邻域半径和密度阈值对DBSCAN的聚类结果影响

樊泽羲 2200010816

问题定义

考虑如下的聚类任务：

$$
\begin{aligned}
&Dataset: D = \{x_i\}_{i=1}^N && (1)\\
&Purity\ Metrics: \mathscr{S}: All\ subsets\ of\ D \to \mathbb{R}_{\geq 0} && (2)\\
&Difference\ Metrics: \mathscr{D}: All\ partitions\ of\ D \to \mathbb{R}_{\geq 0} && (3)\\
&Cluster\ Score: S(\mathscr{S}, \mathscr{D}): All\ partitions\ of\ D \to \mathbb{R}_{\geq 0} && (4)\\
&Goal: \{D_j\}_{j=1}^K = argmax_{All\ partitions\ of\ D} S(\mathscr{S}, \mathscr{D}) && (5)
\end{aligned}
$$

在DBSCAN模型中，数据集D中的全部数据点被分类为核心点、密度相连点和噪声，其具体定义如下：[1]

假设给定邻域半径ϵ和密度阈值$minPts$

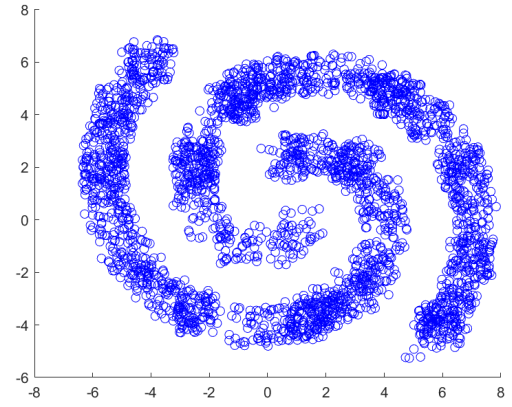
1. $\forall p \in D$，若$U_\epsilon(p)$内至少有$minPts$个样本点，则将p标记为核心点。此时，所有$U_\epsilon(p)$内的样本点被称为p密度可达的
2. 如果存在一条道路$p_1, \ldots, p_n$，有 $p_1 = p$和$p_n = q$， 且每个 p_{i+1} 都是由 p_i 密度可达的(道路上除了q以外所有点都一定是核心点)，则称q 是由 p 密度相连的
3. 所有不与任何点密度相连的点都被称为噪声
4. 对于每个核心点，所有由它密度相连的点分为一个簇

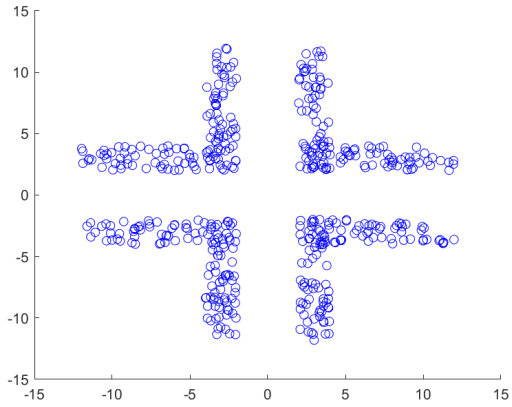
由上述定义可见，DBSCAN的分类结果依赖于超参数邻域半径ϵ和密度阈值$minPts$，不恰当的参数选择可能会导致大量样本被标记为噪声，或者相反地，被粗略地分到不具有代表性的少数类别中。为了保证算法的稳健性，针对二者的敏感度分析无疑是必要的

在本篇报告中，我们用matlab实现了DBSCAN算法，在两个人工生成的数据集twospirals和corners上[2]测试了不同ϵ和$minPts$下的聚类结果，并从聚类数量、分类用时、轮廓值三个角度进行了比较。这将有助于理解DBSCAN对参数选择的敏感程度，并帮助设计其为特定模式数据集执行聚类任务时的超参数设计

数据预处理

我们采用人工生成的数据集twospirals和corners，二者的散点图绘制如下：





我们选择这两个数据集的理由如下：

1. 簇的形状：twospirals和corners具有良好非球形形状，而DBSCAN的一个关键优势恰恰是是它能够检测任意形状的簇。这些数据集允许我们测试此功能
2. 簇的界限：twospirals的两个簇之间有明确的分离，这使得它们的识别很简单。corners的簇在端点处相交，需要一个小的ϵ来区分它们，这有助于分析敏感性
3. 密度变化：在twospirals的每个簇内，密度沿螺旋长度变化。corners具有近似均匀的高密度区域，这些区域被低密度分界面分隔开来，这可以展示DBSCAN在内部密度变化数据集下的表现
4. 低维度：作为2D数据集，它们可以很容易地绘制和可视化。这对于在不同的超参数设置下验证聚类结果很有用，高维数据可能会使得这种检查变得困难
5. 经典性：两个数据集都是经典的聚类问题。使用它们有助于与之前的研究进行直接比较，并帮助建立所实现的DBSCAN方法的可靠性

数据集的具体生成方法参见附录。生成二维数据集之后，我们将其作为数值矩阵导入matlab并命名为data，下面我们将具体展示算法的具体设计以及各个超参数组合下的DBSCAN的聚类表现

算法设计

我们的DBSCAN算法实现为matlab函数my_DBSCAN，其具体运行过程如下：

Algorithm：DBSCAN聚类（采用$Euclid$距离）

输入：数据$data$, 邻域半径ϵ, 密度阈值$minPts$

输出：类别标记idx, 噪声判别$noise$

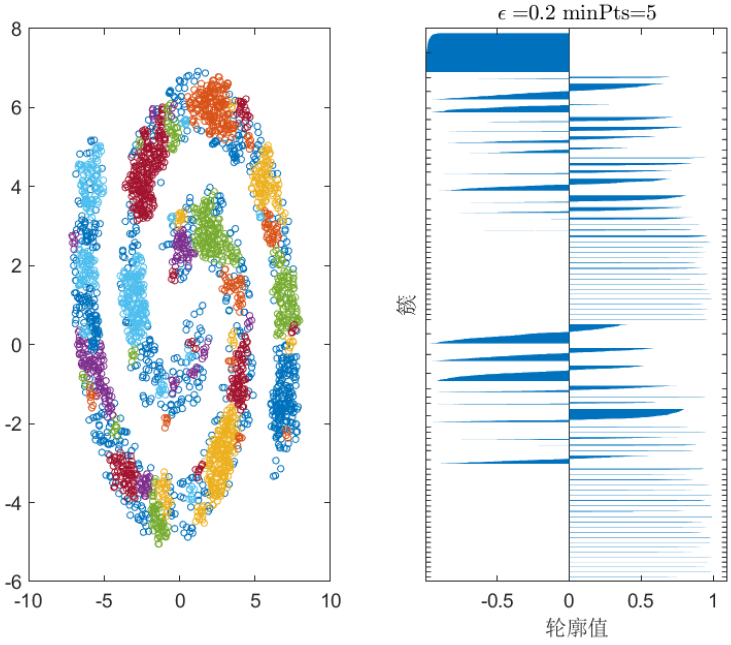
1. 初始化变量：总样本量$datasize$，类别标记idx，访问判别$visited-or-not$，噪声判别$noise$
2. 计算$Euclid$距离：用$pdist2$函数计算各个任意两个样本点之间的距离，这样之后就可以用查找代替距离计算，降低复杂度

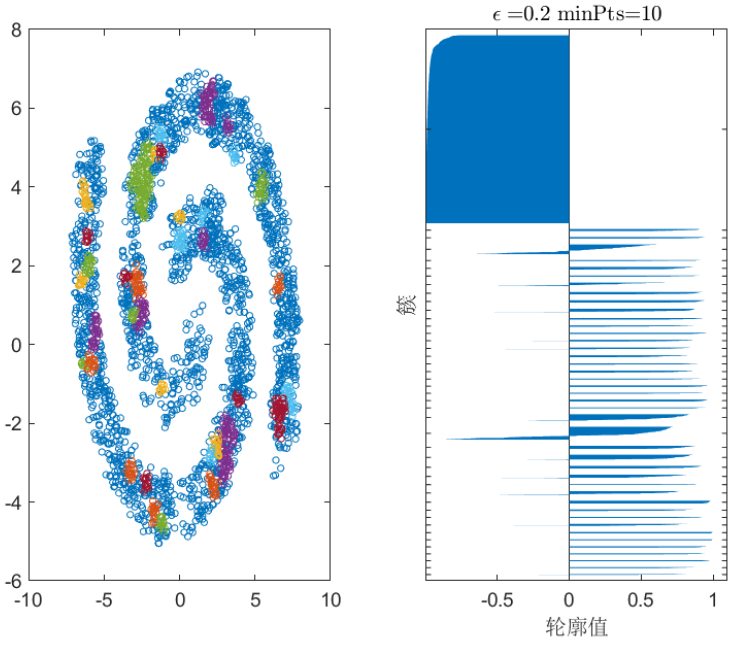
3. 核心点判别：对于每个未访问的样本点p，调用$regionQuery$函数通过距离矩阵寻找$U_\epsilon(p)$，其邻域内样本点数量记为$neighbors$。若$neighbors < minPts$，将p标记未噪声，否则，生成一个新簇，调用$ExpandCluster$函数
4. 簇扩展：$ExpandCluster$收到调用指令后，先给予核心点新的类标记。然后，用$while$循环依次访问$U_\epsilon(p)$内的每个未被访问的样本点，再次使用$regionQuery$获取扩展后的邻域，如果该样本点是核心点，则将其邻域更新进p的可达列表。如果所有邻域内的点都被处理，则将p的可达列表中的点都标记为p的类别
5. 返回结果：所有样本点都访问后，返回更新后的类别标记idx和噪声标记$noise$

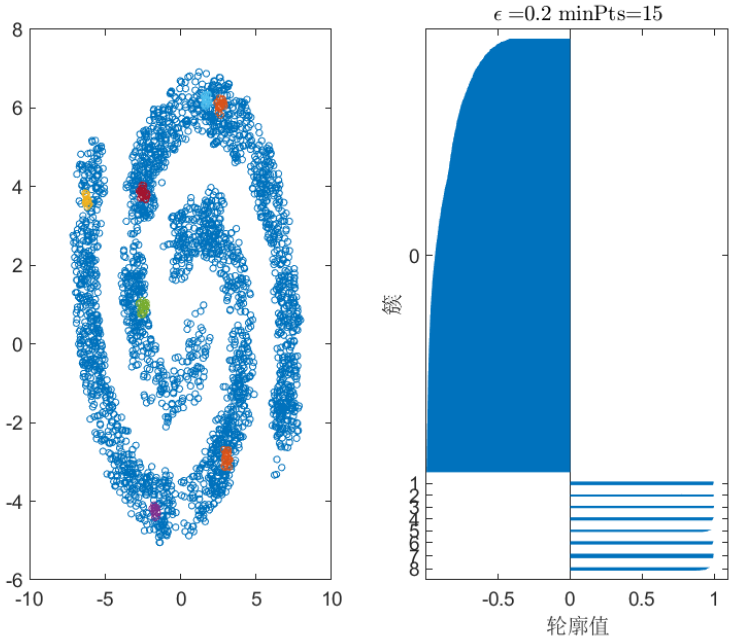
实验结果

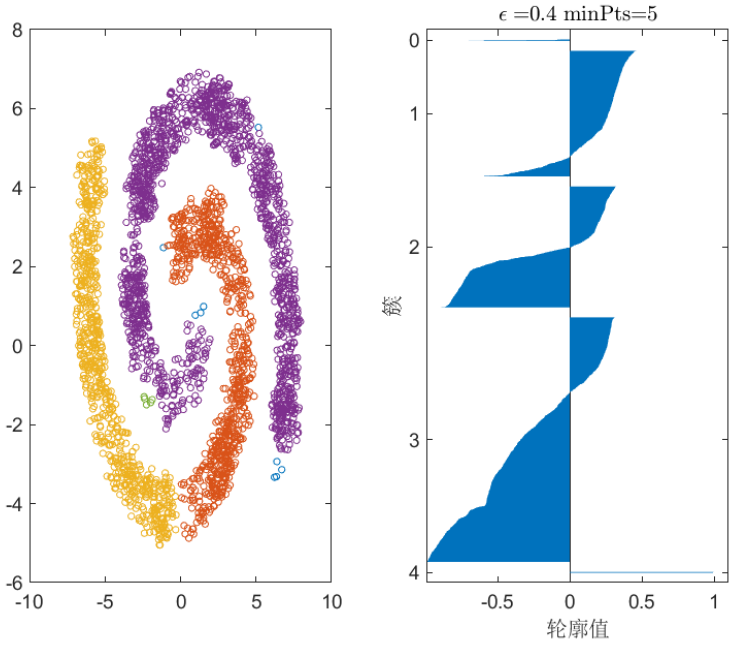
Twospirals

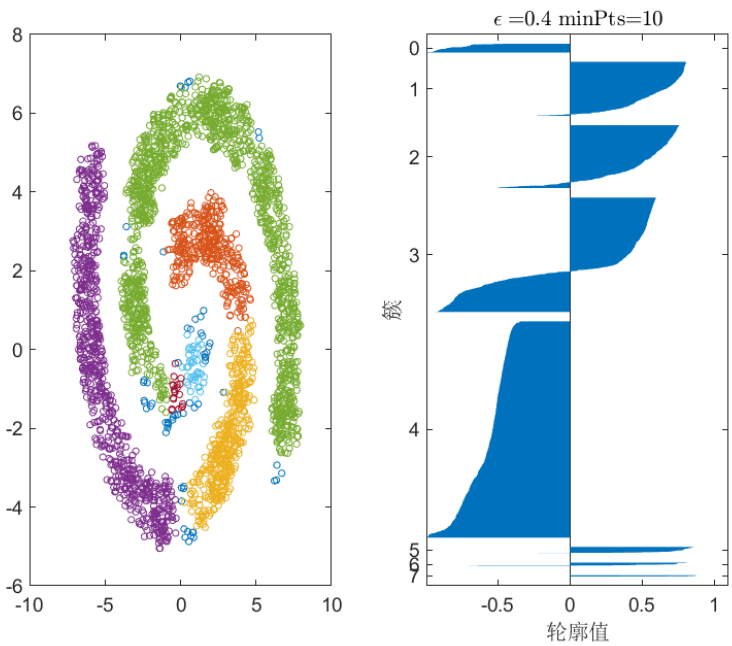
分类结果及轮廓系数：

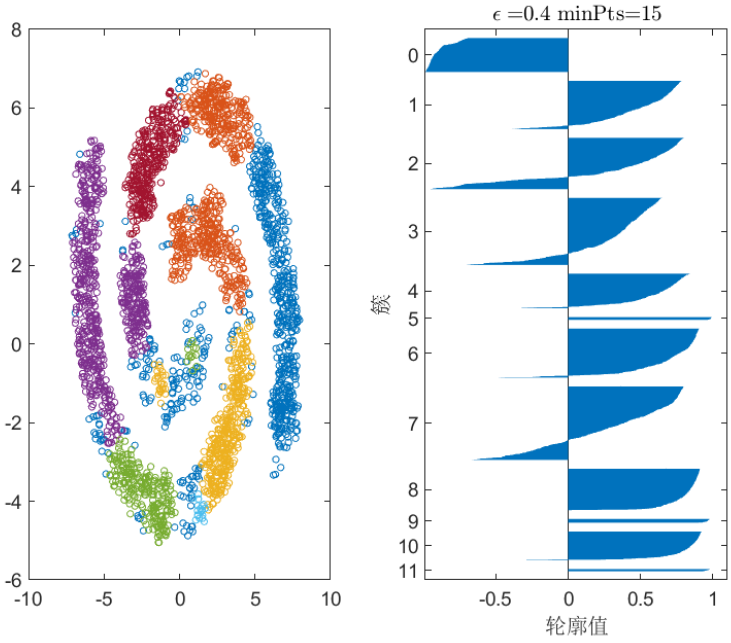


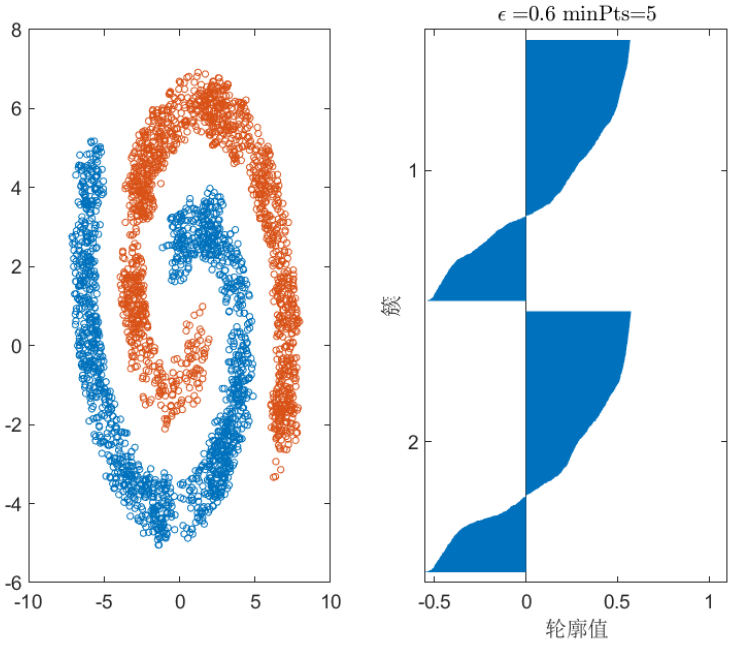


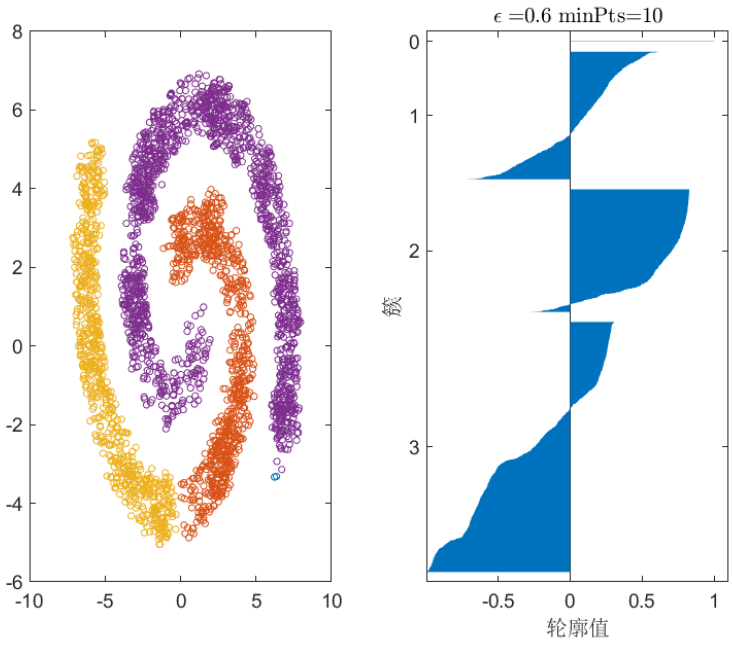

ϵ =0.2 minPts=15
簇
轮廓值

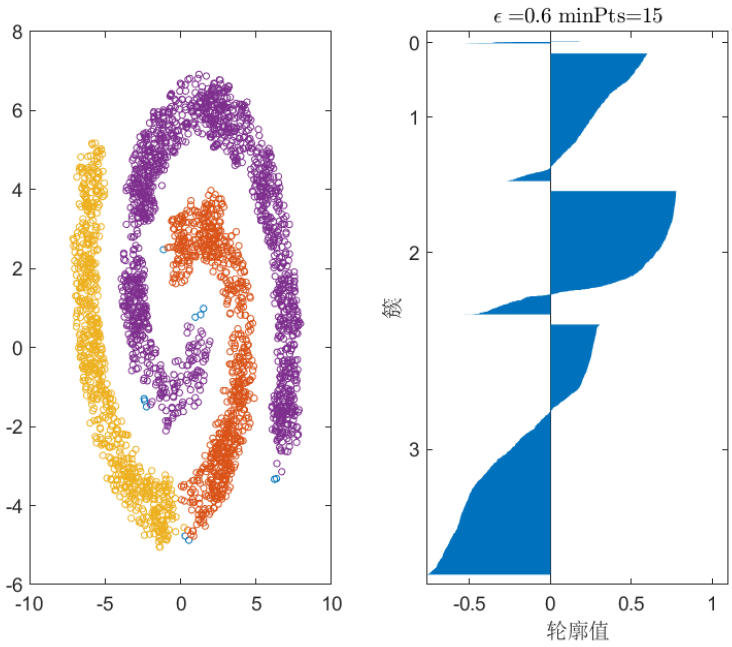

ϵ =0.4 minPts=5
簇
轮廓值


ϵ =0.4 minPts=10
簇
轮廓值

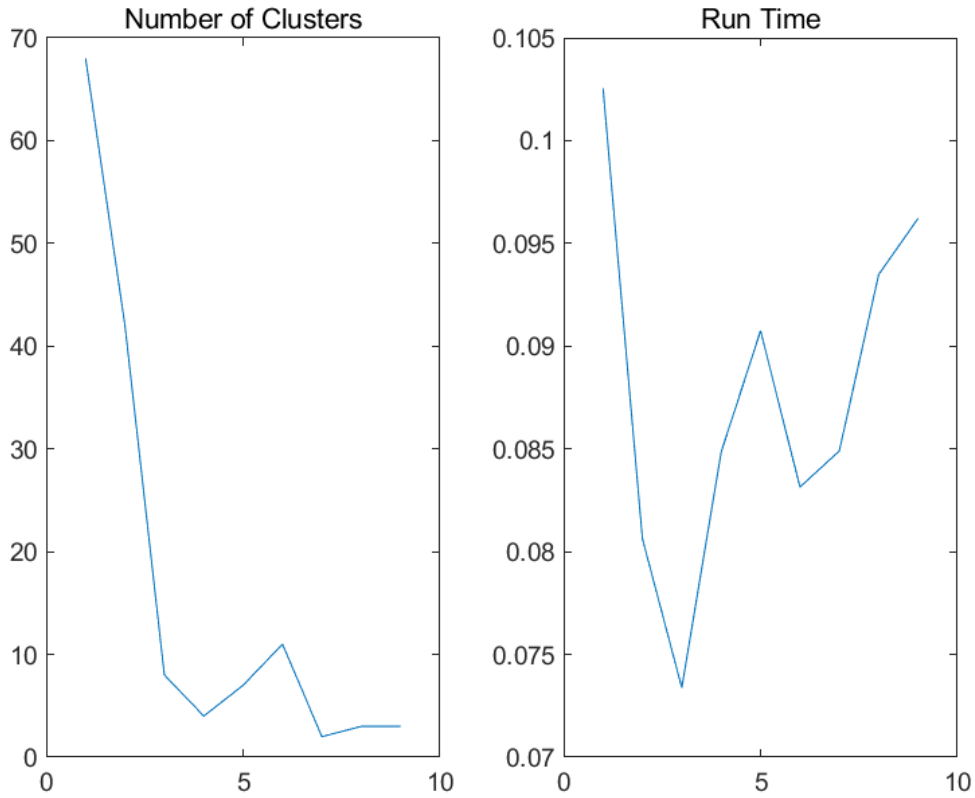

ϵ =0.4 minPts=15
簇
轮廓值


ϵ =0.6 minPts=5
簇
轮廓值





聚类数目及运行用时：



1. $\epsilon = 0.2$

此时大量样本点被划分到单独的簇，由于簇较小，大多数簇具有较高的轮廓系数。同时，随着$minPts$的改变，被分为噪声的类别显著增多，当$minPts$达到15时，绝大多数点都被分为了噪声，说明twospirals集在$\epsilon = 0.2$的尺度下相对稀疏

2. $\epsilon = 0.4$

此时样本点基本被分为$5 - 20$簇，并且随着$minPts$的增加逐渐增多。相较前者，聚类效果有了显著的改进，但是轮廓系数仍然摇摆不定，可能是因为簇的形状相互缠绕，不利于用基于距离的指标给予衡量。特别地，可以观察到，在下方螺线的分界处有一片稀疏区域，所有的三种情况均在这里将簇截断，但这不是我们想要的，说明ϵ还有继续改大的余地

3. $\epsilon = 0.6$

此时样本点被分为了$4 - 10$簇，但是由左侧图像可见，主要的簇只有两三个左右。当$\epsilon = 0.6, minPts = 5$时，聚类最为理想，DBSCAN恰好将两个螺线清晰地勾勒出来。之后随着$minPts$的增大，出现了多余的类别以及前述的截断现象。从轮廓稀疏来看，主要簇的得分均相对理想，基本实现了聚类任务的目标

基于上述观察我们得出以下结论：

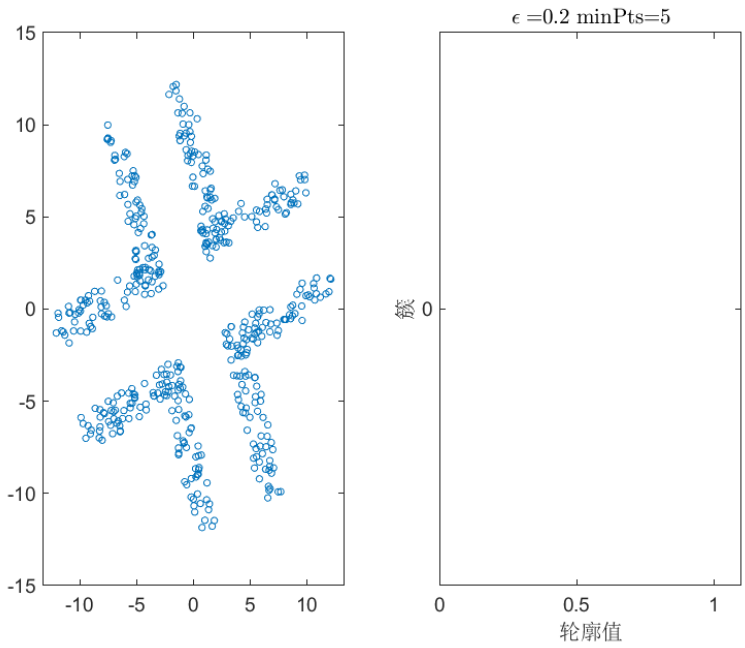
1. ϵ参数对识别出的簇的数量和组成有很大的影响。较小的ϵ能更好地区分内部簇，而较大的ϵ能导致合并。换言之，ϵ决定了邻域考虑的尺度，因此它的值需要适应簇内密度的自然变化，以实现准确的分组

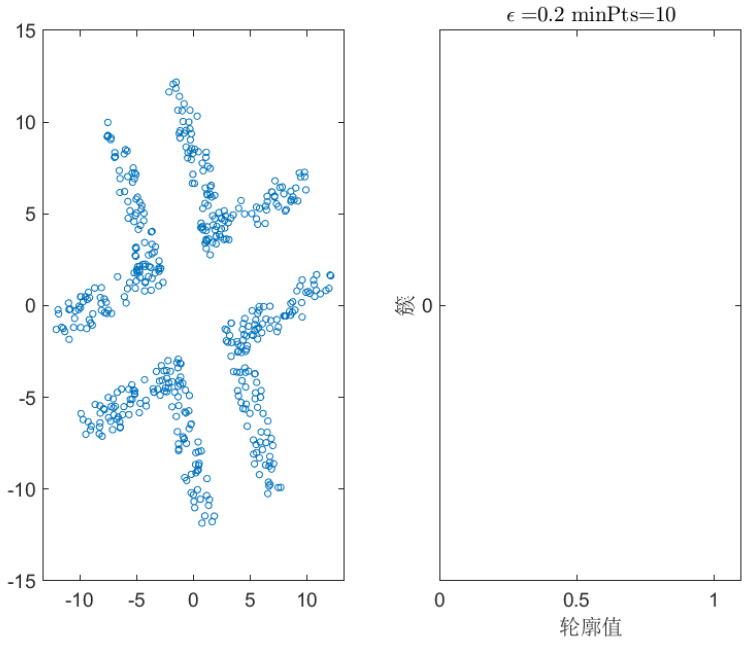
2. $minPts$的影响相对较小，但增加$minPts$时，由于一些稀疏区域的点的邻域大小低于阈值，它们会被整合到不同的簇中。但是，对于像螺旋形这样具有明显聚类分离的数据，$minPts$可能没有ϵ那么重要，因为ϵ刻画了邻域的拓扑结构

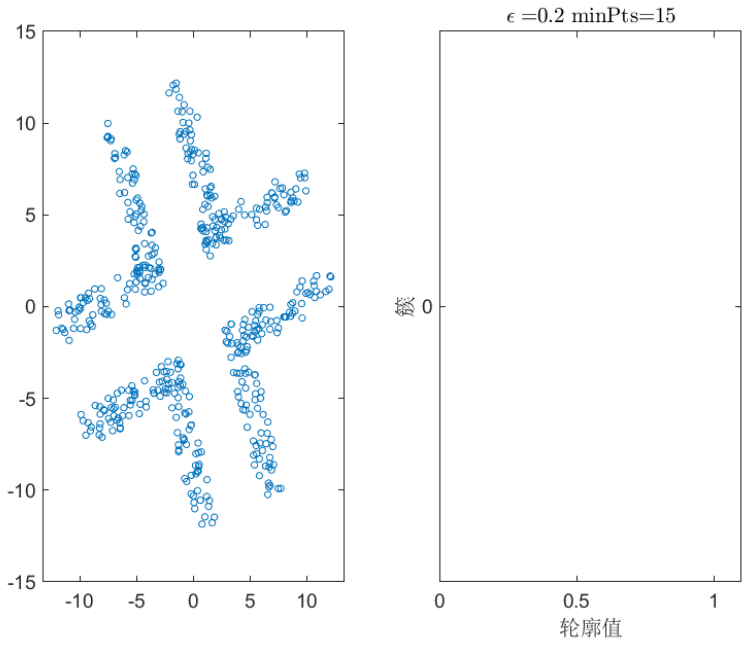
综上所述，ϵ在指定正确的邻域规模的基础上，对提取的簇的数量和粒度具有主导控制作用。$minPts$对凝聚准则进行微调，但在大多数情况下对聚类的影响较小。因此，适当的ϵ设置对于DBSCAN发现有效的内在分组是最重要的

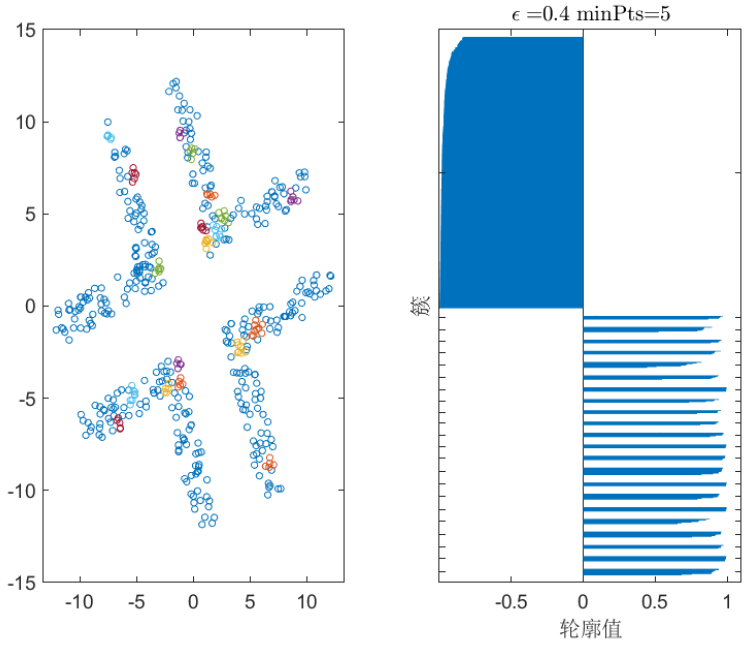
Corners

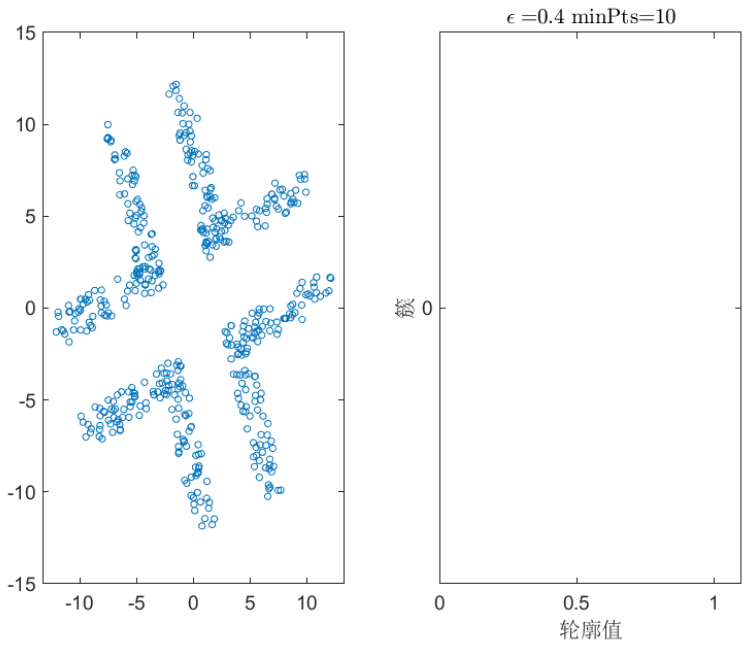
分类结果及轮廓系数：

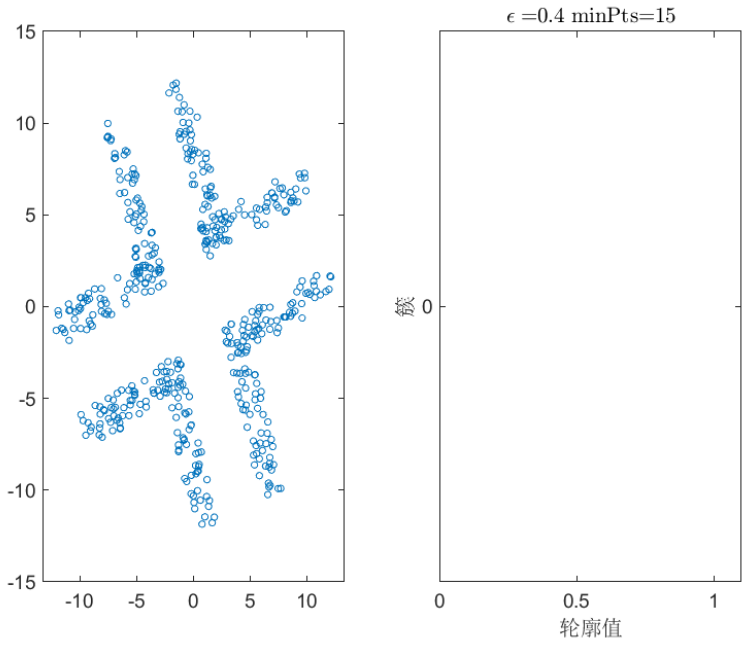


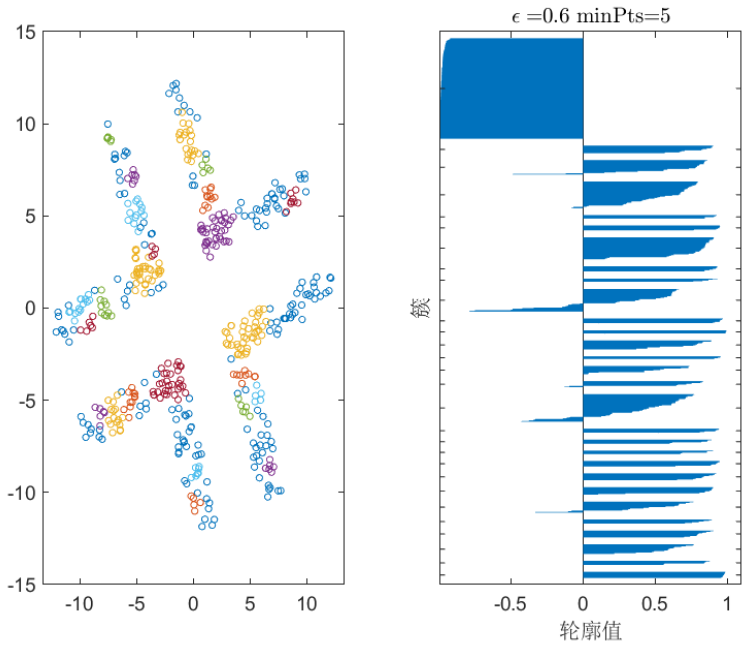


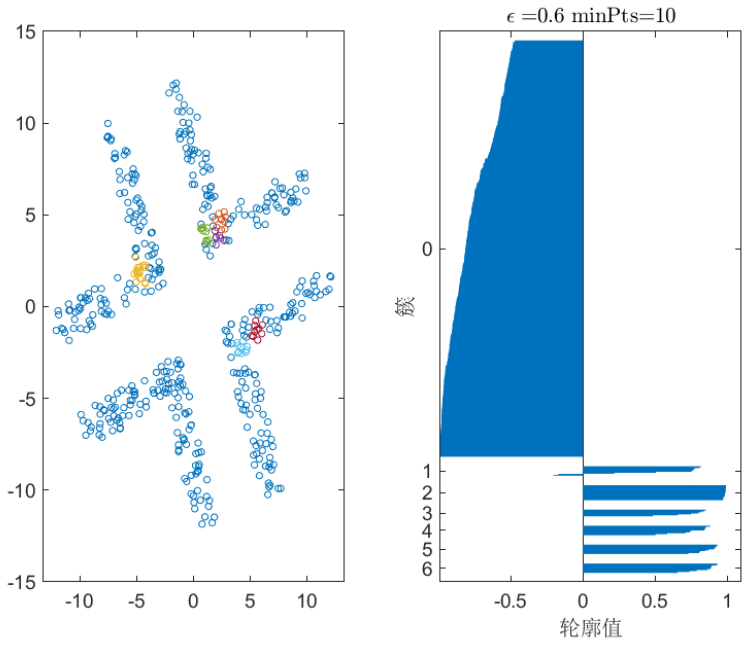

ϵ =0.2 minPts=15
簇
0
0
0.5
1
轮廓值

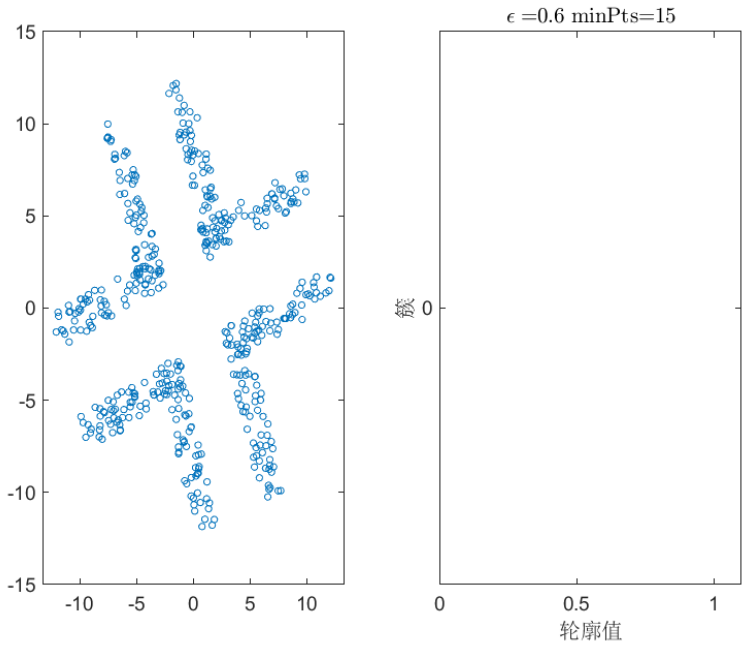

ϵ =0.4 minPts=5
簇
-0.5
0
0.5
1
轮廓值


ϵ =0.4 minPts=10
簇
0
0
0.5
1
轮廓值

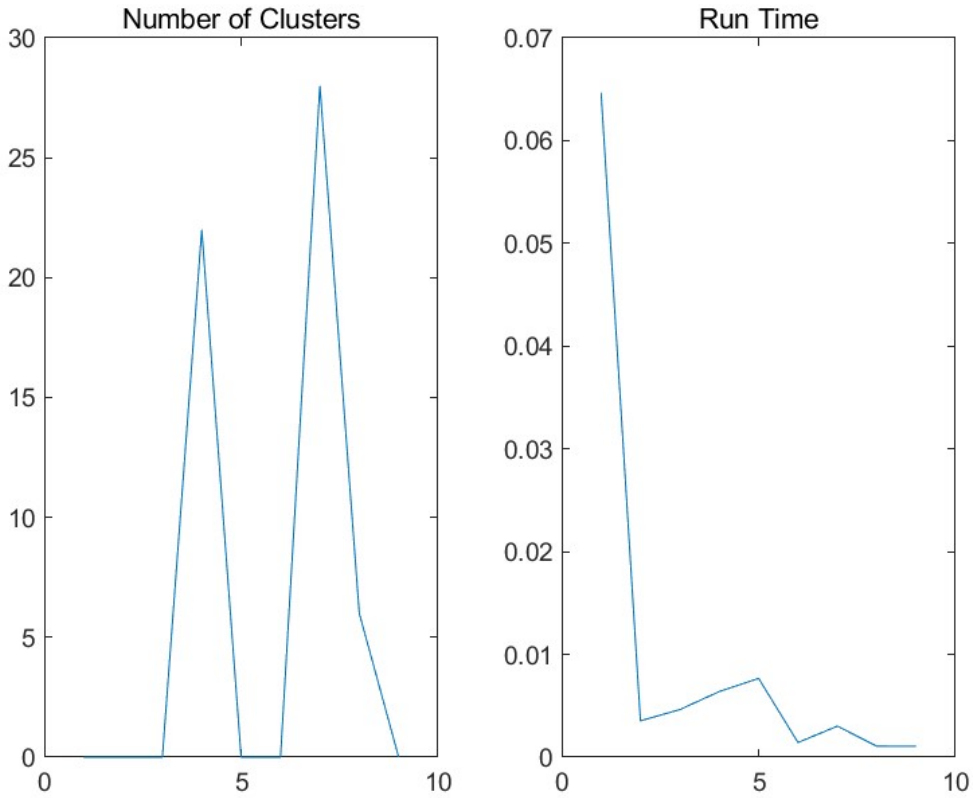

ϵ =0.4 minPts=15
15
10
5
0
-5
-10
-15
-10
-5
0
5
10
簇
0
0
0.5
1
轮廓值


ϵ =0.6 minPts=5
15
10
5
0
-5
-10
-15
-10
-5
0
5
10
簇
-0.5
0
0.5
1
轮廓值





聚类数目及运行用时：



从图上可以明显地看见，相较于twospirals数据集，DBSCAN在corners数据集上的表现明显较差

1. $\epsilon = 0.2$

无论如何调整$minPts$，均无法正常进行聚类，所有点均被标记为噪声

2. $\epsilon = 0.4$

当ϵ和$minPts$均较小时，模型出现了分簇现象，但是几乎每个非噪声点都单独成簇，这些簇因为规模极小导致其轮廓系数很高。这样的结果可能是由于四个直角部分与其它部分密度相差过大，导致DBSCAN无法在同一个尺度上有效聚类

3. $\epsilon = 0.6$

基本与$\epsilon = 0.4$的状况相似，这进一步佐证了我们的猜测。值得注意的是，在$minPts = 10$时，$\epsilon = 0.4$的情况下全部分为噪声，无法聚类，但是在$\epsilon = 0.6$的情况下，实现了聚类（虽然类别过多，效果不好），说明二者间存在一定的对抗关系，这与我们的直觉是一致的

通过比较两个数据集上的表现，可以对差异的原因做出如下分析：

1. 簇的界限：两个螺旋簇之间边界清晰，分隔得很好，使得簇的划分很直观。相反，角点簇的接触范围很窄，需要非常小的ϵ来区分它们
2. 簇的密度：在每个螺旋簇内，密度沿螺旋臂逐渐变化。然而，角点簇已经界限分明地划分出密度的高低区域，这无疑是一个更困难的问题
3. $minPts$的影响：对于螺旋，即使$minPts$不同，基于核心点的扩展也能跨越整个集群。但在角点簇，$minPts$可能直接决定是否将转折点和延伸出的两条直角边划分为同一个簇
4. 噪声处理：与螺旋不同，在角点数据集中，一些标记为噪声的点会显著影响其分割。DBSCAN对拓扑结构更为简单的数据集(如螺旋)中对噪声更稳健

因此，与corners数据集相比，分离良好的螺旋簇内逐渐变化的密度更符合DBSCAN基于密度的拓扑假设，从而在twospirals数据集上取得了相对更好的性能

结论

在本篇报告中，我们用matlab实现了DBSCAN算法，在两个人工生成的数据集twospirals和corners上测试了不同ϵ和$minPts$下的聚类结果，并从聚类数量、分类用时、轮廓值三个角度进行了比较。

主要发现包括:

1. ϵ决定了邻域的定义，主要影响簇的数量和粒度。它的设置应当与簇内部的自然密度相一致，并且对于簇是否能忠实地表示概念至关重要。
2. $minPts$细化了ϵ邻域内进行的抉择并对簇的边界较为敏感。它对聚类稳定性的影响总体上小于控制扩展模式的ϵ但是在个别情况下仍可影响聚类结果
3. DBSCAN偏爱界限分明的、密度逐渐变化的数据，如螺旋，这样的数据与DBSCAN用邻域堆叠出簇的拓扑假设更一致。在这样的数据上面，算法表现得比在复杂的集群结构上面更加稳健
4. 噪声不成比例地影响复杂的集群结构和表现良好的数据集，对前者的分割影响程度显著地高于后者

综上所述，这两个测试证实了DBSCAN确实具有非凸聚类的独特威力，然而，这种威力的发挥是取决于是否明智地规定了匹配簇密度的超参数ϵ，并小心地选择了与边缘适配的$minPts$，这两个超参数对于DBSCAN算法的表现无疑是至关重要的

附录

1. DBSCAN主程序：my_DBSCAN.m

```
function [idx, noise] = my_DBSCAN(data, epsilon, minPts)

    bundle = 0;
    datasize = size(data,1);
    idx = zeros(datasize,1);

    distmat = pdist2(data,data);

    visited_or_not = false(datasize,1);
    noise = false(datasize,1);


    for index = 1 : datasize
        if visited_or_not(index) == false
            visited_or_not(index) = true;

            Neighbors = regionQuery(index, epsilon);
            if numel(Neighbors) < minPts
                noise(index) = true;
            else
                bundle = bundle + 1;
                ExpandCluster(index, Neighbors, bundle, epsilon, minPts)
            end
        end
    end

    function ExpandCluster(index, Neighbors, cluster, eps, minPts)
        idx(index) = cluster;
        temp = 1;

        while true
            neighbor = Neighbors(temp);
```

```
            if visited_or_not(neighbor) == false
                visited_or_not(neighbor) = true;
                NeighborsSecond = regionQuery(neighbor, eps);
                if numel(NeighborsSecond) >= minPts
                   Neighbors = [Neighbors NeighborsSecond];
                end
            end

            if idx(neighbor) == 0
                idx(neighbor) = cluster;
            end

            temp = temp + 1;
            if temp > numel(Neighbors)
                break;
            end
        end
    end

    function Neighbors = regionQuery(i, eps)
        Neighbors = find(distmat(i,:)<=eps);
    end
end
```

2. 测试DBSCAN并可视化结果：DBSCAN.m

```
clear all;
%twospirals
sp=twospirals(200,360,50,1.5,15);
%corners
% cor=corners(500);
% data=cor(:,1:2);
data=sp(:,1:2);
figure();
scatter(data(:,1),data(:,2),36,'blue','o')
saveas(gcf,'spiral_uncluster','png');
% saveas(gcf,'corner_uncluster','png');
% Specify range of epsilon and minPts values to test
epsilon_values = [0.2 0.4 0.6];
minPts_values = [5 10 15];
% Loop through parameter combinations
results = [];
for i = 1:length(epsilon_values)
   epsilon = epsilon_values(i);
   for j = 1:length(minPts_values)
       minPts = minPts_values(j);
       tic;
       % Run DBSCAN
       labels = my_DBSCAN(data, epsilon, minPts);

       % Store clustering results
       nClusters = max(labels);
       runtime = toc;
       results = [results; nClusters runtime];

       % Plot clusters on PCA projection for visualization
       coeff = pca(data);
```

```
        pcaData=data*coeff;
        figure();
        subplot(1,2,1);
        gscatter(pcaData(:,1),pcaData(:,2),labels,[],"o",3,'off')
        subplot(1,2,2);
        silhouette(data,labels)
        title(['$\epsilon=$' num2str(epsilon) ' minPts='
num2str(minPts)],'Interpreter','latex')
        name=sprintf('spiral_cluster_%d_%d',i,j);
        % name=sprintf('corner_cluster_%d_%d',i,j);
        saveas(gcf,name,'png')
    end
end

% Analyze results
figure;
subplot(1,2,1);
plot(results(:,1));
title('Number of Clusters');
subplot(1,2,2);
plot(results(:,2));
title('Run Time');
saveas(gcf,'experiment_result','png')
```

3. 生成twospirals数据：twospirals.m

```
function data = twospirals(N, degrees, start, noise,rep)
% Generate "two spirals" dataset with N*rep instances.
% degrees controls the length of the spirals
% start determines how far from the origin the spirals start, in degrees
% noise displaces the instances from the spiral.
%  0 is no noise, at 1 the spirals will start overlapping

    if nargin < 1
        N = 200;
    end
    if nargin < 2
        degrees = 570;
    end
    if nargin < 3
        start = 90;
    end
    if nargin < 5
        noise = 0.2;
    end

    deg2rad = (2*pi)/360;
    start = start * deg2rad;

    N1 = floor(N/2);
    N2 = N-N1;

    n = start + sqrt(rand(N1,1)) * degrees * deg2rad;
    d1=[];
    for i=1:rep
        d1 = [d1;[-cos(n).*n + rand(N1,1)*noise sin(n).*n+rand(N1,1)*noise
zeros(N1,1)]];
```

```
    end

    n = start + sqrt(rand(N1,1)) * degrees * deg2rad;
    d2=[];
    for i=1:rep
        d2 = [d2;[cos(n).*n+rand(N2,1)*noise -sin(n).*n+rand(N2,1)*noise
ones(N2,1)]];
    end

    data = [d1;d2];
end
```

4. 生成corners数据：corners.m

```
function data = corners(N, scale, gapwidth, cornerwidth)

    if nargin < 1
        N = 1000;
    end
    if mod(N,8) ~= 0
        N = round(N/8) * 8;
    end

    if nargin < 2
        scale = 10;
    end
    if nargin < 3
        gapwidth = 2;
    end
    if nargin < 4
        cornerwidth = 2;
    end

    perCorner = N/4;

    xplusmin = [ones(perCorner,1); -1*ones(perCorner,1); ones(perCorner,1);
-1*ones(perCorner,1)];
    yplusmin = [ones(perCorner,1); -1*ones(2*perCorner,1); ones(perCorner,1)];

    horizontal = [xplusmin(1:2:end) * gapwidth + xplusmin(1:2:end) * scale .*
rand(N/2,1), ...
                  yplusmin(1:2:end) * gapwidth + cornerwidth * yplusmin(1:2:end)
.* rand(N/2,1), ...
                  floor((0:N/2-1)'/(perCorner*.5))];

    vertical = [xplusmin(2:2:end) * gapwidth + cornerwidth * xplusmin(2:2:end)
.* rand(N/2,1), ...
                yplusmin(2:2:end) * gapwidth + yplusmin(2:2:end) * scale .*
rand(N/2,1), ...
                floor((0:N/2-1)'/(perCorner*.5))];

    data=  [horizontal; vertical];

end
```

参考文献

1. DBSCAN - 维基百科，自由的百科全书 (wikipedia.org)
2. DBSCAN/datasets at master · captainjtx/DBSCAN (github.com)