机器学习实验 11. 聚类（二）密度聚类

密度聚类算法原理概述

DBSCAN(Density-Based Spatial Clustering of Applications with Noise，具有噪声的基于密度的聚类方法)是一种基于密度的聚类算法，这类密度聚类算法一般假定类别可以通过样本分布的紧密程度决定。同一类别的样本，他们之间的紧密相连的，也就是说，在该类别任意样本周围不远处一定有同类别的样本存在。通过将紧密相连的样本划为一类，这样就得到了一个聚类类别。通过将所有各组紧密相连的样本划为各个不同的类别，就得到了最终的所有聚类类别结果。

DBSCAN 是基于一组邻域来描述样本集的紧密程度的，参数(ε，MinPts)用来描述邻域的样本分布紧密程度。其中，ε 描述了某一样本的邻域距离阈值，MinPts 描述了某一样本的距离为 ε 的邻域中样本个数的阈值。

DBSCAN 算法中的密度描述

假设样本集是 $D=(x_1, x_2,..., x_m)$,则 DBSCAN 具体的密度描述定义如下：

1）ε-邻域：对于 $x_j \in D$，其 ε-邻域包含样本集 D 中与 x_j 的距离不大于 ε 的子样本集，即 $N_\varepsilon(x_j)=\{x_i \in D | \text{distance}(x_i,x_j) \le \varepsilon\}$,这个子样本集的个数记为 $|N_\varepsilon(x_j)|$。

2) 核心对象：对于任一样本 $x_j \in D$，如果其 ε-邻域对应的 $N_\varepsilon(x_j)$至少包含 MinPts 个样本，即如果 $|N_\varepsilon(x_j)| \ge$ MinPts，则 x_j 是核心对象。

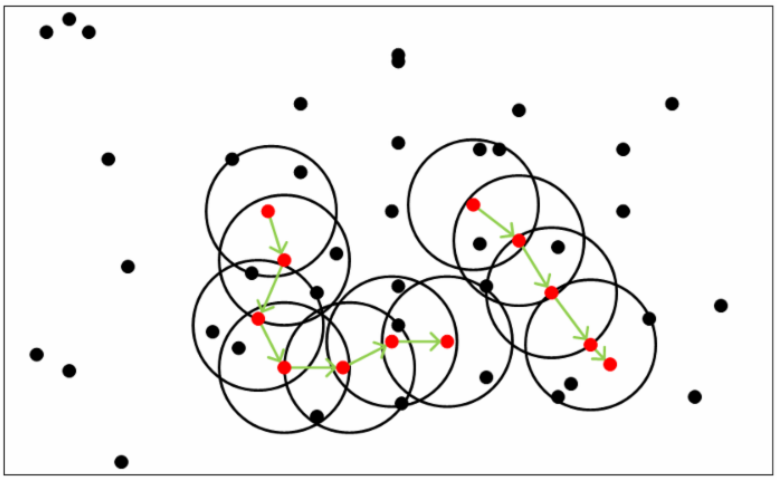
3）密度直达：如果 x_i 位于 x_j 的 ε-邻域中，且 x_j 是核心对象，则称 x_i 由 x_j 密度直达。注意反之不一定成立，即此时不能说 x_j 由 x_i 密度直达，除非且 x_i 也是核心对象。

4）密度可达：对于 x_i 和 x_j，如果存在样本序列 $p_1, p_2,..., p_T$，满足 $p_1=x_i$，$p_T=x_j$，且 p_{t+1} 由 p_t 密度直达，则称 x_j 由 x_i 密度可达。也就是说，密度可达满足传递性。

此时序列中的传递样本 $p_1, p_2, ..., p_{T-1}$ 均为核心对象，因为只有核心对象才能使其他样本密度直达。注意密度可达也不满足对称性，这个可以由密度直达的不对称性得出。

5）密度相连：对于 x_i 和 x_j，如果存在核心对象样本 x_k，使 x_i 和 x_j 均由 x_k 密度可达，则称 x_i 和 x_j 密度相连。注意密度相连关系是满足对称性的。

从下图可以很容易看出理解上述定义，图中 MinPts=5，红色的点都是核心对象，因为其 ε-邻域至少有 5 个样本。黑色的样本是非核心对象。所有核心对象密度直达的样本在以红色核心对象为中心的超球体内，如果不在超球体内，则不能密度直达。图中用绿色箭头连起来的核心对象组成了密度可达的样本序列。在这些密度可达的样本序列的 ε-邻域内所有的样本相互都是密度相连的。



DBSCAN 的聚类定义很简单：由密度可达关系导出的最大密度相连的样本集合，即为最终聚类的一个类别，或者说一个簇。

这个 DBSCAN 的簇里面可以有一个或者多个核心对象。如果只有一个核心对象，则簇里其他的非核心对象样本都在这个核心对象的 ε-邻域里；如果有多个核心对象，则簇里的任意一个核心对象的 ε-邻域中一定有一个其他的核心对象，否则这两个核心对象无法密度可达。这些核心对象的 ε-邻域里所有的样本的集合组成的一个 DBSCAN 聚类簇。

那么怎么才能找到这样的簇样本集合呢？DBSCAN 使用的方法很简单，它任意选择一个没有类别的核心对象作为种子，然后找到所有这个核心对象能够密

度可达的样本集合，即为一个聚类簇。接着继续选择另一个没有类别的核心对象去寻找密度可达的样本集合，这样就得到另一个聚类簇。一直运行到所有核心对象都有类别为止。

DBSCAN 聚类算法描述

1）初始化核心对象集合 Ω=∅，初始化聚类簇数 k=0，初始化未访问样本集合 Γ = D，簇划分 C=∅；

2）对于 j=1,2,...m，按下面的步骤找出所有的核心对象：

a)通过距离度量方式，找到样本 x_j 的 ε-邻域子样本集 $N_\varepsilon(x_j)$；

b) 如果子样本集样本个数满足$|N_\varepsilon(x_j)|$≥MinPts，将样本 x_j 加入核心对象样本集合：Ω=Ω∪{xj}；

3）如果核心对象集合 Ω=Ø，则算法结束，否则转入步骤 4；

4）在核心对象集合 Ω 中，随机选择一个核心对象 o，初始化当前簇核心对象队列 Ω_{cur}={o}，初始化类别序号 k=k+1，初始化当前簇样本集合 C_k={o}，更新未访问样本集合 Γ=Γ−{o}；

5）如果当前簇核心对象队列 Ω_{cur}=Ø，则当前聚类簇 C_k 生成完毕，更新簇划分 C={C_1, C_2,..., C_k},更新核心对象集合 Ω=Ω−C_k，转入步骤 3；

6）在当前簇核心对象队列 Ω_{cur} 中取出一个核心对象 o'，通过邻域距离阈值 ϵ 找出所有的 ε-邻域子样本集 $N_\varepsilon(o')$，令 Δ=$N_\varepsilon(o')$∩Γ,更新当前簇样本集 C_k=C_k∪Δ，更新未访问样本集合 Γ=Γ−Δ,更新 Ω_{cur}=Ω_{cur}∪(Δ∩Ω)−o'，转入步骤 5。

DBSCAN 聚类算法在模拟数据中的一般实现

1）计算相邻的数据点的距离

```
def dist(p1, p2):
    return ((p1[0]-p2[0])**2+ (p1[1]-p2[1])**2)**(0.5)
```

2）随机生成约 100 个坐标点

```
all_points=[]
for i in range(100):
    randCoord = [random.randint(1,50), random.randint(1,50)]
    if not randCoord in all_points:
        all_points.append(randCoord)
```

3）取领域的最大半径 Eps = 8， 阈值 minPts = 8

```
E = 8
minPts = 8
```

4）找出核心点

```
other_points =[]
core_points=[]
plotted_points=[]
for point in all_points:
    point.append(0)
    total = 0
    for otherPoint in all_points:
        distance = dist(otherPoint,point)
        if distance<=E:
            total+=1

    if total > minPts:
        core_points.append(point)
        plotted_points.append(point)
    else:
        other_points.append(point)
```

5）找出边界点

```
border_points=[]
for core in core_points:
    for other in other_points:
        if dist(core,other)<=E:
            border_points.append(other)
            plotted_points.append(other)
```

6）执行算法

```
cluster_label=0
for point in core_points:
    if point[2]==0:
        cluster_label+=1
        point[2]=cluster_label

    for point2 in plotted_points:
        distance = dist(point2,point)
        if point2[2] ==0 and distance<=E:
            print (point, point2)
        point2[2] =point[2]
```

7）给点加上簇标签

```
cluster_list = defaultdict(lambda: [[],[]])
for point in plotted_points:
    cluster_list[point[2]][0].append(point[0])
    cluster_list[point[2]][1].append(point[1])

markers = ['*','+','.','d','^','v','>','<','p']
```

8）画出簇

```
i=0
print (cluster_list)
for value in cluster_list:
    cluster= cluster_list[value]
    plot(cluster[0], cluster[1],markers[i])
    i = i%10+1
```

9）画出噪声点

```
noise_points=[]
for point in all_points:
    if not point in core_points and not point in border_points:
        noise_points.append(point)
noisex=[]
noisey=[]
for point in noise_points:
    noisex.append(point[0])
    noisey.append(point[1])
plot(noisex, noisey, "o")
title(str(len(cluster_list))+" clusters created with E ="+str(E)+" Min Points="+str(minPts)+" total points="+str(len(all_points))+" noise Points = "+ str(len(noise_points)))
axis((0,60,0,60))
show()
```

从图中可以清楚地看出 DBSCAN 聚类算法的原理。