: 112. for i in range(self.k): 113. random = np.random.randint(0, self.data\_x.shape[0]) 114. *#random = np.random.randint(i\*(self.data\_x.shape[0]/self.k), (i+1)\*(self.data\_x.shape[0]/self.k))* 115. self.mean.append(self.data\_x[random]) 116. self.cov.append(np.eye(self.data\_x.shape[1])) 117. self.alpha.append(1 / self.k) 118. def gaussian(self, x, mean, cov): 119. return np.exp(-0.5 \* np.dot(np.dot((x - mean), np.linalg.inv(cov)), (x - mean).T)) / (np.sqrt(np.linalg.det(cov)) \* np.power(2 \* np.pi, self.data\_x.shape[1] / 2)) 121. def likelihood(self): 122. sum = 0.0 123. for k in range(self.k): 124. sum += self.alpha[k] \* self.gaussian(self.data\_x, self.mean[k], self.cov[k]) 125. return np.sum(np.log(sum)) 126. *#return np.sum(np.log(np.sum(self.alpha[k] \* self.gaussian(self.data\_x, self.mean[k], self.cov[k]) for k in range(self.k))))* 127. def E\_step(self): 128. self.gamma = np.zeros((self.data\_x.shape[0], self.k)) 129. for i in range(self.data\_x.shape[0]): 130. for j in range(self.k): 131. self.gamma[i, j] = self.alpha[j] \* self.gaussian(self.data\_x[i], self.mean[j], self.cov[j]) 132. self.gamma[i] /= np.sum(self.gamma[i]) 134. def M\_step(self): 135. for j in range(self.k): 136. Nk = np.sum(self.gamma[:, j]) 137. self.mean[j] = np.sum(self.gamma[:, j].reshape(-1, 1) \* self.data\_x, axis=0) / Nk 138. self.cov[j] = np.dot((self.data\_x - self.mean[j]).T, 139. (self.data\_x - self.mean[j]) \* self.gamma[:, j].reshape(-1, 1)) / Nk 140. self.alpha[j] = Nk / self.data\_x.shape[0] 142. def train(self, limit=100000): 143. self.init\_gauss() 144. likelihood = [] 145. likelihood.append(self.likelihood()) 146. for i in range(limit): 147. self.E\_step() 148. self.M\_step() 149. likelihood.append(self.likelihood()) 150. if np.abs(likelihood[i + 1] - likelihood[i]) < 0.1: 151. break 152. plt.figure() 153. plt.plot(likelihood) 154. plt.show() 155. return self.mean, self.cov, self.alpha 157. def accuary(self, classes): 158. z = np.argmax(self.gamma, axis=1) 159. classes = list(itertools.permutations(classes, len(classes))) 160. acc = [] 161. for cls in classes: 162. count = 0 163. for i in range(self.data\_x.shape[0]): 164. if z[i] == cls[int(self.data\_y[i])]: 165. count += 1 166. acc.append(count / self.data\_x.shape[0]) 167. *#print(acc)* 168. return max(acc) 170. def draw(self): 171. z = np.argmax(self.gamma, axis=1) 172. plt.figure() 173. for i in range(self.k): 174. plt.scatter(self.data\_x[z == i, 0], self.data\_x[z == i, 1]) 175. plt.show() 176. if \_\_name\_\_ == '\_\_main\_\_': 178. *# random\_seed(177)* 179. *# k = 4* 180. *# count = 1000* 181. *# means, data, classes = generate\_data(k, count)* 182. *# print(classes)* 183. *# k\_means = K\_means(data, k)* 184. *# clusters, center = k\_means.train()* 185. *# k\_means.draw()* 186. *# print("K-means正确率： ", k\_means.accuary(classes))* 187. *# print()* 188. *# gmm = GMM(data, k, center)* 189. *# mean, cov, alpha = gmm.train()* 190. *# print(mean, cov, alpha)* 191. *# gmm.draw()* 192. *# print("GMM正确率： ", gmm.accuary(classes))* 193. random\_seed(6) 194. data = iris() 195. k\_means = K\_means(data, 3) 196. clusters, center = k\_means.train() 197. k\_means.draw() 198. print("K-means正确率： ", k\_means.accuary([0, 1, 2])) 199. print(center) 200. gmm = GMM(data, 3, center) 201. mean, cov, alpha = gmm.train() 202. print(mean, cov, alpha) 203. gmm.draw() 204. print("GMM正确率： ", gmm.accuary([0, 1, 2])) |

1. **参考文献**

[1]李航. 统计学习方法[M]. 清华大学出版社, 2012.

[2]周志华. 机器学习[M]. 清华大学出版社, 2016.

[3]Bishop C .Pattern Recognition and Machine Learning[M]. 2006.