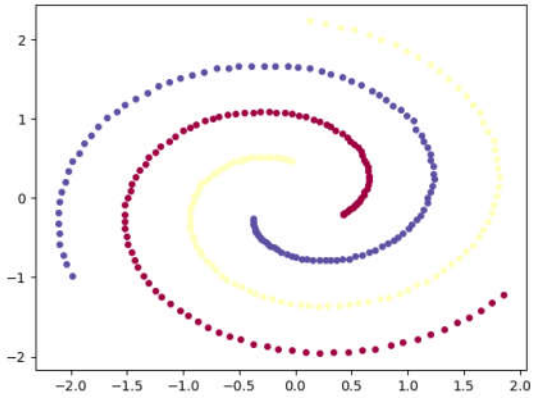
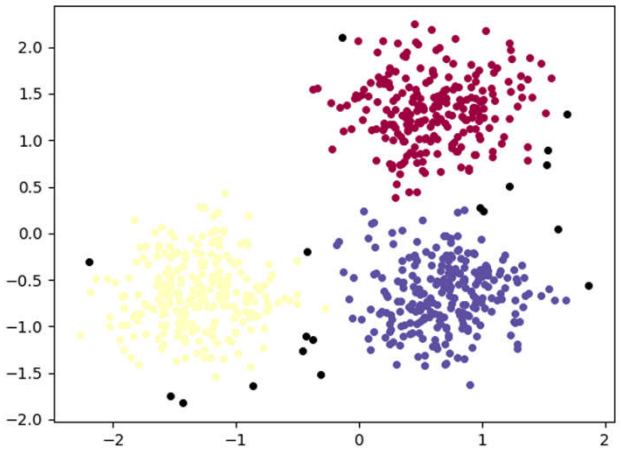
Python编程与人工智能实践

算法篇： DBSCAN
(Density-Based Spatial Clustering of Applications with Noise)





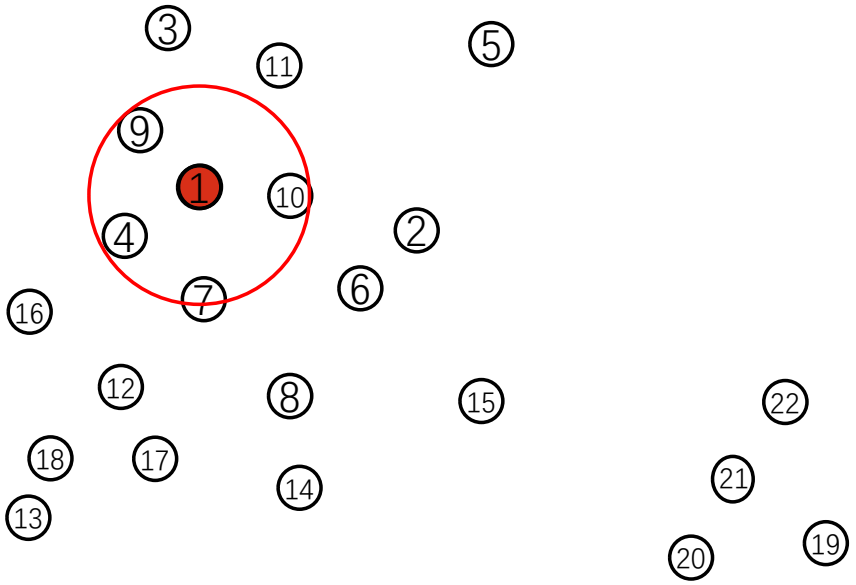
于泓
鲁东大学
信息与电气工程学院
2021.8.27

DBSCAN

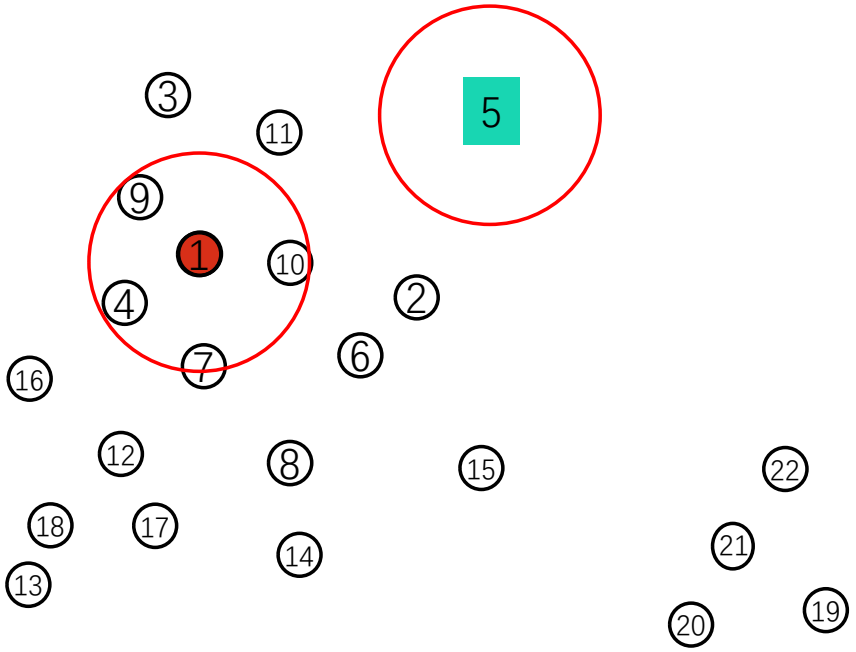
- DBSCAN(Density-Based Spatial Clustering of Applications with Noise)是一个比较有代表性的基于密度的聚类算法。与划分和层次聚类方法不同，它将簇定义为密度相连的点的最大集合，能够把具有足够高密度的区域划分为簇，并可在噪声的空间数据库中发现任意形状的聚类。

算法流程:

（1） 选取一个点，以eps为半径，画一个圈，看圈内有几个临近点？如果大于某个阈值min_points, 则认为该点为某一簇的点



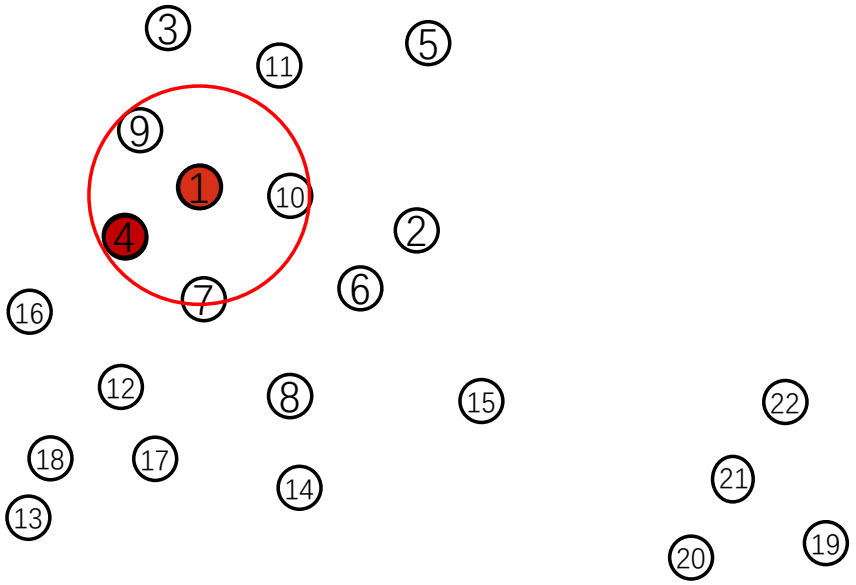
如果小于 min_points
则被标记为噪声点



算法流程:

（1） 选取一个点，以eps为半径，画一个圈，看圈内有几个临近点？如果大于某个阈值min_points, 则认为该点为某一簇的点

如果小于 min_points
则被标记为噪声点，然后处理下一个点



算法流程:

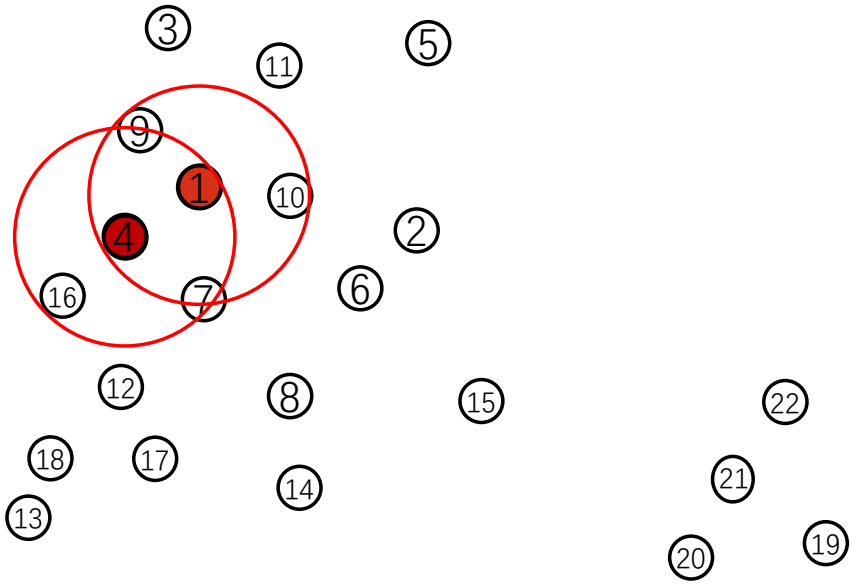
（1） 选取一个点，以eps为半径，画一个圈，看圈内有几个临近点？如果大于某个阈值min_points, 则认为该点为**某一簇**的点

（2）将临近点作为种子点

seeds = [**4**,7,9,10]

遍历所有种子点

如果该点被标为**噪声点**，则重标为**聚类点**
如果该点没有被标记过，则标记为**聚类点**



算法流程:

（1） 选取一个点，以eps为半径，画一个圈，看圈内有几个临近点？如果大于某个阈值min_points, 则认为该点为**某一簇**的点

（2）将临近点作为种子点

seeds = [**4**,7,9,10]

遍历所有种子点

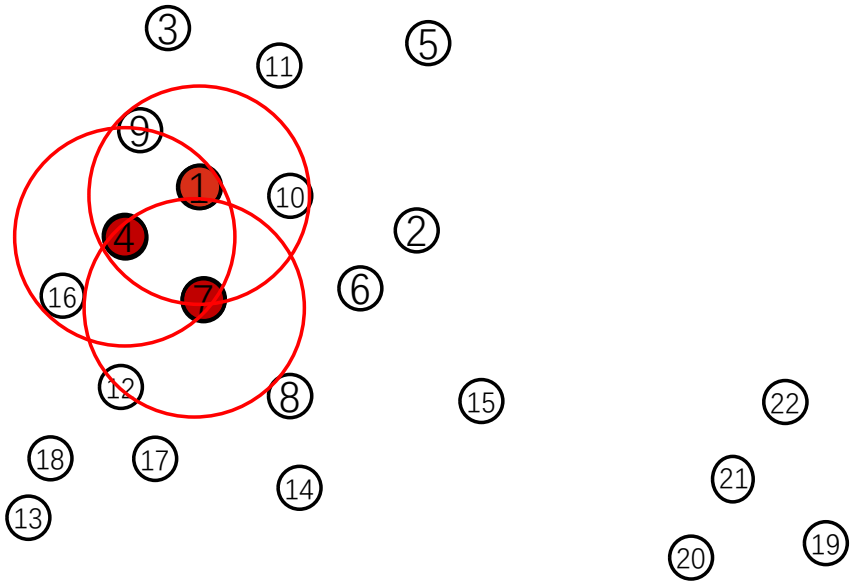
如果该点被标为**噪声点**，则重标为**聚类点**

如果该点没有被标记过，则标记为**聚类点**

并且以该点为圆心，以eps为半径 画圈

如果圈内点大于min_points,
将圈内点，添加到种子点中

seeds = [**4**,7,9,10,1,7,9,16]



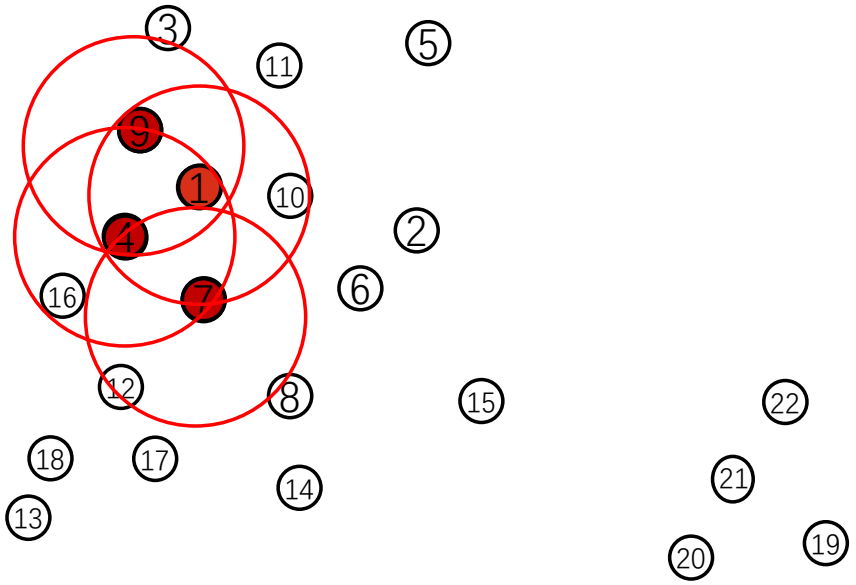
算法流程:

(3) 重复步骤2，直到遍历完所有的种子点

seeds = [**4**,7,9,10,1,7,9,16]

seeds = [**4**,**7**,9,10,1,7,9,16]

7的周围有 12， 4 少于 min_points
seed 不扩展



算法流程:

(3) 重复步骤2，直到遍历完所有的种子点

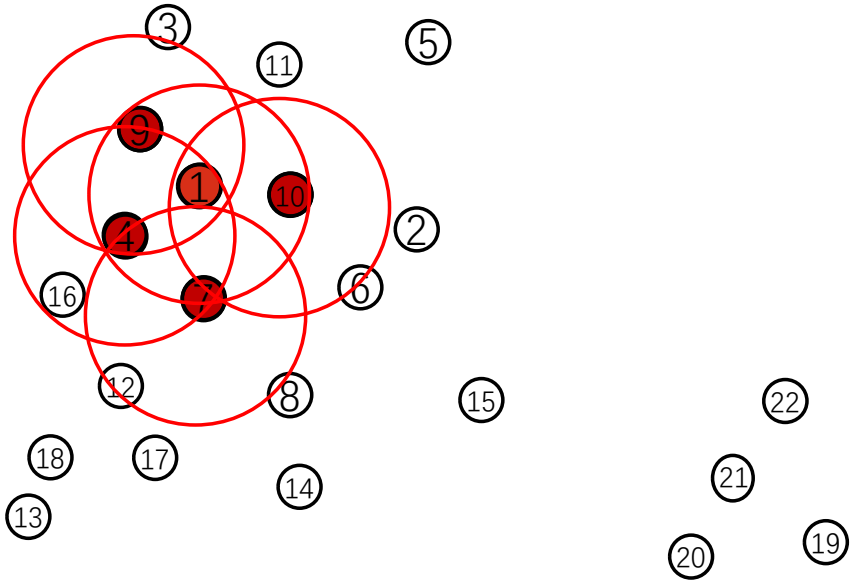
seeds = [**4**,7,9,10,1,7,9,16]

seeds = [**4**,**7**,9,10,1,7,9,16]

seeds = [**4**,**7**,**9**,10,1,7,9,16]

9的周围1,4,3 ,可以添加

seeds = [**4**,**7**,**9**,10,1,7,9,16,1,4,3]



算法流程:

(3) 重复步骤2，直到遍历完所有的种子点

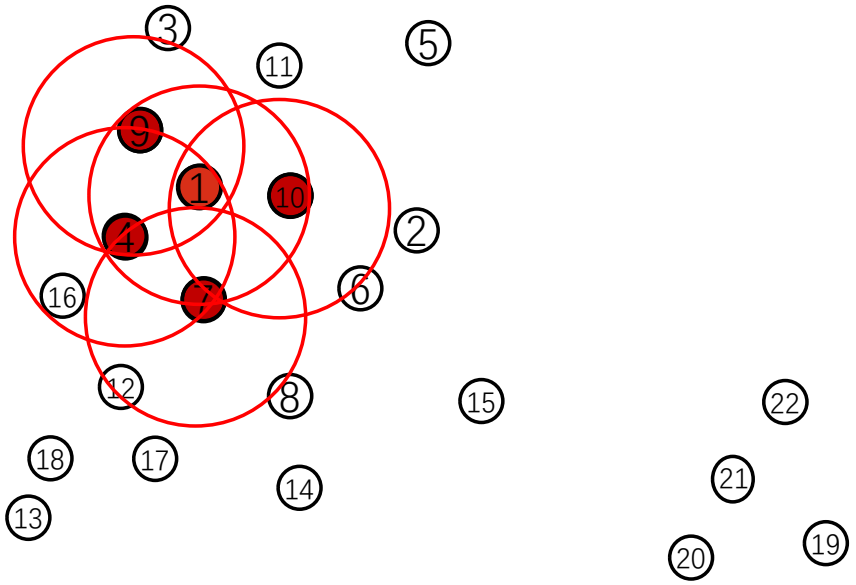
seeds = [**4**,7,9,10,1,7,9,16]

seeds = [**4**,**7**,9,10,1,7,9,16]

seeds = [**4**,**7**,**9**,10,1,7,9,16,1,4,3]

10的周围，1,6,7

seeds = [**4**,**7**,**9**,**10**,1,7,9,16,1,4,3,1,6,7]



算法流程:

(3) 重复步骤2，直到遍历完所有的种子点

seeds = [**4**,7,9,10,1,7,9,16]

seeds = [**4**,**7**,9,10,1,7,9,16]

seeds = [**4**,**7**,**9**,10,1,7,9,16,1,4,3]

seeds = [**4**,**7**,**9**,**10**,1,7,9,16,1,4,3,1,6,7]

1 已经标记过继续下个点

seeds = [**4**,**7**,**9**,**10**,**1**,7,9,16,1,4,3,1,6,7]

7 已经标记过继续下个点

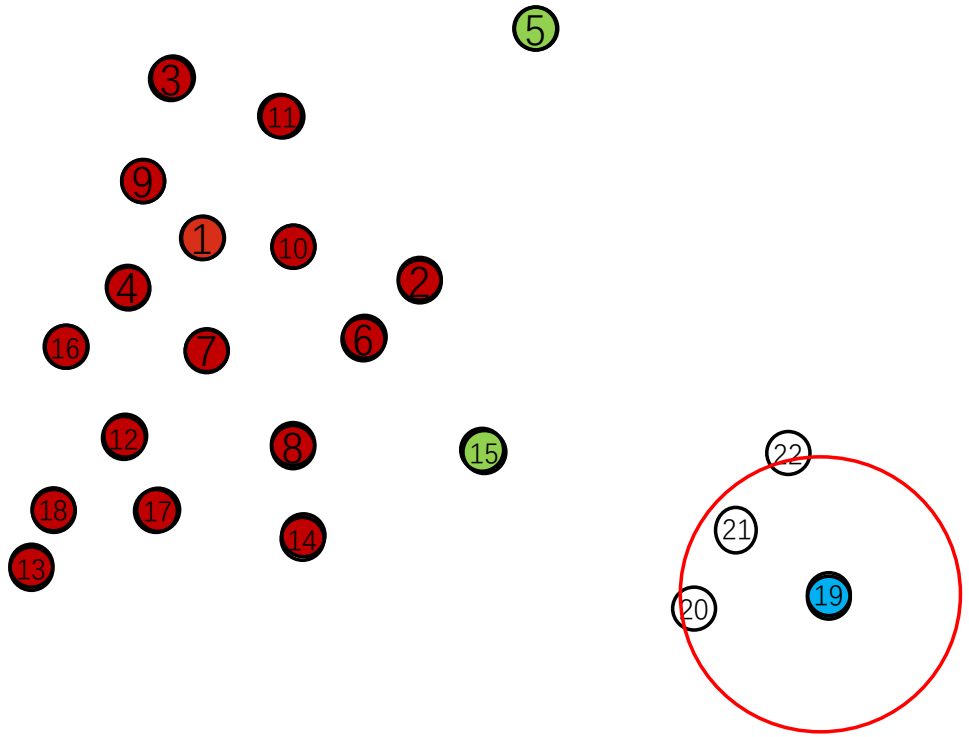
seeds = [**4**,**7**,**9**,**10**,**1**,**7**,9,16,1,4,3,1,6,7]

9 已经标记过继续下个点

seeds = [**4**,**7**,**9**,**10**,**1**,**7**,**9**,16,1,4,3,1,6,7]

16 周围点过少

seeds = [**4**,**7**,**9**,**10**,**1**,**7**,**9**,**16**,1,4,3,1,6,7]



算法流程:

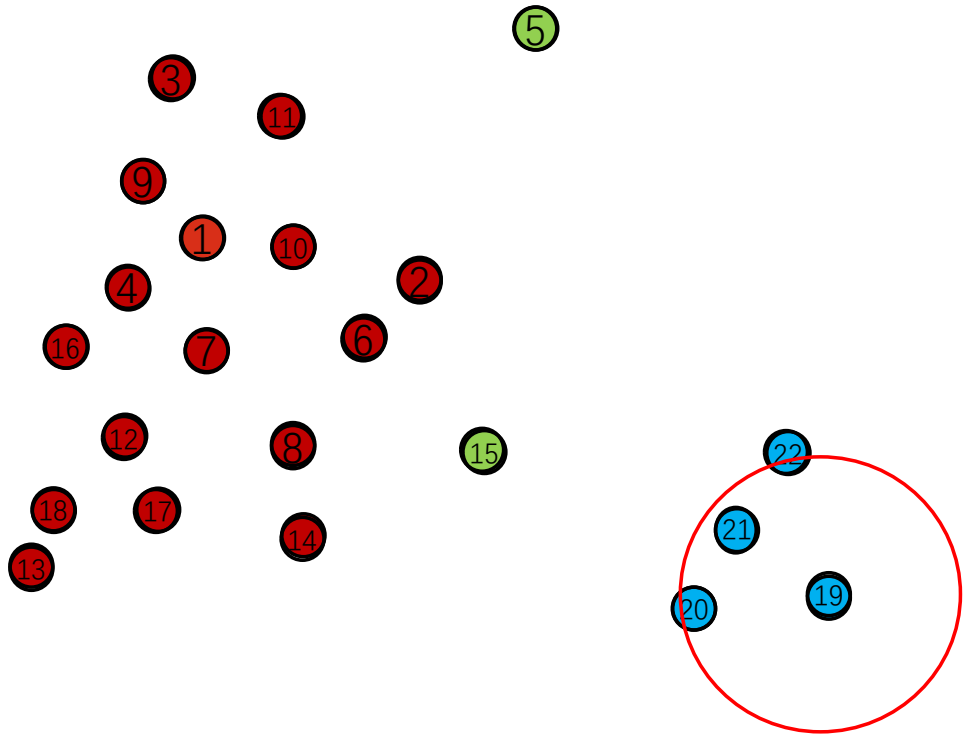
(4) 标记完一 簇后

寻找一个未被标记的点，开始新的一轮聚类

如果 找到点5， 周围点过少 标记为 NOISE

找到点15， 周围点过少，标记为NOISE

找到点 19 开始新的一轮聚类



算法流程:

(4) 标记完一 簇后

寻找一个未被标记的点，开始新的一轮聚类

如果 找到点5， 周围点过少 标记为 NOISE

找到点15， 周围点过少，标记为NOISE

找到点 19 开始新的一轮聚类

代码实现：

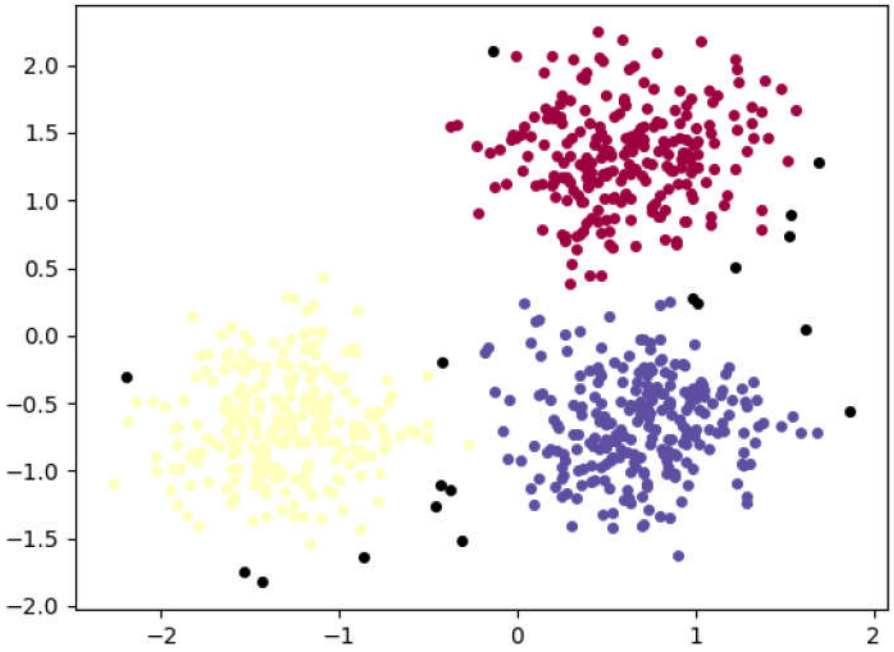
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN
from sklearn import metrics

UNCLASSIFIED = 0
NOISE = -1

# 计算数据点两两之间的距离
def getDistanceMatrix(datas):
    N,D = np.shape(datas)
    dists = np.zeros([N,N])

    for i in range(N):
        for j in range(N):
            vi = datas[i,:]
            vj = datas[j,:]
            dists[i,j]= np.sqrt(np.dot((vi-vj),(vi-vj)))
    return dists


#  寻找以点cluster_id 为中心，eps 为半径的圆内的所有点的id
def find_points_in_eps(point_id,eps,dists):
    index = (dists[point_id]<=eps)
    return np.where(index==True)[0].tolist()
```

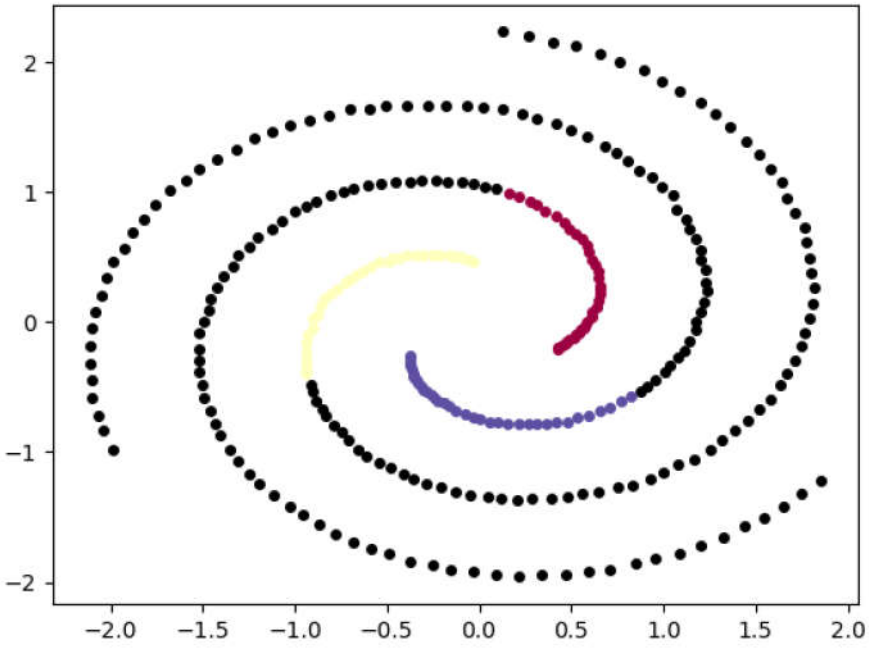
```python
def dbscan(datas, eps, min_points):

    # 计算 所有点之间的距离
    dists = getDistanceMatrix(datas)


    # 将所有点的标签初始化为UNCLASSIFIED
    n_points = datas.shape[0]
    labs = [UNCLASSIFIED]*n_points

    cluster_id = 0
    # 遍历所有点
    for point_id in range(0, n_points):
        # 如果当前点已经处理过了
        if not(labs[point_id] == UNCLASSIFIED):
            continue

        # 没有处理过则计算临近点
        seeds = find_points_in_eps(point_id,eps,dists)

        # 如果临近点数量过少则标记为 NOISE
        if len(seeds)<min_points:
            labs[point_id] = NOISE
        else:
            # 否则就开启一轮簇的扩张
            cluster_id = cluster_id+1
            # 标记当前点
            labs[point_id] = cluster_id
            expand_cluster(dists, labs, cluster_id, seeds,eps, min_points)
    return labs, cluster_id
```

```python
# 聚类扩展
# dists :  所有数据两两之间的距离  N x N
# labs :   所有数据的标签 labs N,
# cluster_id :  一个簇的标号
# eps :  密度评估半径
# seeds:  用来进行簇扩展的点
# min_points:  半径内最少的点数
def expand_cluster(dists, labs, cluster_id, seeds, eps, min_points):


    i = 0
    while i< len(seeds):
        # 获取一个临近点
        Pn = seeds[i]
        # 如果该点被标记为NOISE 则重新标记
        if labs[Pn] == NOISE:
            labs[Pn] = cluster_id
        # 如果该点没有被标记过
        elif labs[Pn] == UNCLASSIFIED:
            # 进行标记，并计算它的临近点 new_seeds
            labs[Pn] = cluster_id
            new_seeds = find_points_in_eps(Pn,eps,dists)

            # 如果 new_seeds 足够长则把它加入到seed 队列中
            if len(new_seeds) >=min_points:
                seeds = seeds + new_seeds

        i = i+1
```

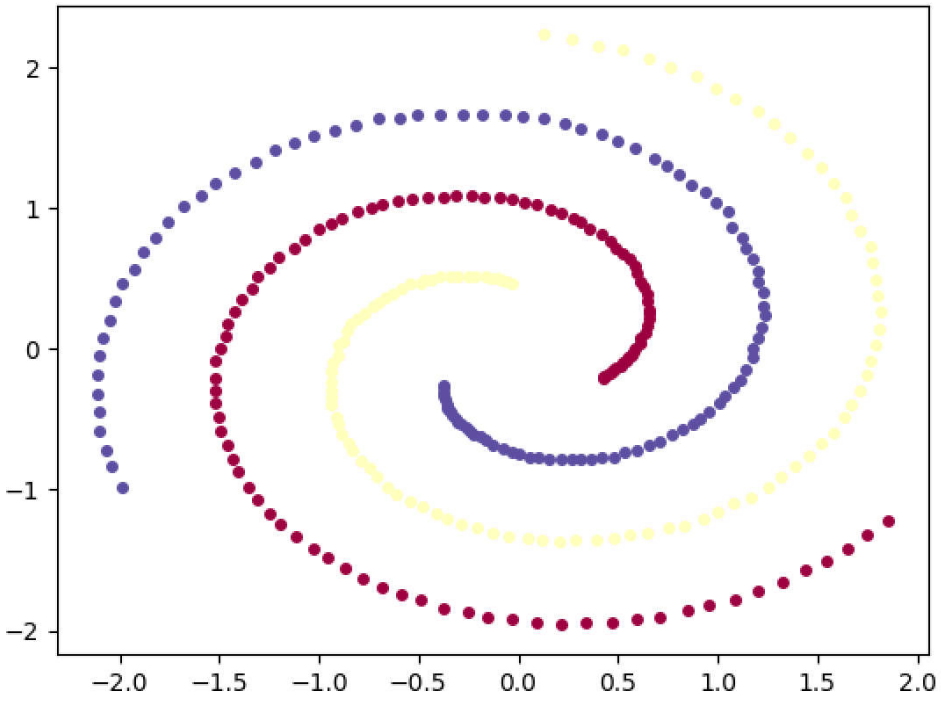
```
# 绘图
def draw_cluster(datas,labs, n_cluster):
    plt.cla()

    colors = [plt.cm.Spectral(each)
          for each in np.linspace(0, 1,n_cluster)]


    for i,lab in enumerate(labs):
        if lab ==NOISE:
            plt.scatter(datas[i,0],datas[i,1],s=16.,color=(0,0,0))
        else:
            plt.scatter(datas[i,0],datas[i,1],s=16.,color=colors[lab-1])
    plt.show()
```

```python
if __name__ == "__main__":

    ## 数据1
    # centers = [[1, 1], [-1, -1], [1, -1]]
    # datas, labels_true = make_blobs(n_samples=750, centers=centers, cluster_std=0.4,
                            # random_state=0)

    ## 数据2
    file_name = "spiral"
    with open(file_name+".txt","r",encoding="utf-8") as f:
        lines = f.read().splitlines()
    lines = [line.split("\t")[:-1] for line in lines]
    datas = np.array(lines).astype(np.float32)


    # 数据正则化
    datas = StandardScaler().fit_transform(datas)
    eps = 0.45
    min_points = 5
    labs, cluster_id = dbscan(datas, eps=eps, min_points=min_points)
    print("labs of my dbscan")
    print(labs)



    db = DBSCAN(eps=eps, min_samples=min_points).fit(datas)
    skl_labels = db.labels_
    print("labs of sk-DBSCAN")
    print(skl_labels)


    draw_cluster(datas,labs, cluster_id)
```





eps = 0.3
min_points = 10



eps = 0.45
min_points = 5