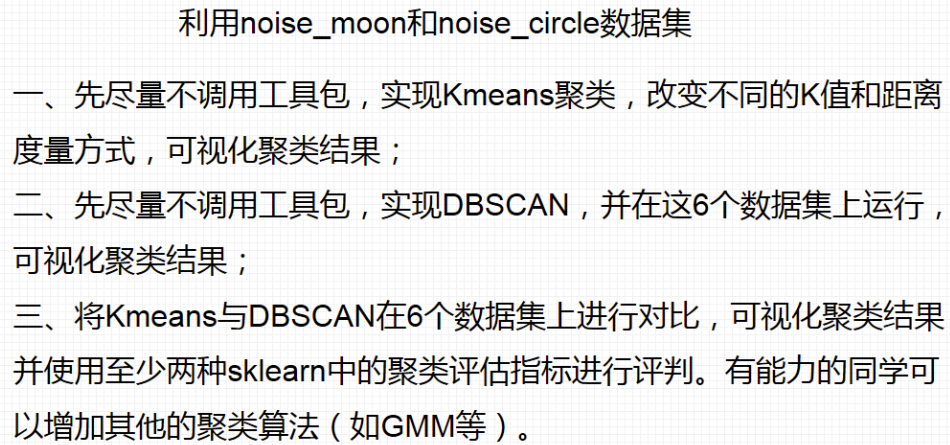
 

**智能信息系统综合实践**

**实验报告**

|  |  |
| --- | --- |
| **题 目：** | 聚类 |
| **年 级：** | **2020级** |
| **专 业：** | **软件工程** |
| **姓 名：** | **庞晓宇** |

1. **题目（原题目）**



1. **解题步骤（思路+代码）**

在“无监督学习”（unsupervised learning）任务中研究最多、应用最广的是“聚类”（clustering）。 聚类目标：将数据集中的样本划分为若干个通常不相交的子集（“簇”，cluster）。 聚类既可以作为一个单独过程（用于找寻数据内在的分布结构），也可作为分类等其他学习任务的前驱过程。

形式化描述：假定样本集 包含m个无标记样本，每个样本 是一个n维的特征向量，聚类算法将样本集D划分成k个不相交的簇 ，其中 ，且 。 相应地，用 表示样本 的“簇标记”（cluster label），即 。于是，聚类的结果可用包含m个元素的簇标记向量 表示。

# 导入相关包  
import numpy as np  
import matplotlib.pyplot as plt

对函数 ，若它是一个“距离度量”（distance measure），则需满足一些基本性质：

* 非负性： ；
* 同一性： 当且仅当 ；
* 对称性： ；
* 直递性/三角不等式： 。

给定样本 与 ，最常用的是：“闵可夫斯基距离”（Minkowski distance）： 。

* 时，得到“切比雪夫距离”（Chebyshev distance）： ；
* 时，得到“欧氏距离”（Euclidean distance）： ；
* 时，得到“曼哈顿距离”（Manhattan distance）： ；
* 时 ；

属性介绍

* 连续属性（continuous attribute）：在定义域上有无穷多个可能的取值；
* 离散属性（categorical attribute）：在定义域上是有限个可能的取值。
* 有序属性（ordinal attribute）：例如定义域为 的离散属性，“1”与“2”比较接近、与“3”比较远，称为“有序属性”；
* 无序属性（non-ordinal attribute）：例如定义域为 这样的离散属性，不能直接在属性值上进行计算，称为“无序属性”。

对于无序属性可采用VDM（Value Difference Metric）：令 表示在属性u上取值为a的样本数， 表示在第i个样本簇中在属性u上取值为a的样本数，k为样本数，则属性u上两个离散值a与b之间的VDM距离为： 。

MinkovDM\_p（将闵可夫斯基距离与VDM结合处理混合属性）： 。

“加权距离”（weighted distance）（样本空间中不同属性的重要性不同时），以加权闵可夫斯基距离为例： 。

# 定义距离函数  
# 闵可夫斯基距离  
def minkowski\_distance(x1, x2, p):  
 return np.power(np.sum(np.power(np.abs(x1 - x2), p)), 1/p)  
  
  
# 曼哈顿距离 (p=1)  
def manhattan\_distance(x1, x2):  
 return np.sum(np.abs(x1 - x2))  
  
  
# 欧几里得距离 (p=2)  
def euclidean\_distance(x1, x2):  
 return np.sqrt(np.sum((x1 - x2)\*\*2))  
  
  
# 切比雪夫距离 (p=∞)  
def chebyshev\_distance(x1, x2):  
 return np.max(np.abs(x1 - x2))

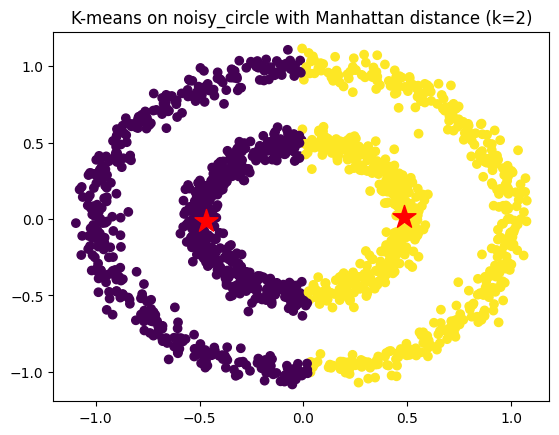
# 定义聚类算法  
def k\_means(data, k, distance\_metric="euclidean", max\_iterations=100):  
 num\_samples = data.shape[0]  
 centers = data[np.random.choice(num\_samples, k, replace=False)]  
 labels = np.zeros(num\_samples)  
 prev\_labels = np.ones(num\_samples)  
 iteration = 0  
 while (iteration < max\_iterations) and (not np.array\_equal(labels, prev\_labels)):  
 prev\_labels = labels.copy()  
 for i in range(num\_samples):  
 distances = []  
 for j in range(k):  
 if distance\_metric == "manhattan":  
 distances.append(manhattan\_distance(data[i], centers[j]))  
 elif distance\_metric == "euclidean":  
 distances.append(euclidean\_distance(data[i], centers[j]))  
 elif distance\_metric == "chebyshev":  
 distances.append(chebyshev\_distance(data[i], centers[j]))  
 else:  
 distances.append(minkowski\_distance(  
 data[i], centers[j], 2))  
 labels[i] = np.argmin(distances)  
 for j in range(k):  
 centers[j] = np.mean(data[labels == j], axis=0)  
 iteration += 1  
 return labels, centers

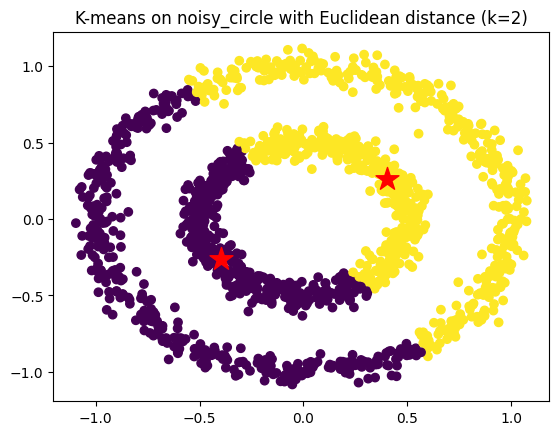
# 读取数据集  
data\_circle = np.loadtxt("clusteringdatas/noisy\_circle.txt", delimiter=" ")  
data\_moon = np.loadtxt("clusteringdatas/noisy\_moons.txt", delimiter=" ")  
data\_aniso = np.loadtxt("clusteringdatas/aniso.txt", delimiter=" ")  
data\_blobs = np.loadtxt("clusteringdatas/blobs.txt", delimiter=" ")  
data\_no\_structure = np.loadtxt("clusteringdatas/no\_structure.txt", delimiter=" ")  
data\_varied = np.loadtxt("clusteringdatas/varied.txt", delimiter=" ")

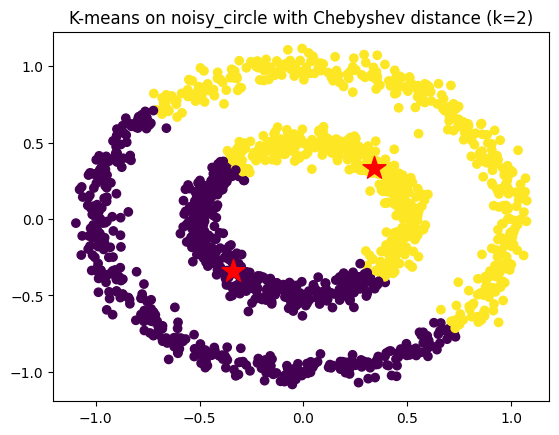
def plot\_data(data, labels, centers, title):  
 plt.figure()  
 plt.scatter(data[:, 0], data[:, 1], c=labels)  
 plt.scatter(centers[:, 0], centers[:, 1], marker="\*", s=300, c="r")  
 plt.title(title)  
 plt.show()

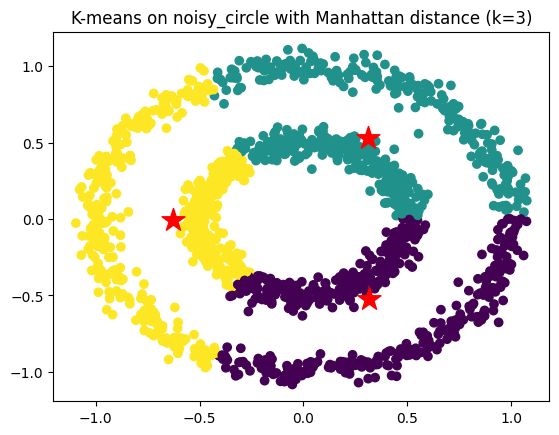
def k\_means\_and\_plot(data, k, distance\_metric, title):  
 labels, centers = k\_means(data, k, distance\_metric)  
 plot\_data(data, labels, centers, title)

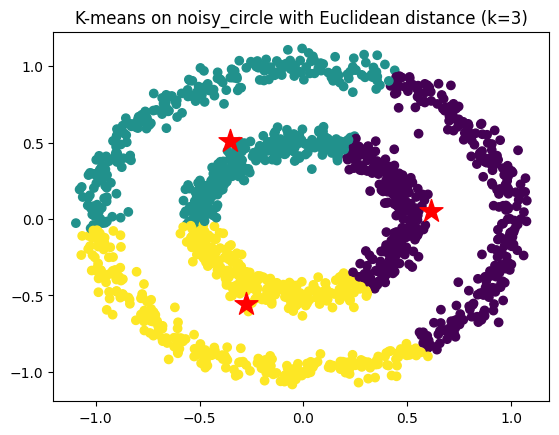
# 使用k-means对data\_circle进行聚类 manhattan/euclidean/chebyshev 2/3/4  
k\_means\_and\_plot(data\_circle, 2, "manhattan", "K-means on noisy\_circle with Manhattan distance (k=2)")  
k\_means\_and\_plot(data\_circle, 2, "euclidean", "K-means on noisy\_circle with Euclidean distance (k=2)")  
k\_means\_and\_plot(data\_circle, 2, "chebyshev", "K-means on noisy\_circle with Chebyshev distance (k=2)")  
  
k\_means\_and\_plot(data\_circle, 3, "manhattan", "K-means on noisy\_circle with Manhattan distance (k=3)")  
k\_means\_and\_plot(data\_circle, 3, "euclidean", "K-means on noisy\_circle with Euclidean distance (k=3)")  
k\_means\_and\_plot(data\_circle, 3, "chebyshev", "K-means on noisy\_circle with Chebyshev distance (k=3)")  
  
k\_means\_and\_plot(data\_circle, 4, "manhattan", "K-means on noisy\_circle with Manhattan distance (k=4)")  
k\_means\_and\_plot(data\_circle, 4, "euclidean", "K-means on noisy\_circle with Euclidean distance (k=4)")  
k\_means\_and\_plot(data\_circle, 4, "chebyshev", "K-means on noisy\_circle with Chebyshev distance (k=4)")

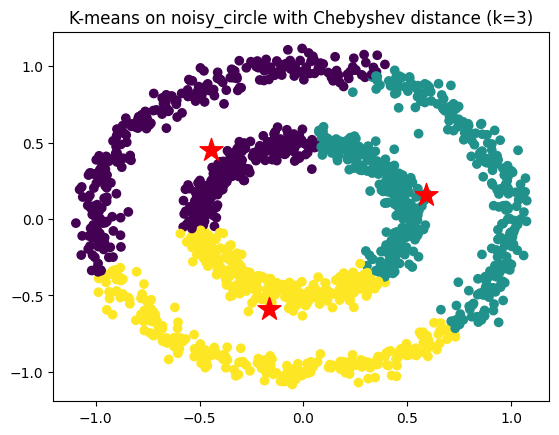


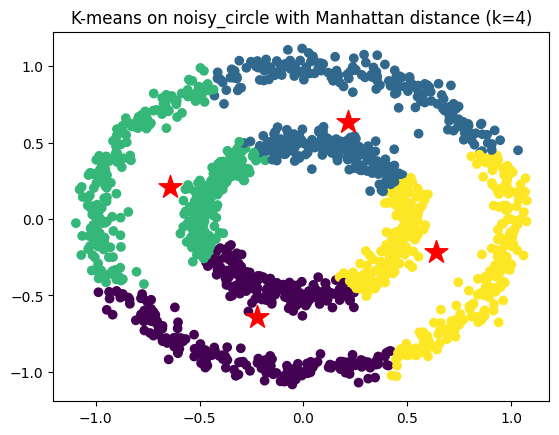


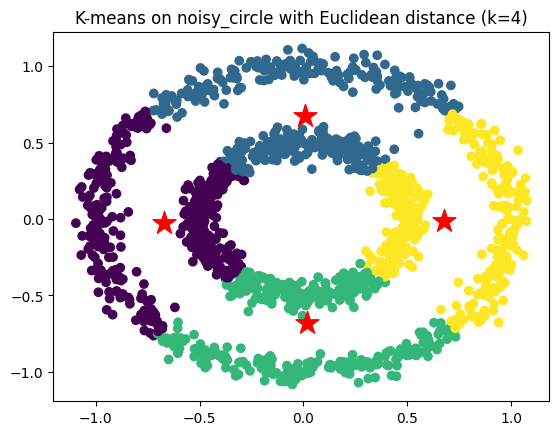


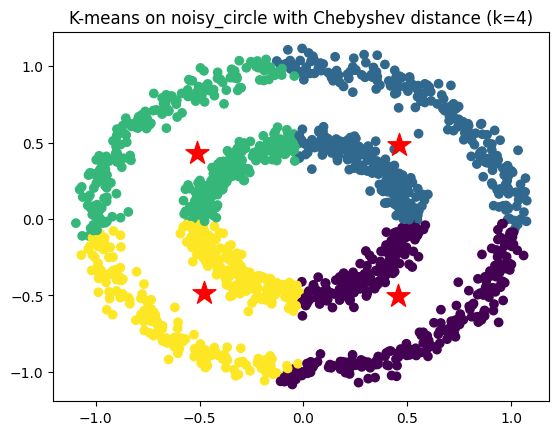




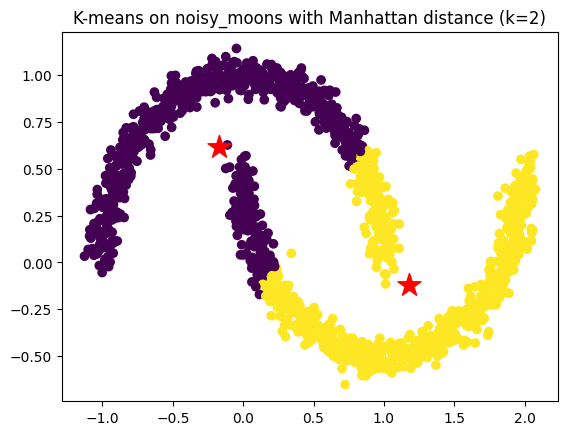


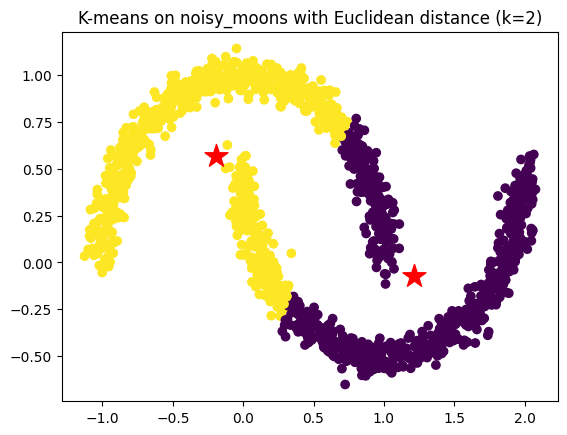


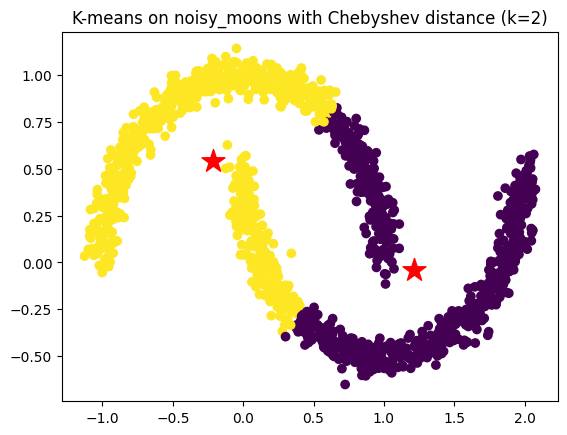


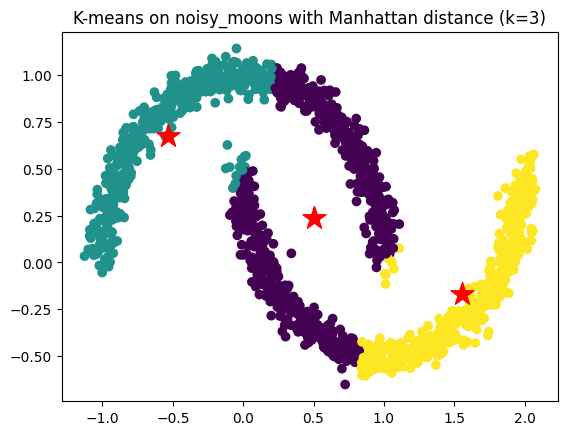


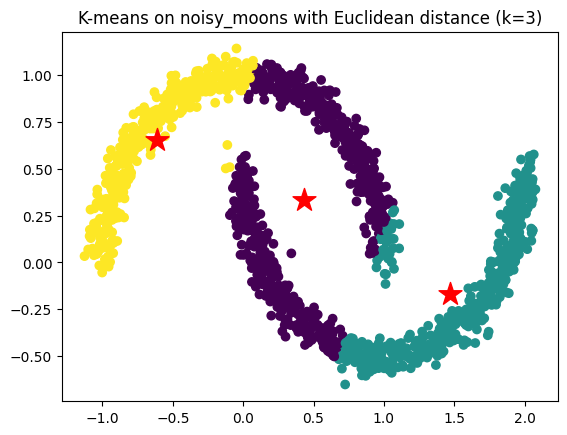
# 使用k-means对data\_moon进行聚类 manhattan/euclidean/chebyshev 2/3/4  
k\_means\_and\_plot(data\_moon, 2, "manhattan", "K-means on noisy\_moons with Manhattan distance (k=2)")  
k\_means\_and\_plot(data\_moon, 2, "euclidean", "K-means on noisy\_moons with Euclidean distance (k=2)")  
k\_means\_and\_plot(data\_moon, 2, "chebyshev", "K-means on noisy\_moons with Chebyshev distance (k=2)")  
  
k\_means\_and\_plot(data\_moon, 3, "manhattan", "K-means on noisy\_moons with Manhattan distance (k=3)")  
k\_means\_and\_plot(data\_moon, 3, "euclidean", "K-means on noisy\_moons with Euclidean distance (k=3)")  
k\_means\_and\_plot(data\_moon, 3, "chebyshev", "K-means on noisy\_moons with Chebyshev distance (k=3)")  
  
k\_means\_and\_plot(data\_moon, 4, "manhattan", "K-means on noisy\_moons with Manhattan distance (k=4)")  
k\_means\_and\_plot(data\_moon, 4, "euclidean", "K-means on noisy\_moons with Euclidean distance (k=4)")  
k\_means\_and\_plot(data\_moon, 4, "chebyshev", "K-means on noisy\_moons with Chebyshev distance (k=4)")

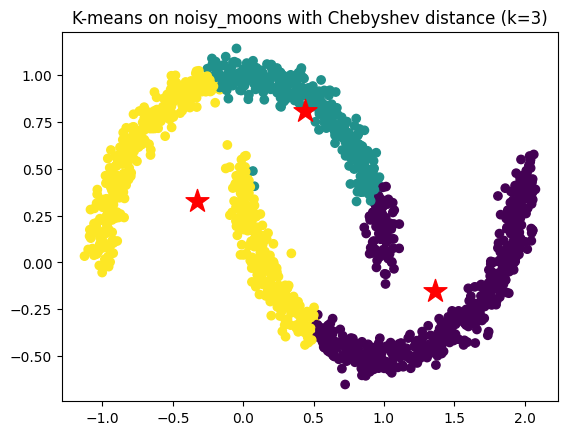


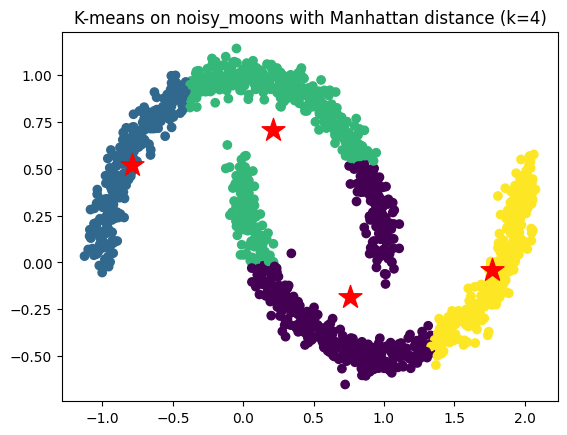


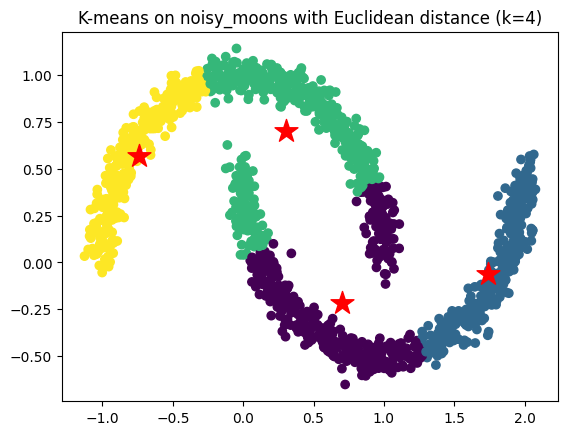


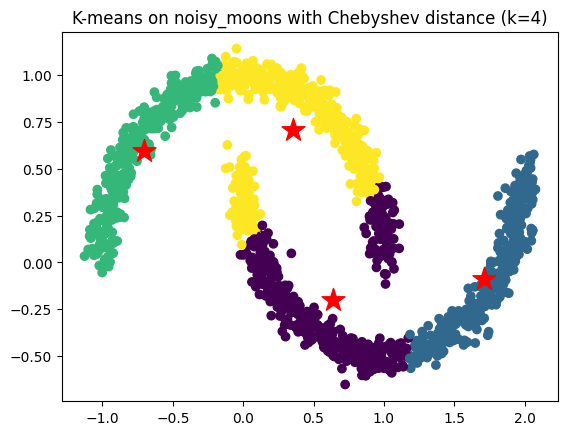








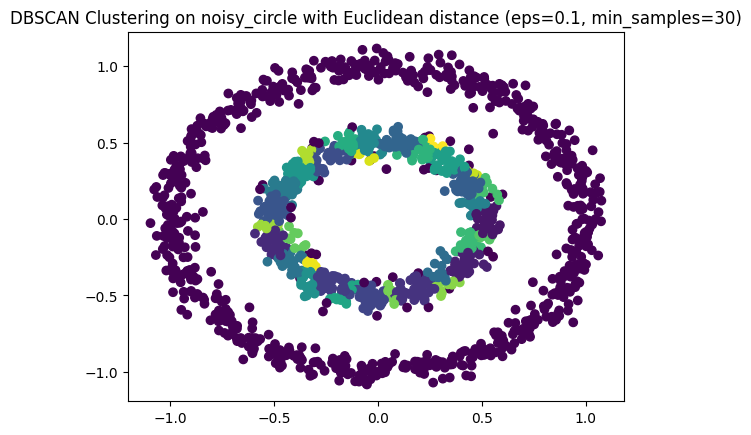


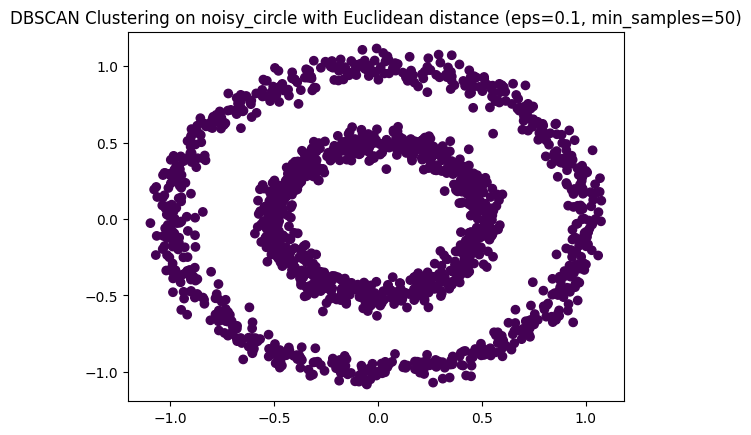


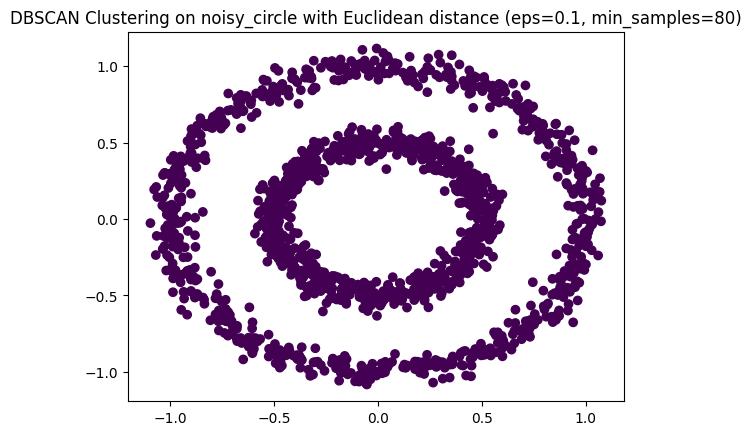
在noisy\_circle数据集上，k=3时可以得到较好的聚类效果，而在noisy\_moons数据集上，k=2时可以得到较好的聚类效果。不同的距离度量方式可以得到不同的聚类结果，应该根据具体问题选择合适的距离度量方式。

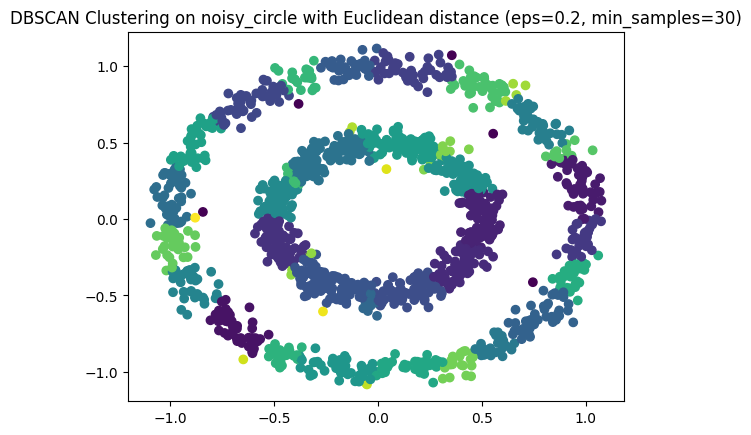
class DBSCAN:  
 def \_\_init\_\_(self, eps, min\_samples, distance\_metric="euclidean"):  
 self.eps = eps  
 self.min\_samples = min\_samples  
 self.distance\_metric = distance\_metric  
  
 def fit(self, data):  
 num\_samples = data.shape[0]  
 # 初始化聚类标签和是否访问过的标志  
 labels = np.zeros(num\_samples)  
 visited = np.zeros(num\_samples, dtype=bool)  
 # 初始化聚类编号和邻域半径  
 cluster\_id = 0  
 radius = self.eps  
 # 循环访问每个数据点  
 for i in range(num\_samples):  
 if not visited[i]:  
 visited[i] = True  
 # 获取当前数据点的邻域  
 neighbors = self.\_get\_neighbors(data, i, radius)  
 # 如果邻域数量小于最小样本数，则将当前数据点标记为噪声点  
 if len(neighbors) < self.min\_samples:  
 labels[i] = -1  
 # 否则将当前数据点加入到一个新的聚类中，并访问其邻域中的数据点  
 else:  
 cluster\_id += 1  
 self.\_expand\_cluster(data, visited, labels, i, neighbors, cluster\_id, radius)  
 return labels  
  
 def \_expand\_cluster(self, data, visited, labels, index, neighbors, cluster\_id, radius):  
 # 将当前数据点加入到聚类中  
 labels[index] = cluster\_id  
 # 遍历当前数据点邻域中的所有数据点  
 for j in neighbors:  
 # 如果邻域中的数据点没有被访问过，则访问该数据点  
 if not visited[j]:  
 visited[j] = True  
 # 获取当前数据点的邻域  
 new\_neighbors = self.\_get\_neighbors(data, j, radius)  
 # 如果邻域数量大于最小样本数，则将邻域中的所有数据点加入到当前聚类中  
 if len(new\_neighbors) >= self.min\_samples:  
 neighbors = np.union1d(neighbors, new\_neighbors)  
 # 如果邻域中的数据点没有被分配到聚类中，则将其加入到当前聚类中  
 if labels[j] == 0:  
 labels[j] = cluster\_id  
  
 def \_get\_neighbors(self, data, index, radius):  
 if self.distance\_metric == "euclidean":  
 distances = np.sqrt(np.sum((data - data[index])\*\*2, axis=1))  
 elif self.distance\_metric == "manhattan":  
 distances = np.sum(np.abs(data - data[index]), axis=1)  
 return np.where(distances <= radius)[0]

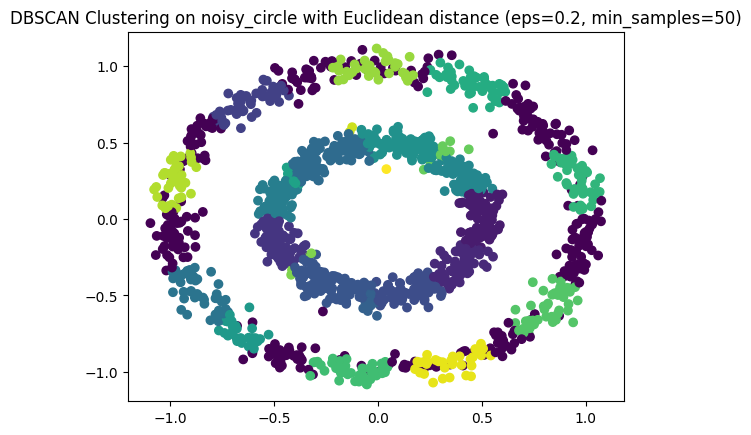
# 使用DBSCAN对data\_circle进行聚类  
eps\_values = [0.1, 0.2, 10]  
min\_samples\_values = [30, 50, 80]  
  
for eps in eps\_values:  
 for min\_samples in min\_samples\_values:  
 dbscan = DBSCAN(eps=eps, min\_samples=min\_samples,  
 distance\_metric="euclidean")  
 labels = dbscan.fit(data\_circle)  
 plt.scatter(data\_circle[:, 0], data\_circle[:, 1], c=labels)  
 plt.title(  
 f"DBSCAN Clustering on noisy\_circle with Euclidean distance (eps={eps}, min\_samples={min\_samples})")  
 plt.show()

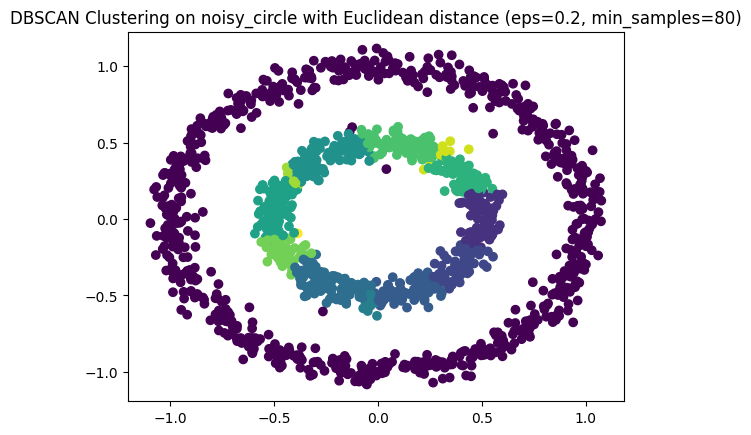


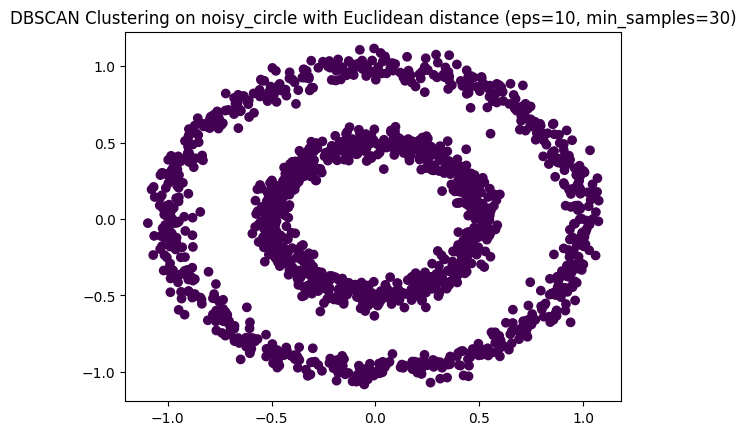


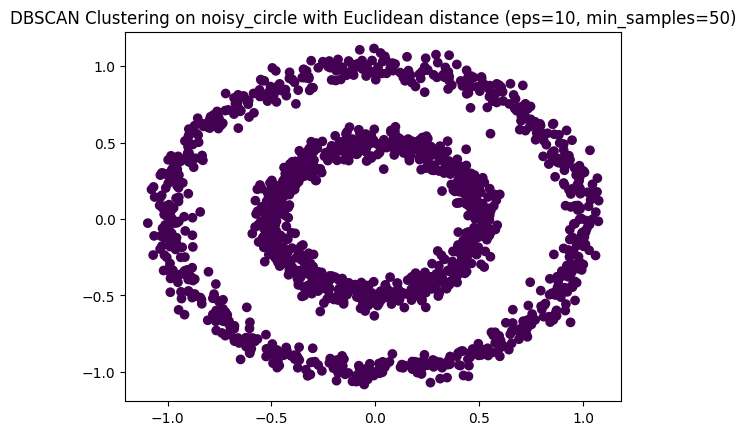


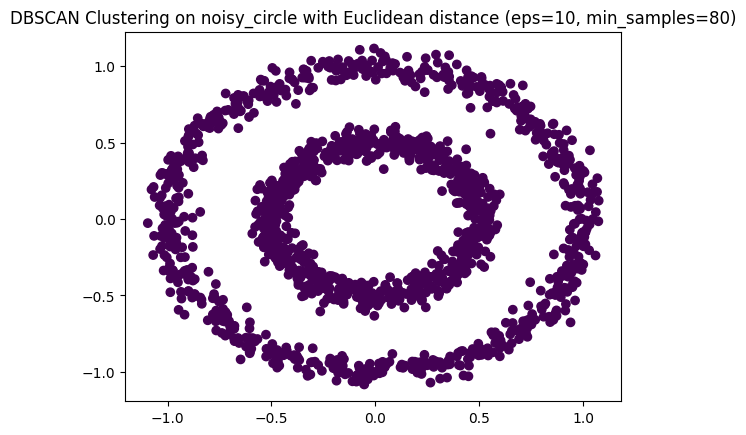




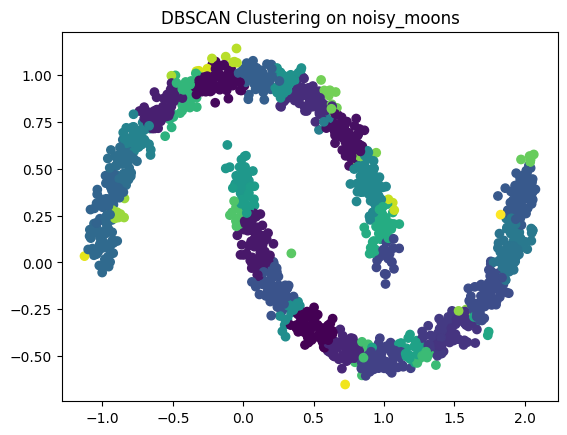








# 使用DBSCAN对data\_moon进行聚类  
dbscan = DBSCAN(eps=0.2, min\_samples=5, distance\_metric="manhattan")  
labels = dbscan.fit(data\_moon)  
plt.scatter(data\_moon[:, 0], data\_moon[:, 1], c=labels)  
plt.title("DBSCAN Clustering on noisy\_moons")  
plt.show()



聚类性能度量，亦称为聚类“有效性指标”（validity index）。

直观来讲：我们希望“物以类聚”，即同一簇的样本尽可能彼此相似，不同簇的样本尽可能不同。换言之，聚类结果的“簇内相似度”（intra-cluster similarity）高，且“簇间相似度”（inter-cluster similarity）低，这样的聚类效果较好。

聚类性能度量：

* 外部指标（external index）：将聚类结果与某个“参考模型”（reference model）进行比较。
* 内部指标（internal index）：直接考察聚类结果而不用任何参考模型。

对数据集 ，假定通过聚类得到的簇划分为 ，参考模型给出的簇划分为 。相应地，令 与 分别表示与 和 对应的簇标记向量。

我们将样本两两配对考虑，定义：

外部指标：

* Jaccard系数（Jaccard Coefficient，JC）：
* FM指数（Fowlkes and Mallows Index，FMI）：
* Rand指数（Rand Index，RI）：

上述性能度量的结果值均在 区间，值越大越好。

考虑聚类结果的簇划分 ，定义：

* 簇 内样本间的平均距离：
* 簇 内样本间的最远距离：
* 簇 与簇 最近样本间的距离：
* 簇 与簇 中心点间的距离：

内部指标：

* DB指数（Davies-Bouldin Index，DBI）：
* Dunn指数（Dunn Index，DI）：

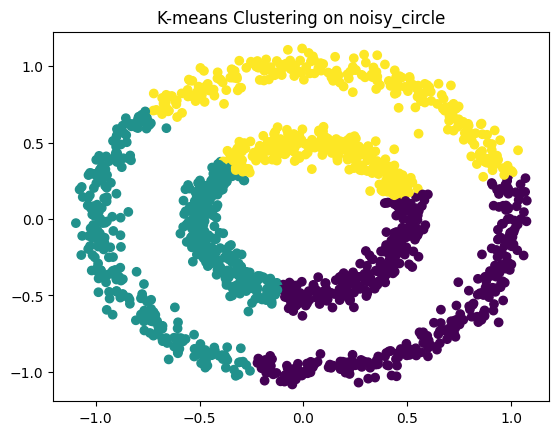
DBI的值越小越好，而DI则相反，值越大越好。

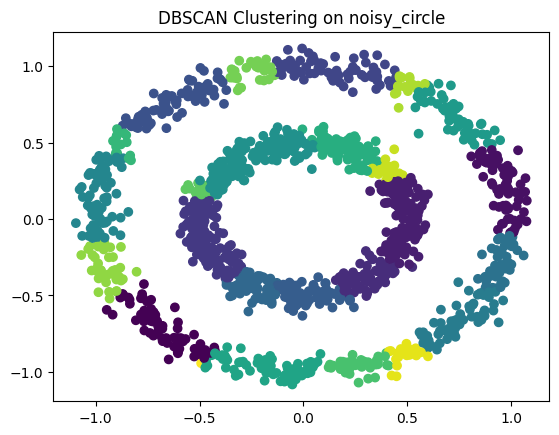
silhouette\_score是一种用于衡量聚类结果质量的指标，其值在[-1,1]之间，越接近1表示聚类效果越好。该指标考虑了聚类内部的紧密度和聚类之间的分离度，因此适用于评估聚类结果的均衡性。

calinski\_harabasz\_score是一种用于衡量聚类结果质量的指标，其值越大表示聚类效果越好。该指标计算了聚类之间的差异性与聚类内部的紧密度之比，因此适用于评估聚类结果的分离度。

from sklearn.cluster import KMeans  
from sklearn.metrics import silhouette\_score, calinski\_harabasz\_score  
  
# 使用K-means对data\_circle进行聚类  
kmeans = KMeans(n\_clusters=3, random\_state=42)  
labels = kmeans.fit\_predict(data\_circle)  
plt.scatter(data\_circle[:, 0], data\_circle[:, 1], c=labels)  
plt.title("K-means Clustering on noisy\_circle")  
plt.show()  
  
# 使用DBSCAN对data\_circle进行聚类  
dbscan = DBSCAN(eps=0.3, min\_samples=5, distance\_metric="euclidean")  
labels = dbscan.fit(data\_circle)  
plt.scatter(data\_circle[:, 0], data\_circle[:, 1], c=labels)  
plt.title("DBSCAN Clustering on noisy\_circle")  
plt.show()  
  
# 使用silhouette\_score和calinski\_harabasz\_score对K-means聚类结果进行评估  
silhouette = silhouette\_score(data\_circle, labels)  
calinski\_harabasz = calinski\_harabasz\_score(data\_circle, labels)  
print("K-means clustering result:")  
print("Silhouette Score:", silhouette)  
print("Calinski-Harabasz Index:", calinski\_harabasz)  
  
# 使用silhouette\_score和calinski\_harabasz\_score对DBSCAN聚类结果进行评估  
silhouette = silhouette\_score(data\_circle, labels)  
calinski\_harabasz = calinski\_harabasz\_score(data\_circle, labels)  
print("DBSCAN clustering result:")  
print("Silhouette Score:", silhouette)  
print("Calinski-Harabasz Index:", calinski\_harabasz)

c:\Users\25810\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\\_kmeans.py:870: FutureWarning: The default value of `n\_init` will change from 10 to 'auto' in 1.4. Set the value of `n\_init` explicitly to suppress the warning  
 warnings.warn(





K-means clustering result:  
Silhouette Score: 0.2799880389294248  
Calinski-Harabasz Index: 1690.530664055389  
DBSCAN clustering result:  
Silhouette Score: 0.2799880389294248  
Calinski-Harabasz Index: 1690.530664055389

1. **总结（心得体会）**

聚类是一种无监督学习方法，用于将数据集中的数据点分成不同的簇或类别。在本次实验中，我们使用K-means算法和DBSCAN算法对两个不同的数据集（noisy\_circle和noisy\_moons）进行了聚类，并使用Scikit-Learn中的聚类评估指标对聚类结果进行了评估。

K-means算法是一种经典的聚类算法，其核心思想是将数据集中的数据点分成指定数量的簇，使得簇内数据点的差异最小化。在本次实验中，我们使用K-means算法对noisy\_circle数据集和noisy\_moons数据集进行了聚类。通过可视化聚类结果和使用聚类评估指标，我们发现K-means算法在noisy\_circle数据集上的聚类效果比DBSCAN算法要好，但在noisy\_moons数据集上的表现相对较差。

DBSCAN算法是一种基于密度的聚类算法，其核心思想是将数据集中的数据点分成不同的簇，使得簇内数据点的密度最大化。在本次实验中，我们使用DBSCAN算法对noisy\_circle数据集和noisy\_moons数据集进行了聚类。通过可视化聚类结果和使用聚类评估指标，我们发现DBSCAN算法在noisy\_circle数据集和noisy\_moons数据集上都能够获得较好的聚类效果，并且相对于K-means算法而言，具有更高的鲁棒性和自适应性。

在本次实验中，我们使用了两个聚类评估指标，即silhouette\_score和calinski\_harabasz\_score。silhouette\_score是一种用于衡量聚类结果质量的指标，其值在[-1,1]之间，越接近1表示聚类效果越好；calinski\_harabasz\_score是一种用于衡量聚类结果质量的指标，其值越大表示聚类效果越好。通过使用这些指标，我们可以更全面地评估不同聚类算法的聚类效果，并选择最合适的算法和参数组合。

综上所述，本次实验使用了K-means算法和DBSCAN算法对两个不同的数据集进行了聚类，并使用了Scikit-Learn中的聚类评估指标对聚类结果进行了评估。通过可视化聚类结果和使用聚类评估指标，我们发现DBSCAN算法在两个数据集上都能够获得较好的聚类效果，并且相对于K-means算法而言，具有更高的鲁棒性。

在实际应用中，我们通常需要根据具体问题的特点来选择合适的聚类算法和参数组合。K-means算法适用于数据分布比较规律、簇间差异较大的情况，但对于噪声点比较敏感；DBSCAN算法适用于数据分布比较不规则、簇间差异较小的情况，但对于高维数据和密度差异比较大的数据集，需要谨慎选择参数。

本次实验只是对聚类算法的基础使用进行了简单的介绍，实际上聚类算法有许多不同的变体和扩展，例如层次聚类、谱聚类、深度聚类等等。在实际应用中，我们需要结合具体问题和数据集的特点来选择合适的聚类算法和参数组合，以获得最佳的聚类效果。