**《机器学习基础》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2022级计算机科学与技术（卓越）01** | | | **姓名** |  |
| **实验题目** | **聚类算法实践** | | | | | |
| **实验时间** | **2024/05/28** | | **实验地点** | **DS3401** | | |
| **实验成绩** |  | | **实验性质** | **□验证性 □设计性 □综合性** | | |
| 教师评价：  **□**算法/实验过程正确； **□**源程序/实验内容提交 **□**程序结构/实验步骤合理；  **□**实验结果正确； **□**语法、语义正确； **□**报告规范；  其他：  评价教师签名： | | | | | | |
| 一、实验目的  掌握聚类算法原理。 | | | | | | |
| 二、实验项目内容  1.理解并**描述**各一种原型聚类算法、密度聚类算法的原理。  2.**编程**实践，将k均值算法应用于合适的数据集（西瓜数据集、鸢尾花数据集或其它合适数据集），设置三组以上不同的k值，分别使用三组不同初始中心点，对比实验结果，分析聚类结果的优劣。 | | | | | | |
| 1. 实验过程或算法（源程序）   原型聚类算法：K-Means  原理：  1. 初始化：  - 随机选择K个初始聚类中心（称为质心）。这些质心可以从数据集中随机选择，也可以使用一些启发式方法如K-Means++。  2. 分配：  - 对于数据集中的每一个数据点，计算它到所有质心的距离（通常使用欧几里得距离），将该数据点分配给最近的质心。这样，数据集被分K个簇。  3.更新：  - 重新计算每个簇的质心，质心是簇中所有数据点的平均值。  4. 迭代：  - 重复步骤2和3，直到质心不再变化或者变化量在预设阈值以内。  优点：  - 算法简单，易于实现和理解。  - 对于大数据集，算法的收敛速度较快。  缺点：  - 需要预先指定簇的数量K。  - 对初始质心敏感，可能会收敛到局部最优解。  - 适用于球状簇，对于非球状或复杂形状的簇效果不佳。  - 对噪声和离群点敏感。  密度聚类算法：DBSCAN (Density-Based Spatial Clustering of Applications with Noise)  原理：  1. 参数设定：  - 设定两个参数：半径 和 最小点数。  2.核心点、边界点和噪声点：  - 核心点：在半径内包含至少 MinPts 个数据点。  - 边界点：在半径内包含的数据点数少于MinPts，但在核心点的邻域内。  - 噪声点：既不是核心点也不是边界点的点。  3.聚类构建：  - 任意选择一个未被访问的点，如果它是核心点，从该点出发将所有密度可达的点标记为同一个簇。密度可达性指从一个核心点出发，通过连续的核心点可以达到其他点。  4. 重复：  - 重复上述过程，直到所有点都被访问和标记。  优点：  - 不需要预先指定簇的数量。  - 能够发现任意形状的簇。  - 对噪声和离群点有很好的处理能力。  缺点：  - 对参数 \( \epsilon \) 和 MinPts 较为敏感，不同的数据集需要不同的参数设定。  - 对于高维数据，计算密度邻域的复杂度较高，可能导致计算开销较大。  西瓜数据集聚类代码：   1. import numpy as np 2. import pandas as pd 3. import matplotlib.pyplot as plt 4. *# 功能: 设置随机种子, 确保结果可复现* 5. def make\_seed(SEED=42): 6. np.random.seed(SEED) 7. *#计算样本与聚类中心的距离，返回离簇中心最近的类别* 8. def distance(sample,centers): 9. d=np.power(sample-centers,2).sum(axis=1) 10. cls=d.argmin() 11. return cls 12. def clusters\_show(clusters,step): 13. color=["red","blue","pink"] 14. marker=["\*","^","."] 15. plt.figure(figsize=(8,8)) 16. plt.title("step:{}".format(step)) 17. plt.xlabel("Density",loc="center") 18. plt.ylabel("Sugar Content",loc="center") 19. *# 用颜色区分k个簇的数据样本* 20. for i,cluster in enumerate(clusters): 21. cluster=np.array(cluster) 22. plt.scatter(cluster[:,0],cluster[:,1],c=color[i],marker=marker[i],s=150) 23. plt.show() 24. *#k均值算法* 25. def k\_means(samples,k): 26. data\_number=len(samples) 27. centers\_flag=np.zeros((k,)) 28. *#随机在数据中选择k个聚类中心* 29. centers=samples[np.random.choice(data\_number,k,replace=False)] 30. print(centers) 31. step=1 32. while True: 33. *#计算每个样本中心距离簇中心的距离，然后分到距离最短的簇中心* 34. clusters=[[] for i in range(k)] 35. for sample in samples: 36. ci=distance(sample,centers) 37. clusters[ci].append(sample) 38. *#可视化当前聚类结构* 39. clusters\_show(clusters,step) 40. *#更新每个簇的中心点，得到簇中心进行下一步聚类* 41. for i,sub\_clusters in enumerate(clusters): 42. new\_center=np.array(sub\_clusters).mean(axis=0) 43. *# 如果数值有变化则更新, 如果没有变化则设置标志位为1，当所有的标志位为1则退出循环* 44. if(centers[i]!=new\_center).all(): 45. centers[i]=new\_center 46. else: 47. centers\_flag[i]=1 48. step+=1 49. print("step:{}".format(step),"\n","centers:{}".format(centers)) 50. if centers\_flag.all(): 51. break 52. return centers 53. *#根据簇中心对簇进行分类，获取最后的分类结果* 54. def split\_data(samples, centers): 55. *# 根据中心样本得知簇数* 56. k = len(centers) 57. clusters = [[] for i in range(k)] 58. for sample in samples: 59. ci = distance(sample, centers) 60. clusters[ci].append(sample) 61. return clusters 62. if \_\_name\_\_ == '\_\_main\_\_': 63. make\_seed() 64. *# 导入数据* 65. data = pd.read\_csv(r"西瓜数据集4.0.csv",encoding='GBK') 66. samples = data[["密度", "含糖率"]].values 67. *# print(samples)* 68. centers = k\_means(samples=samples, k=3) 69. clusters = split\_data(samples=samples, centers=centers) 70. print(clusters)   鸢尾花数据集聚类代码：   1. import numpy as np 2. import pandas as pd 3. import matplotlib.pyplot as plt 4. from mpl\_toolkits.mplot3d import Axes3D 5. from sklearn.preprocessing import LabelEncoder 6. *# 设置随机种子* 7. def make\_seed(SEED=42): 8. np.random.seed(SEED) 9. *# 计算样本与聚类中心的距离* 10. def distance(sample, centers): 11. d = np.power(sample - centers, 2).sum(axis=1) 12. cls = d.argmin() 13. return cls 14. *# 可视化聚类结果* 15. def clusters\_show\_3d(clusters, step): 16. color = ["red", "blue", "pink"] 17. marker = ["\*", "^", "."] 18. fig = plt.figure(figsize=(8, 8)) 19. ax = fig.add\_subplot(111, projection='3d') 20. ax.set\_title("step:{}".format(step)) 21. ax.set\_xlabel("Sepal.Length") 22. ax.set\_ylabel("Sepal.Width") 23. ax.set\_zlabel("Petal.Length") 24. for i, cluster in enumerate(clusters): 25. cluster = np.array(cluster) 26. ax.scatter(cluster[:, 0], cluster[:, 1], cluster[:, 2], c=color[i], marker=marker[i], s=150) 27. plt.show() 28. *# K-means 算法* 29. def k\_means(samples, k): 30. centers = samples[np.random.choice(len(samples), k, replace=False)] 31. centers\_flag = np.zeros((k,)) 32. step = 1 33. while True: 34. clusters = [[] for \_ in range(k)] 35. for idx, sample in enumerate(samples): 36. ci = distance(sample, centers) 37. clusters[ci].append(idx)  *# 存储样本的索引* 38. *# 可视化时使用样本数据* 39. visual\_clusters = [[samples[idx] for idx in cluster] for cluster in clusters] 40. clusters\_show\_3d(visual\_clusters, step) 41. for i, sub\_clusters in enumerate(clusters): 42. new\_center = np.array([samples[idx] for idx in sub\_clusters]).mean(axis=0) 43. if not np.array\_equal(centers[i], new\_center): 44. centers[i] = new\_center 45. else: 46. centers\_flag[i] = 1 47. step += 1 48. print("step:{}".format(step), "\n", "centers:{}".format(centers)) 49. if centers\_flag.all(): 50. break 51. return centers, clusters 52. *# 数据分裂* 53. def split\_data(samples, centers): 54. k = len(centers) 55. clusters = [[] for \_ in range(k)] 56. for idx, sample in enumerate(samples): 57. ci = distance(sample, centers) 58. clusters[ci].append(idx)  *# 存储样本的索引* 59. return clusters 60. if \_\_name\_\_ == '\_\_main\_\_': 61. make\_seed() 62. *# 导入数据* 63. iris\_data = pd.read\_csv("iris.csv") 64. *# 提取特征和标签* 65. features = iris\_data[['Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width']].values 66. labels = iris\_data['Species'].values 67. *# 将标签编码为数字* 68. label\_encoder = LabelEncoder() 69. labels\_encoded = label\_encoder.fit\_transform(labels) 70. *# 执行 K-means 聚类* 71. centers, clusters = k\_means(samples=features, k=3) 72. *# 打印每个簇的类别* 73. for i, cluster in enumerate(clusters): 74. cluster\_labels = labels[cluster] 75. print(f"Cluster {i+1} labels: {cluster\_labels}") | | | | | | |
| 四、实验结果及分析  IMG_256  图一：西瓜数据集结果可视化  IMG_256  图二：鸢尾花数据集聚类可视化  由上方可视化图像可以得知，我们成功的实现了无监督学习的聚类功能，无论是西瓜集还是鸢尾花集我们都将其分成了三类。但是由于我们是最基础的K-means算法，对于鸢尾花数据集中我们无法将分类出的簇和其种类标签完全对应，这一方面可能有以下原因：   1. k-means假设数据集是球形的，并且各类具有相似的方差。这意味着算法倾向于寻找大小相似且形状规则的簇。然而，实际数据集（如鸢尾花数据集）中的簇可能不是球形的，且各类簇的大小和形状可能不同。 2. 均值点的选择：k-means通过最小化点到质心的距离进行分配，因此在面对形状不规则或分布不均匀的数据时，效果会较差。   IMG_256  如上图所示：簇1中无法完全的将versicolor类别的样本完全聚类，会有virginica类别的出现。  当我们修改初始随机中心点的时候的时候，最终分类得到的簇可能会改变。  分析聚类结果的优劣：  轮廓系数（Silhouette Score）：该值越高，聚类效果越好。值介于-1和1之间，接近1表示簇内数据点更紧密，簇间数据点更分离。  簇内误差平方和（Inertia）：该值越低，表示簇内数据点距离其簇中心越近。  最终将这两个评价指标应用在不同的K值（K=2、3、4）上，得到当K值为3时，得到的分类效果最好。K=4时，会导致聚类结果的细粒度增加，最终导致结果过拟合。K=2时，对鸢尾花数据集而言，因为其本身就含有三个标签，使用K=2聚类并不能起到什么实质性作用。 | | | | | | |