法律声明

- 本课件包括演示文稿、示例、代码、题库、视频和声音等内容，小象学院和主讲老师拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意及内容，我们保留一切通过法律手段追究违反者的权利。

- 课程详情请咨询
  - 微信公众号：小象
  - 新浪微博：ChinaHadoop

聚类(下)

邹博

本次目标

- 理解密度聚类并能够应用于实践
  - DBSCAN
  - DensityPeak密度最大值聚类
- 掌握谱聚类的算法
  - 考虑谱聚类和PCA的关系
- 半监督LPA算法
  - 社区发现

密度聚类方法

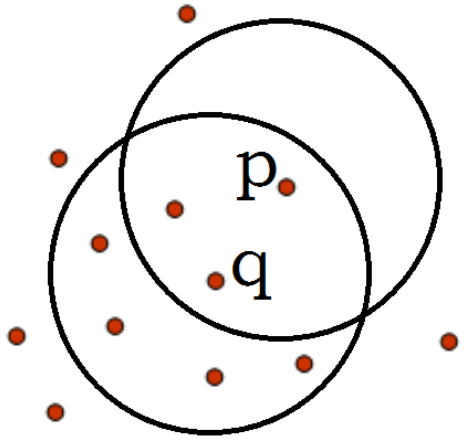
- 密度聚类方法的指导思想是，只要样本点的密度大于某阈值，则将该样本添加到最近的簇中。
- 这类算法能克服基于距离的算法只能发现“类圆形”(凸)的聚类的缺点，可发现任意形状的聚类，且对噪声数据不敏感。但计算密度单元的计算复杂度大，需要建立空间索引来降低计算量。
  - DBSCAN
  - 密度最大值算法

DBSCAN算法

- DBSCAN(Density-Based Spatial Clustering of Applications with Noise)
- 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，并可在有“噪声”的数据中发现任意形状的聚类。

DBSCAN算法的若干概念

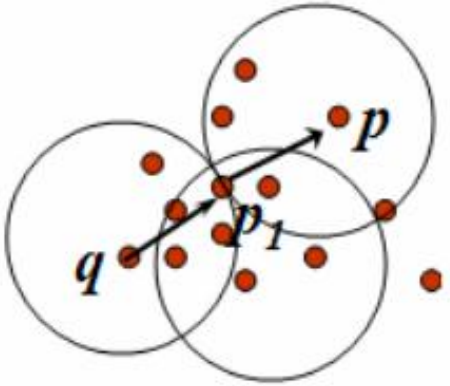
- 对象的ε-邻域：给定对象在半径ε内的区域。
- 核心对象：对于给定的数目m，如果一个对象的ε-邻域至少包含m个对象，则称该对象为核心对象。
- 直接密度可达：给定一个对象集合D，如果p是在q的ε-邻域内，而q是一个核心对象，我们说对象p从对象q出发是直接密度可达的。

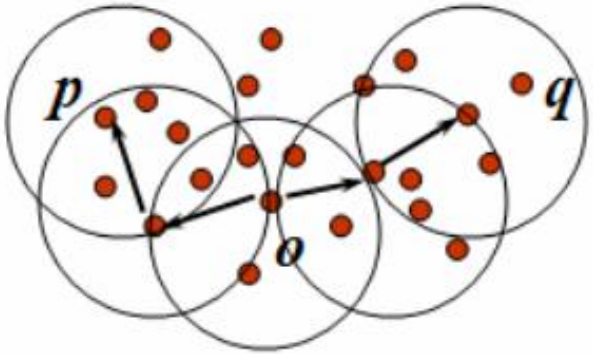


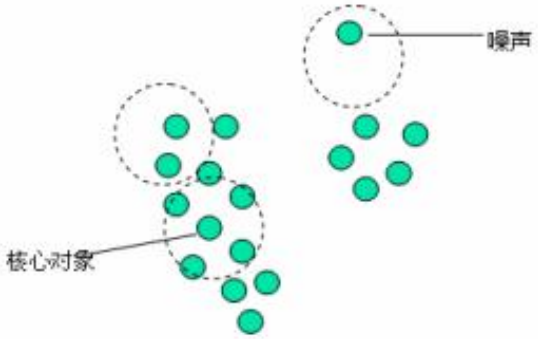
- 如图ε=1cm，m=5，q是一个核心对象，从对象q出发到对象p是直接密度可达的。

DBSCAN算法的若干概念

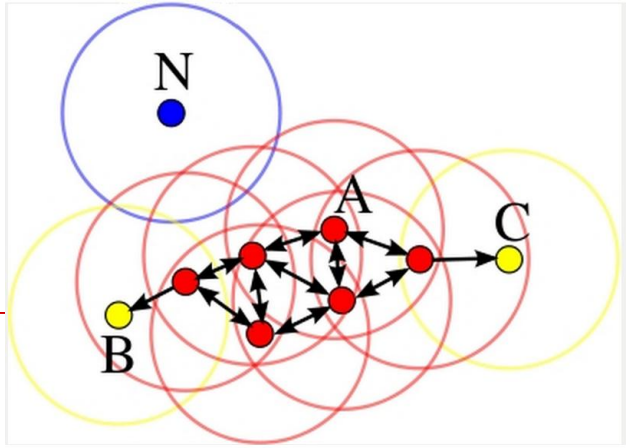
- 密度可达：如果存在一个对象链$p_1p_2...p_n$，$p_1=q$，$p_n=p$，对$p_i \in D$，$(1 \le i \le n)$，p_{i+1}是从p_i关于ε和m直接密度可达的，则对象p是从对象q关于ε和m密度可达的。
- 密度相连：如果对象集合D中存在一个对象o，使得对象p和q是从o关于ε和m密度可达的，那么对象p和q是关于ε和m密度相连的。
- 簇：一个基于密度的簇是最大的密度相连对象的集合。
- 噪声：不包含在任何簇中的对象称为噪声。



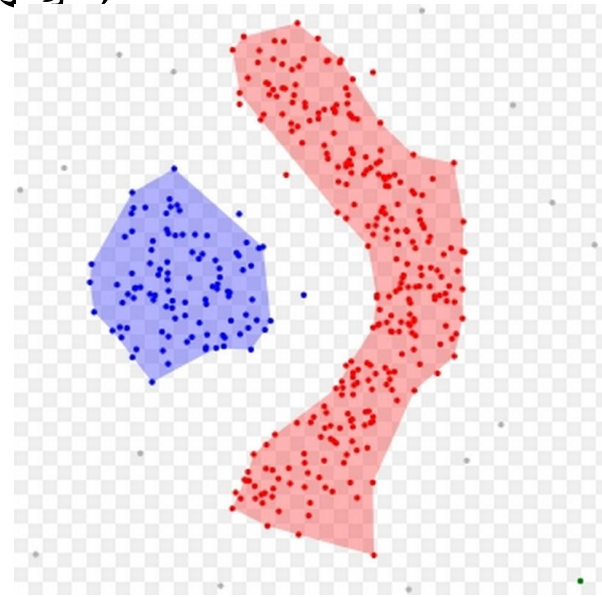




DBSCAN算法



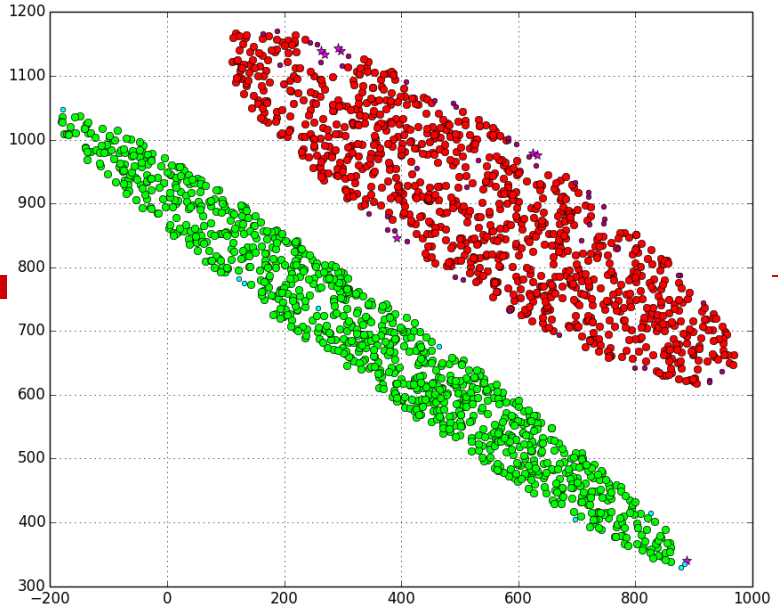
- DBSCAN算法流程：
  - 如果一个点p的ε-邻域包含多于m个对象，则创建一个p作为核心对象的新簇；
  - 寻找并合并核心对象直接密度可达的对象；
  - 没有新点可以更新簇时，算法结束。
- 有上述算法可知：
  - 每个簇至少包含一个核心对象；
  - 非核心对象可以是簇的一部分，构成了簇的边缘(edge)；
  - 包含过少对象的簇被认为是噪声。



Code

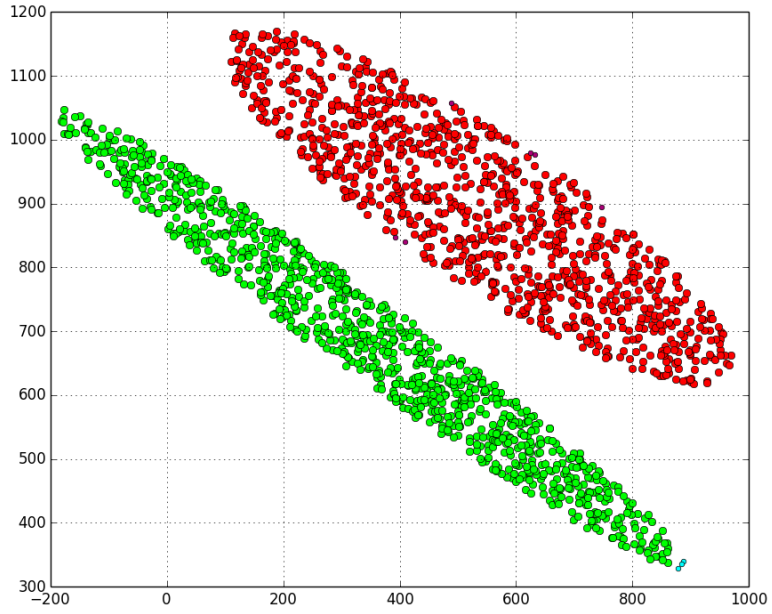
```
def density_cluster(data):
    # 计算核心对象
    m = len(data)       # 样本数目
    sim = [[] for i in range(m)] * m   # sim[i]:第i个样本的近邻
    r2 = r * r
    for i in range(m):  # 距离
        print i
        for j in range(m):
            if distance2(data[i], data[j]) < r2:
                sim[i].append(j)

    # 根据核心对象合并
    uf = uf_init(m)   # 并查集初始化
    for i in range(m):
        if len(sim[i]) > kernel_num: # i是核心对象
            for j in sim[i]:
                uf_union(uf, i, j)

    # 计算并查集的每个对象所属类型
    types = {}
    t = 0
    for i in range(m):
        c = uf_find(uf, i)
        if not types.has_key(c):
            if c != i or len(sim[i]) > kernel_num:    # 去除孤立点
                t += 1
                types[c] = t

    # 根据字典映射关系，计算每个样本所属的簇
    cluster = [0] * m
    for i in range(m):
        c = uf_find(uf, i)
        if (c != i) or (len(sim[i]) > kernel_num):
            if len(sim[i]) > kernel_num:  # 是核心对象
                cluster[i] = types[c]
            else:
                cluster[i] = k+types[c]
    return cluster
```

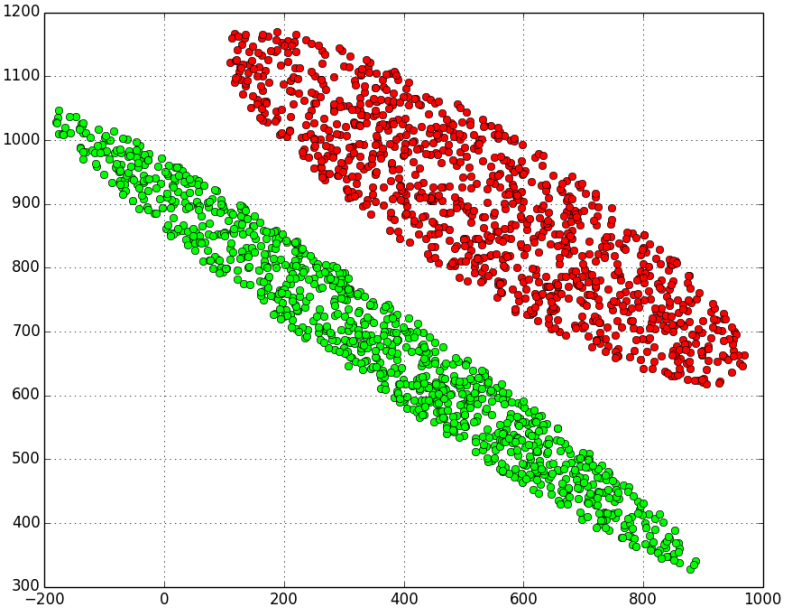
参数： n=5


1200
1100
1000
900
800
700
600
500
400
300
−200
0
200
400
600
800
1000

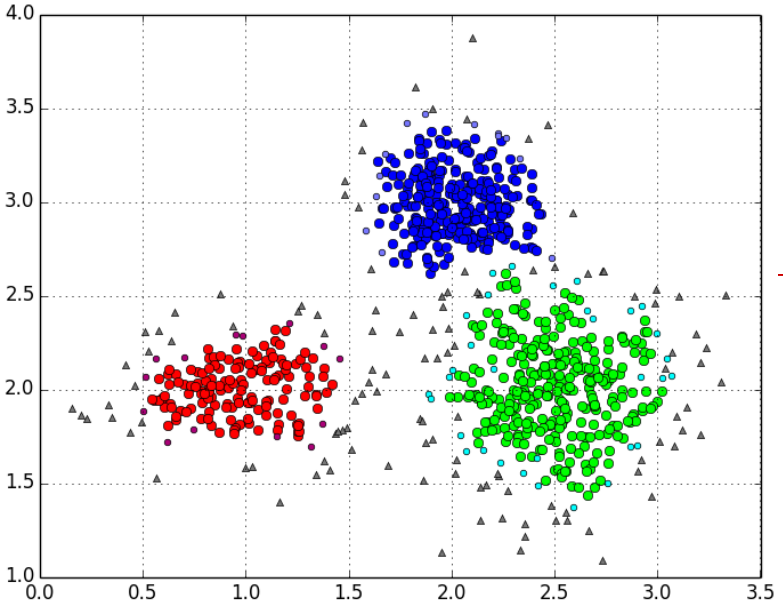
25

30 | 40


1200
1100
1000
900
800
700
600
500
400
300
−200
0
200
400
600
800
1000

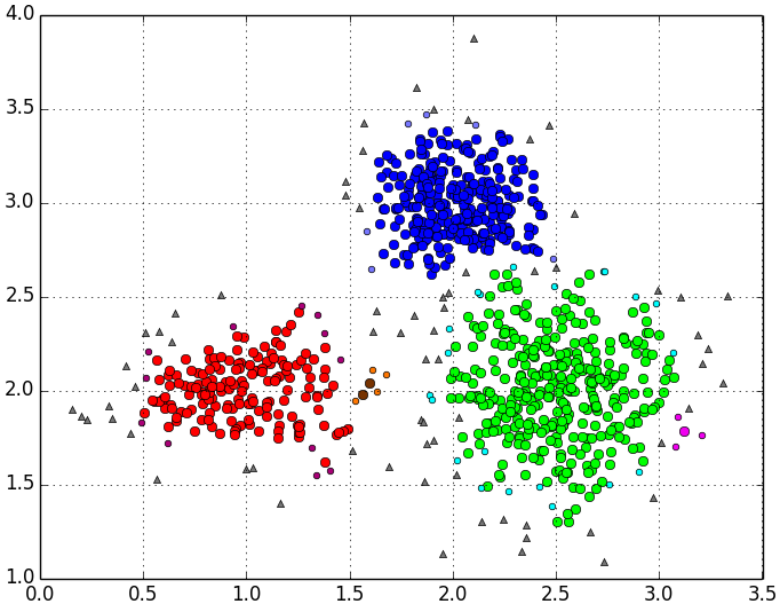

1200
1100
1000
900
800
700
600
500
400
300
−200
0
200
400
600
800
1000

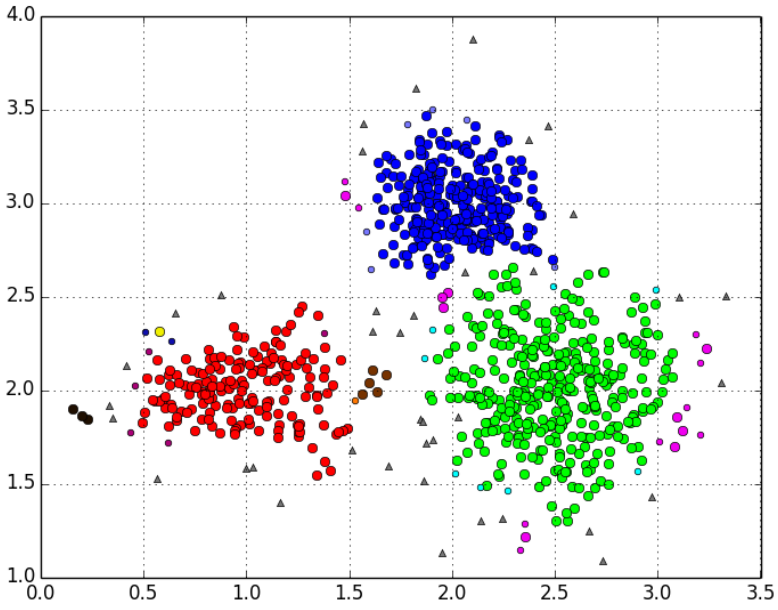
参数： r=0.1


4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5

5

3 | 2


4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5


4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5

密度最大值聚类

- 密度最大值聚类是一种简洁优美的聚类算法，可以识别各种形状的类簇，并且参数很容易确定。
- 定义：局部密度ρi

$$\rho_i = \sum_j \chi(d_{ij} - d_c), \quad 其中，\ \chi(x) = \begin{cases} 1 & x < 0 \\ 0 & otherwise \end{cases}$$

  - d_c是一个截断距离，ρ_i即到对象i的距离小于d_c的对象的个数。由于该算法只对ρ_i的相对值敏感，所以对d_c的选择是稳健的，一种推荐做法是选择d_c，使得平均每个点的邻居数为所有点的1%-2%
- 定义：高局部密度点距离δi

局部密度的其他定义

- 截断值：

$$\rho_i = \sum_j \chi(d_{ij} - d_c), \quad 其中，\ \chi(x) = \begin{cases} 1 & x < 0 \\ 0 & otherwise \end{cases}$$

- 高斯核相似度：

$$\rho_i = \sum_{j \in I_S \setminus \{i\}} \exp\left(-\left(\frac{d_{ij}}{d_c}\right)^2\right)$$

- K近邻均值：

$$\rho_i = \frac{1}{K}\sum_{j=1}^{K} d_{ij}, \quad 其中，\ d_{i1} > d_{i2} > \cdots d_{iK} > d_{i,K+1} > \cdots$$

高局部密度点距离

- 高局部密度点距离 $\delta_i = \min_{j:\rho_j > \rho_i} \left(d_{ij} \right)$

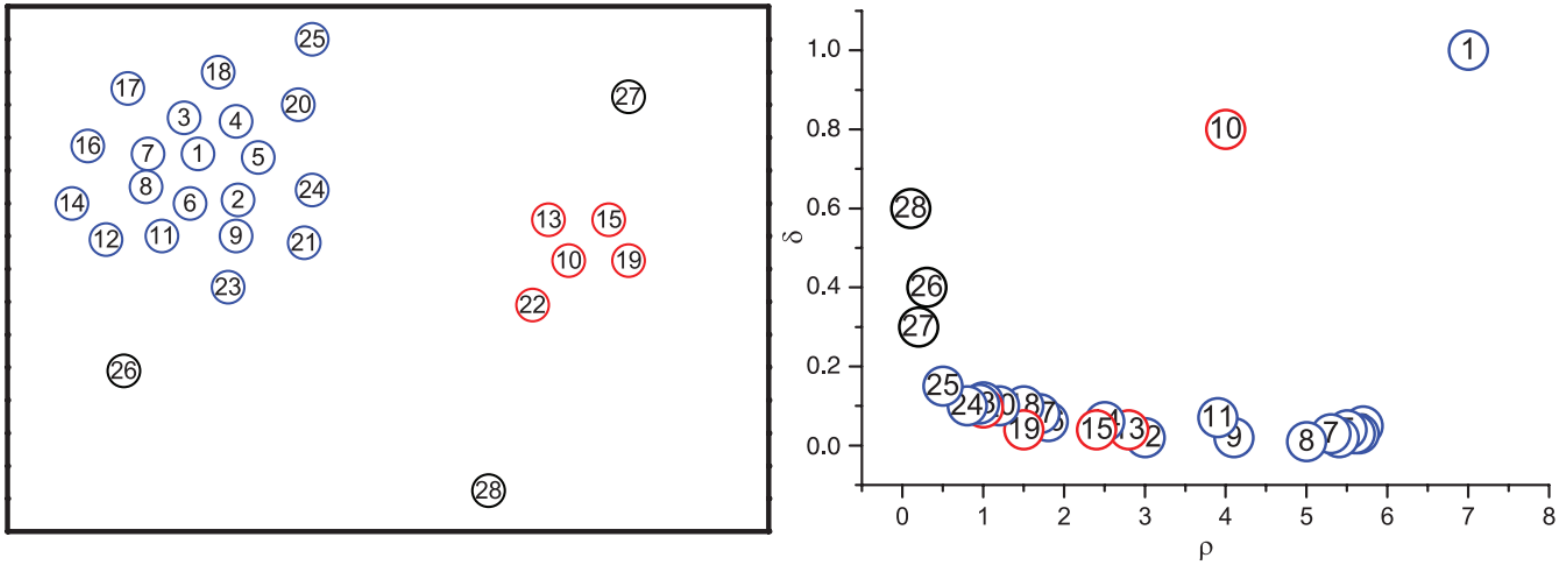
- 在密度高于对象i的所有对象中，到对象i最近的距离，即高局部密度点距离。
- 对于密度最大的对象，设置$\delta_i=\max(d_{ij})$(即：该问题中的无穷大)。
  - 只有那些密度是局部或者全局最大的点才会有远大于正常值的高局部密度点距离。

簇中心的识别

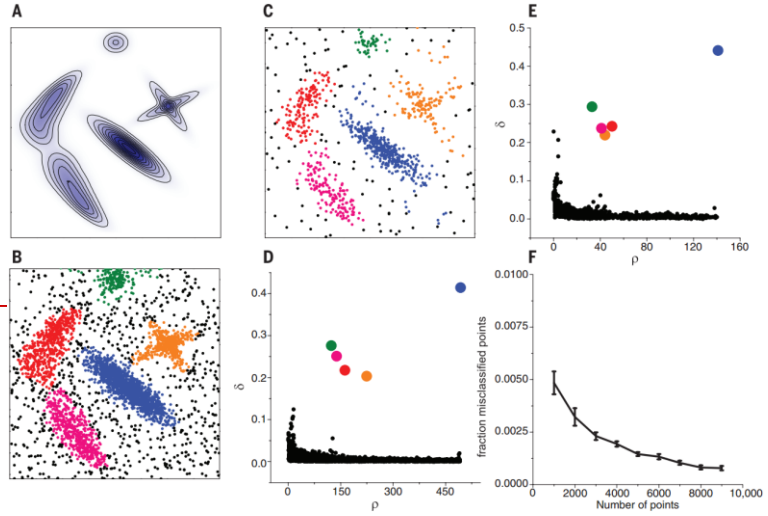
- 那些有着比较大的局部密度ρ_i和很大的高密距离δ_i的点被认为是簇的中心；
- 高密距离δ_i较大但局部密度ρi较小的点是异常点；
  - 确定簇中心之后，其他点按照距离已知簇的中心最近进行分类
    - 注：也可按照密度可达的方法进行分类。

DensityPeak与决策图Decision Graph

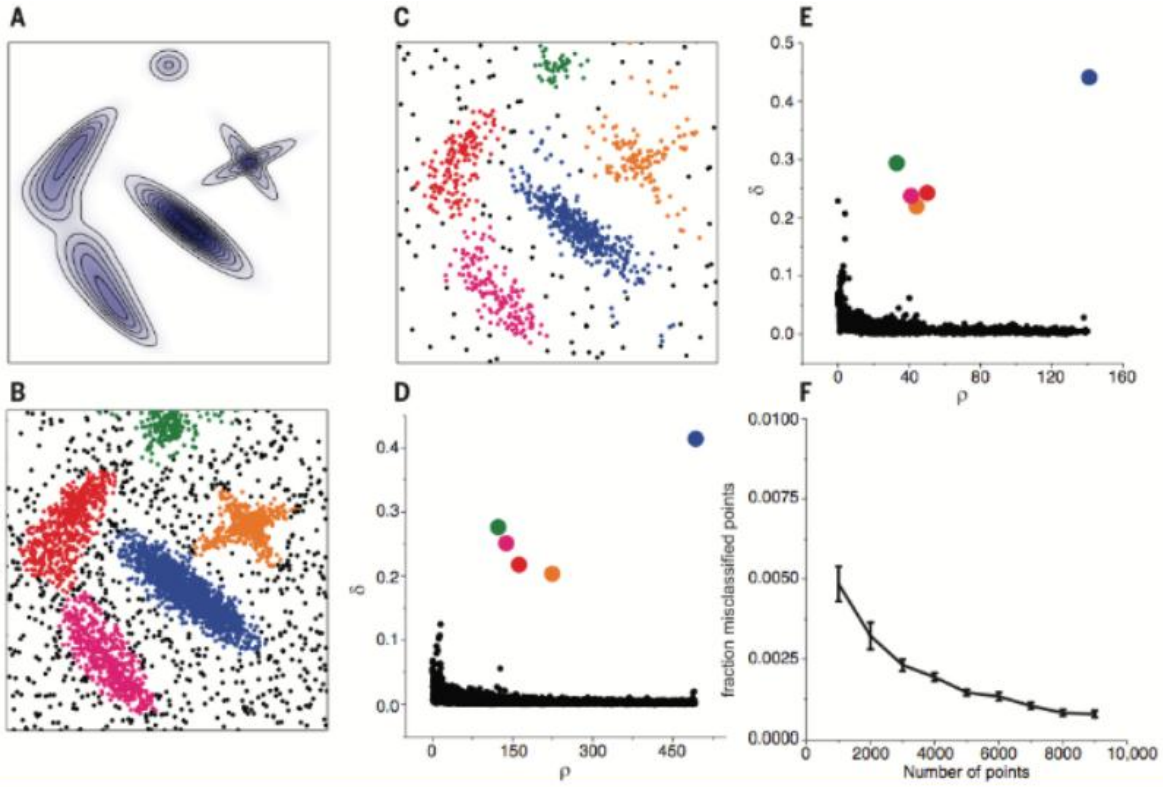
- 左图是所有点在二维空间的分布，右图是以ρ为横坐标，以δ为纵坐标绘制的决策图。可以看到，1和10两个点的ρ_i和δ_i都比较大，作为簇的中心点。26、27、28三个点的δ_i也比较大，但是ρ_i较小，所以是异常点。



边界和噪声的重认识

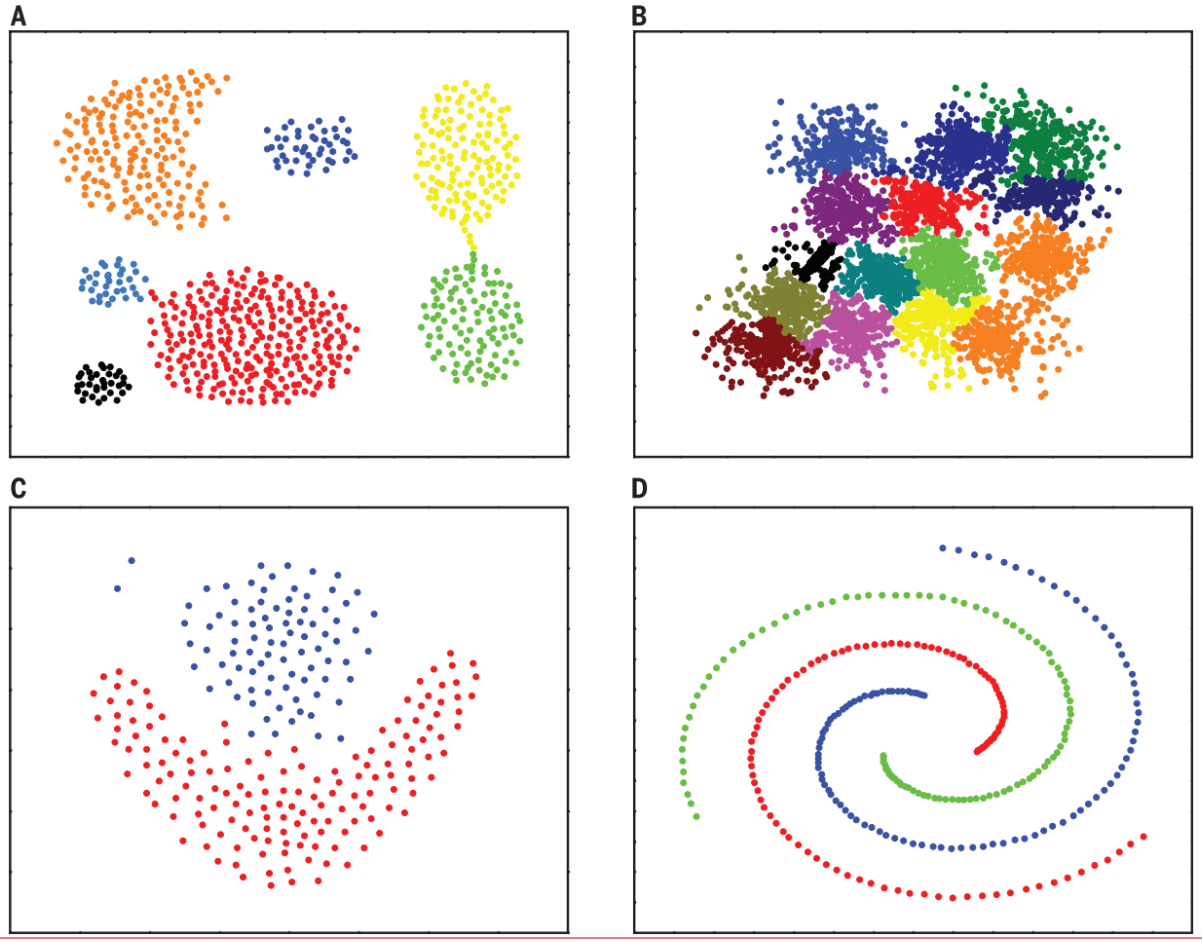


- 在聚类分析中，通常需要确定每个点划分给某个簇的可靠性：
- 在该算法中，可以首先为每个簇定义一个边界区域(border region)，亦即划分给该簇但是距离其他簇的点的距离小于d_c的点的集合。然后为每个簇找到其边界区域的局部密度最大的点，令其局部密度为ρ_h。
- 该簇中所有局部密度大于ρ_h的点被认为是簇核心的一部分(亦即将该点划分给该类簇的可靠性很大)，其余的点被认为是该类簇的光晕(halo)，亦即可以认为是噪声。
  - 注：关于可靠性问题，在EM算法中仍然会有相关涉及。



A图为生成数据的概率分布, B, C二图为分别从该分布中生成了4000, 1000个点. D, E分别是B, C两组数据的决策图(decision tree), 可以看到两组数据都只有五个点有比较大的ρ i 和很大的δ i . 这些点作为类簇的中心, 在确定了类簇的中心之后, 每个点被划分到各个类簇(彩色点), 或者是划分到类簇光晕(黑色点). F图展示的是随着抽样点数量的增多, 聚类的错误率在逐渐下降, 说明该算法是鲁棒的.

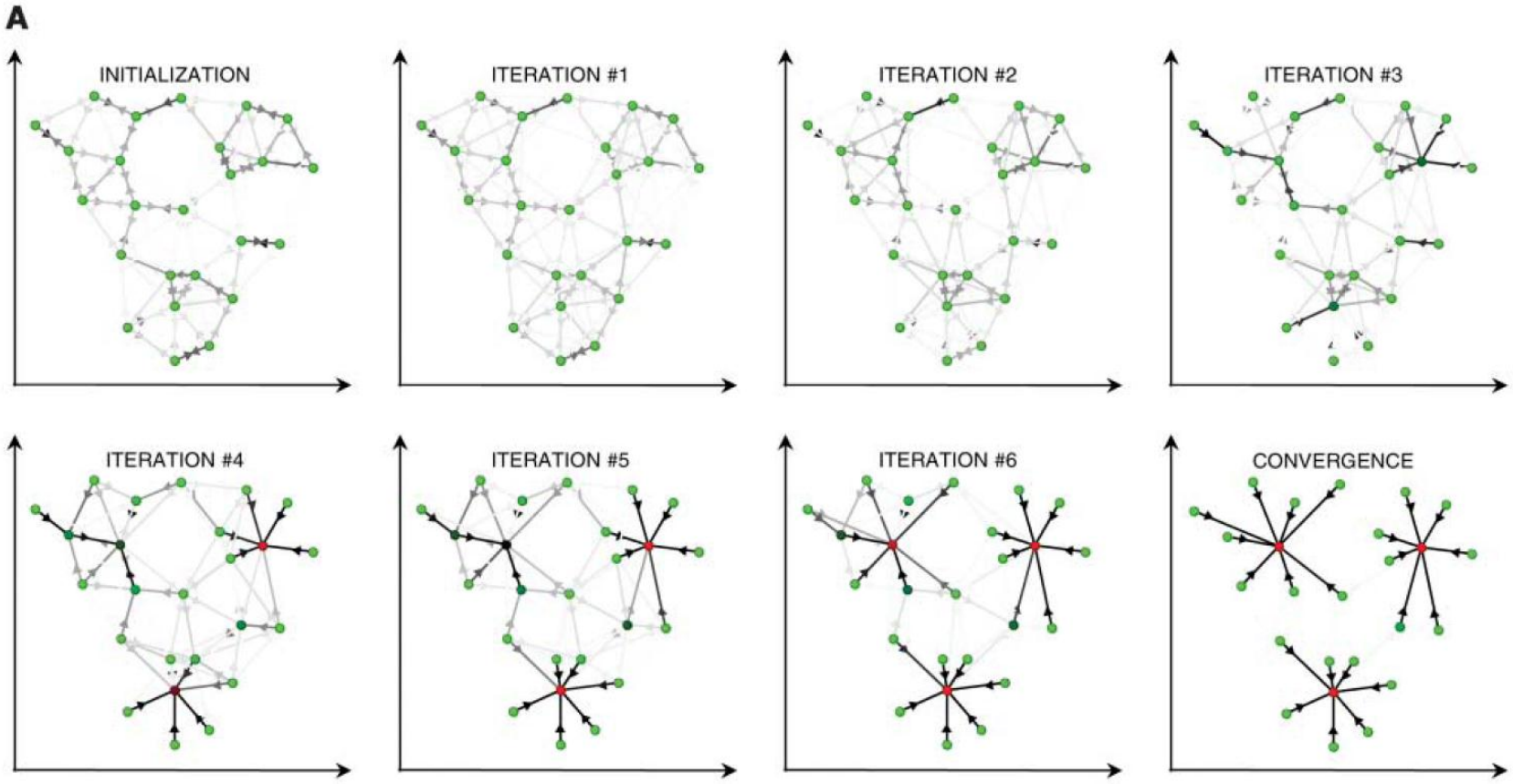
不同数据下密度最大值聚类的效果



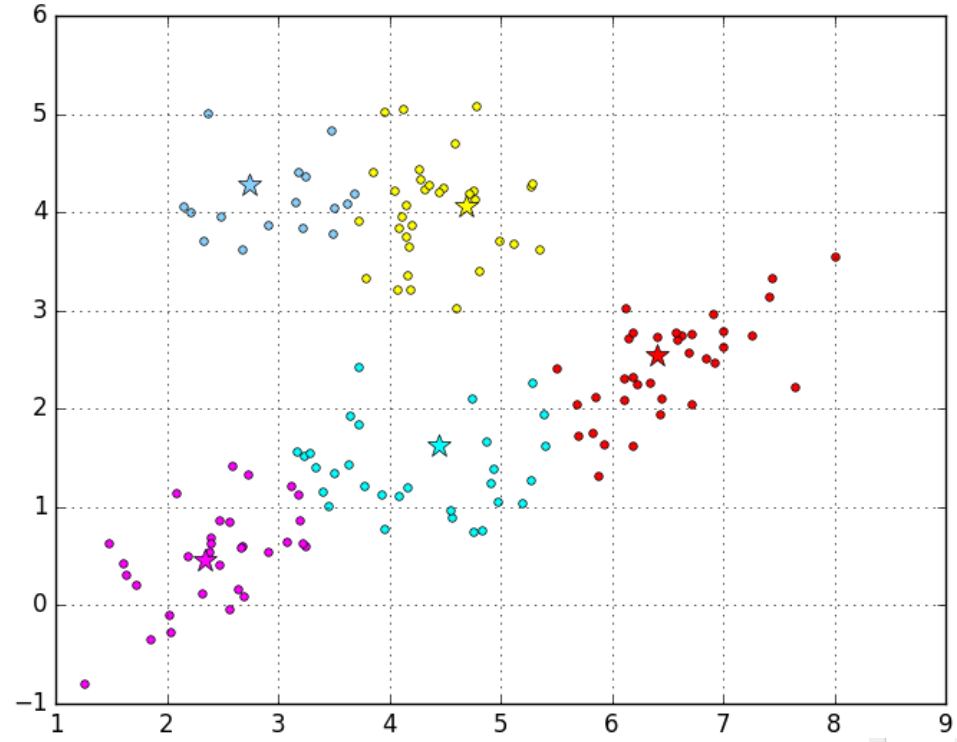
$$r(i,k) \leftarrow s(i,k) - \max_{k' \text{ s.t. } k' \neq k} \{a(i,k') + s(i,k')\}$$

Affinity Propagation

$$a(i,k) \leftarrow \min\left\{0, r(k,k) + \sum_{i' \text{ s.t. } i' \notin \{i,k\}} \max\{0, r(i',k)\}\right\}$$



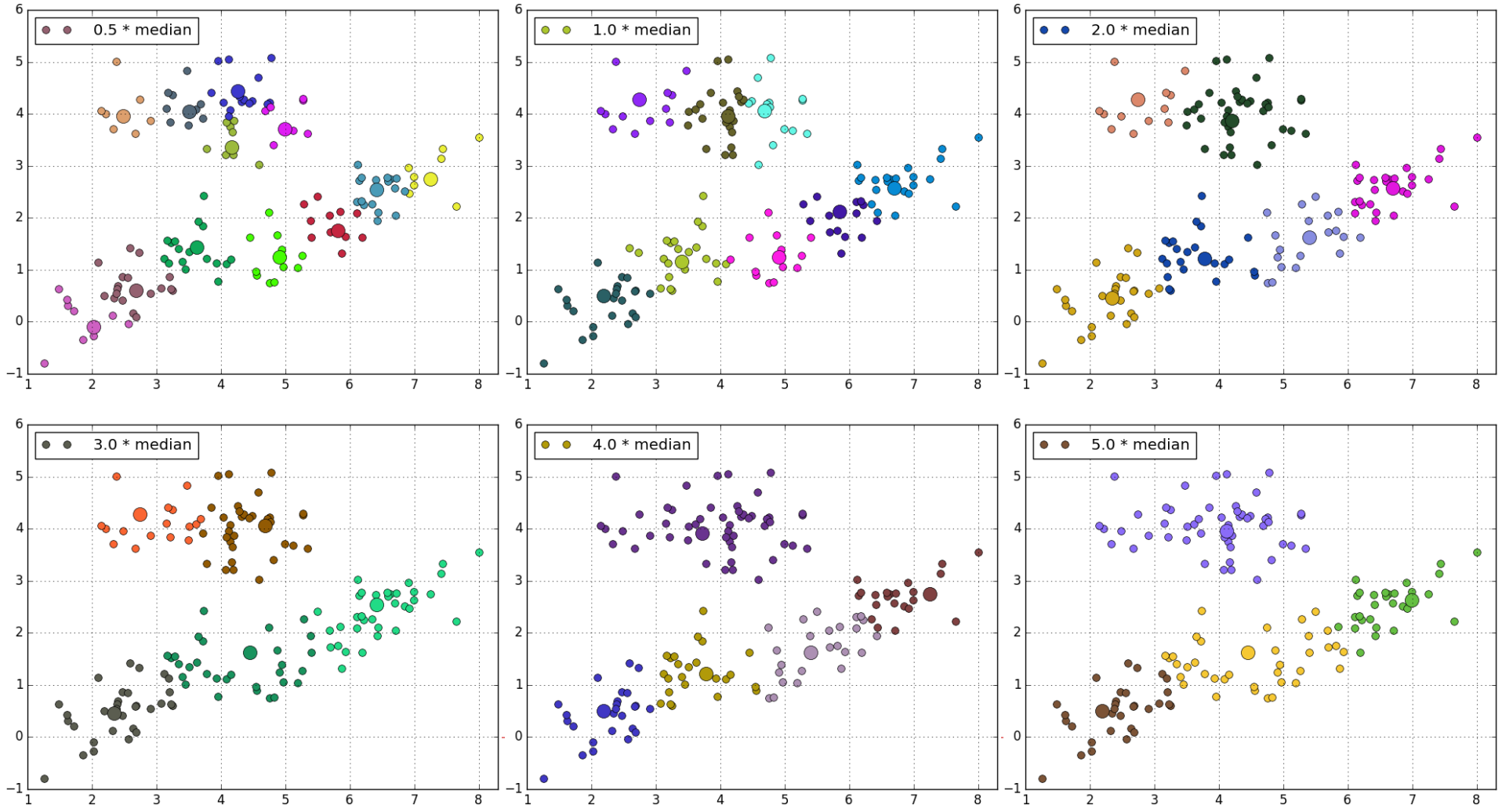
Code




```
    # Cluster
```

```
def affinity_propagation(data, factor):
    m, n = data.shape            # m样本个数，n样本维度
    s = np.zeros((m, m))
    for i in np.arange(m):
        for j in np.arange(i+1, m):
            s[i][j] = s[j][i] = -((data[i] - data[j]) ** 2).sum()
    p = factor*np.median(s)      # 自聚类因子
    print p
    for i in np.arange(m):
        s[i][i] = p
    r = np.zeros((m, m))     # r(i,k):i对k的依赖度
    a = np.zeros((m, m))     # a(i,k):k度i的适合度
    times = 100              # 迭代次数
    lamda = 0.5              # 阻尼因子
    cluster = np.zeros(m, dtype=np.int)
    center = {}
    for tt in np.arange(times):
        # r
        for i in np.arange(m):
            for k in np.arange(m):
                a_s = None  # a_s:a+s
                for t in np.arange(m):
                    if t != k:
                        if (a_s is None) or (a_s < a[i][t] + s[i][t]):
                            a_s = a[i][t] + s[i][t]
                r[i][k] = lamda * (s[i][k] - a_s) + (1-lamda)*r[i][k]

        # a
        for i in np.arange(m):
            for k in np.arange(m):
                if k == i:  # a[i][i]单独计算
                    continue
                r_s = r[k][k]     # sum(r[:k])
                for t in np.arange(m):
                    if (t != i) and (t != k):
                        r_s += max(r[t][k], 0)
                        # r_s += r[t][k]
                a[i][k] = lamda * min(0, r_s) + (1-lamda)*a[i][k]
            # 计算a[i][i]
            r_s = 0
            for t in np.arange(m):
                if t != i:
                    r_s += max(r[t][i], 0)
                    # r_s += r[t][i]
            a[i][i] = lamda * r_s + (1-lamda)*a[i][i]
```

AP算法调参



复习：实对称阵的特征值是实数

- 首先 $A\bar{x} = \bar{A}\bar{x} = \overline{Ax} = \overline{\lambda x} = \bar{\lambda}\bar{x}$
- 因为 $\bar{x}^T(Ax) = \bar{x}^T(Ax) = \bar{x}^T\lambda x = \lambda\bar{x}^T x$

  $\bar{x}^T(Ax) = (\bar{x}^T A^T)x = (A\bar{x})^T x = (\bar{\lambda}\bar{x})^T x = \bar{\lambda}\bar{x}^T x$
- 从而

  $$\lambda\bar{x}^T x = \bar{\lambda}\bar{x}^T x \Rightarrow (\lambda - \bar{\lambda})\bar{x}^T x = 0$$
- 而

  $$\bar{x}^T x = \sum_{i=1}^{n} \bar{x}_i x_i = \sum_{i=1}^{n} |x_i|^2 \neq 0$$
- 所以

  $$\lambda - \bar{\lambda} = 0 \Rightarrow \lambda = \bar{\lambda}$$

实对称阵不同特征值的特征向量正交

- 令实对称矩阵为A，其两个不同的特征值$\lambda_1\lambda_2$对应的特征向量分别是$\mu_1\mu_2$；
  - $\lambda_1\lambda_2\,\mu_1\mu_2$都是实数或是实向量。

$$\begin{cases} A\mu_1 = \lambda_1\mu_1 \\ A\mu_2 = \lambda_2\mu_2 \Rightarrow \mu_1^T A\mu_2 = \mu_1^T \lambda_2\mu_2 \end{cases}$$

$$\Rightarrow \left(A^T\mu_1\right)^T\mu_2 = \lambda_2\mu_1^T\mu_2 \Rightarrow \left(A\mu_1\right)^T\mu_2 = \lambda_2\mu_1^T\mu_2$$

$$\Rightarrow \left(\lambda_1\mu_1\right)^T\mu_2 = \lambda_2\mu_1^T\mu_2$$

$$\Rightarrow \lambda_1\mu_1^T\mu_2 = \lambda_2\mu_1^T\mu_2$$

$$\xrightarrow{\lambda_1\neq\lambda_2} \mu_1^T\mu_2 = 0$$

谱和谱聚类

- 方阵作为线性算子，它的所有特征值的全体统称方阵的**谱**。
  - 方阵的谱半径为最大的特征值
  - 矩阵A的**谱半径**：(A^TA)的最大特征值
- 谱聚类是一种基于图论的聚类方法，通过对样本数据的**拉普拉斯矩阵**的**特征向量**进行聚类，从而达到对样本数据聚类的目的。

谱分析的整体过程

- 给定一组数据$x_1,x_2,...x_n$，记任意两个点之间的相似度(“距离”的减函数)为$s_{ij}=<xi,xj>$，形成相似度图(similarity graph)：G=(V,E)。如果x_i和x_j之间的相似度s_{ij}大于一定的阈值，那么，两个点是连接的，权值记做s_{ij}。
- 接下来，可以用相似度图来解决样本数据的聚类问题：找到图的一个划分，形成若干个组(Group)，使得不同组之间有较低的权值，组内有较高的权值。

若干概念

- 无向图G=(V,E)
- 邻接矩阵$W = (w_{ij})_{i,j=1,\ldots,n}$
- 顶点的度di → 度矩阵D(对角阵)

$$d_i = \sum_{j=1}^{n} w_{ij}$$

若干概念

- 子图A的指示向量

$$\mathbb{1}_A = (f_1, \ldots, f_n)' \in \mathbb{R}^n$$
$$f_i = 1 \text{ if } v_i \in A$$
$$f_i = 0 \text{ otherwise}$$

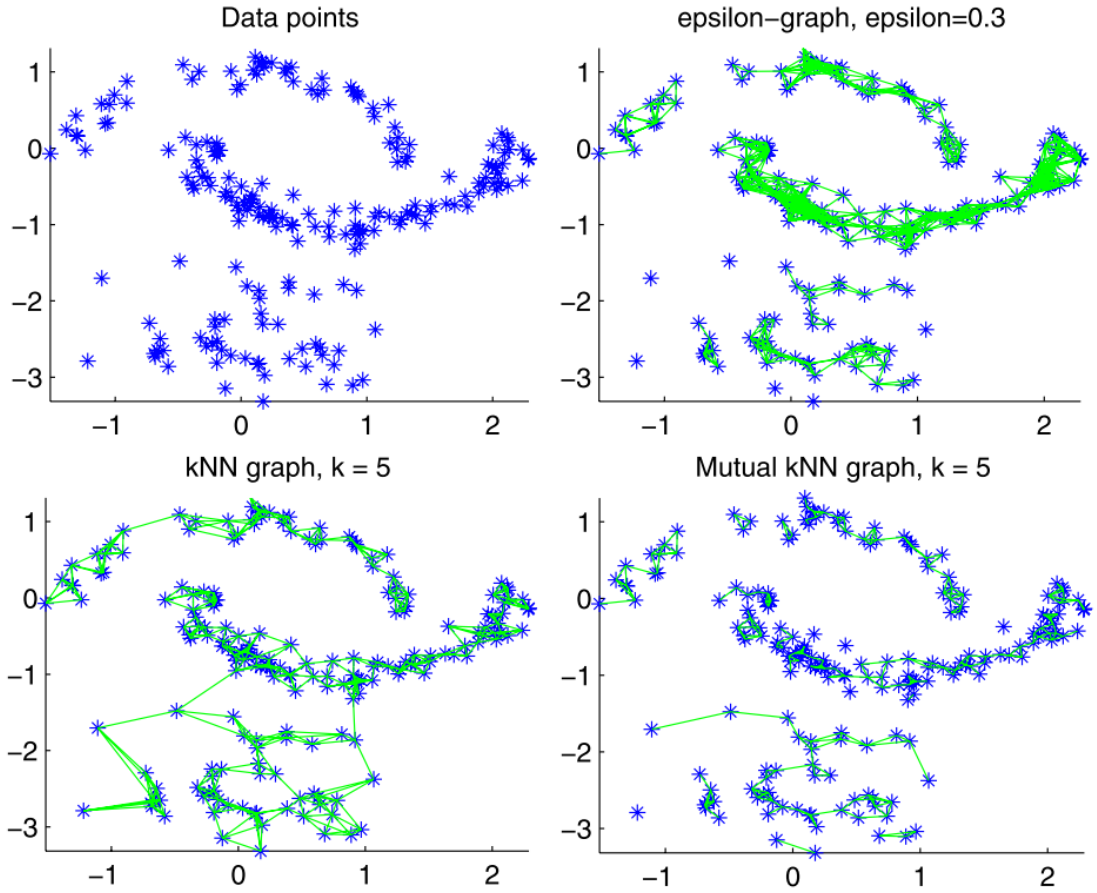
- A和B是图G的不相交子图，则定义子图的连接权：

$$W(A, B) := \sum_{i \in A, j \in B} w_{ij}$$

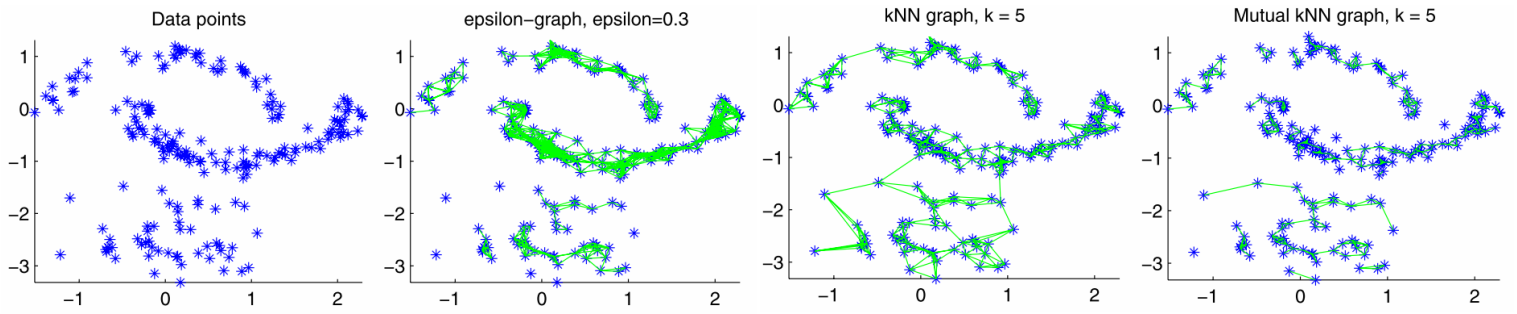
相似度图G的建立方法

- 全连接图：距离越大，相似度越小
  - 高斯相似度 $s(x_i, x_j) = \exp\left(\frac{-\|x_i - x_j\|^2}{2\sigma^2}\right)$
- ε近邻图
  - 给定参数ε/如何选择ε?
    - 图G的权值的均值
    - 图G的最小生成树的最大边
- k近邻图(k-nearest neighbor graph)
  - 若vi的k最近邻包含vj，vj的k最近邻不一定包含vi：有向图
  - 忽略方向的图，往往简称“k近邻图”
  - 两者都满足才连接的图，称作“互k近邻图(mutual)”

相似度图G的举例


Data points
epsilon−graph, epsilon=0.3
kNN graph, k = 5
Mutual kNN graph, k = 5

权值比较



- ε近邻图：ε=0.3，“月牙部分”非常紧的连接了，但“高斯部分”很多都没连接。当数据有不同的“密度”时，往往发生这种问题。
- k近邻图：可以解决数据存在不同密度时有些无法连接的问题，甚至低密度的“高斯部分”与高密度的“月牙部分”也能够连接。同时，虽然两个“月牙部分”的距离比较近，但k近邻还可以把它们区分开。
- 互k近邻图：它趋向于连接相同密度的部分，而不连接不同密度的部分。这种性质介于ε近邻图和k近邻图之间。如果需要聚类不同的密度，这个性质非常有用。
- 全连接图：使用高斯相似度函数可以很好的建立权值矩阵。但缺点是建立的矩阵不是稀疏的。
- 总结：首先尝试使用k近邻图。

拉普拉斯矩阵及其性质

- 拉普拉斯矩阵：$L = D - W$

$$f^T L f = f^T D f - f^T W f = \sum_{i=1}^{n} d_i f_i^2 - \sum_{i,j=1}^{n} f_i f_j w_{ij}$$

$$= \frac{1}{2}\left(\sum_{i=1}^{n} d_i f_i^2 - 2\sum_{i,j=1}^{n} f_i f_j w_{ij} + \sum_{j=1}^{n} d_j f_j^2 \right)$$

$$= \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}\left(f_i - f_j\right)^2$$

- L是对称半正定矩阵，最小特征值是0，相应的特征向量是全1向量。

拉普拉斯矩阵的定义

- 计算点之间的邻接相似度矩阵W
  - 若两个点的相似度值越大，表示这两个点越相似；
  - 同时，定义$w_{ij}=0$表示v_i,v_j两个点没有任何相似性(无穷远)
- W的第i行元素的和为v_i的度。形成顶点度对角阵D
  - d_{ii}表示第i个点的度
  - 除主对角线元素，D其他位置为0
- 未正则的拉普拉斯矩阵：$L=D-W$
- 正则拉普拉斯矩阵
  - 对称拉普拉斯矩阵 $L_{sym}=D^{-\frac{1}{2}}\cdot L\cdot D^{\frac{1}{2}}=I-D^{-\frac{1}{2}}\cdot W\cdot D^{\frac{1}{2}}$
  - 随机游走拉普拉斯矩阵 $L_{rw}=D^{-1}L=I-D^{-1}W$
    - Random walk

谱聚类算法：未正则拉普拉斯矩阵

- 输入：n个点$\{p_i\}$，簇的数目k
  - 计算n×n的相似度矩阵W和度矩阵D；
  - 计算拉普拉斯矩阵L=D-W；
  - 计算L的前k个特征向量$u_1,u_2,...,u_k$；
  - 将k个列向量$u_1,u_2,...,u_k$组成矩阵U，$U \in R^{n \times k}$；
  - 对于i=1,2,...,n,令$y_i \in R^k$是U的第i行的向量；
  - 使用k-means算法将点$(y_i)_{i=1,2,...,n}$聚类成簇$C_1,C_2,...C_k$；
  - 输出簇$A_1,A_2,...A_k$，其中，$Ai=\{j|y_j \in Ci\}$

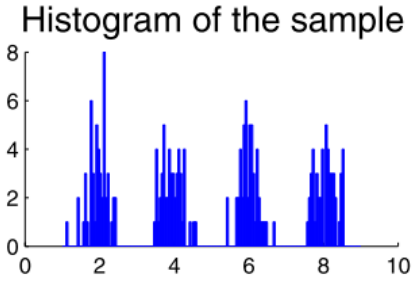
谱聚类算法：随机游走拉普拉斯矩阵

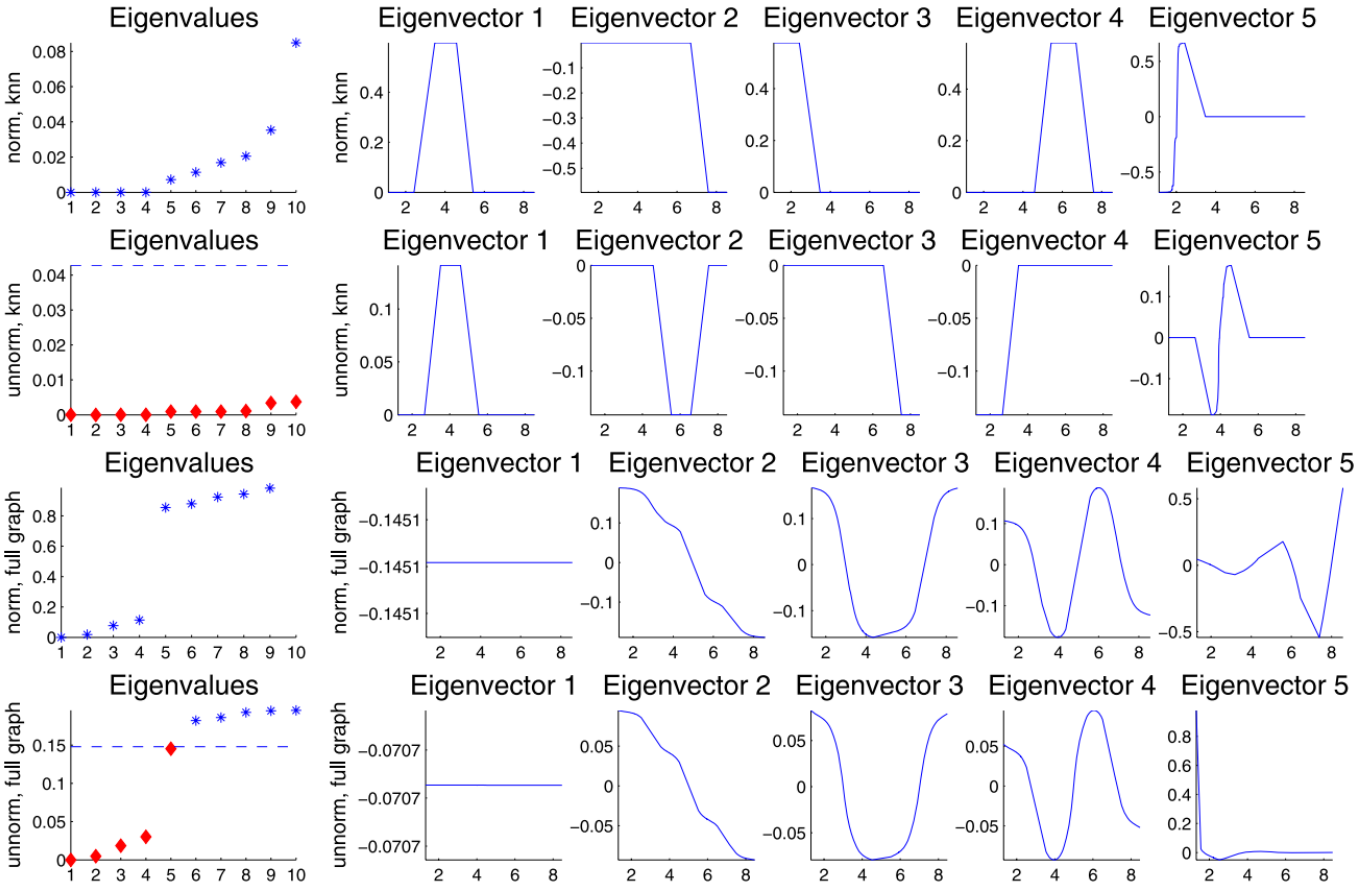
- 输入：n个点$\{p_i\}$，簇的数目k
  - 计算n×n的相似度矩阵W和度矩阵D；
  - 计算正则拉普拉斯矩阵$L_{rw}=D^{-1}(D-W)$；
  - 计算L_{rw}的前k个特征向量$u_1,u_2,...,u_k$；
  - 将k个列向量$u_1,u_2,...,u_k$组成矩阵U，$U\in R^{n\times k}$；
  - 对于i=1,2,...,n,令$y_i\in R^k$是U的第i行的向量；
  - 使用k-means算法将点$(y_i)_{i=1,2,...,n}$聚类成簇$C_1,C_2,...C_k$；
  - 输出簇$A_1,A_2,...A_k$，其中，$Ai=\{j|y_j\in Ci\}$

谱聚类算法：对称拉普拉斯矩阵

- 输入：n个点$\{p_i\}$，簇的数目k
  - 计算n×n的相似度矩阵W和度矩阵D；
  - 计算正则拉普拉斯矩阵$L_{sym}=D^{-1/2}(D-W)\ D^{-1/2}$；
  - 计算L_{sym}的前k个特征向量$u_1,u_2,...,u_k$；
  - 将k个列向量$u_1,u_2,...,u_k$组成矩阵U，$U\in R^{n\times k}$；
  - 对于i=1,2,...,n,令$y_i\in R^k$是U的第i行的向量；
  - 对于i=1,2,...,n,将$y_i\in R^k$依次单位化，使得$|y_i|=1$；
  - 使用k-means算法将点$(y_i)_{i=1,2,...,n}$聚类成簇$C_1,C_2,...C_k$；
  - 输出簇$A_1,A_2,...A_k$，其中，$Ai=\{j|y_j\in Ci\}$

一个实例


Histogram of the sample

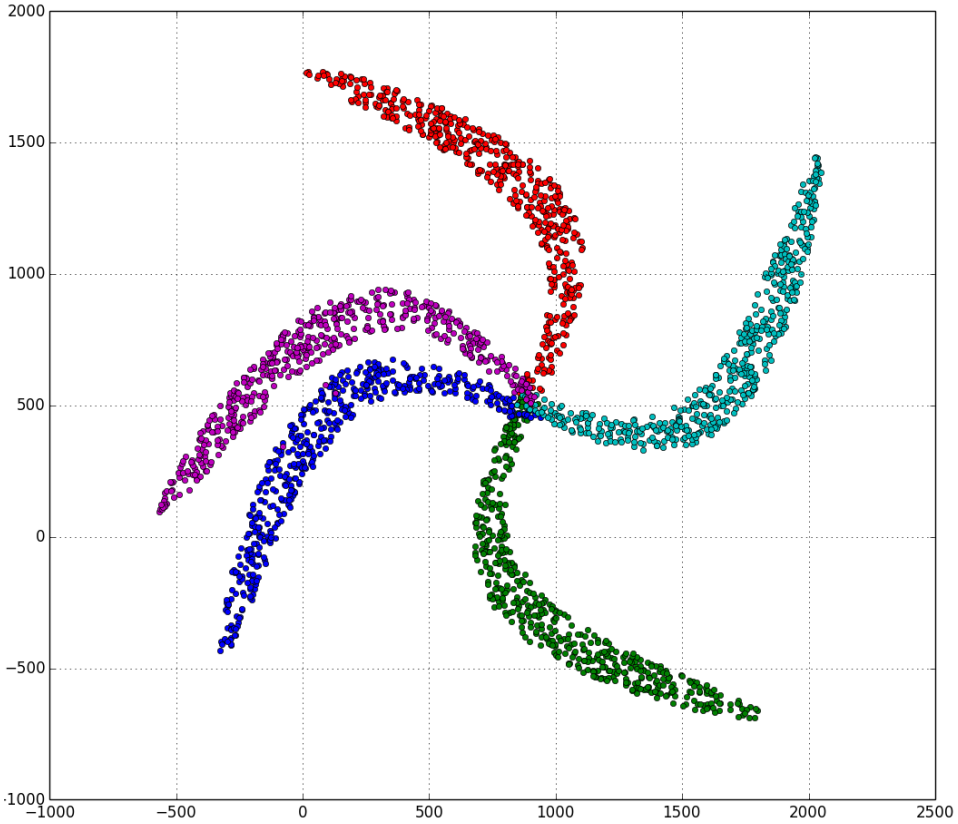

Eigenvalues
Eigenvector 1
Eigenvector 2
Eigenvector 3
Eigenvector 4
Eigenvector 5
norm, knn
unnorm, knn
norm, full graph
unnorm, full graph

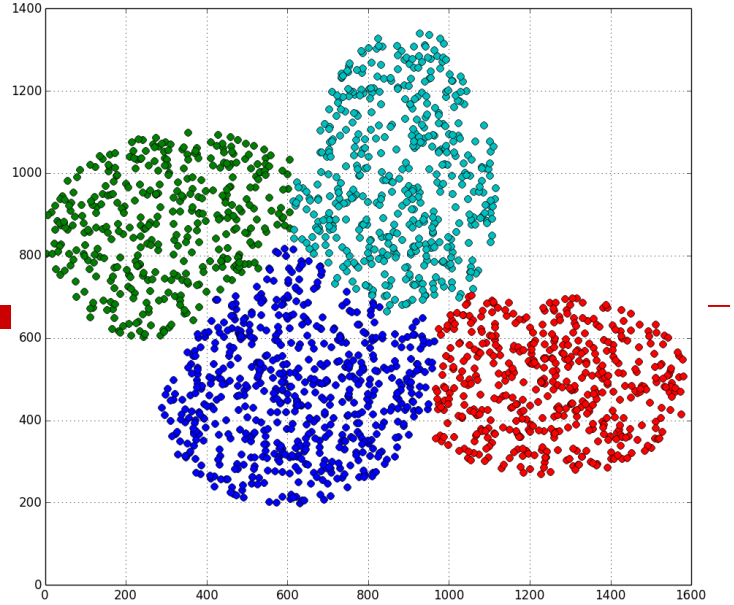
Code

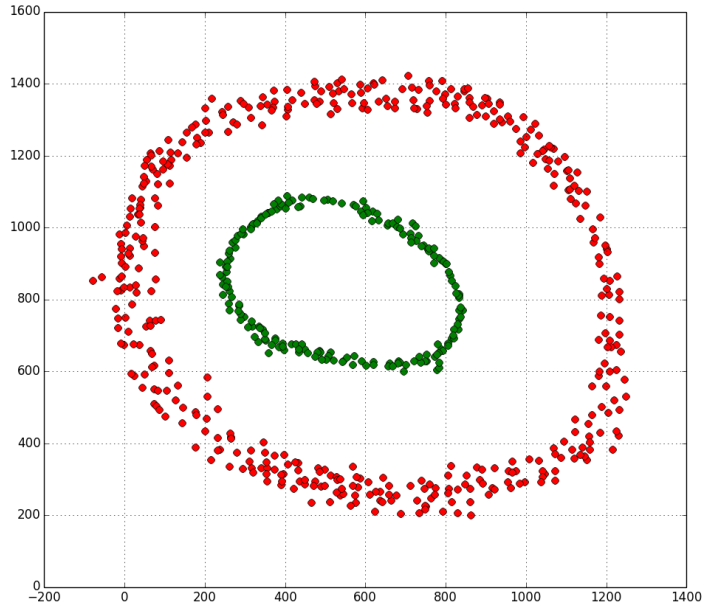
```
def spectral_cluster(data):
    lm = laplace_matrix(data)
    eg_values, eg_vectors = linalg.eig(lm)
    idx = eg_values.argsort()
    eg_vectors = eg_vectors[:, idx]

    m = len(data)
    eg_data = [[] for x in range(m)]
    for i in range(m):
        eg_data[i] = [0 for x in range(k)]
        for j in range(k):
            eg_data[i][j] = eg_vectors[i][j]
    return k_means(eg_data)
```

```
def laplace_matrix(data):
    m = len(data)
    w = [[] for x in range(m)]
    for i in range(m):
        w[i] = [0 for x in range(m)]
    nearest = [0 for x in range(neighbor)]

    for i in range(m):
        zero_list(nearest)
        for j in range(i+1, m):
            w[i][j] = similar(data, i, j)
            if not is_neighbor(w[i][j], nearest):
                w[i][j] = 0
            w[j][i] = w[i][j]   #对称
        w[i][i] = 0
    for i in range(m):
        s = 0
        for j in range(m):
            s += w[i][j]
        if s == 0:
            print "矩阵第", i, "行全为0"
            continue
        for j in range(m):
            w[i][j] /= s
            w[i][j] = -w[i][j]
        w[i][i] += 1    #单位阵主对角线为1
    return w
```

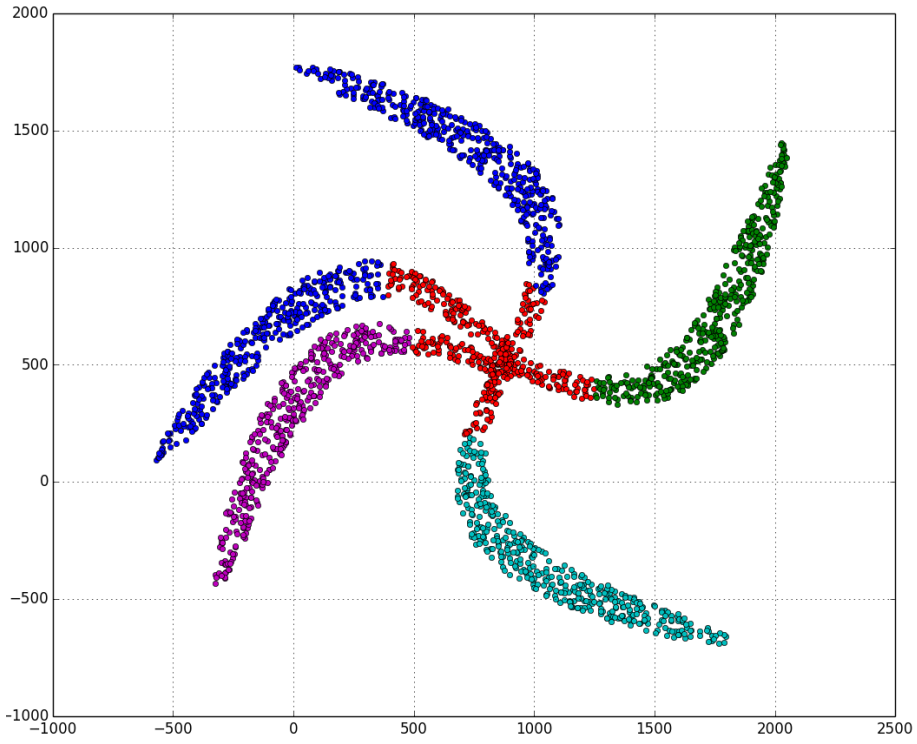
聚类效果

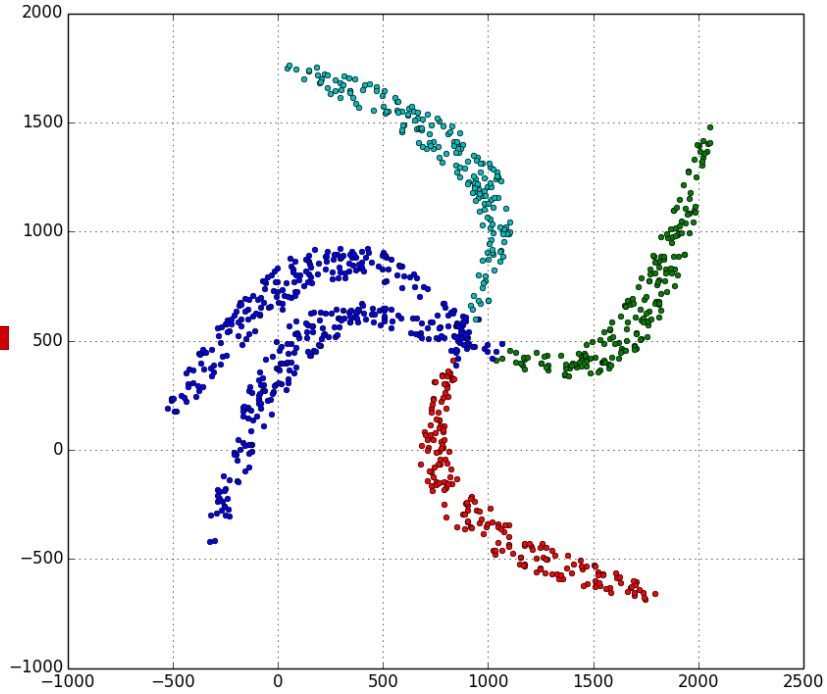

2000
1500
1000
500
0
−500
−1000
−1000
−500
0
500
1000
1500
2000
2500

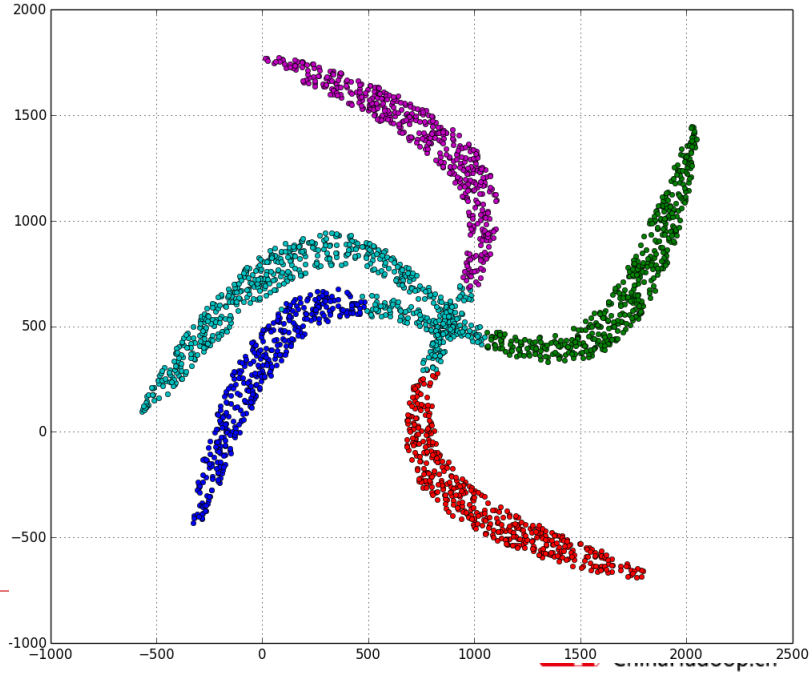

1400
1200
1000
800
600
400
200
0
0
200
400
600
800
1000
1200
1400
1600


1600
1400
1200
1000
800
600
400
200
0
−200
0
200
400
600
800
1000
1200
1400

聚类失败的情况


2000
1500
1000
500
0
−500
−1000
−1000
−500
0
500
1000
1500
2000
2500


2000
1500
1000
500
0
−500
−1000
−1000
−500
0
500
1000
1500
2000
2500


2000
1500
1000
500
0
−500
−1000
−1000
−500
0
500
1000
1500
2000
2500

进一步思考

- 谱聚类中的K如何确定？$k^* = \arg\max_k \left|\lambda_{k+1} - \lambda_k\right|$
- 最后一步K-Means的作用是什么？
  - 目标函数是关于子图划分指示向量的函数，该向量的值根据子图划分确定，是离散的。该问题是NP的，转换成求连续实数域上的解，最后用K-Means算法离散化。
- 未正则/对称/随机游走拉普拉斯矩阵，首选哪个？
  - 随机游走拉普拉斯矩阵
- 谱聚类可以用切割图/随机游走/扰动论等解释。

随机游走和拉普拉斯矩阵的关系

- 图论中的随机游走是一个随机过程，它从一个顶点跳转到另外一个顶点。谱聚类即找到图的一个划分，使得随机游走在相同的簇中停留而几乎不会游走到其他簇。
- 转移矩阵：从顶点vi跳转到顶点vj的概率正比于边的权值wij

$$p_{ij} = w_{ij} / d_i \qquad P = D^{-1}W$$

标签传递算法

- 对于部分样本的标记给定，而大多数样本的标记未知的情形，是半监督学习问题。
- 标签传递算法(Label Propagation Algorithm,LPA)，将标记样本的标记通过一定的概率传递给未标记样本，直到最终收敛。

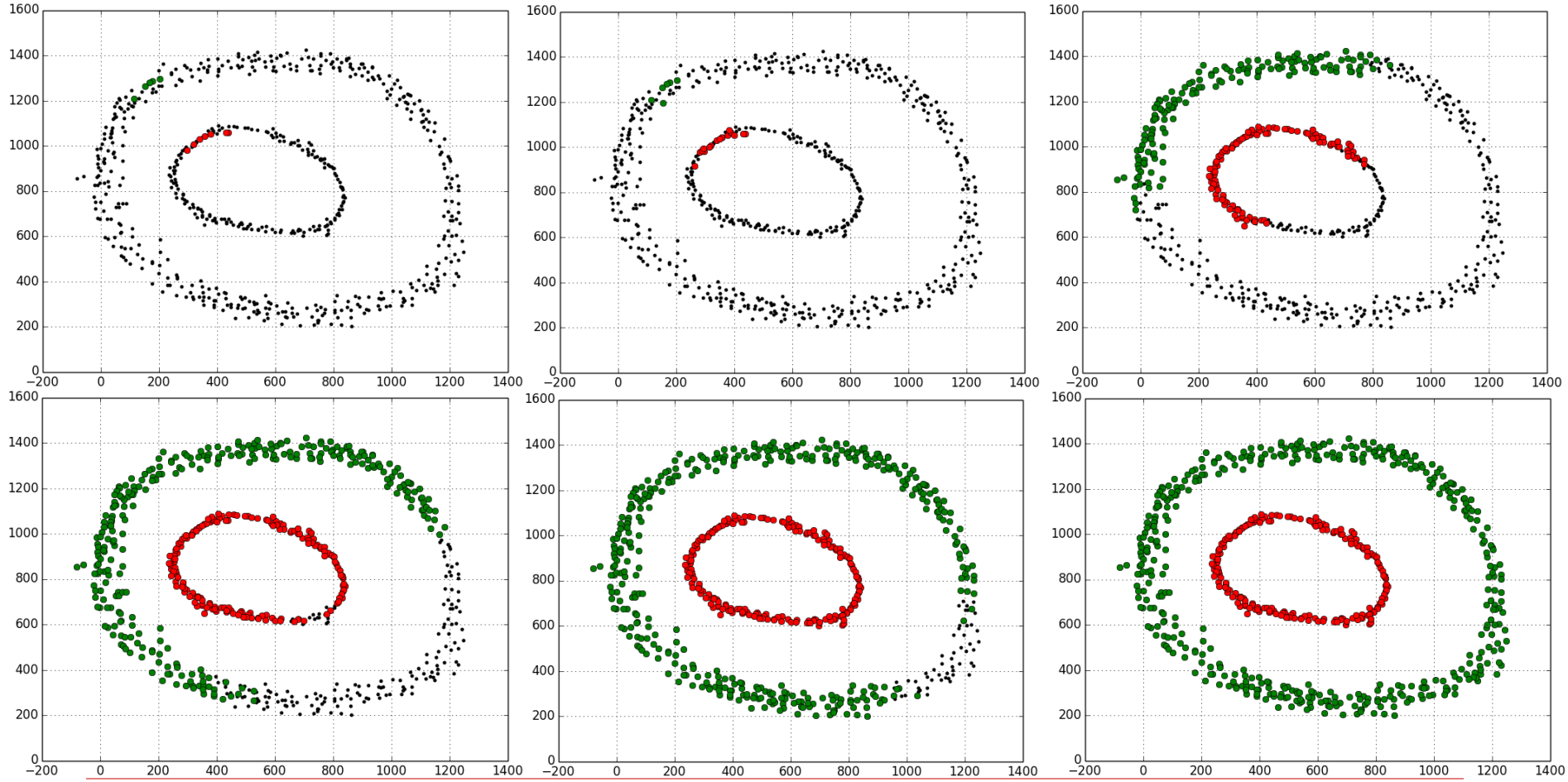
Code

```
def label_propagation(data, a):
    p = transition_matrix(data)
    m = len(data)
    n = len(data[0])
    for times in range(100):
        for i in range(a, m):
            j = calc_label(p, i)
            label = data[j][n-1]
            if label > 0:
                data[i][n-1] = label

def calc_label(p, i):
    n = len(p[i])
    k = random.random()    # k∈[0,1)
    r = n-1
    for j in range(n):
        if p[i][j] > k:
            r = j
            break
    return r
```

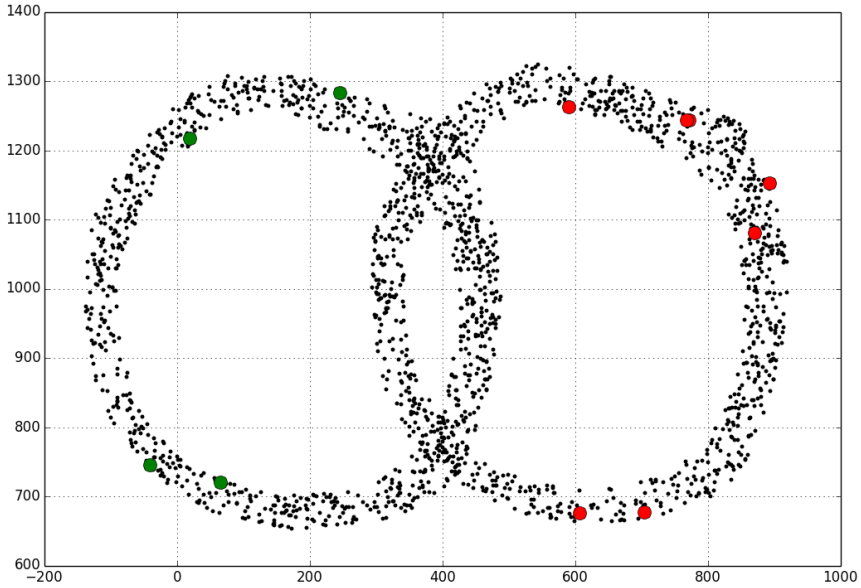
```
def transition_matrix(data):
    m = len(data)
    p = [[] for x in range(m)]
    for i in range(m):
        p[i] = [0 for x in range(m)]
    nearest = [0 for x in range(neighbor)]

    for i in range(m):
        zero_list(nearest)
        for j in range(i+1, m):
            p[i][j] = similar(data, i, j)
            if not is_neighbor(p[i][j], nearest):
                p[i][j] = 0
            p[j][i] = p[i][j]   #对称
        p[i][i] = 0
    for i in range(m):
        s = 0
        for j in range(m):
            s += p[i][j]
        if s == 0:
            print "矩阵第", i, "行全为0"
            continue
        for j in range(m):
            p[i][j] /= s
            if j != 0:
                p[i][j] += p[i][j-1]
    return p
```

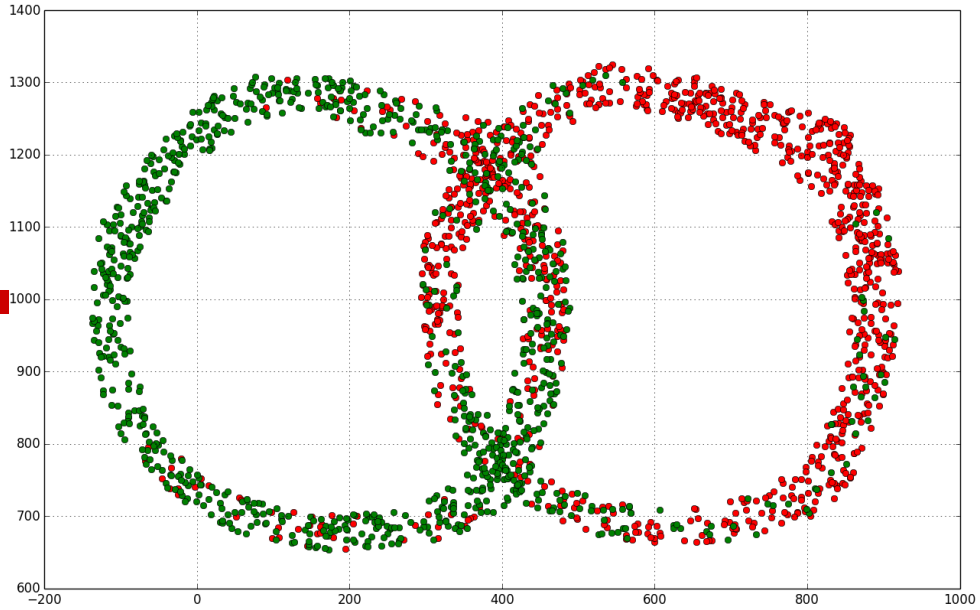
标签传递过程

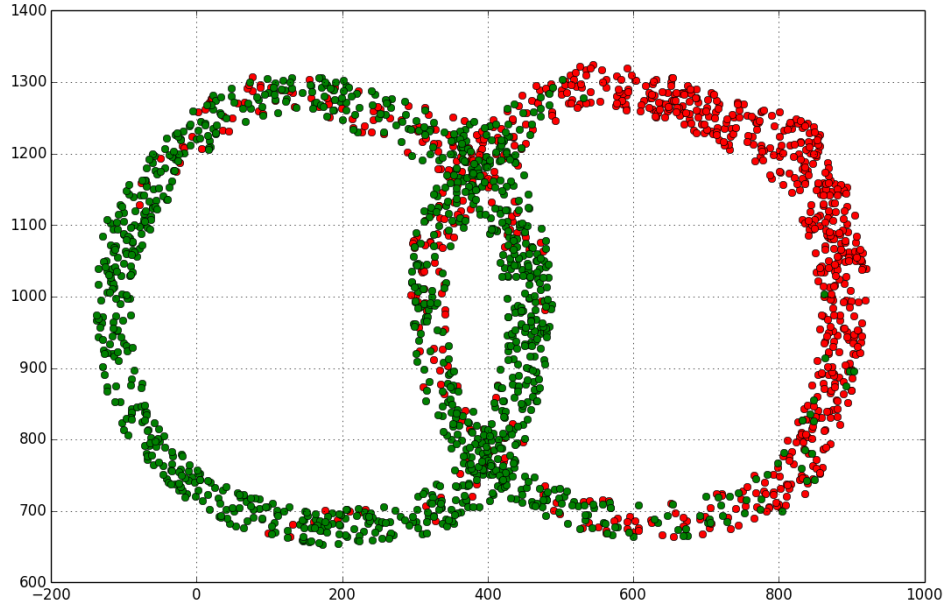
初始	1	10
20	30	40



带宽/邻域影响


1400
1300
1200
1100
1000
900
800
700
600
−200
0
200
400
600
800
1000


1400
1300
1200
1100
1000
900
800
700
600
−200
0
200
400
600
800
1000


1400
1300
1200
1100
1000
900
800
700
600
−200
0
200
400
600
800
1000

参考文献

- Alex Rodriguez, Alessandro Laio. *Clustering by fast search and find of density peak*. Science. 2014
- Ulrike von Luxburg. *A tutorial on spectral clustering*. 2007
- Lang K. *Fixing two weaknesses of the spectral method*. Advances in Neural Information Processing Systems 18, 715–722. MIT Press, Cambridge, 2006
- Bach F, Jordan M. *Learning spectral clustering*. Advances in Neural Information Processing Systems 16 (NIPS). 305–312. MIT Press, Cambridge,2004
- R.J.G.B. Campello, D. Moulavi, A. Zimek and J. Sander Hierarchical Density Estimates for Data Clustering, Visualization, and Outlier Detection, ACM Trans. on Knowledge Discovery from Data, Vol 10, 1 (July 2015), 1-51.

我们在这里

- http://wenda.ChinaHadoop.cn
  - 视频/课程/社区
- 微博
  - @ChinaHadoop
  - @邹博_机器学习
- 微信公众号
  - 小象学院
  - 大数据分析挖掘

感谢大家！

恳请大家批评指正！