**《人工智能》实验报告**

实验名称 聚类算法研究

学生姓名 黄鸿展

学　　　　号 202230441138

学生班级 计算机科学与技术2班

实验日期 2024年12月10日

实验成绩

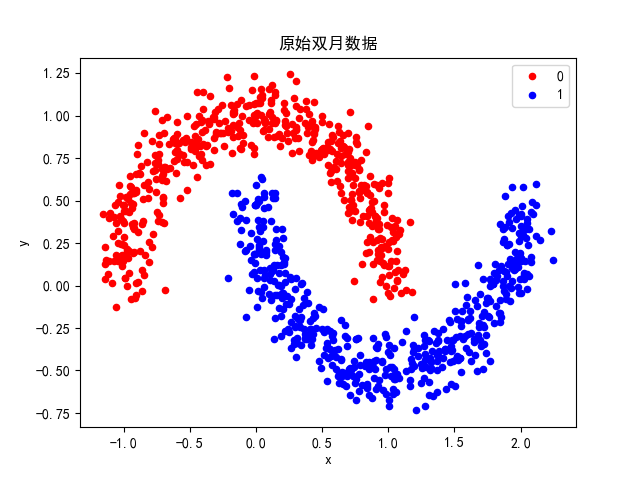
一、实验内容

本实验旨在通过实现k-means算法和DBSCAN算法来对双月数据集进行聚类，并分析展示两种算法的聚类效果。双月数据集是一个二维分类数据集，通过make\_moons函数生成，包含1000个样本点，每个样本点有两个特征，且噪声水平为0.1。

而k-means算法是一种划分方法，旨在将数据集分成K个簇，使得簇内样本之间的距离尽可能小，簇间样本的距离尽可能大。DBSCAN（Density-Based Spatial Clustering of Applications with Noise）算法是一种基于密度的聚类方法，能够识别出任意形状的簇，并且能够处理噪声点。

二、实验过程

1．数据集生成

使用sklearn库中的make\_moons函数生成双月数据集，参数设置为n\_samples=1000，shuffle=True，noise=0.1，random\_state=10，以确保结果的可重复性。最后生成的图如下所示：  


2. 算法思想

k-means算法：

k-means算法是一种迭代的聚类算法，其目标是将n个样本划分为k个簇，使得簇内的样本尽可能相似，而簇间的样本尽可能不同。算法的基本步骤如下：

1. 初始化：在数据集中随机选择k个样本点作为初始的质心。
2. 分配：对于数据集中的每个点，计算它到每个质心的距离，并将其分配给最近的质心，从而形成k个簇。
3. 更新：对于每个簇，计算簇中所有点的均值，将该均值设置为新的质心。
4. 迭代：重复步骤2和3，直到质心的位置不再发生变化，或达到预设的最大迭代次数。
5. 终止：当满足终止条件时结束，此时簇的质心和簇的分配是最终的聚类。

k-means算法简单、易于实现，对大数据集效果较好，但是它假设簇是凸形的，且大小相似，这在现实世界的数据中不总是成立。因此对于混杂在一起的图形分类效果不佳。

DBSCAN算法：

DBSCAN（Density-Based Spatial Clustering of Applications with Noise）算法是一种基于密度的聚类算法，它可以发现任意形状的簇，并且能够识别出噪声点。算法的基本步骤如下：

1. 参数设置：选择两个参数，eps（邻域半径）和MinPts（邻域最小点数）。
2. 核心点识别：对于每个点，计算其eps-邻域内包含的点数。如果一个点的邻域内至少有MinPts个点，则称该点为核心点。
3. 簇的生成：对于每个核心点，如果它尚未被访问，则以该点为起点，递归地将所有可达的密度可达的点（即在核心点的eps-邻域内的核心点）加入到同一个簇中。
4. 处理边界点和噪声：非核心点（邻域内点数少于MinPts的点）可能被分配到簇中（如果它们在某个核心点的eps-邻域内），或被标记为噪声。
5. 迭代：重复步骤3和4，直到所有点都被访问。
6. 终止：当所有点都被访问并且分类后结束，此时每个簇和噪声点被认为是最终的聚类结果。

DBSCAN算法不需要预先指定簇的数量，可以识别出不同形状和大小的簇，并且对噪声具有良好的鲁棒性。然而，选择合适的eps和MinPts参数可能比较困难，且算法的性能对这些参数非常敏感，因此需要考虑正确的获得方法。

3. 算法设计

首先定义距离：

对于两个点 p和 q在 n维空间中的坐标分别为 (p1,p2,...,pn)(p1​,p2​,...,pn​) 和 (q1,q2,...,qn)(q1​,q2​,...,qn​)，它们之间的欧氏距离 d计算公式为：

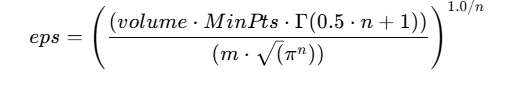


在代码中，这个计算是通过np.linalg.norm(data[i] - data[j])实现的，它计算了两个向量（点）之间的2-范数（即欧氏距离）。

eps计算

在DBSCAN算法中，eps（ε）是一个重要的参数，它定义了点的邻域范围。在代码中，eps的计算是通过epsilon函数实现的，该函数基于数据集和MinPts参数来动态计算eps值。eps的计算方法如下：

1. 计算边界框体积： 首先，计算数据集的边界框体积，即数据集在每个维度上的最大值和最小值之差的产品。这可以通过np.max(data, axis=0)和np.min(data, axis=0)获得。
2. 应用DBSCAN的eps公式： eps的计算公式为：

**

其中，m是数据点的数量，n是数据的维度，volume是边界框体积，Gamma是伽玛函数，用于计算阶乘的推广。这个公式是基于密度的聚类算法中常用的eps估计方法，它考虑了数据的分布和期望的邻域内点的数量。

kmeans函数实现了kmeans算法的主要逻辑：

1. 初始化质心（kmeans）：

从数据集中随机选择k个样本点作为初始的质心。

1. 分配样本到最近的质心：

对于数据集中的每个点，计算它到每个质心的距离，并将其分配给最近的质心，从而形成k个簇。

1. 更新质心：

对于每个簇，计算簇中所有点的均值，并将该均值设置为新的质心。

1. 迭代过程：

重复步骤2和3，直到质心的位置不再变化或达到最大迭代次数。

1. 返回结果：

返回每个点的簇标签。

dbscan函数实现了dbscan算法的主要逻辑：

1. 计算ε（eps）：

根据数据集和MinPts参数，使用公式计算合适的eps值。

1. 计算距离矩阵：

计算数据集中所有点之间的欧氏距离，并将结果存储在一个矩阵中。

1. 寻找ε-邻域内的点（find\_eps）：

对于每个点，找出其ε-邻域内的所有点。

1. DBSCAN聚类（dbscan）：

初始化标签数组labels，所有值设为-1（表示噪声）。

遍历所有点，对于未访问的点：

如果该点的ε-邻域内点的数量小于MinPts，则将其标记为噪声。

如果该点的ε-邻域内点的数量大于或等于MinPts，则将其标记为一个核心点，并递归地将所有密度可达的点加入到同一个簇中。

返回每个点的簇标签。

3. 具体实现

DBSCAN 聚类

epsilon：计算DBSCAN算法中的邻域半径ε（eps）值。

distance：构建数据集中所有点之间的欧氏距离矩阵。

find\_eps：找出距离小于等于ε的所有点的索引。

dbscan：执行DBSCAN聚类算法，返回每个点的聚类标签。

KMeans 聚类

kmeans：执行KMeans聚类算法，返回每个点的聚类标签。

数据处理和可视化

数据生成：使用make\_moons函数生成双月形状的数据集。

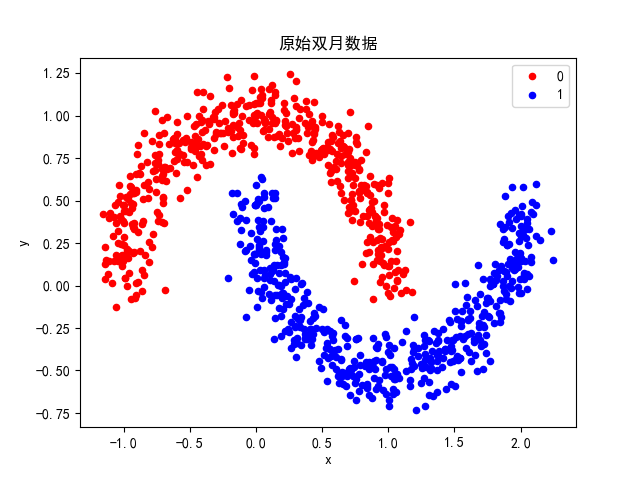
数据可视化：用matplotlib绘制原始数据的散点图，不同类别用不同颜色表示。

聚类参数设置：设置DBSCAN的MinPts参数和计算eps值。

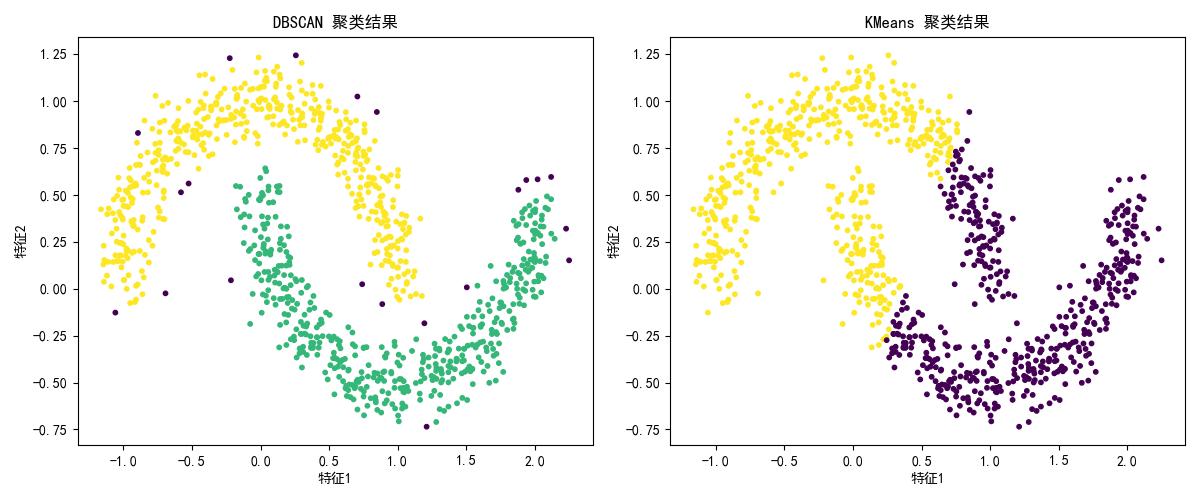
聚类结果展示：绘制DBSCAN和KMeans算法的聚类结果。

三、实验结果

原始数据：双月数据集，其中数据点被分为两类，分别用红色和蓝色表示。这些数据点形成了两个相互交错的月牙形状。



聚类结果：



第二张图展示了DBSCAN算法的聚类结果：

DBSCAN算法成功地将数据点分为了三个主要的簇，分别用黄色、绿色和紫色表示。

紫色簇包含了被视为位于两个月亮形状之间的噪声点。

绿色和黄色簇分别对应于两个月亮形状的主体部分。

第三张图展示了KMeans算法的聚类结果：

KMeans算法将数据点分为了两个簇，用黄色和紫色表示。

黄色簇包含了图案左侧的数据点，紫色簇则包含了图案右侧数据点。

可以看出，DBSCAN在处理具有复杂形状和噪声的数据集时表现更好，能够识别出任意形状的簇。KMeans无法很好地分类图形。在这个特定的双月数据集上，DBSCAN能够更细致地区分簇，而KMeans则将两个月亮形状的部分数据点合并到一个簇中。

四、实验心得

通过本次实验，我深入理解了kmeans和DBSCAN两个聚类算法的原理和实现过程。在实验过程中，我首先生成了双月数据集，并使用DBSCAN和KMeans算法对其进行了聚类分析。通过调整DBSCAN的eps和MinPts参数，我观察到了不同参数设置对聚类结果的影响。我也意识到了KMeans算法在处理非凸形簇时的局限性。

五、附录（关键代码）

|  |
| --- |
| import pandas as pd import numpy as np import matplotlib.pyplot as plt import random import math  plt.rcParams["font.sans-serif"] = ["SimHei"]  plt.rcParams["axes.unicode\_minus"] = False *# 生成双月数据集* from sklearn.datasets import make\_moons  *# 生成2D分类数据集* X, y = make\_moons(n\_samples=1000, shuffle=True, noise=0.1, random\_state=10)  *# 将生成的数据存入 DataFrame 方便后续处理* df = pd.DataFrame(dict(x=X[:, 0], y=X[:, 1], label=y))  *# 原始数据散点图，按照类值上色* colors = {0: 'red', 1: 'blue'} fig, ax = plt.subplots() grouped = df.groupby('label')  for key, group in grouped:     group.plot(ax=ax, kind='scatter', x='x', y='y', label=key, color=colors[key])  plt.title("原始双月数据") plt.show()  *# DBSCAN 聚类* def epsilon(data, MinPts):     m, n = np.shape(data)     xMax = np.max(data, axis=0)     xMin = np.min(data, axis=0)     volume = np.prod(xMax - xMin)     eps = ((volume \* MinPts \* math.gamma(0.5 \* n + 1)) / (m \* math.sqrt(math.pi \*\* n))) \*\* (1.0 / n)     return eps  def distance(data):     m, n = data.shape     dis = np.zeros((m, m))     for i in range(m):         for j in range(i, m):             dis[i, j] = np.linalg.norm(data[i] - data[j])             dis[j, i] = dis[i, j]     return dis  def find\_eps(distance\_D, eps):     return np.where(distance\_D <= eps)[0].tolist()  def dbscan(data, eps, MinPts):     m = data.shape[0]     labels = np.full(m, -1)  *# -1 indicates noise*     visited = np.zeros(m, dtype=bool)     current\_label = 0     dis = distance(data)      for i in range(m):         if not visited[i]:             visited[i] = True             neighbors = find\_eps(dis[i], eps)              if len(neighbors) < MinPts:                 labels[i] = -1  *# Noise*             else:                 current\_label += 1                 labels[i] = current\_label                  for neighbor in neighbors:                     if not visited[neighbor]:                         visited[neighbor] = True                         neighbors\_next = find\_eps(dis[neighbor], eps)                         if len(neighbors\_next) >= MinPts:                             neighbors.extend(neighbors\_next)                     if labels[neighbor] == -1:                         labels[neighbor] = current\_label      return labels  *# KMeans 聚类* def kmeans(data, k, max\_iters=100):     m, n = data.shape     centroids = data[np.random.choice(m, k, replace=False)]      for \_ in range(max\_iters):         distances = np.array([np.linalg.norm(data - centroid, axis=1) for centroid in centroids])         labels = np.argmin(distances, axis=0)          new\_centroids = np.array([data[labels == i].mean(axis=0) for i in range(k)])         if np.all(centroids == new\_centroids):             break         centroids = new\_centroids      return labels  *# 聚类参数* MinPts = 5 eps = epsilon(X, MinPts)  *# DBSCAN 结果* dbscan\_labels = dbscan(X, eps, MinPts)  *# KMeans 结果* k = 2 kmeans\_labels = kmeans(X, k)  *# 绘制结果* fig, axs = plt.subplots(1, 2, figsize=(12, 5))  *# DBSCAN 结果* axs[0].scatter(X[:, 0], X[:, 1], c=dbscan\_labels, cmap='viridis', marker='o', s=10) axs[0].set\_title("DBSCAN 聚类结果") axs[0].set\_xlabel("特征1") axs[0].set\_ylabel("特征2")  *# KMeans 结果* axs[1].scatter(X[:, 0], X[:, 1], c=kmeans\_labels, cmap='viridis', marker='o', s=10) axs[1].set\_title("KMeans 聚类结果") axs[1].set\_xlabel("特征1") axs[1].set\_ylabel("特征2")  plt.tight\_layout() plt.show() |