**Lab 4 实验报告**

PB20000296 郑滕飞

**目的/原理：**

**1、实验目的**

用DPC算法实现聚类，达到结合k-means与DBSCAN优点且计算复杂度较低的聚类效果。本文中，之后的复杂度分析都假设总点数为n，聚类簇数为m。

**2、实验原理**

此聚类算法基于一个假设：簇中心的邻域内的点的局部密度低于中心，且中心到比其局部密度更大的点的距离会显著更远。由此，确定中心只需要寻找局部密度较大且到局部密度更大的点显著远的点[定义比某点局部密度更大的点中与其最近的点为“高密度近邻”，定义此距离为“高密度距离”](若没有比其局部密度更大的点，将它与其他点的最远距离作为高密度距离)。

[值得注意的是，此假设确定了簇中心的性质，但并不代表符合假设的点全部为簇中心，之后实验步骤处会进一步解释。]

实验中采用dc为半径的邻域内点的个数作为局部密度的定义，并通过设置局部密度阈值与高密度距离阈值来控制怎样的点是可以选作簇中心的。此外，若一个点的高密度距离很高，但局部密度不大，则这个点有较大可能是孤立的，或者看作一点成为一簇。

在确定簇中心后，进一步决定每簇中的点时，要求每个点与其高密度近邻所在的簇一致。由于每次取高密度近邻若不结束，一定可以找到密度最大的点，而其高密度距离是与其他点的最远距离，必然显著大，因此必然最后能找到某个簇中心。

**实验步骤：**

**1、整体框架**

这次的框架分为四个部分：绘制、评估、聚类与测试，前三个部分都是定义了一个函数。由于这个聚类算法并不存在内部状态，而是在输入数据后直接判断，我选择用函数直接实现，不需要利用类。

绘制部分，输入每点第一个坐标、第二个坐标与标签组成的数组，绘制出对应的散点图。评估部分，由于我自己的sklearn包并没有包含DBI计算，更新出现了一些问题，自己构建了计算DBI的函数。聚类部分即为输入数据后计算聚类并画图、计算DBI。最后的测试部分，输入数据并进行测试。

**2、绘制散点图与DBI计算**

在我的算法中，簇的标签是簇中心元素的下标，也即标签等于下标的点就是簇中心，而孤立点的标签统一为-1。

画图时，将这三类点分别画图：

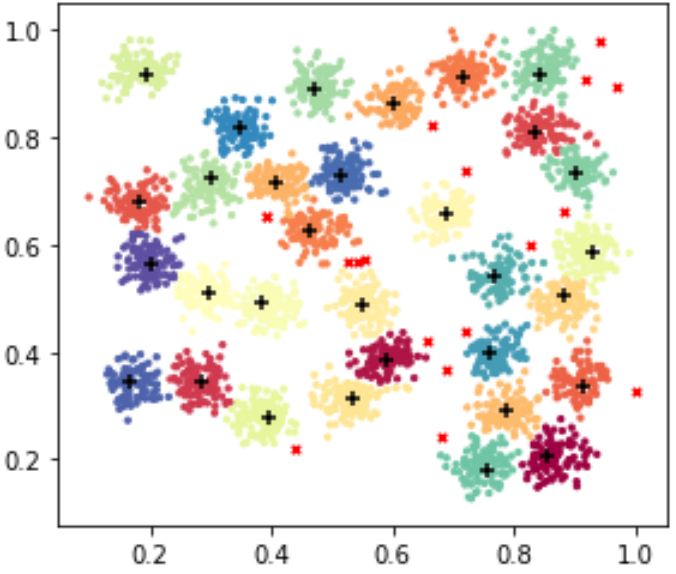
now.set\_title(name)

now.scatter(x1[normal],x2[normal],c=y[normal],cmap=plt.cm.Spectral, s=5)

now.scatter(x1[centers], x2[centers], c="black", marker="+")

now.scatter(x1[out], x2[out], c="red", marker="x", s=12)

normal代表不是中心也不是孤立点的点，根据matplotlib提供的Spectral颜色映射绘制；centers代表中心，绘制为黑色的加号；out代表孤立点，绘制为红色的叉号，示例效果如下：



DBI直接按照公式计算，其中的范数为二范数。步骤为：区分中心、构建每类分开的二维数组X、计算S、计算M、计算R。分类过程如下：

a = []

for i in centers:

a.append([i])

for i in range(y.size):

if y[i] >= 0 and y[i] != i:

for t in range(classes):

if y[i] == centers[t]:

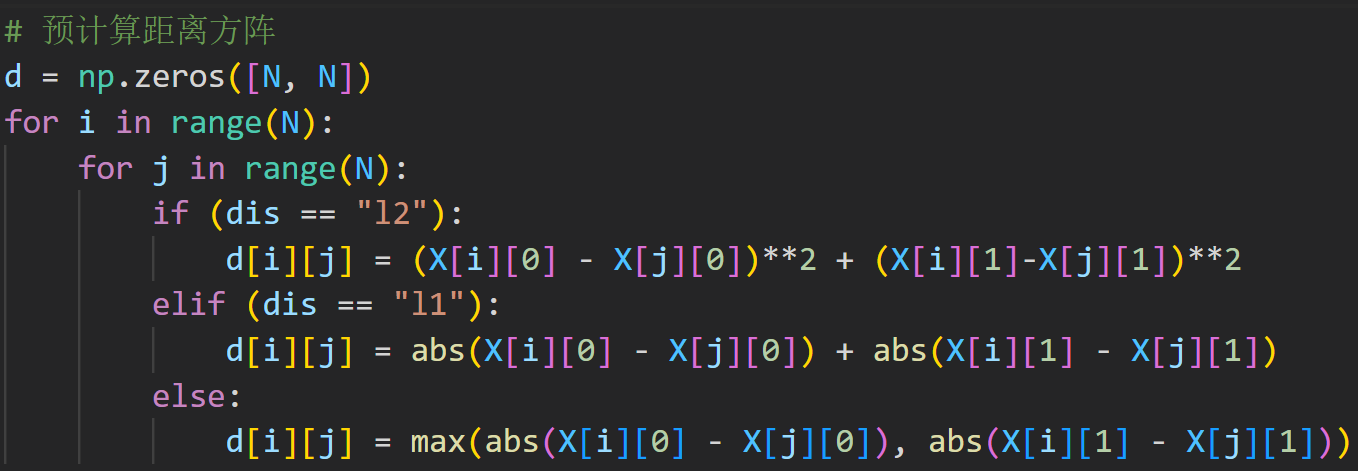
a[t].append(i)

由于a每行的第一个位置就是对应类的中心，后续找中心时直接找a[i][0]即可。

\*DBI计算忽略了被选为孤立点的点，当选取孤立点过多时，小DBI未必优。

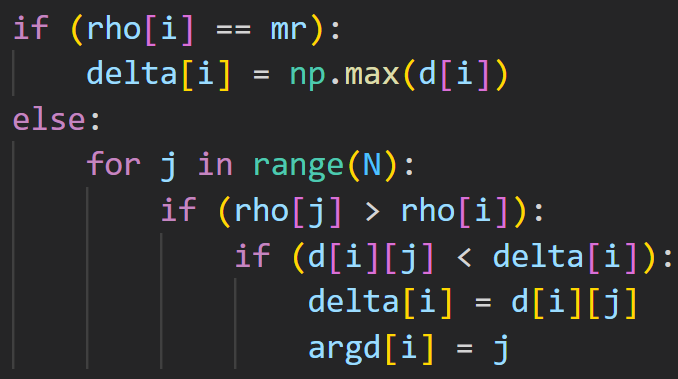
**3、聚类算法**

首先，由于数据尺度的差别，先对数据进行归一化。又因为后面大量涉及距离的比较，先计算出任意两点间的距离矩阵d[i][j]，方便之后直接使用。



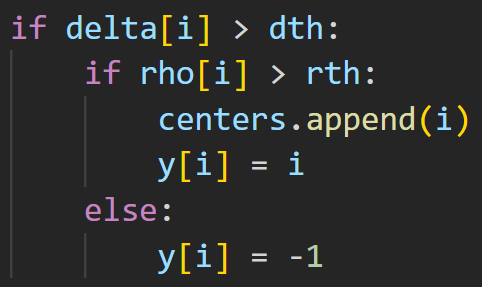
这里采取了不同的距离函数，对应不同的区域形状(由于只涉及大小比较，l2范数没有必要取根号)，在之后的参数比较中会进行比较。

之后，按照算法要求，计算rho与delta。为了之后寻找高密度近邻方便，这里额外记录了delta取到时的i(mr为rho的最大值，已预先计算出)，保存在argd[i]中：



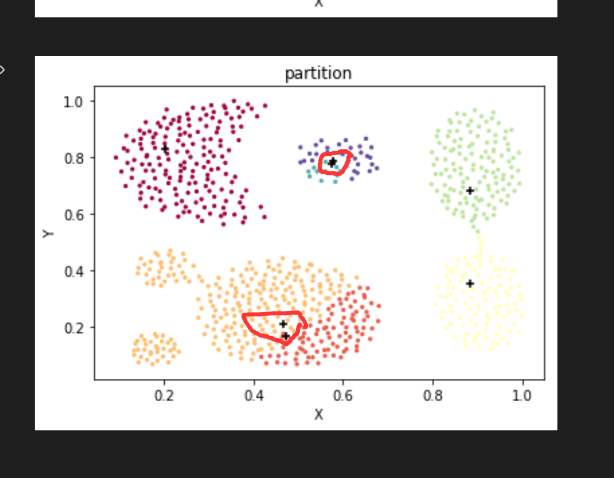
argd的初值均为-1，因此rho为最大值得点的高密度近邻记为-1.

在分析中心时，严格按照算法的中心选取是这样的(这里也同时分出了孤立点)：



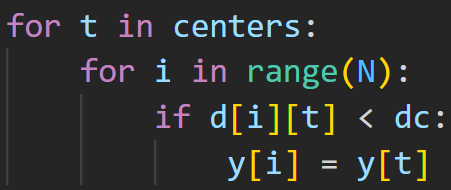
其中dth与rth为提前选定的r与d的阈值。

但是，直接以其作为中心可能会出现如下的情况：



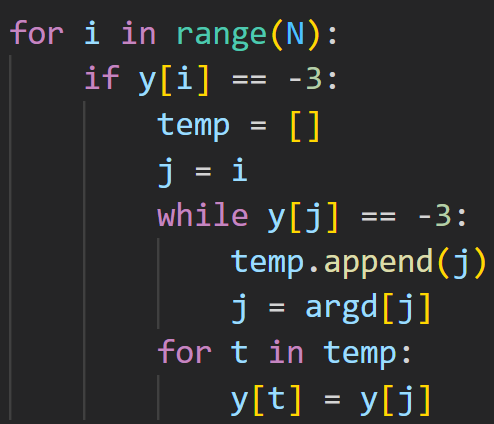
红圈部分出现了过于密集的中心。根据之前的算法过程，这是由于“高密度近邻”需要的是严格大于而非大于等于，于是可能有若干密度相等且接近的点都被选为了中心。事实上，这些“备选中心”中选择任何一个都可以。

于是，在上方的中心算法之后加了一段：



在加上这个处理之后，如果先后选出的中心很接近，后被选出的中心所在的簇会被先选出的覆盖掉，从而不会被视为两个簇。不仅如此，将中心的近邻与中心分为同簇还可以减少之后确定所在的簇的搜索次数。

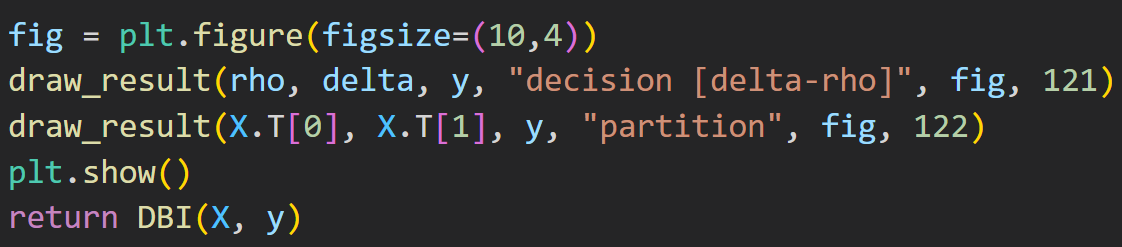
这样从备选中心得到最终中心后，就可以确定簇了，方法如下：



y[i]的初值是-3，也以此表示还没有确定的点。在找高密度近邻的搜索过程中，会出现一条链temp，将链上的所有元素进行赋值即得到了整个结果。

算法复杂度为：预处理、计算rho与delta均为n\*n，寻找中心n，中心周围处理m\*n，确定簇n(每个点最多被访问三次)，因此总复杂度是量级。

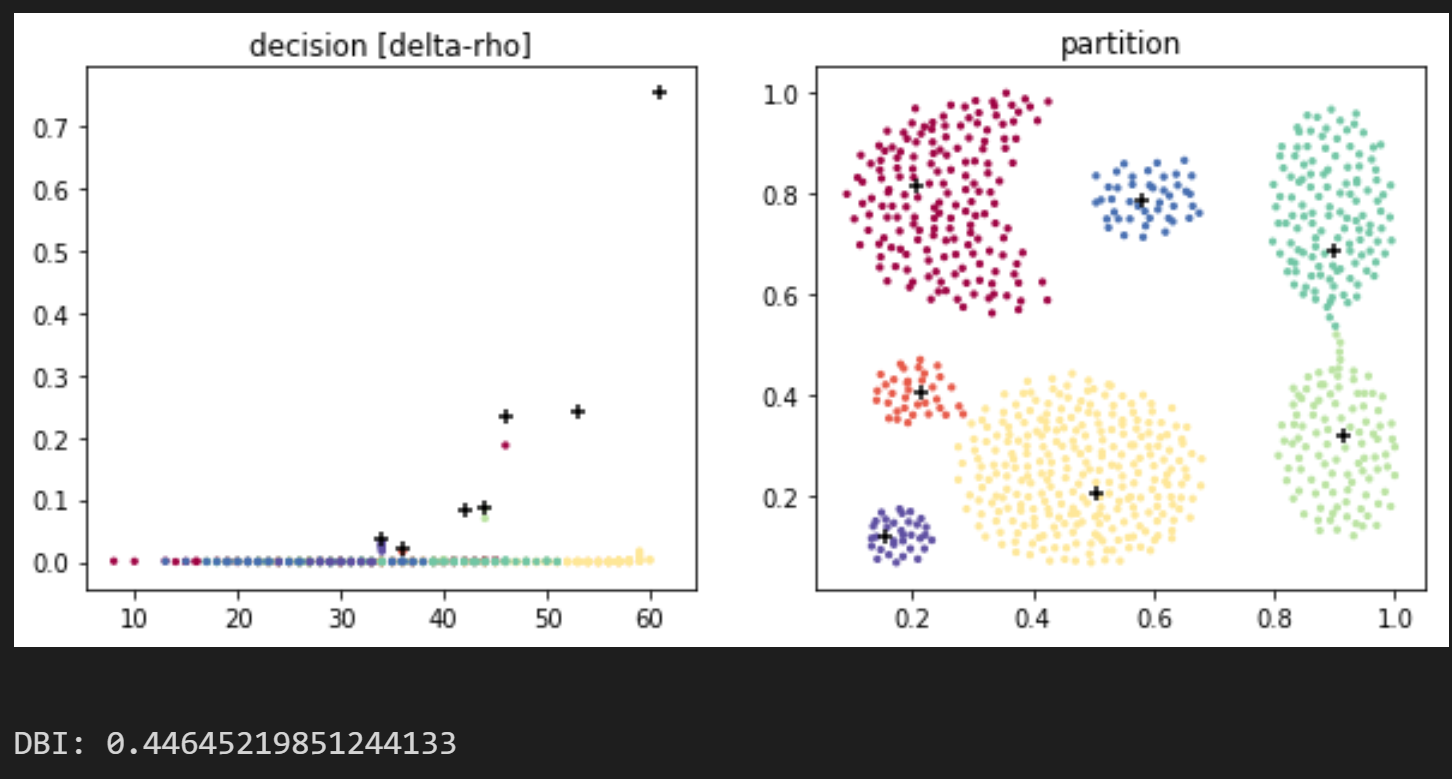
由于过程中需要计算的距离包括小的与大的，哪怕进行预排序，需要提前计算的距离也不能减少，因此复杂度量级改进的可能性不大。最后，绘制两个图，并返回计算的DBI：

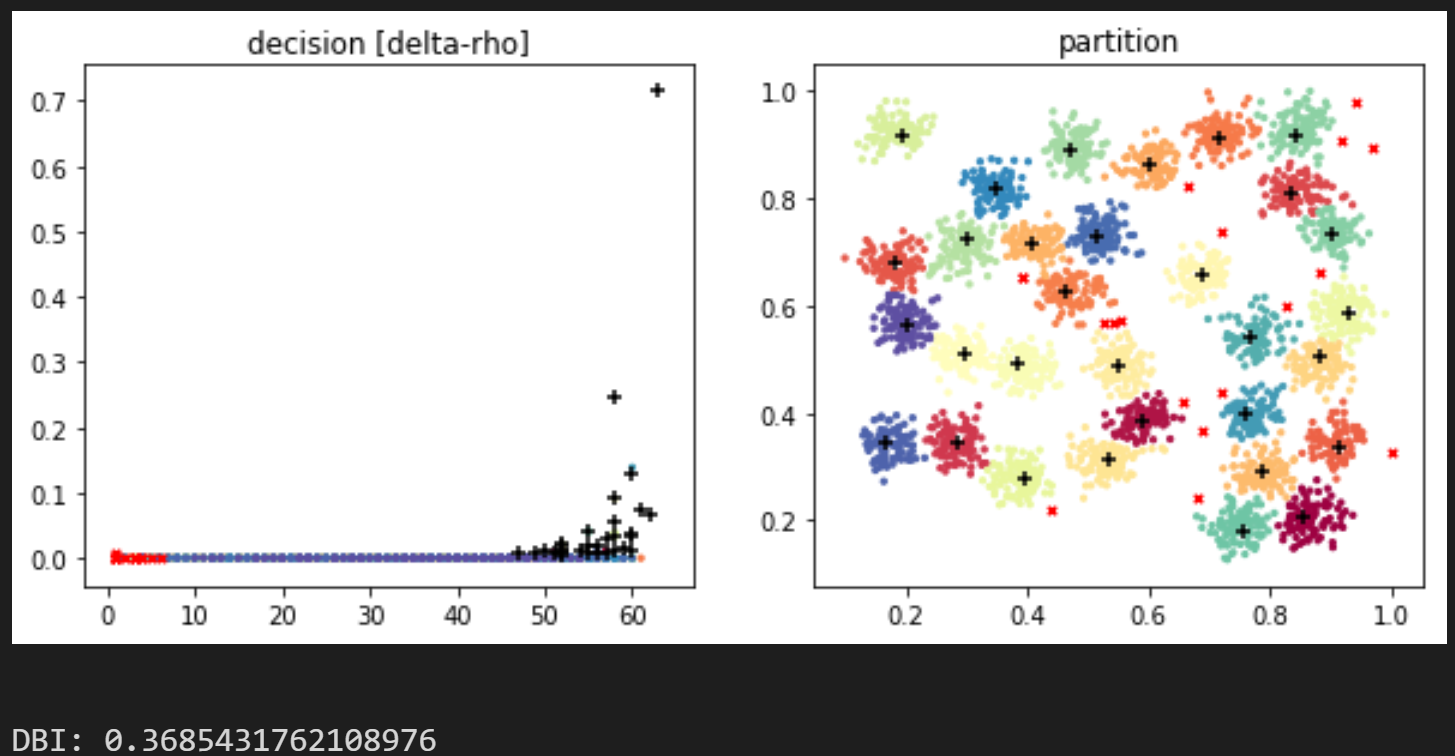


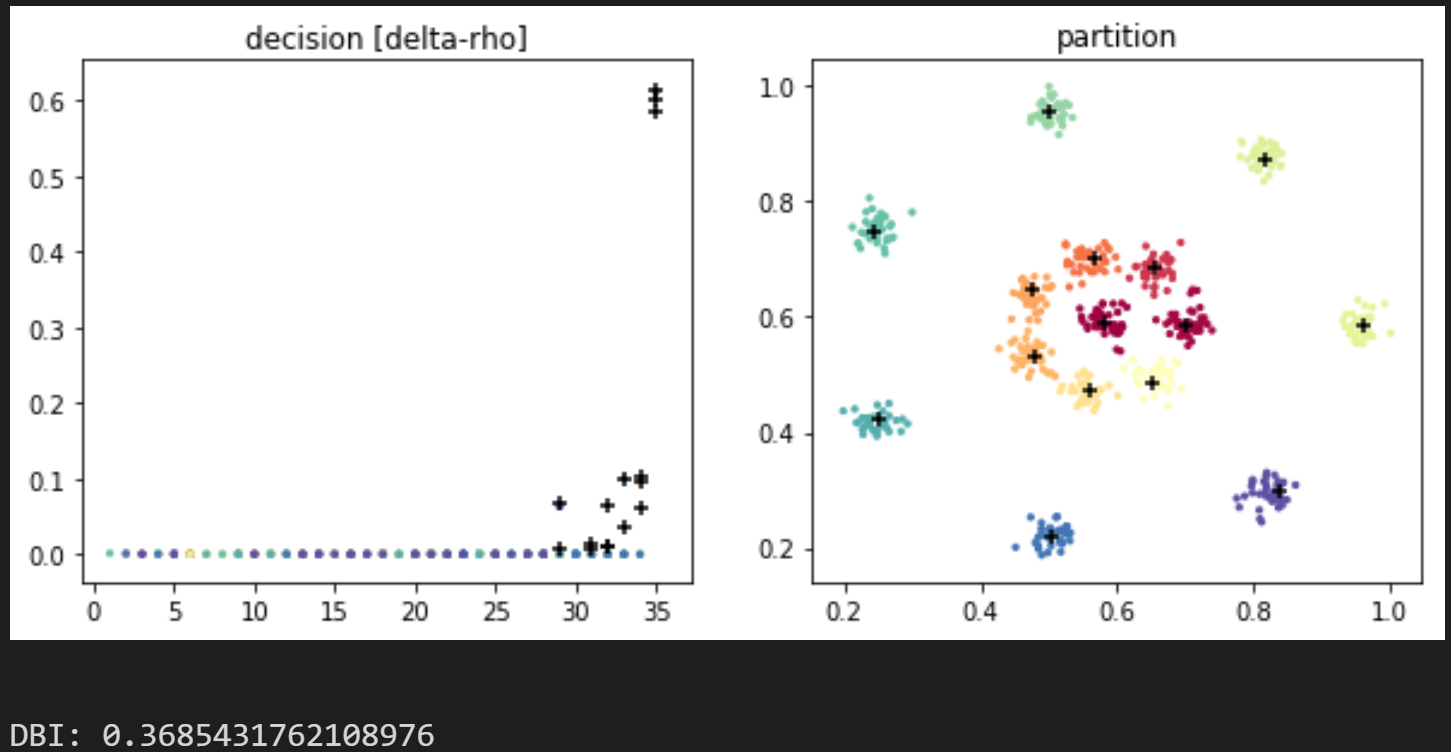
**展示/分析：**

**1、测试样例输出**

以下为三个样例在选取到合适参数后的输出结果：



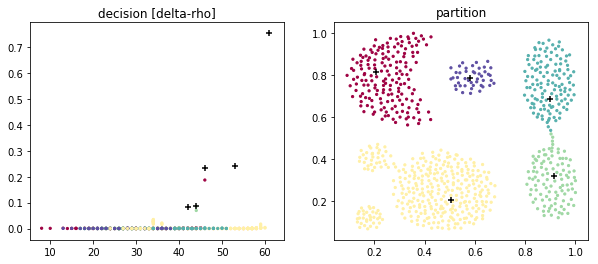


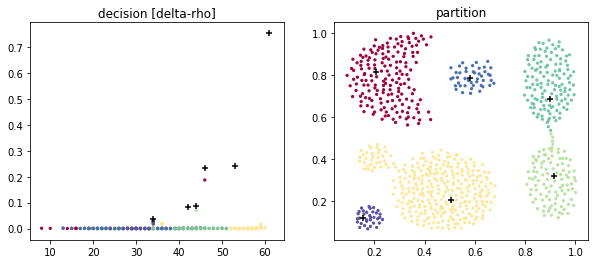


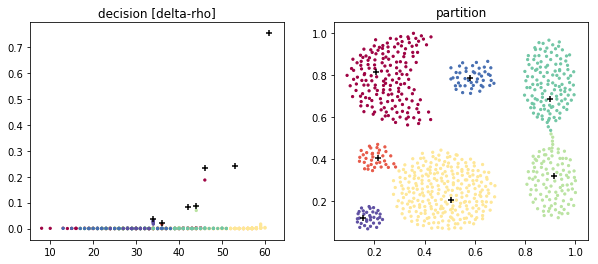
第一张图中，很明显发现有rho相同的点最后只选取了一个为中心，这也是对中心进行进一步处理的效果。

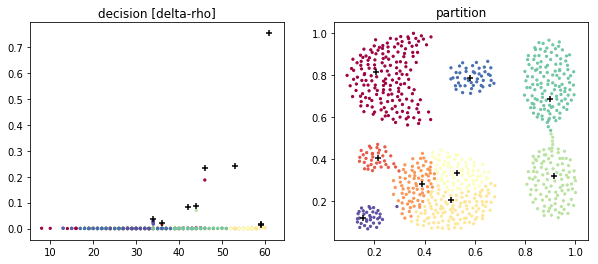
**2、阈值相关对比**

delta的阈值对结果的影响是明显的，因为这直接关系到分出的簇的数量，以下四张图为delta逐渐减少时的结果：

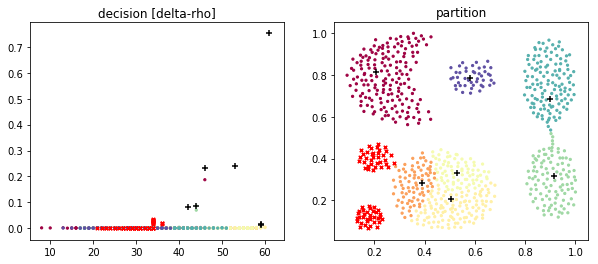








rho的阈值则是主要影响将多大的簇视为孤立，例如上一张图中将rho的阈值调大后，会出现如下情况：



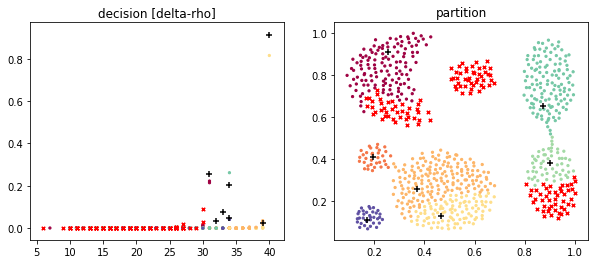
左边的两个小簇现在不再被视为簇，而是看作孤立点，这是由于两个对应的中心被舍弃了。

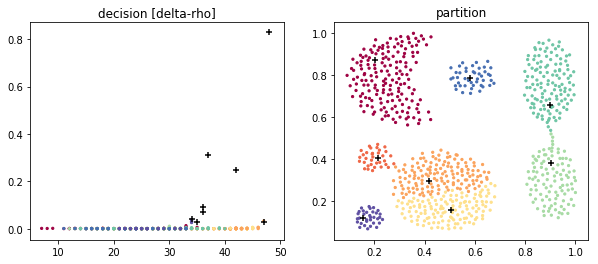
在上方的例子中，第三张图，也即聚类与目测较为一致时，对应的DBI最小。

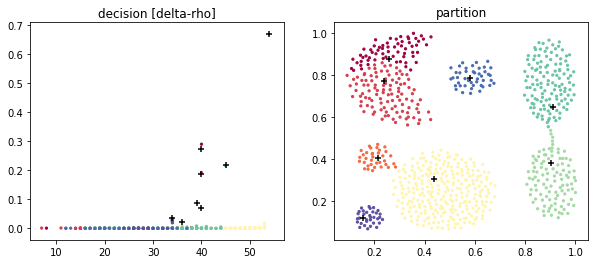
另一个改变d阈值的例子见代码文件后方的“参数比较”部分。

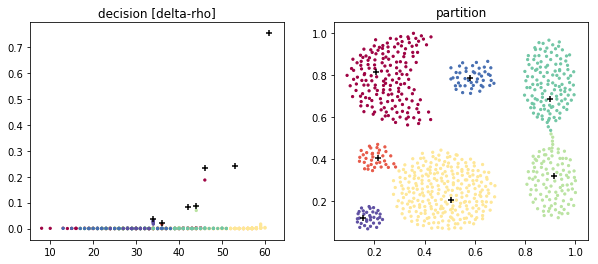
**3、距离相关对比**

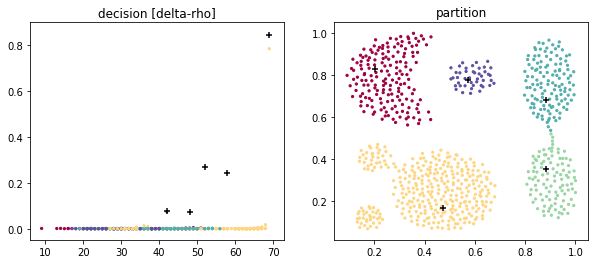
距离dc的影响比较微妙。假设两个阈值不变，在dc越小时划分得越精细(这是由于dc的意义事实上是容许的簇半径)[第一张图中由于rho的阈值设置，有些簇被视为孤立]：



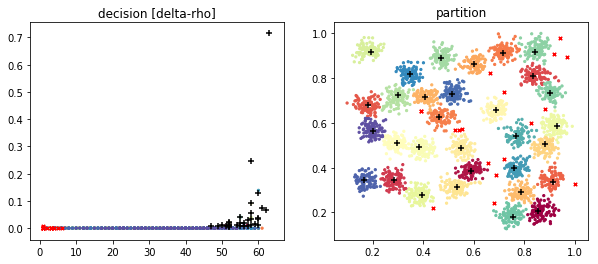


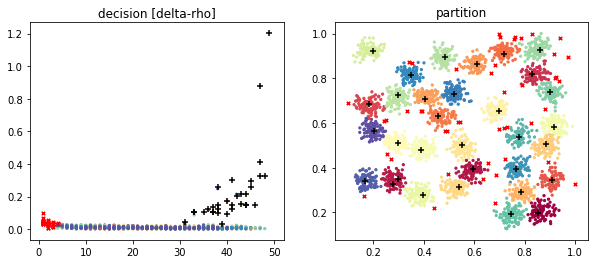


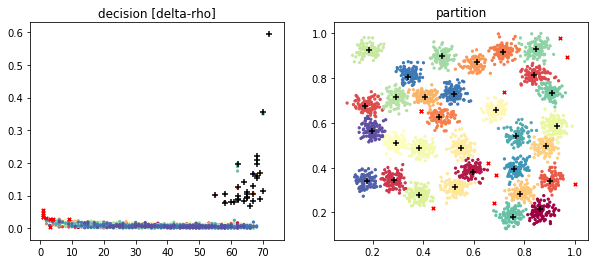




而范数的选取则是关乎区域的形状。值得注意的是，为了对比不同的参数，在1范数与无穷范数时需要给阈值和dc作平方根，这是由于之前计算二范数为了节省时间没有进行平方，对比结果如下：







从上到下依次为二范数、一范数、无穷范数的结果，DBI最小的事实上是无穷范数。

不同范数的选取会导致区域形状的差别，二范数为圆，一范数为竖着的正方形，而无穷范数是横着的正方形。当实际区域形状与对应的情况越接近时，实际效果越好，而之外的点会被视为孤立点。

更多对比见代码文件后方的“参数比较”部分。

**总结：**

这次实验的任务难度主要在于，不能尽信论文。在自己进行了一些处理之后，可以获得比直接按论文操作更好的结果。除了文中涉及的部分外，局部密度、高密度近邻的选取算法也都可以进行一些处理(如将直接设置阈值变为加权软阈值)，或许能获得更好的结果。