聚类

聚类，属于无监督的机器学习方式。聚类根据未知标签样本的数据集内部的数据特征，将数据集划分成多个不同的类，使得同一类的数据样本尽可能地相似，不同类的数据样本之间相似度尽可能地小。

Kmeans

1. 算法解析

   Kmeans是一种划分式的聚类算法，其基于相似度将数据样本划分到"最近"的类中，而每个类则由其类中心（质心）作为代表，故而实则将每个数据样本划分到与其相似度最大的类中心（质心）所对应的类中。算法主要步骤如下：

   （1）初始化K个类中心

   （2）将数据样本划分到"最近"的类中（与类中心的相似度最大）

   （3）更新类中心

   （4）重复（2）（3）步骤直至满足收敛条件

   在Kmeans的算法中，我们可以看到其划分的标准是相似度，而相似度的度量方式有很多，常见的如下：

   - 闽可夫斯基距离：$d(x,y)=\sqrt[p]{\sum_{i=1}^{v}|x_i-y_i|^p}$

     当p=1时，上式即为街区距离；当p=2时，为欧式距离；

   - 余弦相似度：$cos(x,y)=\frac{x\cdot y}{|x||y|}$

     其用以描述向量之间的相似度时，忽略向量本身的大小

   - Jaccard相似度：$J(A,B)=\frac{|A\bigcap B|}{|A\bigcup B|}$

     一般地，其用于描述两个集合A,B的相似度

   - Person相关系数：$\rho(x,y)=\frac{Cov(x,y)}{\sigma_x\sigma_y}$

     其中$Cov(x,y)$表示x,y的协方差，σ_x表示x的标准差，相关系数实则通过除于标准差来剔除协方差受变化幅度的影响，试图标准化协方差

   - KL散度：$D(p||q)=\sum_x p(x)log\frac{p(x)}{q(x)}$

     可以用以描述两个随机变量的距离；越相似，KL散度（相对熵）越小；当KL散度=0时，两个随机分布相同

   而算法的收敛条件可以是：迭代次数、均方误差等。

   Kmeans的优缺点：

   - 优点：时空复杂度较低；具有优化迭代功能，能修正迭代过程中的误分类；
   - 缺点：需要确定类的个数K；对初始选取的类中心敏感；

2. 算法实现

（1）初始化K个质心

最简单的方案就是随机不重复地进行选择K个数据样本，具体实现可以将数据样本的下标索引进行随机打乱排列，再选择前K个即可：（当然这样做效果会差一些，因为Kmeans对初始质心比较敏感，后面的内容会给出一些优化的策略和思考）

```
%初始化质心
rand('seed',10); %实验的可重复性
temp = randperm(length(X)); %随机打乱
Centeridx = temp(1:K); %K个随机数
Center = X(Centeridx,:); %选取K个质心
```

（2）分配类标

给每个样本点分配类标，实则就是计算样本点到哪个质心的距离最小

```
%分配类标
Cennum(:,:) = 0;
for i = 1:length(X)
    [v,index]=min(sum(abs(X(i,:)-Center),2)); %曼哈顿距离
    C(i)=index;
    Cennum(index) = Cennum(index)+1;
end
```

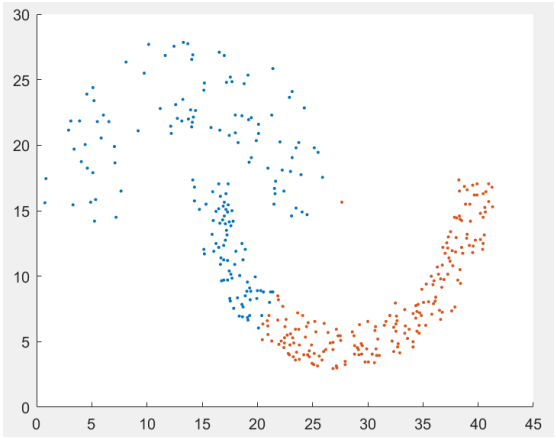
（3）重新计算质心

质心的计算只需取各维度的均值，将这些均值构成一个数据点即认为其为该类的类中心

```
%重新计算质心
for k = 1:K
    Center(k,:) = sum(X(C==k,:),1);
end
Center = Center./Cennum;
```

3. 算法效果及一些优化策略

由上述基础版的Kmeans算法得到的结果比较一般，从下图可见：

Jain	Aggregation
	

由Kmeans算法原理可知，其聚类效果为球状，故前者Jain无法合理地分开；后者Aggregation聚类效果较前者好了不少，但左下方的三个类依然分类地较差。我根据图形的分类情况猜测这和初始质心的选取有关，故打算对初始化K个质心这一算法步骤进行优化，优化的思路便是尽可能使得K个质心分离地开一些：

（1）优化的第一个思路，便是迭代地生成K个质心，每生成1个质心，均要求其距离前面已确定的质心尽可能地远；具体来说，可以随机从数据集中选取第一个质心，后面每一个新的质心，均按距离作为概率从数据集中选取，这个"距离"定义为：该数据点到各个质心距离中的最小值。故而按此步骤下来，可以使得初始的质心有更大的概率变好，此思路即为Kmeans++

（2）第二种思路，即是每次随机选取K个数据样本点，重复log(n)次，对得到的Klogn个样本点进行聚类，聚类得到的K个样本点作为初始质心，此思路即为Kmeans||。但在对Klogn个样本点聚类时若采用Kmeans同样面临初始质心的选取问题，此时便只能采用随机的策略

以下为我对上述第一种优化初始化质心思路的实现：

```
%初始化K个质心
Centeridx = zeros(K,1);
rand('seed',10); %实验的可重复性
first = randi([1,length(X)]); %先随机找第一个质心
Centeridx(1) = first;
for i = 2:K %尽可能选远的K-1个
    distance = zeros(length(X),1);
    for j = 1:length(X)
        distance(j) = min( sum(abs(X(j,:)-X(Centeridx(1:i-1),:)),2) ); %曼哈顿距离最小值
    end
    distance = distance/sum(distance); %归一化距离
    temp = 0;
    while(ismember(temp,Centeridx)) %剔除重复质心
        temp = randsrc(1,1,[1:length(X);distance']); %按概率抽取
    end
    Centeridx(i)=temp;
end
Center = X(Centeridx,:);
```

采用该思路后，Aggregation数据集左下方的两个小类得以区分，整体的聚类情况也理想了很多，但是依然存在不少部分的分类不当：

同样地，跑有类标的数据集，基础版的Kmeans和Kmeans++相比，三个聚类指标如下：

	ACC聚类准确率	NMI标准互信息	PUR聚类纯度
Kmeans	0.6575	0.7050	0.7055
Kmeans++	0.7820	0.7584	0.7970

此外，观察有类标的data数据集，发现数据之间的量纲差距较大，故对原数据集进行z-score标准化再使用Kmeans进行聚类，实现以及效果对比如下：

```
%标准化
for i = 1:size(X,2)
    X(:,i) = (X(:,i)-mean(X(:,i)))/std(X(:,i));
end
```

	ACC聚类准确率	NMI标准互信息	PUR聚类纯度
未标准化	0.6575	0.7050	0.7055
标准化	0.7880	0.7563	0.7975

由上述两组对比可见，对初始化质心的优化和对数据集的标准化均使得Kmeans的性能变优。

DBSCAN

1. 算法解析

   DBSCAN是基于密度的聚类策略，其将簇定义为密度相连的点的最大集合。该算法引入两个参数来定义邻域和密度，能够聚类出任意形状的簇。DBSCAN引入了三个关键的概念和两个相关的参数，这三个概念为：核心点、边界点和噪声点，两个参数分别是：eps半径、Minpts密度阈值：

   - 核心点：其邻域内（距离小于eps）的数据点的个数大于Minpts；
   - 边界点：其不满足核心点的条件，但位于核心点的邻域内；
   - 噪声点：其不满足核心点的条件，且不位于任何一个核心点的邻域内；

   其算法的主要步骤如下：

   （1）计算距离矩阵

   （2）找到核心点

   - 根据距离矩阵和eps找到每个数据点的所有邻居

   - 根据Minpts判断哪些点属于核心点，得到核心点集合

   （3）标签传递

   - 遍历核心点集合，若其未被标记过，则标记为新的类，并将其类标赋给其所有邻居

   - 若该核心点被标记过，则直接将其类标赋给其所有邻居

   DBSCAN的优缺点如下：

   - 优点：不需知道类的个数；能够聚类任意形状；

- 缺点：参数不好确定；当空间密度不均时，聚类效果较差；

2. 算法实现

（1）计算距离矩阵

由于这里我们定义的距离为欧氏距离，故而矩阵是对称的，只需要计算一半即可：

```
%计算距离矩阵
dis = zeros(length(X),length(X)); %距离矩阵
for i = 1:length(X)
    for j = 1:i
        if i==j
            dis(i,j)=0;
        else
            dis(i,j)=sqrt(sum((X(i,:)-X(j,:)).^2,2)); %欧式距离
            dis(j,i)=dis(i,j);
        end
    end
end
```

（2）找核心点

对数据集中每个数据点计算其邻域内的邻居数量，判断其数量是否大于密度阈值，若大于则该数据点为核心点

```
%找核心点
arr = zeros(length(X),length(X)); %每个点的领域点
arr(dis<=eps) = 1;
core = find(sum(arr,2)>=minpts); %核心点
```

（3）标签传递

遍历核心点集合，若核心点标记过，则直接将类标赋给其所有邻居；而若未被标记，则先标记为新的类，再将其类标赋给其所有邻居；最后类标仍未初始类标-1的数据点为噪声点

```
%标签传递
c=1;
for i=1:length(core)
    if C(core(i)) == -1
        C(core(i)) = c; %第一次分配
        C(find(arr(core(i),:)==1)) = C(core(i)); %标签传递
        c = c+1;
    else
        C(find(arr(core(i),:)==1)) = C(core(i)); %标签传递
    end
end
```

但是对于DBSCAN算法而言，两个参数半径eps、密度阈值minpts起到了较为重要的作用，而这对参数值的大小关系搭配较难把握，调参存在较大的难度，故而在查阅相关网上资料和论文后，采用了以下的调参策略：

（4）K距离法确定参数

通过k-距离来确定DBSCAN的参数问题，属于一种经验法，能够较快地得到较为理想的参数，但不一定能保证找到的参数就能使DBSCAN跑出好的聚类效果。其步骤是这样的：

①首先找到各个数据点第k近的距离值

②将这些k距离按升序进行排序

③拟合升序后的k距离值为一条曲线

④曲线急剧变化的地方对应的k距离值即为eps半径参数，而此时的k即为minpts密度阈值

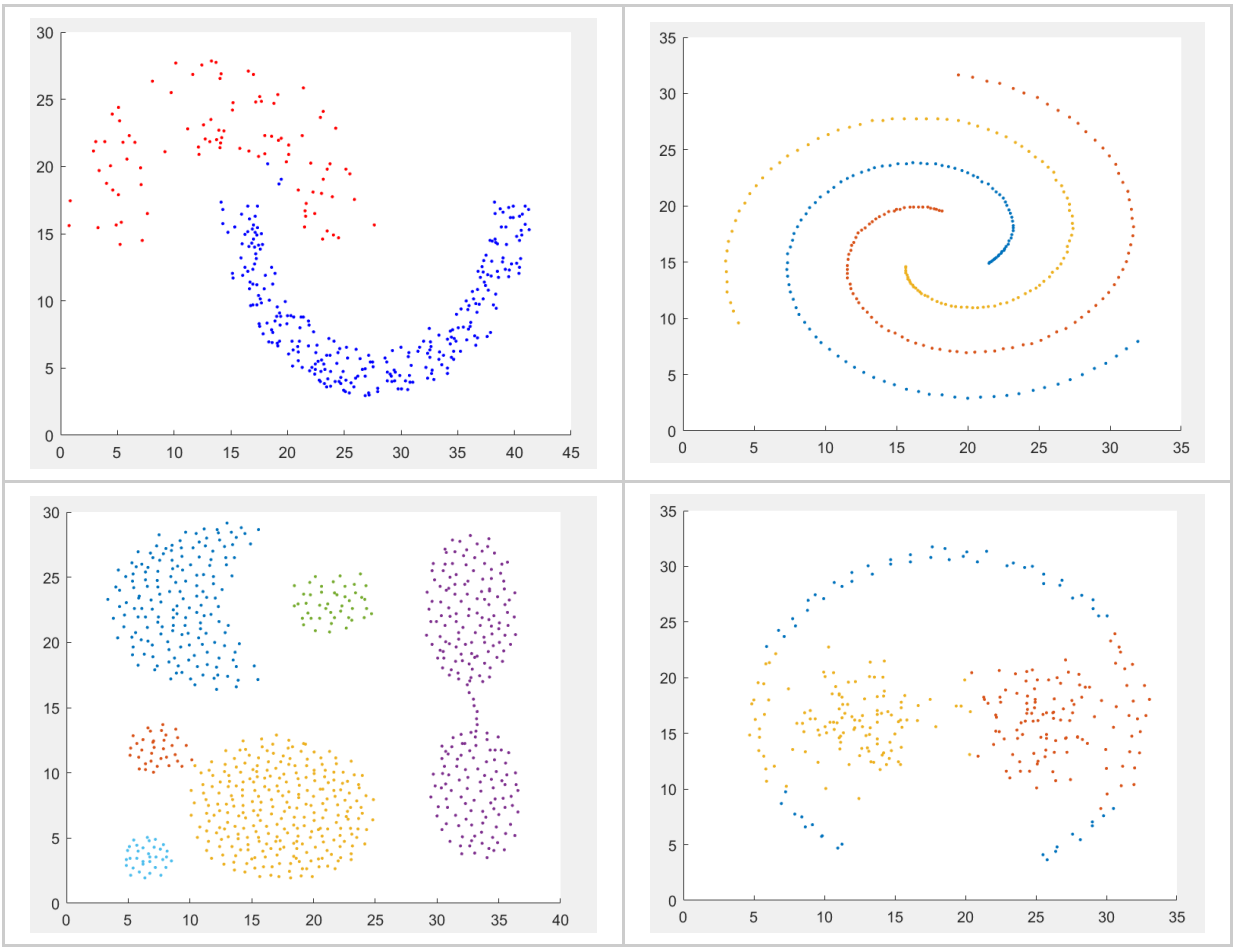
```
%k-距离确定参数
sdis = sort(dis,2); %排序
max_dif = 0;
for k = 1:KK
    kcolumn = sort(sdis(:,k)); %有序k-距离
    [v,p] = max(diff(kcolumn)); %导数最大处
    if v>max_dif
        max_dif = v;
        eps = kcolumn(p);
        minpts = k;
    end
end
```

上述方法为经验法，往往不一定能直接奏效，但可以通过经验法得到eps和Minpts后，通过手动调参对比每次的结果，找到更好的参数组合：

- 控制MinPts不变，Eps过大，簇数变少，很多点都聚到同一个簇中；Eps过小，簇数过多，导致一个簇的分裂
- 控制Eps不变，MinPts过大，簇数过多，可能导致同一簇中的点被判断为噪声点；而MinPts过小，簇数过少，会导致出现大量的核心点

3. 算法效果

应用上述DBSCAN算法到测试集上，效果如下：



可以看到，DBSCAN比起Kmeans在这些数据集上的效果明显要好很多，尤其是上面两张条状线性的图，几乎可以得到很完美的聚类结果；而下方的两张图，聚类效果则只能说是马马虎虎：

- 其中左下图的聚类效果纯度很高，精确率也不错，但是我们人工可能会分成7个类，而其只分成了6个类，有两个簇由于有小量密集的点连接，故而DBSCAN将这两个右边的簇（紫色）归为了同一个簇，这一点可以看作是DBSCAN的劣势之一。
- 此外再来看看右下方的图，本来人工可能会分成3个类，左右两个"眼睛"各一个类，一圈数据点再一个类，而DBSCAN在这张图上没有得到较好的聚类结果，这是由于样本在空间中的密度不均，故而DBSCAN很难得到较好的结果；而此时这种怪异的聚类结果（图中左右下方各有一小撮蓝色的类，上方也有蓝色的类）表面DBSCAN将这些蓝色的点全部当成了噪声点（原因自然是密度不均，这些蓝色点密度相对黄色群体和红色群体较低）

谱聚类 Spectral Clustering

1. 算法解析

谱聚类是基于图论角度的聚类算法。它的主要思想是将每个数据样本看成空间中的点，这些点用边连接起来形成图，相似度高的数据点之间的边权重大，相似度低的数据点之间的边权重小。而聚类的目的就是对该图进行切分，找到"最佳"的切分方式：使得子图内的边权重和尽可能高，子图间的边权重和尽可能低。

以下是谱聚类算法相关的数学基础知识：

- 相似度矩阵W：数据集的图连接表示可以用相似度矩阵来描述，其中相似度用高斯核函数来描述相最为常用：$w_{ij} = e^{-\frac{\|x_i - x_j\|_2^2}{2\sigma^2}}$
- 度矩阵D：度矩阵是一个对角矩阵，只在主对角线上有值：$d_{ii} = \sum_{j=1}^{n} w_{ij}$

- 拉普拉斯矩阵L：$L = D - W$，其为半正定矩阵，对应的n个实数特征值都大于等于0,最小的特征值为0，即$0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$

算法步骤：

（1）根据数据集构建相似度矩阵W

（2）由相似度矩阵W构建度矩阵D

（3）由W和D得到拉普拉斯矩阵L

（4）构建标准化拉普拉斯矩阵$D^{-\frac{1}{2}}LD^{-\frac{1}{2}}$

（5）从第二小的特征值开始找k个最小的特征值对应的特征向量构造$n \times k$维度的特征矩阵F

（6）对特征矩阵F按行进行标准化后，进行Kmeans聚类得到最后的簇划分C

谱聚类的优缺点：

- 优点：聚类效果较为优秀，能够处理稀疏数据；由于有降维思想，计算复杂度要小；
- 局限性：需要类的个数作为参数；聚类效果较为依赖相似度矩阵；相似度衡量中高斯核函数的超参数选取；

2. 算法实现

（1）通过高斯核函数构建相似度矩阵W，由于其为对称矩阵，计算一半即可；特别地，我们忽略自身的相似度，故矩阵中主对角线上的值为0：

```
%构建相似度矩阵
W = zeros(length(X),length(X)); %忽略自身的相似度
for i = 1:length(X)
    for j = 1:i-1
        W(i,j) = exp( - sum((X(i,:)-X(j,:)).*(X(i,:)-X(j,:))) / (2*sigma*sigma) ); %高斯核函
数
        W(j,i) = W(i,j); %对称矩阵
    end
end
```

（2）计算度矩阵D，一个节点的度是其连接的所有边的权重之和，故度矩阵为对角矩阵，只有主对角线上有值，且值为相似度矩阵W中每行数据值之和：

```
%构建度矩阵
D = eye(length(X));
D = D .* sum(W,2); %根据相似度计算度矩阵
```

（3）计算标准化的拉普拉斯矩阵，由度矩阵D和相似度矩阵W可得拉普拉斯矩阵L，再计算 $D^{-\frac{1}{2}}LD^{-\frac{1}{2}}$ 对拉普拉斯矩阵进行标准化：

```
%得到标准化拉普拉斯矩阵
L = D-W;
L = D^(-.5)*L*D^(-.5);
```

（4）构建特征矩阵，计算标准化拉普拉斯矩阵的特征值和特征向量，从第二小的特征值开始选取K个最小的特征值对应的特征向量组成特征矩阵：

```
%得到特征矩阵
[eigVector,eigvalue] = eig(L); %特征向量矩阵 & 特征值矩阵
[eigvalue,index] = sort(diag(eigvalue)); %升序排序
F = eigVector(:,index(2:K+1)); %选取特征向量
F = mynormalize(F); %按行标准化
```

其中，mynormalize()函数是我自己写的一个对特征矩阵F按行进行z-score标准化的函数，目的是消去同一数据点不同维度上量纲的差别：

```
%按行进行标准化
function X = mynormalize(X)
for i = 1:length(X)
    X(i,:) = ( X(i,:)-mean(X(i,:)) )/ std(X(i,:));
end
```

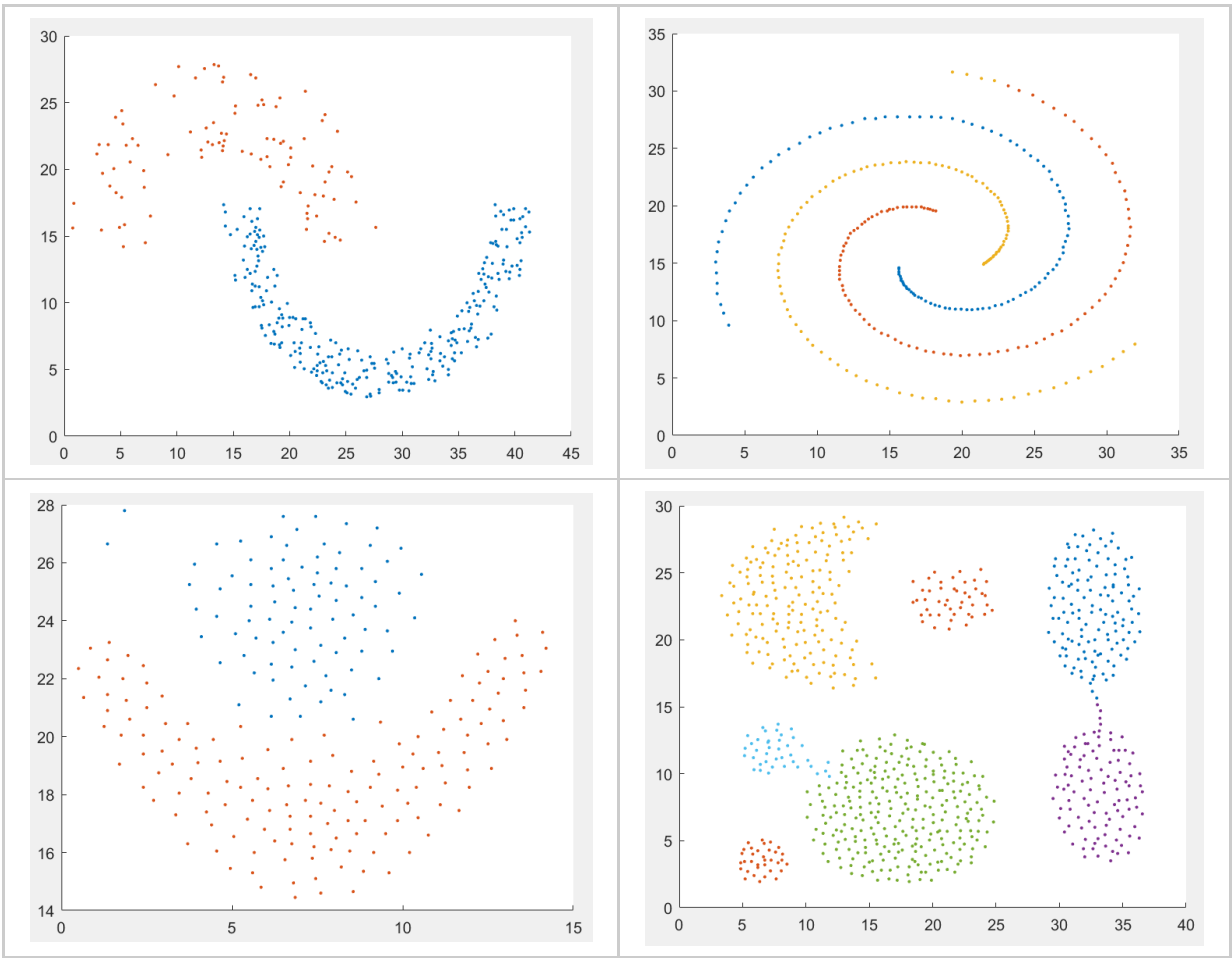
（5）对特征矩阵进行聚类；经由第（4）步后，实则对原数据集做了特征提取的降维处理，故而最后一步只需对得到的特征矩阵按某种方式聚成K类即可，在这里我采用的是Kmeans++的策略：

```
%对特征矩阵进行聚类
rng('default'); %保证实验可重复性
C = kmeans(F,K); % 默认距离 -欧式 默认初始方式 -Kmeans++ 默认最大迭代次数 -100
```

至此，谱聚类的算法实现完毕，让我们来看看它在我们二维测试集上的效果

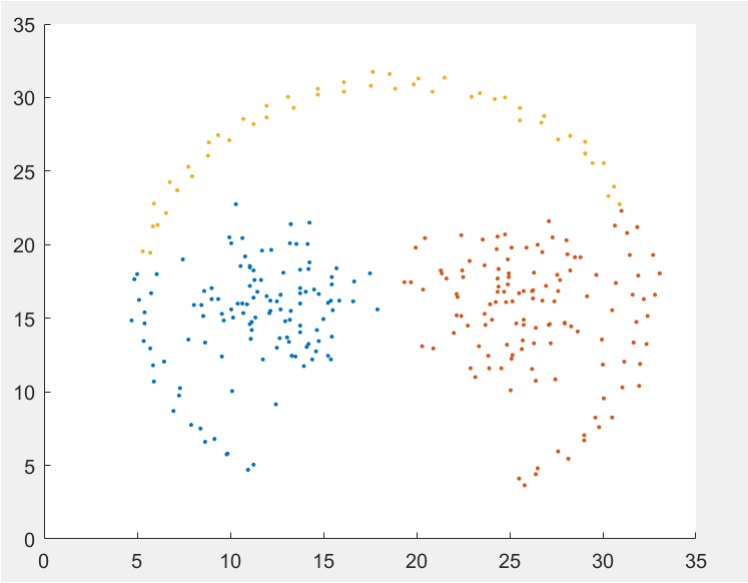
3. 算法效果

谱聚类的效果相对不错，这也使得它在实际中得到了较为广泛的使用，当然它的效果比较依赖于相似度的度量，而我们相似度的度量采用了高斯核函数，里面有一个超参数σ，故而调整到合适的σ值谱聚类才能发挥其更好的效果：（如若采用其他不需参数的相似度，发现效果不如全连接图概念下高斯核函数发挥的效果好）



从上述四辐图可以看到，当超参数σ调整到较为合适的值时，谱聚类Spectral Clustering的效果相当相当地不错，无论是上面两张带状分布的数据集（DBSACN能够得到很好的簇划分，但谱聚类效果更好，实现了我个人人工意义上的完美划分），还是下面两张DBSCAN划分效果相对差一些些的数据集，谱聚类都能得到几乎完美的簇划分。特别地，谱聚类是进行空间映射、特征提取后再进行Kmeans聚类，对比Kmeans上面四图均难以得到较好的簇划分效果可知，进行空间映射、特征提取使得本来只能进行线性划分的Kmeans得以处理非线性的数据集。

但谱聚类用高斯核函数进行相似度衡量时，超参数值的选取是个较大的难题，大多是基于经验值和尝试来进行调参，就如下面这个二维非线性数据集，我无法找到一个较为合适的σ值使得簇划分结果较为理想，不过对比其他数据集能够较快找到合适的超参数σ，可以认为谱聚类在这个极易混淆的数据集上难以有更理想的表现：



而在有类标的数据集上，同样地对σ值依赖性很大，且由于整个data六个特征值量纲有很大的不同，是否做了z-score标准化处理，结果也有较大的区别：（指标为ACC、NMI、PUR三者的均值）

	未标准化	**标准化**
σ=100	0.3334	0.7522
σ=400	0.5823	0.7346

DenPeak

1. 算法解析

   Density peak聚类算法是这四个算法里面最为年轻的算法，其首次于2014年发表在Science上，其很巧妙地将密度和距离结合在了一起，主要提出了下面两个定义：

   - ρ：局部密度；一个数据点$\rho_i = \sum_j \chi\left(d_{ij} - d_c\right)$，$\chi(x) = \begin{cases} 0 & x \geq 0 \\ 1 & x < 0 \end{cases}$

     其中d_c是一个参数，表示邻域的范围，故ρ实则表示其邻域内点的个数

   - δ：距离；一个数据点$\delta_i = min_{j:\rho_j > \rho_i}\left(d_{ij}\right)$

     一个数据点的距离定义为其与最近的密度比它大的数据点之间的距离，特别地，密度最大的数据点的距离定义为$\delta_i = max_j\left(d_{ij}\right)$

   故而整个算法步骤围绕提出的这两个定义可以描述成如下步骤：

   （1）计算所有数据点之间的距离，构造距离矩阵

   （2）选取合适的参数d_c

   （3）根据公式计算每个数据点密度ρ和距离δ

   （4）归一化ρ和δ后计算它们的乘积$\gamma = \rho \cdot \delta$

   （5）根据γ选取合适的类个数和类中心

   （6）将每个数据点归到最近的类中心所在的类

在Science的《Clustering by fast search and find of density peaks》中，①作者提出了对参数d_c的一般化建议：对所有数据而言，d_c能够使得平均邻居个数为总样本数的1%~2%。此时的d_c的值即为作者提出的经验值。此外，②作者还提出了如何更好地选取类个数和类中心，其指出按降序画出γ值的曲线后，非聚类中心的γ较为平滑，类中心和非类中心之间会出现明显的跳跃值，这个数值即可作为截取的选择。在文章中，③作者还提出了如何找出离群点（outliers），其对离群点i的定义如下：在该数据点i的邻域d_c范围内，存在属于不同于c_i的另一个类的点j，它们的平均密度$\bar{\rho}=\frac{1}{2}(\rho_i+\rho_j)$大于该数据点$i$自身的密度$\rho_i$。

2. 算法实现

（1）计算数据点之间的距离，构造距离矩阵；顺便得到由所有距离组成的距离向量，方便下一步（2）中根据设定好的参数比例percent来选择邻域半径d_c的大小

```
%计算距离矩阵
dis = zeros(length(X),length(X)); %距离矩阵
disvec = zeros( length(X)*(length(X)-1)/2 ,1); %距离向量
k = 1;
for i = 1:length(X)
    for j = 1:i-1
        dis(i,j)=sqrt(sum( (X(i,:)-X(j,:)).^2,2 )); %欧式距离
        dis(j,i)=dis(i,j);
        disvec(k) = dis(i,j);
        k = k+1;
    end
end
```

（2）根据paper中提到的经验比例1%~2%来选取参数d_c

```
%选取参数dc
disvec = sort(disvec); %升序排列
dc = disvec( round(percent*length(disvec)) ); %选取邻域参数dc
```

（3）求各个数据点的密度ρ，paper中对密度ρ 的定义是邻域中数据点的个数，此值为离散值，较容易出现密度相等的情况，故而个人对其优化为该数学定义：$\rho_i=e^{-\left(\frac{d_{ij}}{d_c}\right)^2}$ 化离散值为连续值，能够更好地区分和表示数据点的密度。此外，求出最大密度点，便于（4）中分情况计算距离δ值；对密度由大到小的降序排列，得到序列号，便于第（6）步的类标划定

```
%计算密度向量p
p = zeros(length(X),1);
for i = 1:length(X)
    p(i) =  sum( exp(-(dis(i,:)/dc).^2),2 ) - exp(-(dis(i,i)/dc).^2); %高斯核函数 除去自身
end
[psort,pidx] = sort(p,'descend'); %降序排列 密度大到小
[pmax,midx] = max(p); %最大密度点
```

（4）求各个数据点的距离δ，其定义为与密度比它大的数据点的最小距离，特别地，因为密度最大的数据点无法找到比它密度还要大的数据点，故而其特别地定义为与所有样本点的最大距离。也因为要对密度最大的点和其他数据点分开讨论，故而上一步（3）中要计算最大密度点；此外，记录每个数据点的距离δ也同时记录其对应的数据点$arg\delta$，便于第（6）步的类标划定

```
%计算距离向量d
d = zeros(length(X),1);
neighbor = zeros(length(X),1); % arg(d)代表邻居
for i = 1:length(X)
    if p(i)==pmax
        [d(i),neighbor(i)] = max( dis(i,:) );
    else
        idxset = find(p>p(i)); %密度大
        d(i) = min( dis(i,idxset) ); %数组经过截取 第二维度大小发生了变化 返回的下标并非原始数据集下标
        neighs = intersect(find(dis(i,:)==d(i)),idxset);
        neighbor(i) = neighs(1);
    end
end
```

（5）归一化ρ和δ后计算它们的乘积γ，便于选出类中心（密度较大且距离也较大的数据点往往可以作为类中心；根据paper中作者提出的对γ降序排列后变化最明显的地方可作为聚类中心和非聚类中心的划分

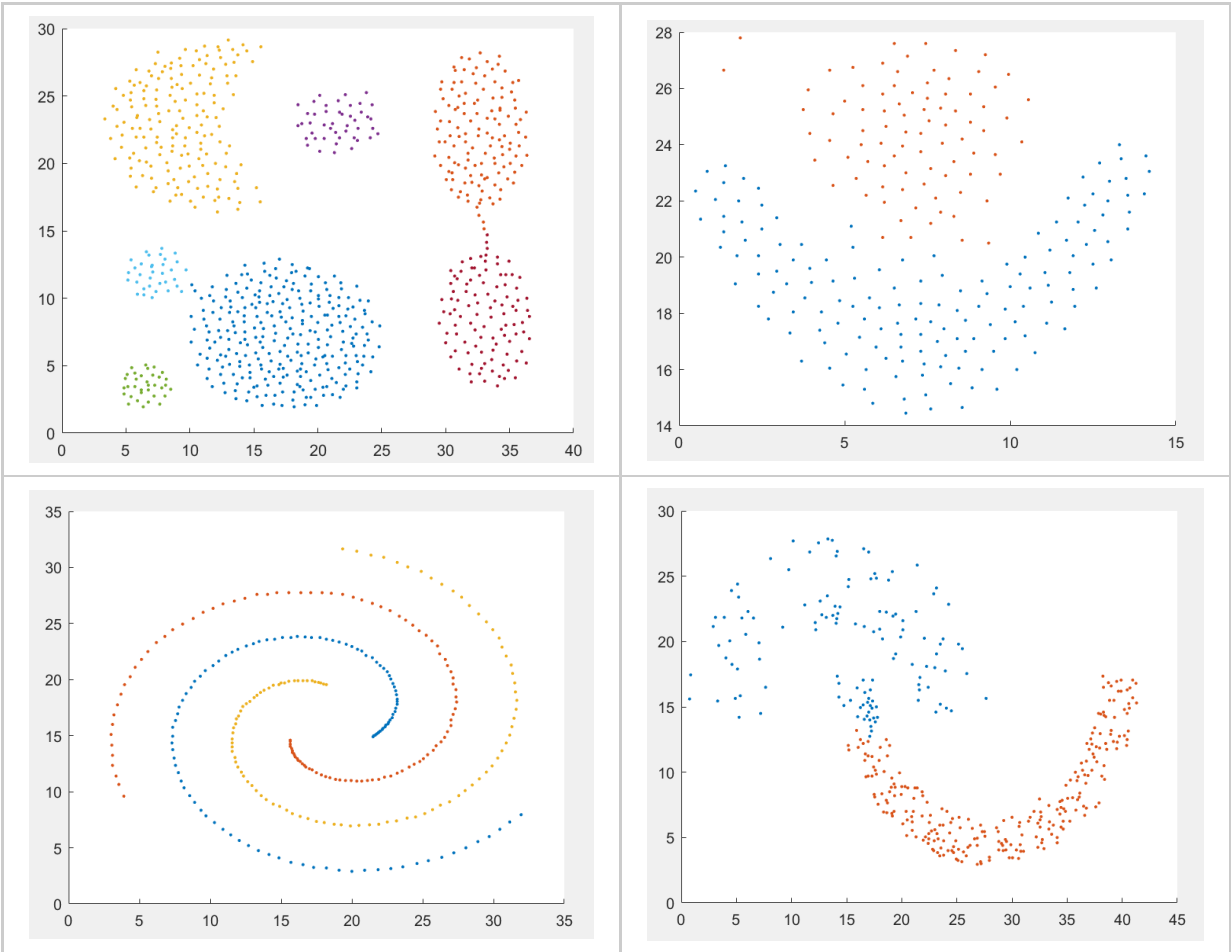
```
r = p.*d;
[rsort,ridx] = sort(r,'descend'); %降序排列
[rmax,rdif] = max(diff(rsort)); %变化最明显处
center = ridx(1:rdif); %类中心
```

（6）划定类标，标签传递。先给每个类中心分配好不同的类标，再根据密度的降序列，从大到小对每个数据点划定类标，如果已被分配则不改变其类标，如果未被分配，则将其距离最小的密度较大数据点的类标分配给它

```
%类中心分配类标
for k = 1:K
    C(center(k))=k;
end
%非类中心分配类标
for i = 1:length(X)
    if C(pidx(i)) == -1
        C(pidx(i)) = C(neighbor(pidx(i)));
    end
end
```

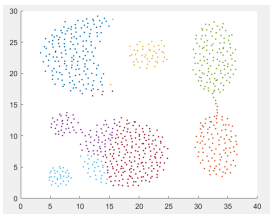
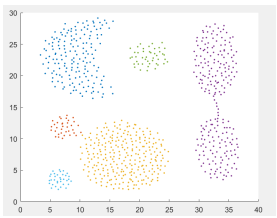
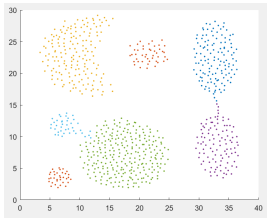
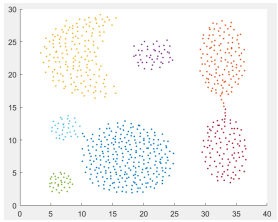
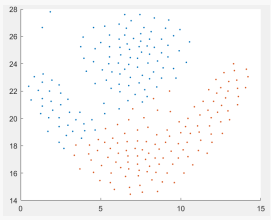
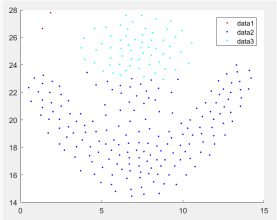
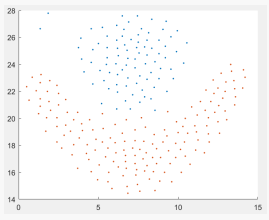
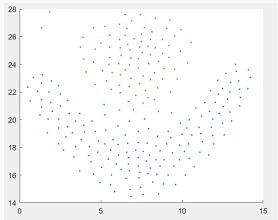
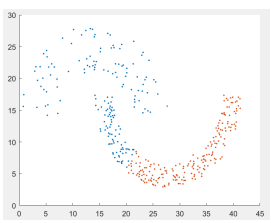
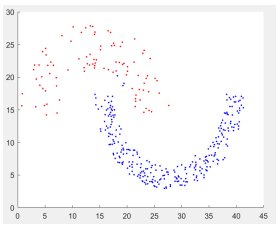
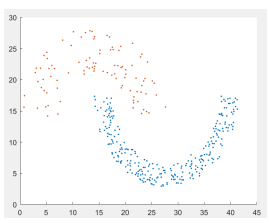
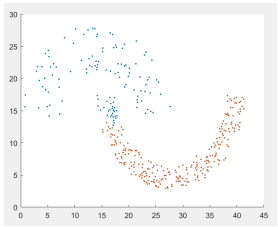
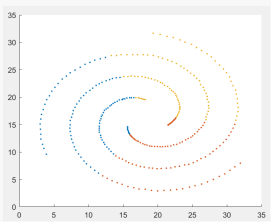
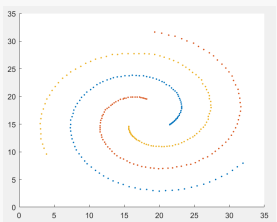
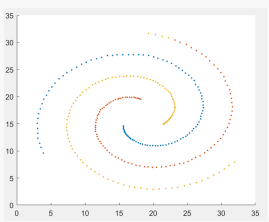
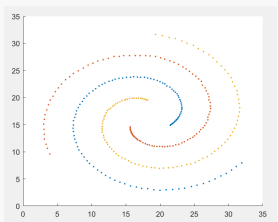
3. 算法效果

Density Peak的算法效果和谱聚类Spectral Clustering一样，均在我们的五个较有特色的二维数据集上有着较好的效果：



如图所示，对左上和右上的数据集都得到了完美划分，左下的非线性数据集也得到了完美的簇划分结果，右下的聚类效果相对较差一些，但依然较为出色。故而从中我们也可以认识到，虽然《Clustering by fast search and density peaks》中的思想很朴素，但正是这么朴素的思想也能得到如此好的结果，simple is beauty，在这个算法得到了淋漓尽致的体现。当然，Density peak算法也和上一个我实现的谱聚类一样，对Pathbased这个数据集的簇划分结果不够理想，这也体现了不同算法在不同数据集上的表现力是不同的，几乎没有通吃的算法，只有具备不同特色、在不同数据集上有着不同表现的算法这一说。

到此，我们已经分析、实现并观察了四个不同的算法Kmeans、DBSCAN、Spectral Clustering、Density Peak，让我们再来横向地比较一下他们在二位数据集上的表现：

Kmeans	DBSCAN	Spectral Clustering	Density Peak
			
			
			
			

先来看第一列Kmeans的效果，在这几个数据集上均不够理想，横向比较其他算法结果可以看出，其的聚簇形状偏向于球状，故而第一列第一幅图是其四个当中表现最好的聚簇结果，其实Kmeans也并非这么弱，在一些团状的数据集中它也能表现的很好，不同算法在不同的数据集上自然有不同的表现，当然其初始化问题是影响其最终效果的一个关键因素之一；此外从第一列最后一幅图也可以看出，非线性的卷曲的簇类Kmeans完全无法划分；

而第二列DBSCAN在带状的密度均匀的数据分布中表现优秀，从第二列最后两幅图可以看到其簇划分的结果几近完美；但其在第二幅图中部的划分不够准确，这和这一部分区域密度不均有直接关系，而DBSCAN在第一幅图的聚类中，结果也是较为优秀，但其将第一幅图中右侧两个我们人工会拆分开的类聚在了一起，便是DBSCAN的另一个局限性，两个类有小簇较为密度的数据点分布相连接，其将会有很大的概率将这两个类分在一起；

第三列谱聚类Spectral Clustering是这组四个算法在这些数据集上跑出最好结果的算法，其表现非常优异，在四个数据集上都能得到完美的划分，当然由于其在得到降维的特征矩阵后，仍然需要其他聚类算法对特征矩阵进行聚类，故而也可以认为Spectral Clustering的一大局限性便是依赖于其他的聚类算法，e.g.，Kmeans这样的话，Kmeans的局限性谱聚类也会有，比如需要知道类的个数K，而在实际的大型数据中，类个数的估计本身便是较为困难的工作；

第四列Density Peak在这四组数据集上的效果也不赖，除了能够完美划分一二四辐图对应的数据集之外，其在第三个数据集上的聚类效果也还算及格，有一部分出现了误分类的情况；Density Peak的优点和DBSCAN一样，不需要提前知道类的个数，此外其算法思想朴素、过程较为简单、数学理论也浅显易懂，却有着不俗的表现，首次将密度和距离结合了起来；但Density Peak也有它的不足之处，比如它在计算密度时未考虑局部的结构，当遇到密度不均

的时候，表现力自然会弱一些，这也是我认为Density Peak在第三幅图对应的数据集上无法跑出谱聚类Spectral Clustering的原因