

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



小象学院
ChinaHadoop.cn

邹博

本次目标

- 理解相似度度量的各种方法与相互联系
- 掌握K-means聚类的思路和使用条件
- 了解层次聚类的思路和方法
- 理解密度聚类并能够应用于实践
 - DBSCAN
 - 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

聚类的定义

- 聚类就是对大量未知标注的数据集，按数据的内在相似性将数据集划分为多个类别，使类别内的数据相似度较大而类别间的数据相似度较小
- 无监督

相似度/距离计算方法总结

- 闵可夫斯基距离 Minkowski/ 欧式距离 $dist(X, Y) = \left(\sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}}$
- 杰卡德相似系数 (Jaccard) $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$
- 余弦相似度 (cosine similarity) $\cos(\theta) = \frac{a^T b}{|a| \cdot |b|}$
- Pearson 相似系数 $\rho_{XY} = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y} = \frac{\sum_{i=1}^n (X_i - \mu_X)(Y_i - \mu_Y)}{\sqrt{\sum_{i=1}^n (X_i - \mu_X)^2} \sqrt{\sum_{i=1}^n (Y_i - \mu_Y)^2}}$
- 相对熵 (K-L 距离) $D(p \parallel q) = \sum_x p(x) \log \frac{p(x)}{q(x)} = E_{p(x)} \log \frac{p(x)}{q(x)}$
- Hellinger 距离 $D_\alpha(p \parallel q) = \frac{2}{1-\alpha^2} \left(1 - \int p(x)^{\frac{1+\alpha}{2}} q(x)^{\frac{1-\alpha}{2}} dx \right)$

Hellinger distance

$$\begin{aligned} D_\alpha(p \parallel q) &= \frac{2}{1-\alpha^2} \left(1 - \int p(x)^{\frac{1+\alpha}{2}} q(x)^{\frac{1-\alpha}{2}} dx \right) \\ \Rightarrow D_H(p \parallel q) &= 2 \left(1 - \int \sqrt{p(x)q(x)} dx \right) \quad s.t. \quad \alpha = 0 \\ &= 2 - 2 \int \sqrt{p(x)q(x)} dx \\ &= \int p(x)dx + \int q(x)dx - \int 2\sqrt{p(x)q(x)} dx \\ &= \int (p(x) - 2\sqrt{p(x)q(x)} + q(x)) dx \\ &= \int (\sqrt{p(x)} - \sqrt{q(x)})^2 dx \end{aligned}$$

□ 该距离满足三角不等式，是对称、非负距离

余弦相似度与Pearson相似系数

- n 维向量 x 和 y 的夹角记做 θ , 根据余弦定理, 其余弦值为:

$$\cos(\theta) = \frac{x^T y}{|x| \cdot |y|} = \frac{\sum_{i=1}^n x_i y_i}{\sqrt{\sum_{i=1}^n x_i^2} \sqrt{\sum_{i=1}^n y_i^2}}$$

- 这两个向量的相关系数是:

$$\rho_{XY} = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y} = \frac{\sum_{i=1}^n (x_i - \mu_X)(y_i - \mu_Y)}{\sqrt{\sum_{i=1}^n (x_i - \mu_X)^2} \sqrt{\sum_{i=1}^n (y_i - \mu_Y)^2}}$$

- 相关系数即为将 x 、 y 坐标向量各自平移到原点后的夹角余弦!
 - 这即解释了为何文档间求距离使用夹角余弦——因为这一物理量表征了文档去均值化后的随机向量间相关系数。

聚类的基本思想

- 给定一个有N个对象的数据集，构造数据的k个簇， $k \leq n$ 。满足下列条件：
 - 每一个簇至少包含一个对象
 - 每一个对象属于且仅属于一个簇
 - 将满足上述条件的k个簇称作一个合理划分
- 基本思想：对于给定的类别数目k，首先给出初始划分，通过迭代改变样本和簇的隶属关系，使得每一次改进之后的划分方案都较前一次好。

k-Means算法

- k-Means算法，也被称为k-平均或k-均值，是一种广泛使用的聚类算法，或者成为其他聚类算法的基础。
- 假定输入样本为 $S=x_1, x_2, \dots, x_m$ ，则算法步骤为：
 - 选择初始的k个类别中心 $\mu_1, \mu_2, \dots, \mu_k$
 - 对于每个样本 x_i ，将其标记为距离类别中心最近的类别，即：

$$label_i = \arg \min_{1 \leq j \leq k} \|x_i - \mu_j\|$$

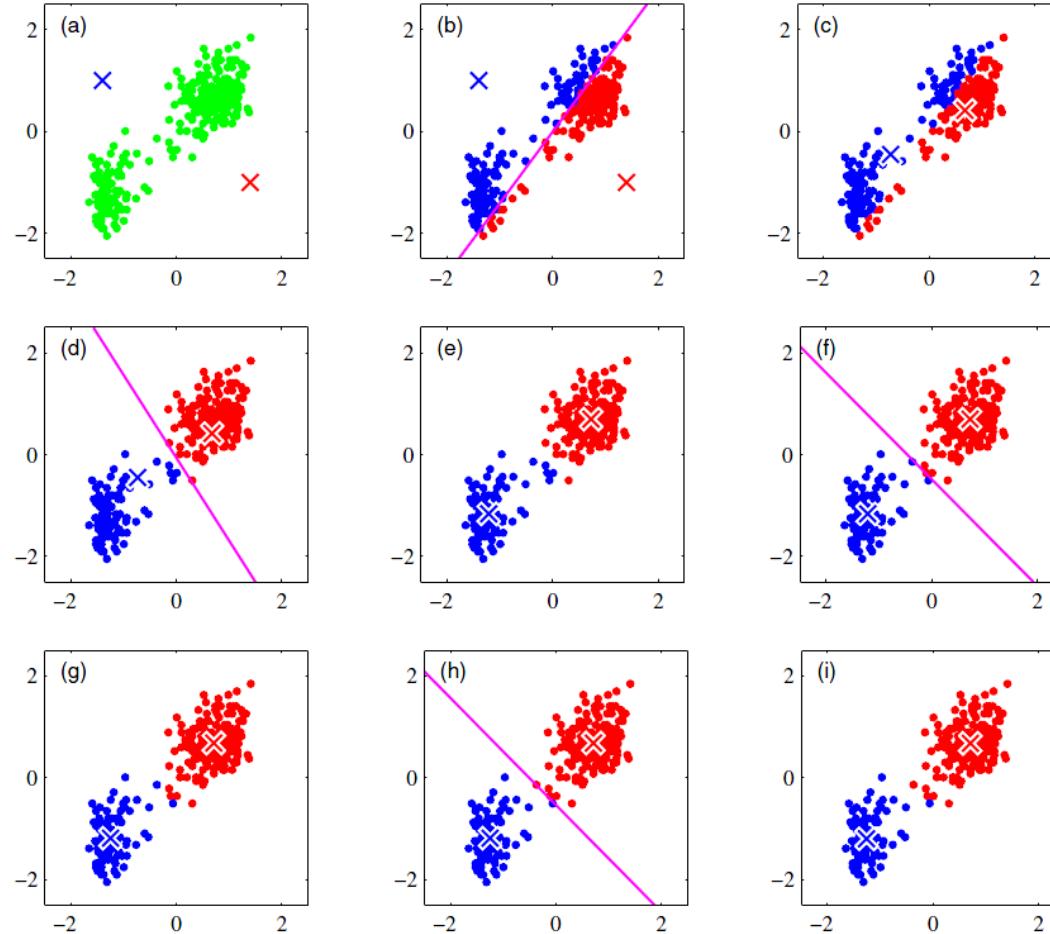
- 将每个类别中心更新为隶属该类别的所有样本的均值

$$\mu_j = \frac{1}{|c_j|} \sum_{i \in c_j} x_i$$

- 重复最后两步，直到类别中心的变化小于某阈值。

- 中止条件：
 - 迭代次数/簇中心变化率/最小平方误差MSE(Minimum Squared Error)

k-Means过程



Code

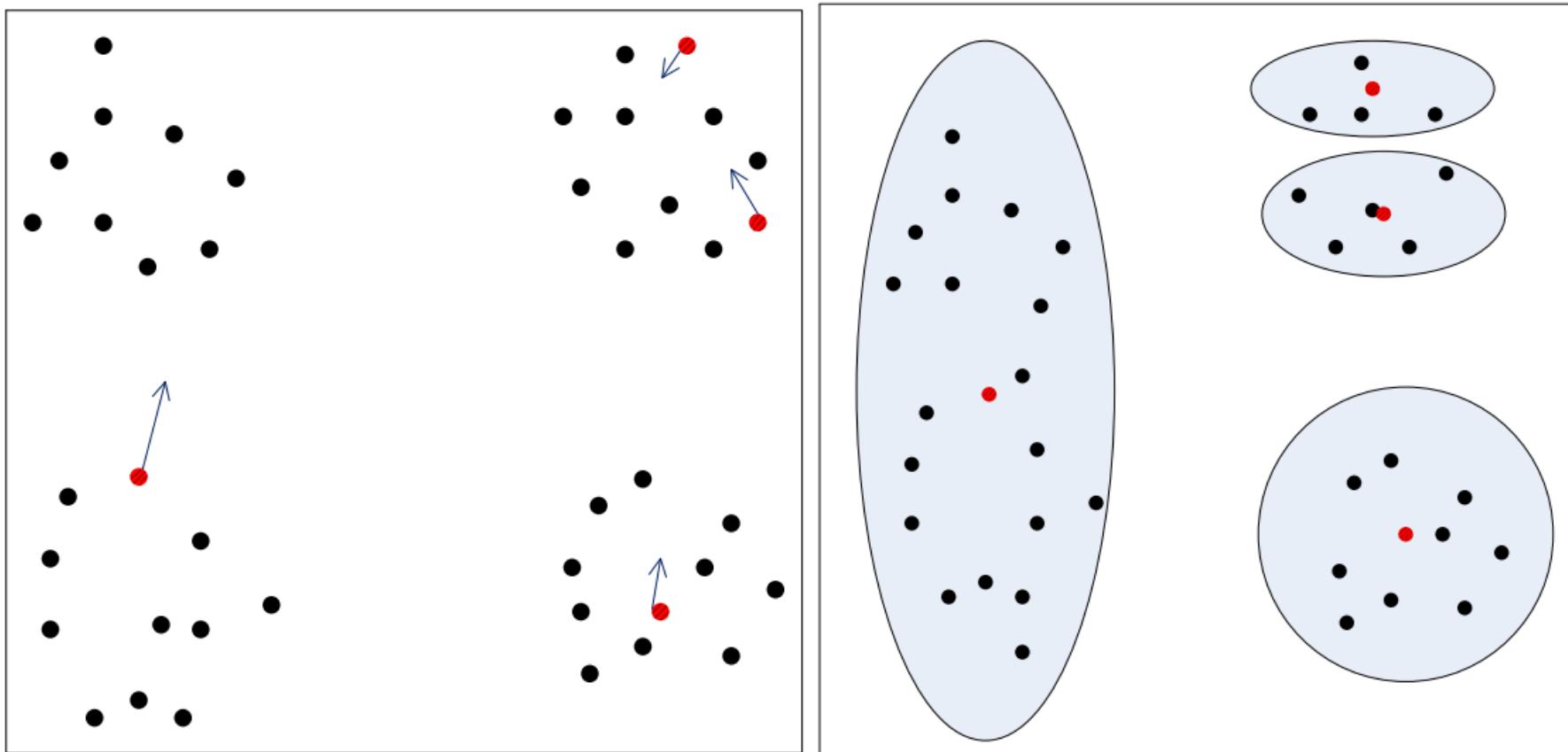
```
def k_means(data):
    m = len(data)
    n = len(data[0])
    cluster = [-1 for x in range(m)]      # 所有样本尚未聚类
    cluster_center = [[] for x in range(k)] # 聚类中心
    cc = [[] for x in range(k)]           # 下一轮的聚类中心
    c_number = [0 for x in range(k)]       # 每个簇中样本的数目

    # 随机选择簇中心
    i = 0
    while i < k:
        j = random.randint(0, m-1)
        if is_similar(data[j], cluster_center):
            continue
        cluster_center[i] = data[j][:]
        cc[i] = [0 for x in range(n)]
        i += 1
    for times in range(40):
        for i in range(m):
            c = nearest(data[i], cluster_center)
            cluster[i] = c      # 第i个样本归于第c簇
            c_number[c] += 1
            add(cc[c], data[i])
        for i in range(k):
            divide(cc[i], c_number[i])
            c_number[i] = 0
            cluster_center[i] = cc[i][:]
            zero_list(cc[i])
        print times, cluster_center
    return cluster
```

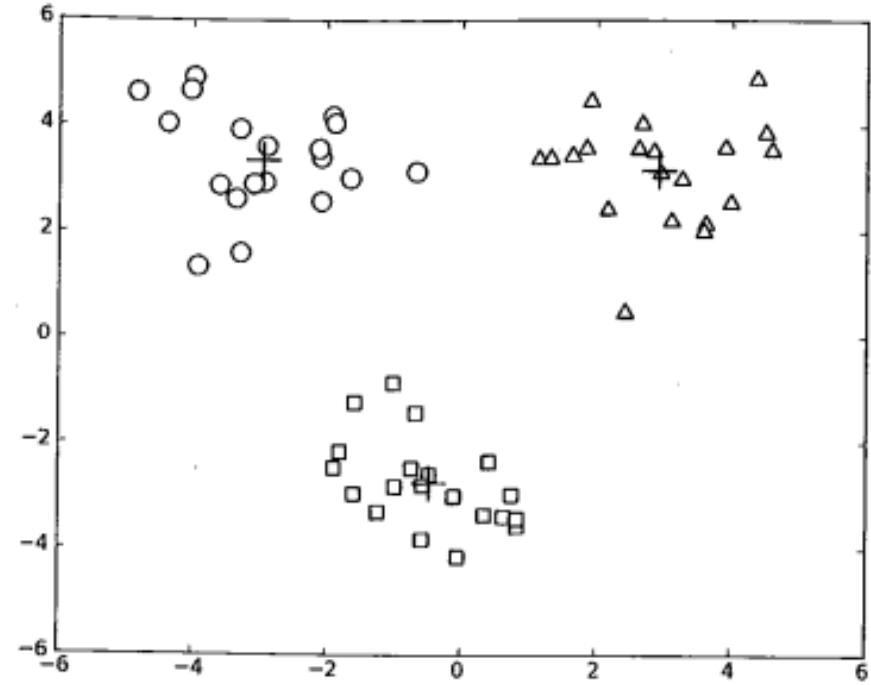
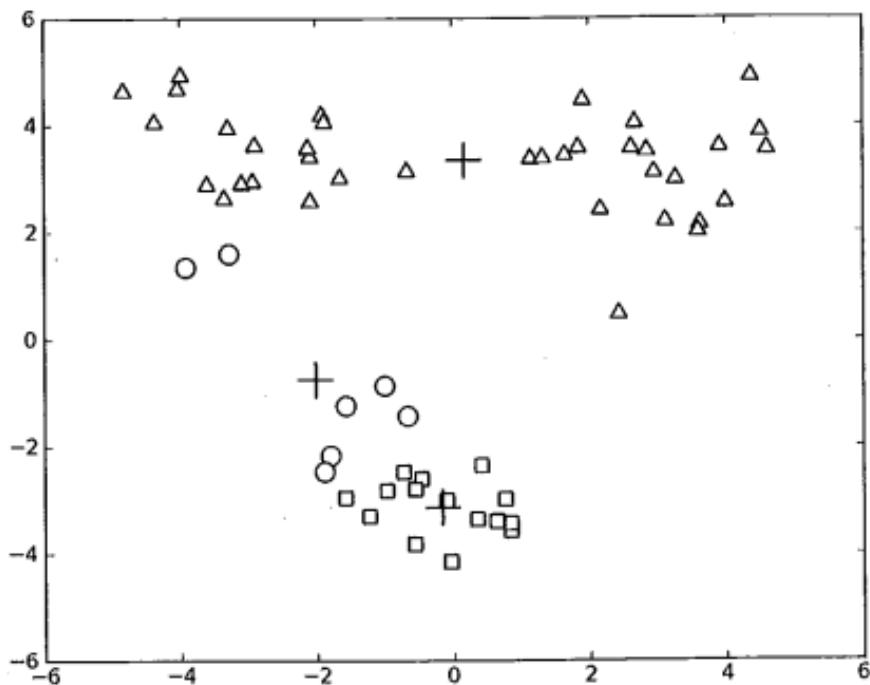
对k-Means的思考

- k-Means将簇中所有点的均值作为新质心，若簇中含有异常点，将导致均值偏离严重。
以一维数据为例：
 - 数组1、2、3、4、100的均值为22，显然距离“大多数”数据1、2、3、4比较远
 - 改成求数组的中位数3，在该实例中更为稳妥。
 - 这种聚类方式即**k-Mediods聚类(K中值距离)**
- 初值的选择，对聚类结果有影响吗？
 - 如何避免？

k-Means是初值敏感的



二分k-Means



Code2

```
def k_means(data):
    m = len(data)          # 样本个数
    n = len(data[0])        # 维度
    cluster_center = np.zeros((k, n))  # 聚类中心

    # 选择合适的初始聚类中心
    j = np.random.randint(m)      # [0, m)
    cluster_center[0] = data[j][:]
    dis = np.zeros(m)-1          # 样本到当前所有聚类中心的最近距离
    i = 0
    while i < k-1:
        for j in range(m):      # j: 样本
            d = (cluster_center[i]-data[j]) ** 2  # data[j] 与最新聚类中心 cluster_center[i] 的距离
            d = np.sum(d)
            if (dis[j] < 0) or (dis[j] > d):
                dis[j] = d
        j = random_select(dis)  # 按照 dis 加权选择样本 j
        i += 1
        cluster_center[i] = data[j][:]

    # 聚类
    cluster = np.zeros(m, dtype=np.int) - 1      # 所有样本尚未聚类
    cc = np.zeros((k, n))                      # 下一轮的聚类中心
    c_number = np.zeros(k)                     # 每个簇中样本的数目
    for times in range(40):
        for i in range(m):
            c = nearest(data[i], cluster_center)
            cluster[i] = c                      # 第 i 个样本归于第 c 簇
            c_number[c] += 1
            cc[c] += data[i]
        for i in range(k):
            cluster_center[i] = cc[i] / c_number[i]
    cc.flat = 0
    c_number.flat = 0
return cluster
```

k-Means的公式化解释

- 记K个簇中心为 $\mu_1, \mu_2, \dots, \mu_k$ ，每个簇的样本数目为 N_1, N_2, \dots, N_k
- 使用平方误差作为目标函数：

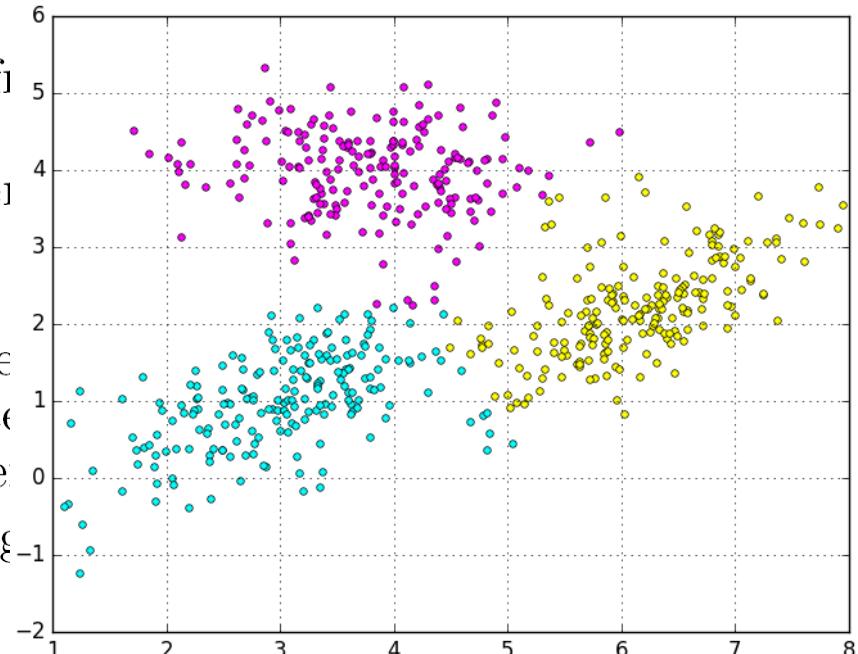
$$J(\mu_1, \mu_2, \dots, \mu_k) = \frac{1}{2} \sum_{j=1}^K \sum_{i=1}^{N_j} (x_i - \mu_j)^2$$

- 该函数为关于 $\mu_1, \mu_2, \dots, \mu_k$ 的凸函数，其驻点为： $\frac{\partial J}{\partial \mu_j} = -2 \sum_{i=1}^{N_j} (x_i - \mu_j) \xrightarrow{\text{令}} 0 \Rightarrow \mu_j = \frac{1}{N_j} \sum_{i=1}^{N_j} x_i$

Mini-batch k-Means算法描述

Mini-batch k -Means

```
1: Given:  $k$ , mini-batch size  $b$ , iterations  $t$ , data set  $X$ 
2: Initialize each  $\mathbf{c} \in C$  with an  $\mathbf{x}$  picked randomly from  $X$ 
3:  $\mathbf{v} \leftarrow 0$ 
4: for  $i = 1$  to  $t$  do
5:    $M \leftarrow b$  examples picked randomly from  $X$ 
6:   for  $\mathbf{x} \in M$  do
7:      $\mathbf{d}[\mathbf{x}] \leftarrow f(C, \mathbf{x})$  // Cache the ce
8:   end for
9:   for  $\mathbf{x} \in M$  do
10:     $\mathbf{c} \leftarrow \mathbf{d}[\mathbf{x}]$  // Get cached ce
11:     $\mathbf{v}[\mathbf{c}] \leftarrow \mathbf{v}[\mathbf{c}] + 1$  // Update per-ce
12:     $\eta \leftarrow \frac{1}{\mathbf{v}[\mathbf{c}]}$  // Get per-cente
13:     $\mathbf{c} \leftarrow (1 - \eta)\mathbf{c} + \eta\mathbf{x}$  // Take  $\xi_{-1}$ 
14:  end for
15: end for
```



Code3

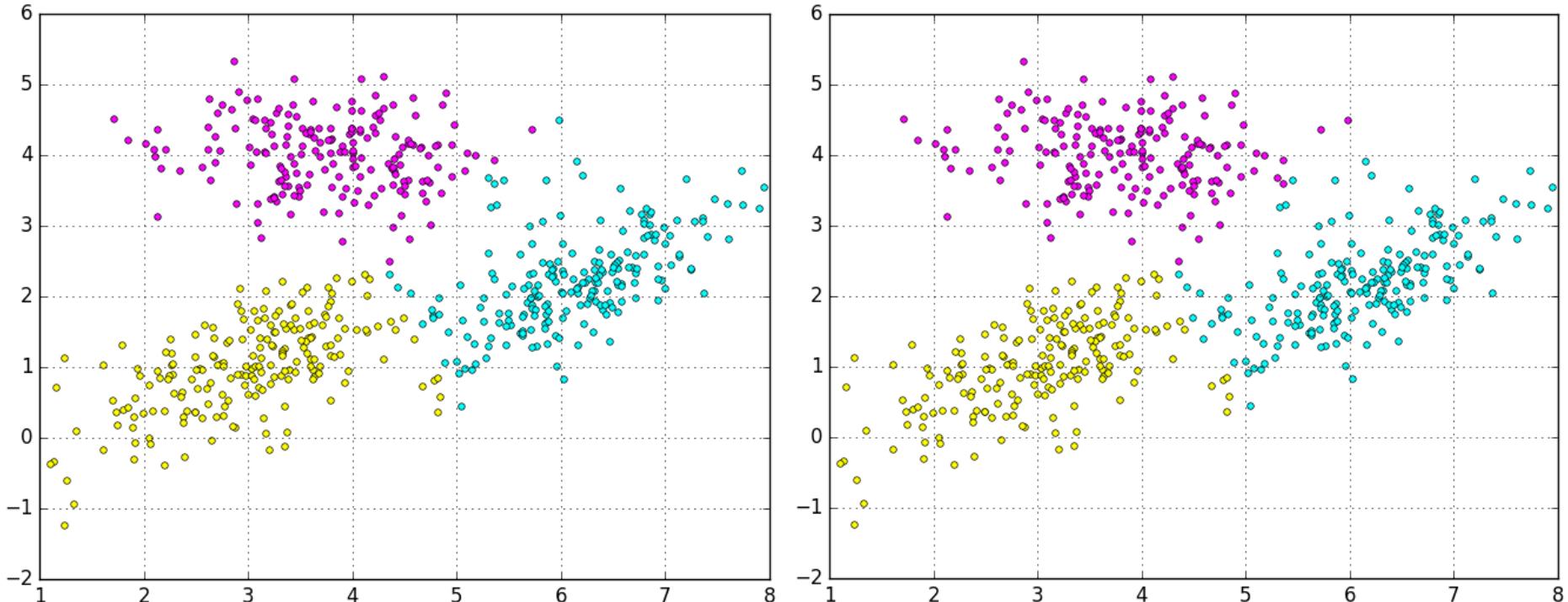
```
def k_means(data, k):
    m = len(data)          # 样本个数
    n = len(data[0])        # 维度
    cluster_center = np.zeros((k, n))  # 聚类中心

    # 选择合适的初始聚类中心
    j = np.random.randint(m)      # [0,m)
    cluster_center[0] = data[j][:]
    dis = np.zeros(m)-1          # 样本到当前所有聚类中心的最近距离
    i = 0

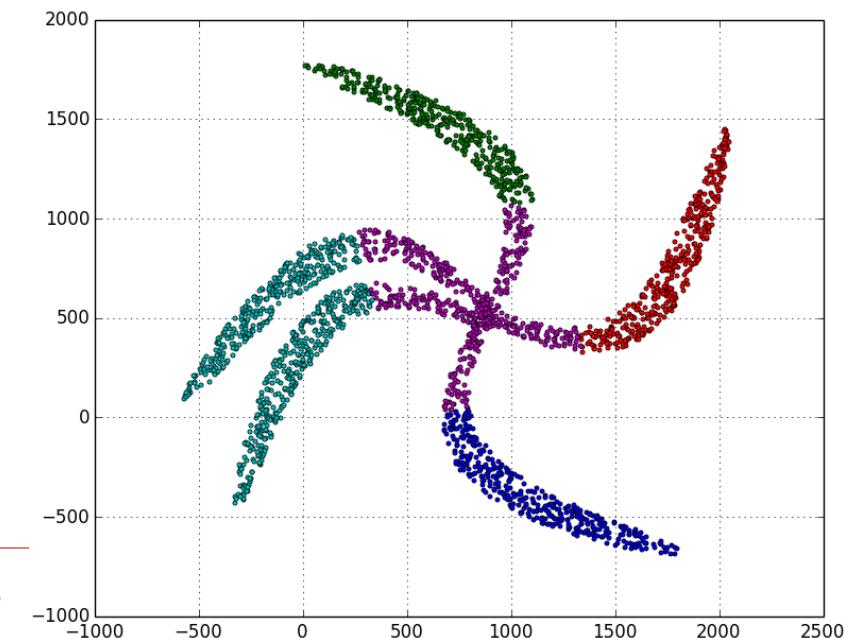
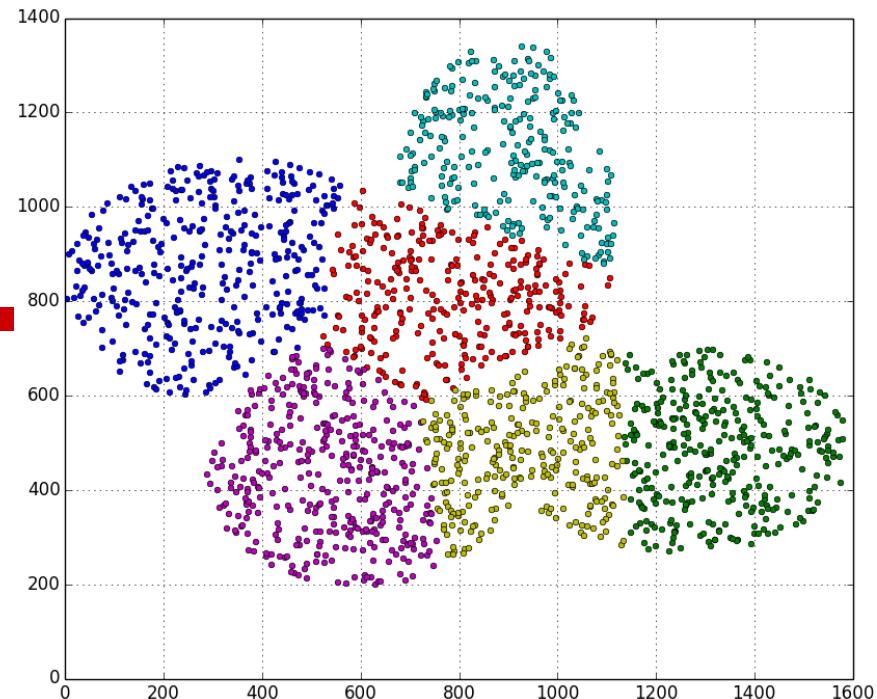
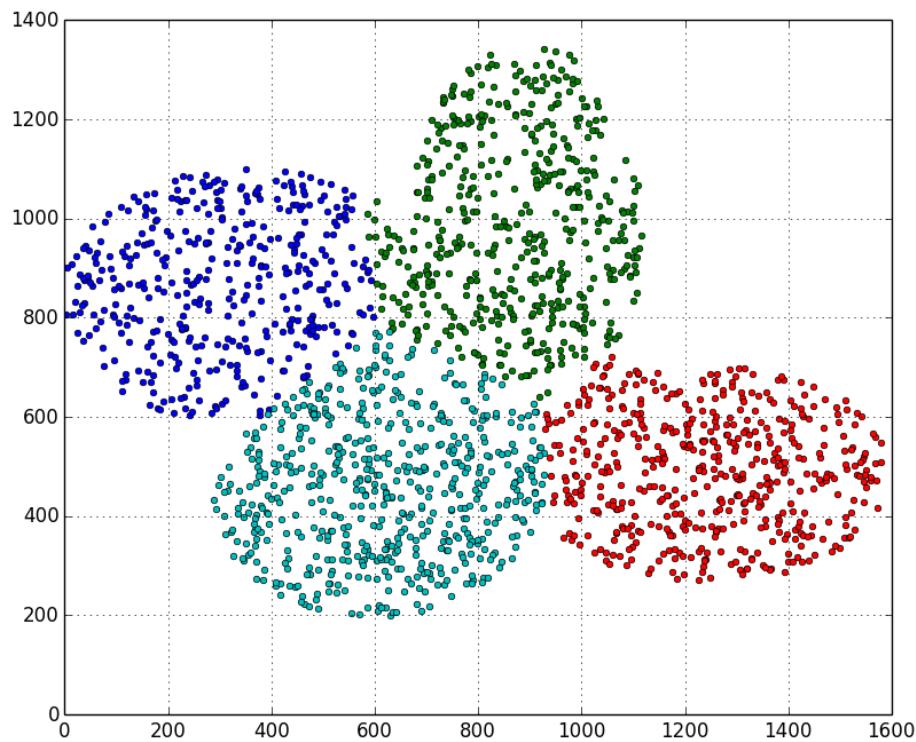
    while i < k-1:
        for j in range(m):      # j:样本
            d = (cluster_center[i]-data[j]) ** 2  # data[j]与最新聚类中心cluster_center[i]的距离
            d = np.sum(d)
            if (dis[j] < 0) or (dis[j] > d):
                dis[j] = d
        j = random_select(dis)  # 按照dis加权选择样本j
        i += 1
        cluster_center[i] = data[j][:]
        print "Find Cluster Center:", i

    # 聚类
    cluster = np.zeros(m, dtype=np.int) - 1      # 所有样本尚未聚类
    cc = np.zeros((k, n))                      # 下一轮的聚类中心
    c_number = np.zeros(k)                      # 每个簇的样本数目
    times = 50
    for t in range(times):
        for i in range(m):
            if np.random.random() * m > 100:
                continue
            c = nearest(data[i], cluster_center)
            cluster[i] = c      # 第i个样本归于第c簇
            cc[c] += data[i]
            c_number[c] += 1
        for i in range(k):
            cluster_center[i] = cc[i] / c_number[i]
        cc.flat = 0
        c_number.flat = 0
        print t, '%.2f%%' % (100 *float(t) / times)
        print cluster_center
    return cluster
```

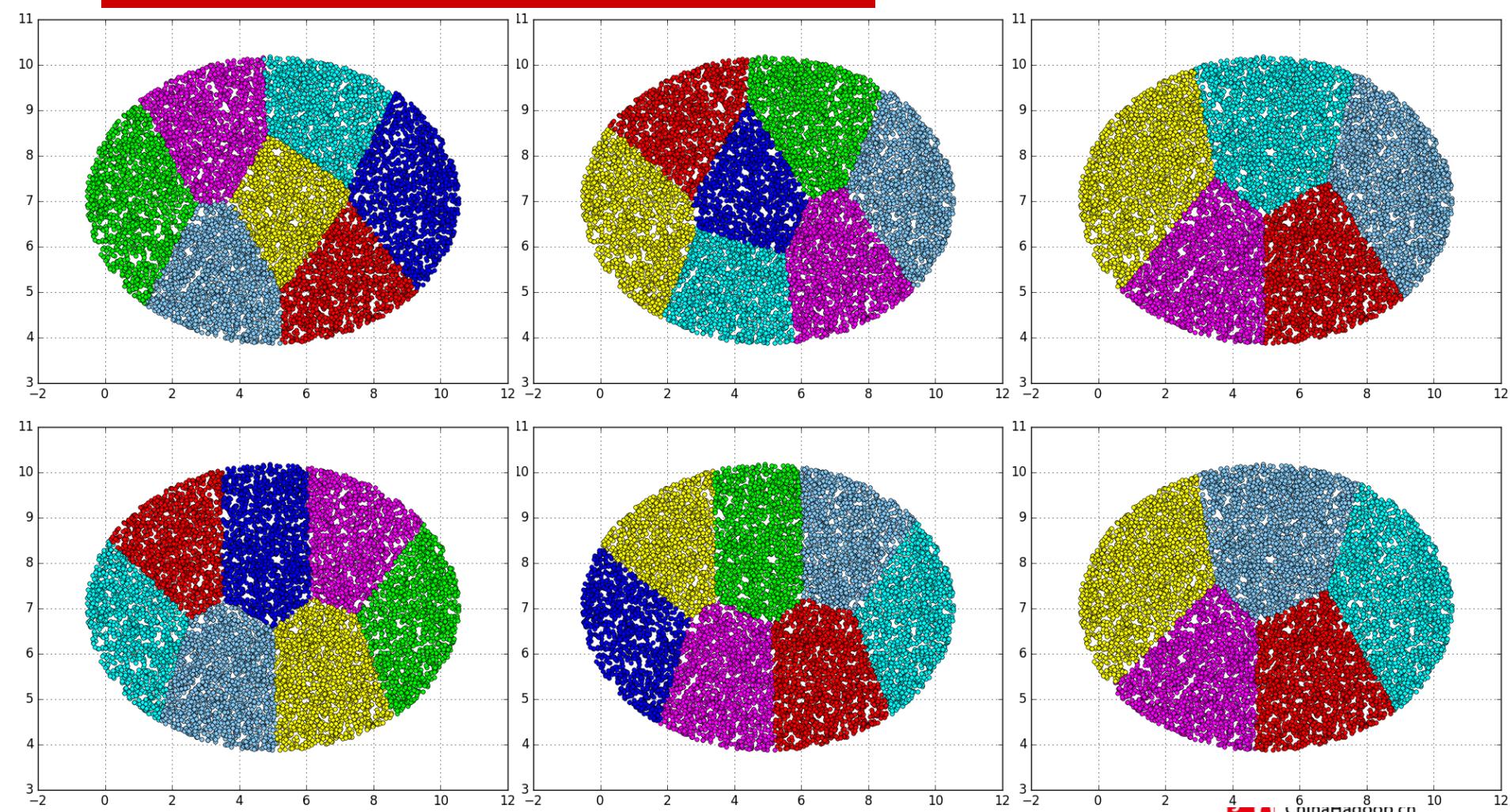
Mini-batch k-Means效果



k-Means适用范围



k-Means++算法测试



k-Means聚类方法总结

□ 优点：

- 是解决聚类问题的一种经典算法，简单、快速
- 对处理大数据集，该算法保持可伸缩性和高效率
- 当簇近似为高斯分布时，它的效果较好

□ 缺点

- 在簇的平均值可被定义的情况下才能使用，可能不适用于某些应用
- 必须事先给出k(要生成的簇的数目)，而且对初值敏感，对于不同的初始值，可能会导致不同结果。
- 不适合于发现非凸形状的簇或者大小差别很大的簇
- 对噪声和孤立点数据敏感

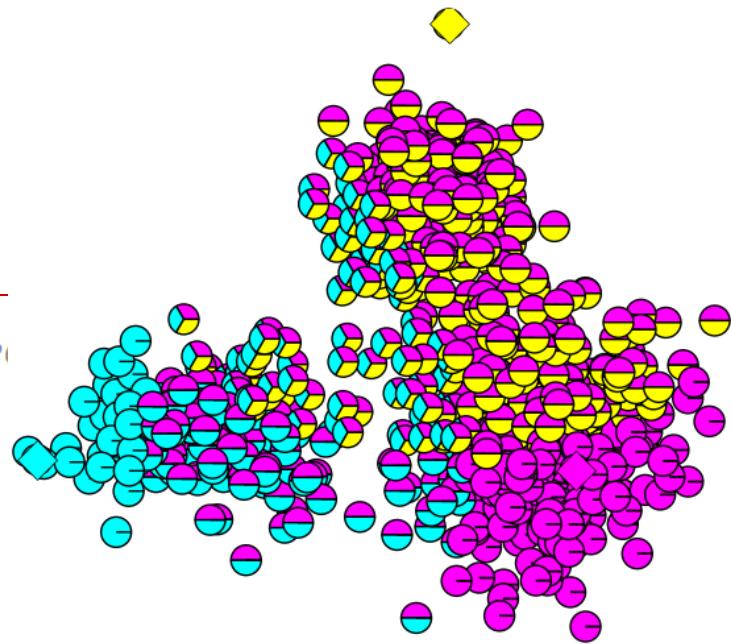
□ 可作为其他聚类方法的基础算法，如谱聚类

Canopy算法

- 虽然Canopy算法可以划归为聚类算法，但更多的可以使用Canopy算法做空间索引，其时空复杂度都很出色，算法描述如下：
- 对于给定样本 $x_1, x_2 \dots x_m$ ，给定先验值 r_1, r_2 , ($r_1 < r_2$)
 - $x_1, x_2 \dots x_m$ 形成列表L；构造 x_j ($1 \leq j \leq m$) 的空列表 C_j ；
 - 随机选择L中的样本c，要求c的列表 C_c 为空：
 - 计算L中样本 x_j 与c的距离 d_j
 - 若 $d_j < r_1$ ，则在L中删除 x_j ，将 C_j 赋值为 {c}
 - 否则，若 $d_j < r_2$ ，则将 C_j 增加 {c}
 - 若L中没有不满足条件的样本c，算法结束。

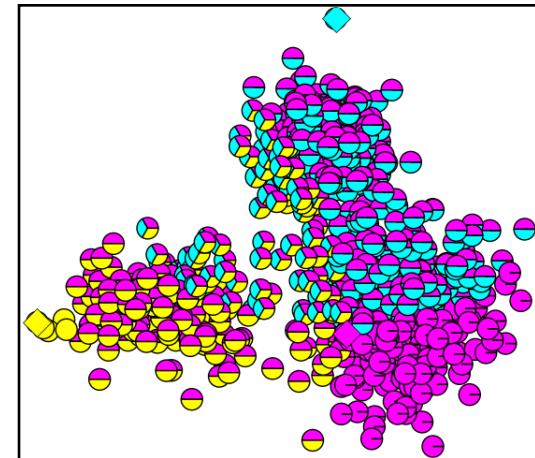
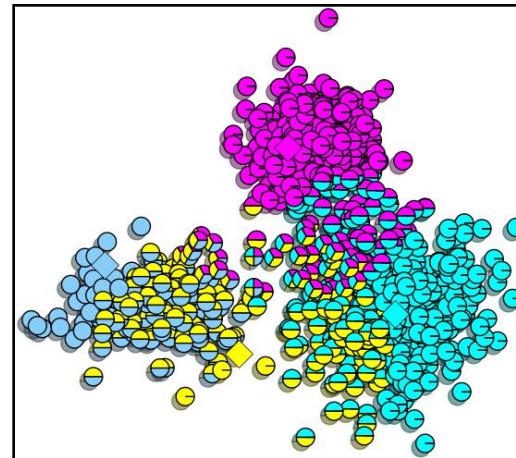
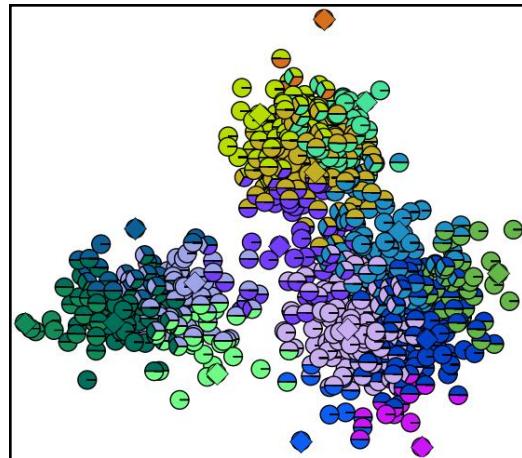
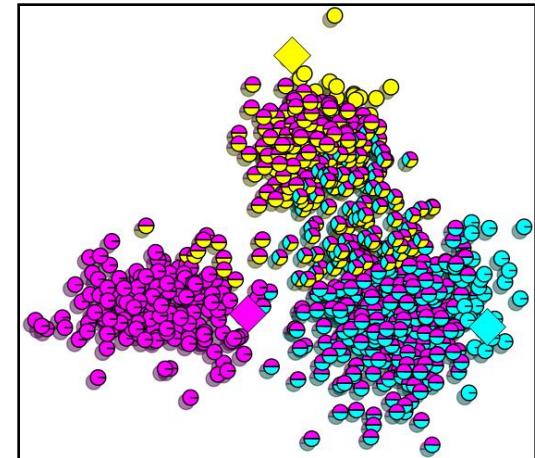
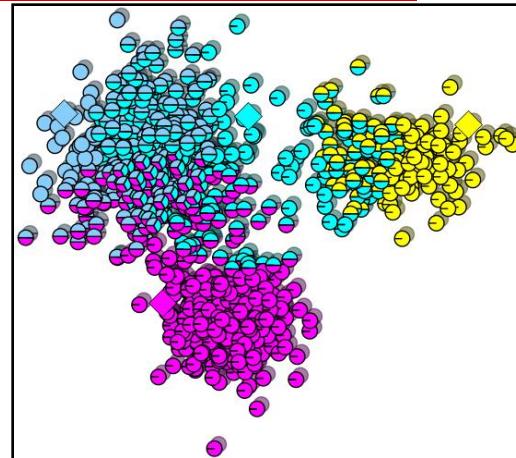
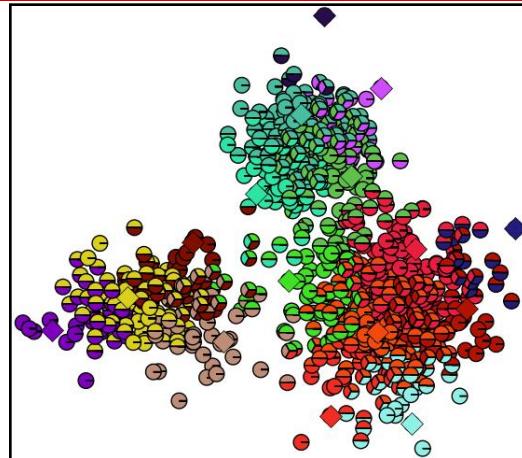
Code

```
def canopy(data, d_near, d_far):          # d_near, d_far 分别是最近邻和最远邻的半径  
    m = len(data)                      # 样本个数  
    cluster = [[] for i in range(m)]  
    center = []  
    has = m  
    while has > 0:  
        i = np.random.randint(has)  
        i = find_new_canopy(cluster, i)      # 查找cluster中第i个为空的值  
        center.append(i)                   # canopy中心  
        if i == -1:  
            break  
        for j in range(m):    # 针对新canopy中心data[i], 计算所有样本与之距离  
            d = distance(data[i], data[j])  
            if d < d_near:  
                cluster[j] = [i]  
            elif d < d_far:  
                cluster[j].append(i)  
    has = calc_candidate(cluster)  
return cluster, center
```



Canopy的调参

0.5	1	1.5
0.5	1	2



聚类的衡量指标

□ 均一性

■ Homogeneity

$$h = \begin{cases} 1 & \text{if } H(C) = 0 \\ 1 - \frac{H(C|K)}{H(C)} & \text{otherwise} \end{cases}$$

■ 一个簇只包含一个类别的样本，则满足均一性

□ 完整性

■ Completeness

$$c = \begin{cases} 1 & \text{if } H(K) = 0 \\ 1 - \frac{H(K|C)}{H(K)} & \text{otherwise} \end{cases}$$

■ 同类别样本被归类到相同簇中，则满足完整性

$$\square \text{ V-measure } v_\beta = \frac{(1 + \beta) \cdot h \cdot c}{\beta \cdot h + c}$$

■ 均一性和完整性的加权平均

ARI

C	Y_1	Y_2	\cdots	Y_s	sum
X_1	n_{11}	n_{12}	\cdots	n_{1s}	a_1
X_2	n_{21}	n_{22}	\cdots	n_{2s}	a_2
\vdots	\vdots	\vdots	\ddots	\vdots	\vdots
X_r	n_{r1}	n_{r2}	\cdots	n_{rs}	a_r
sum	b_1	b_2	\cdots	b_s	N

- 数据集S共有N个元素，两个聚类结果分别是：

$$X = \{X_1, X_2, \dots, X_r\} \quad Y = \{Y_1, Y_2, \dots, Y_s\}$$

- X和Y的元素个数为： $a = \{a_1, a_2, \dots, a_r\}$ $b = \{b_1, b_2, \dots, b_s\}$

- 记： $n_{ij} = |X_i \cap Y_j|$

- 则：

$$ARI = \frac{Index - EIndex}{MaxIndex - EIndex} = \frac{\sum_{i,j} C_{n_{ij}}^2 - \left[\left(\sum_i C_{a_i}^2 \right) \cdot \left(\sum_j C_{b_j}^2 \right) \right] / C_n^2}{\frac{1}{2} \left[\left(\sum_i C_{a_i}^2 \right) + \left(\sum_j C_{b_j}^2 \right) \right] - \left[\left(\sum_i C_{a_i}^2 \right) \cdot \left(\sum_j C_{b_j}^2 \right) \right] / C_n^2}$$

AMI

- 使用与ARI相同的记号，根据信息熵，得：
- 互信息/正则化互信息：

$$MI(X, Y) = \sum_{i=1}^r \sum_{j=1}^s P(i, j) \log \frac{P(i, j)}{P(i)P(j)}$$

$$NMI(X, Y) = \frac{MI(X, Y)}{\sqrt{H(X)H(Y)}}$$

- X服从超几何分布，求互信息期望：

$$E[MI] = \sum_x P(X=x) MI(X, Y) = \sum_{x=\max(1, a_i+b_i-N)}^{\min(a_i, b_i)} \left[MI \cdot \frac{a_i!b_j!(N-a_i)!(N-b_j)!}{N!x!(a_i-x)!(b_j-x)!(N-a_i-b_j+x)!} \right]$$

- 借鉴ARI，有： $AMI(X, Y) = \frac{MI(X, Y) - E[MI(X, Y)]}{\max\{H(X), H(Y)\} - E[MI(X, Y)]}$

轮廓系数(Silhouette)

- Silhouette 系数是对聚类结果有效性的解释和验证，由 Peter J. Rousseeuw 于 1986 提出。
- 计算样本 i 到同簇其他样本的平均距离 a_i 。 a_i 越小，说明样本 i 越应该被聚类到该簇。将 a_i 称为样本 i 的簇内不相似度。
 - 簇 C 中所有样本的 a_i 均值称为簇 C 的簇不相似度。
- 计算样本 i 到其他某簇 C_j 的所有样本的平均距离 b_{ij} ，称为样本 i 与簇 C_j 的不相似度。定义为样本 i 的簇间不相似度： $b_i = \min\{b_{i1}, b_{i2}, \dots, b_{iK}\}$
 - b_i 越大，说明样本 i 越不属于其他簇。

轮廓系数(Silhouette)

- 根据样本*i*的簇内不相似度 a_i 和簇间不相似度 b_i ，定义样本*i*的轮廓系数：

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

$$s(i) = \begin{cases} 1 - \frac{a(i)}{b(i)}, & a(i) < b(i) \\ 0, & a(i) = b(i) \\ \frac{b(i)}{a(i)} - 1, & a(i) > b(i) \end{cases}$$

- s_i 接近1，则说明样本*i*聚类合理； s_i 接近-1，则说明样本*i*更应该分类到另外的簇；若 s_i 近似为0，则说明样本*i*在两个簇的边界上。
- 所有样本的 s_i 的均值称为聚类结果的轮廓系数，是该聚类是否合理、有效的度量。

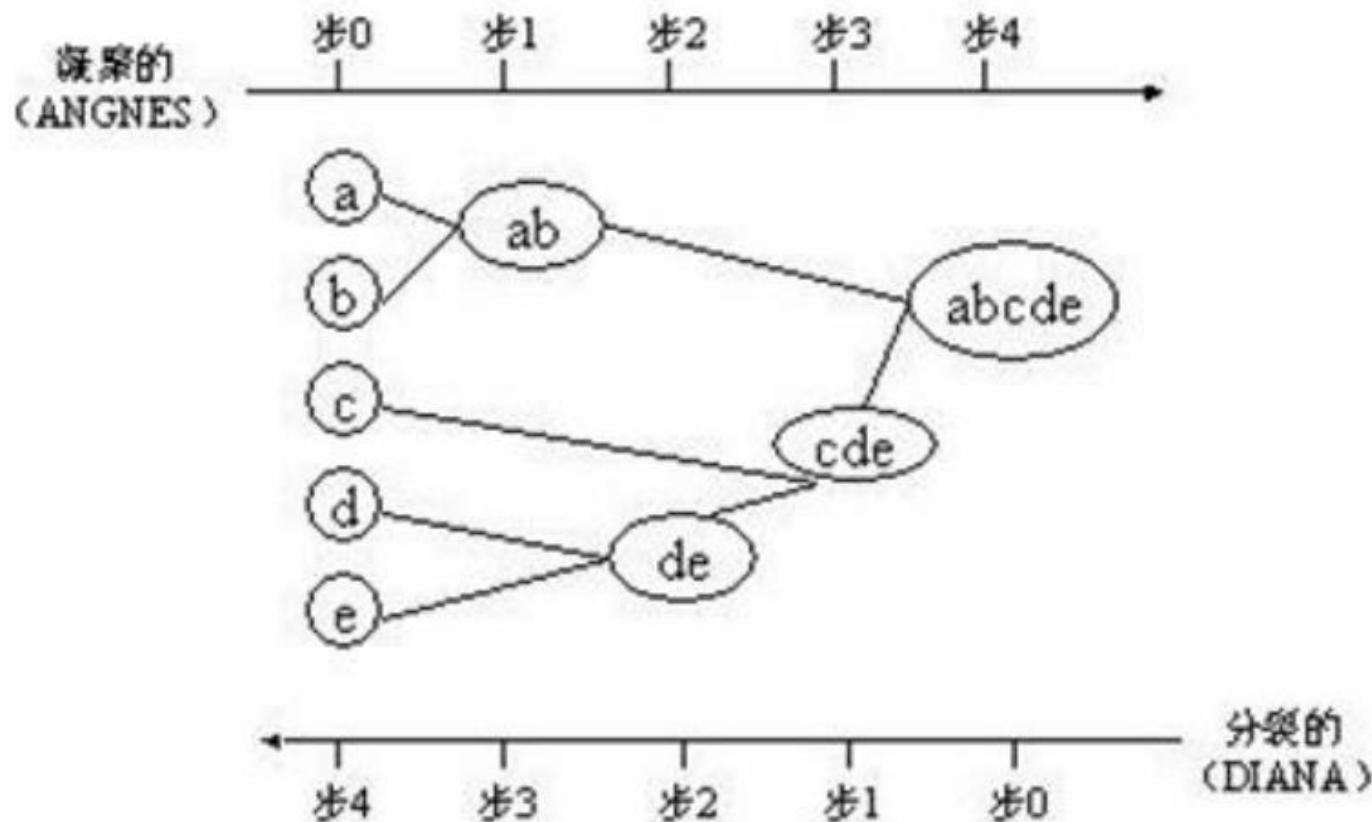
层次聚类方法

- 层次聚类方法对给定的数据集进行层次的分解，直到某种条件满足为止。具体又可分为：
 - 凝聚的层次聚类：AGNES算法
 - 一种自底向上的策略，首先将每个对象作为一个簇，然后合并这些原子簇为越来越大的簇，直到某个终结条件被满足。
 - 分裂的层次聚类：DIANA算法
 - 采用自顶向下的策略，它首先将所有对象置于一个簇中，然后逐渐细分为越来越小的簇，直到达到了某个终结条件。

AGNES和DIANA算法

- AGNES (AGglomerative NESting) 算法最初将每个对象作为一个簇，然后这些簇根据某些准则被一步步地合并。两个簇间的距离由这两个不同簇中距离最近的数据点对的相似度来确定；聚类的合并过程反复进行直到所有的对象最终满足簇数目。
- DIANA (DIvisive ANAlysis) 算法是上述过程的反过程，属于分裂的层次聚类，首先将所有的对象初始化到一个簇中，然后根据一些原则(比如最大的欧式距离)，将该簇分类。直到到达用户指定的簇数目或者两个簇之间的距离超过了某个阈值。

层次聚类



密度聚类方法

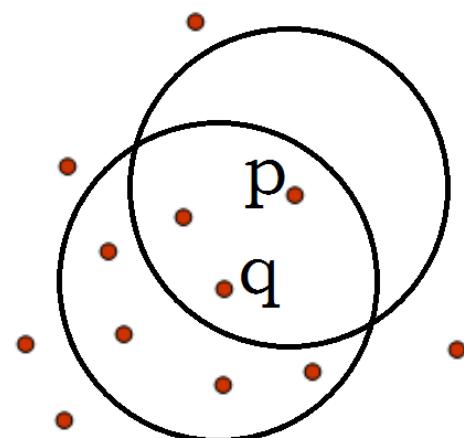
- 密度聚类方法的指导思想是，只要样本点的密度大于某阈值，则将该样本添加到最近的簇中。
- 这类算法能克服基于距离的算法只能发现“类圆形”(凸)的聚类的缺点，可发现任意形状的聚类，且对噪声数据不敏感。但计算密度单元的计算复杂度大，需要建立空间索引来降低计算量。
 - DBSCAN
 - 密度最大值算法

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。

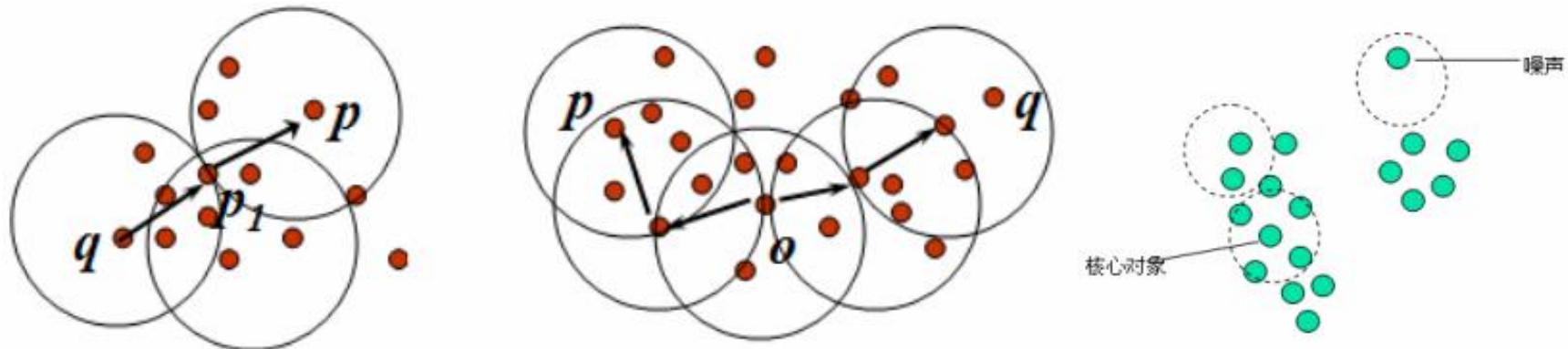
DBSCAN算法的若干概念

- 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 $\varepsilon=1\text{cm}$, $m=5$, 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。

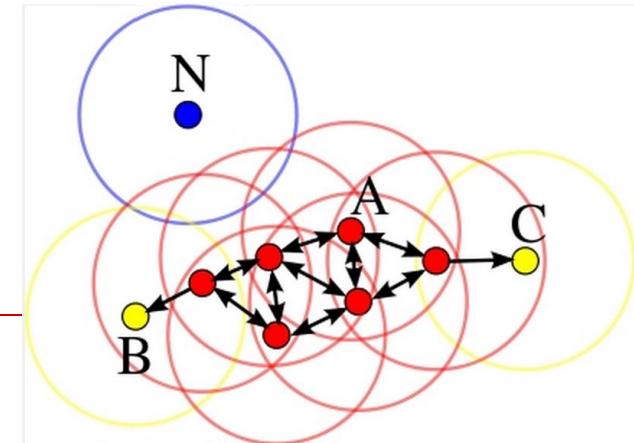


DBSCAN算法的若干概念

- 密度可达：如果存在一个对象链 $p_1 p_2 \dots p_n$, $p_1 = q$, $p_n = p$, 对 $p_i \in D$, ($1 \leq i \leq n$), 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

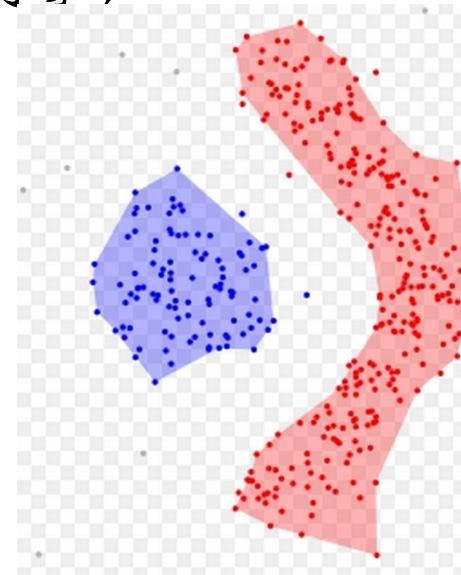


□ DBSCAN算法流程：

- 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

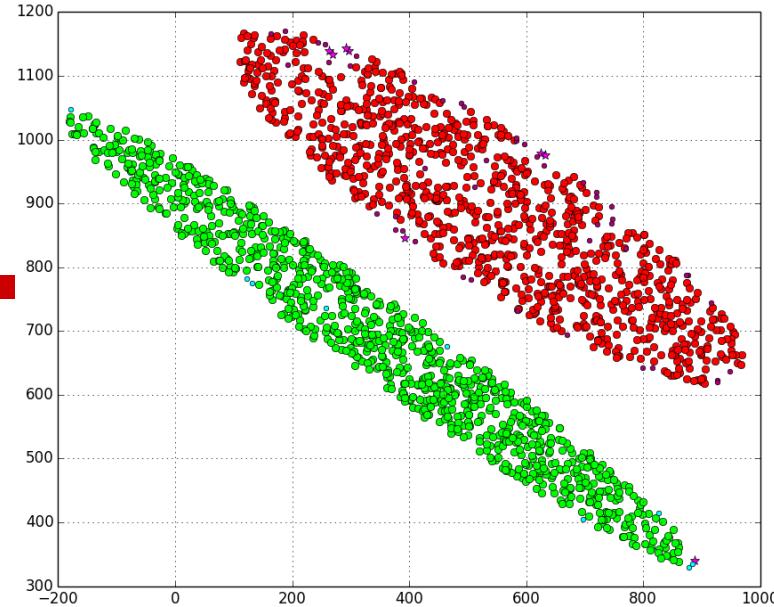
```
def density_cluster(data):
    # 计算核心对象
    m = len(data)          # 样本数目
    sim = [[] for i in range(m)] * m   # sim[i]: 第i个样本的近邻
    r2 = r * r
    for i in range(m):  # 距离
        print i
        for j in range(m):
            if distance2(data[i], data[j]) < r2:
                sim[i].append(j)

    # 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

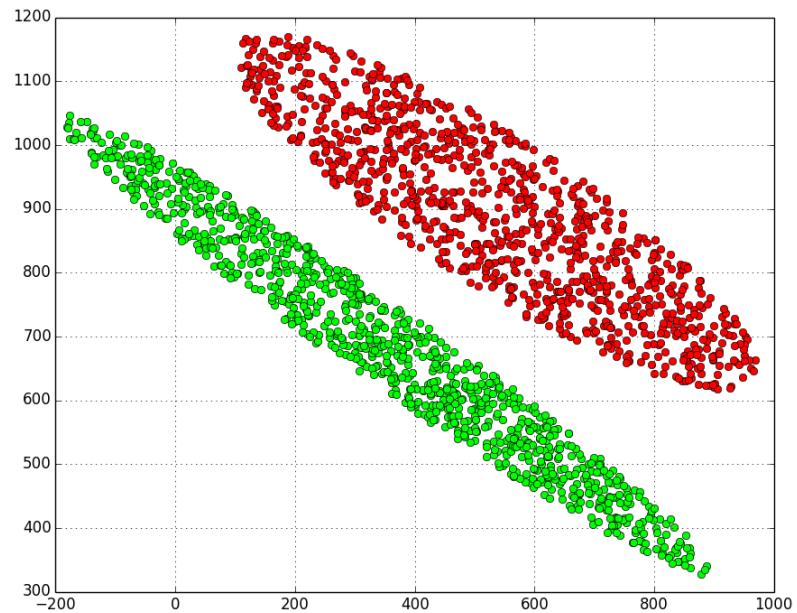
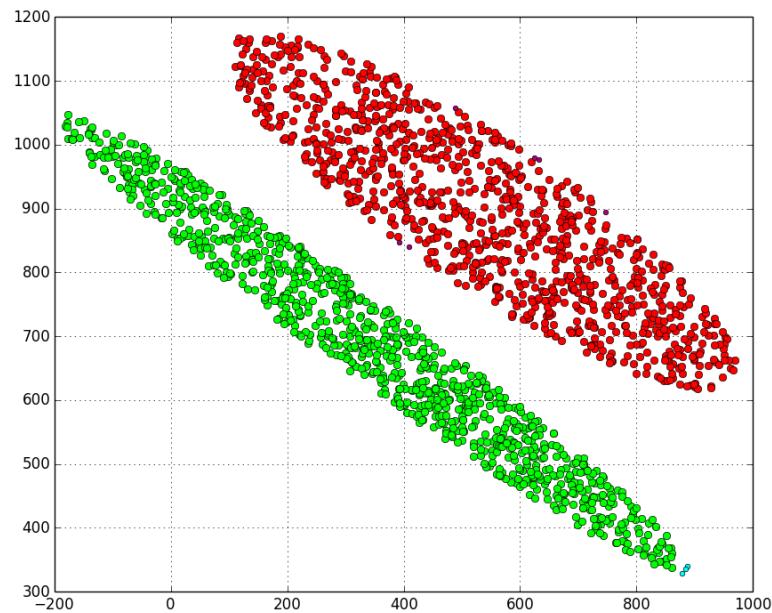
    # 根据字典映射关系，计算每个样本所属的簇
    cluster = [0] * m
    for i in range(m):
        c = uf_find(uf, i)
        if (c != i) or (len(sim[i]) > kernel_num):
            if len(sim[i]) > kernel_num: # 是核心对象
                cluster[i] = types[c]
            else:
                cluster[i] = k+types[c]
    return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

参数： $n=5$

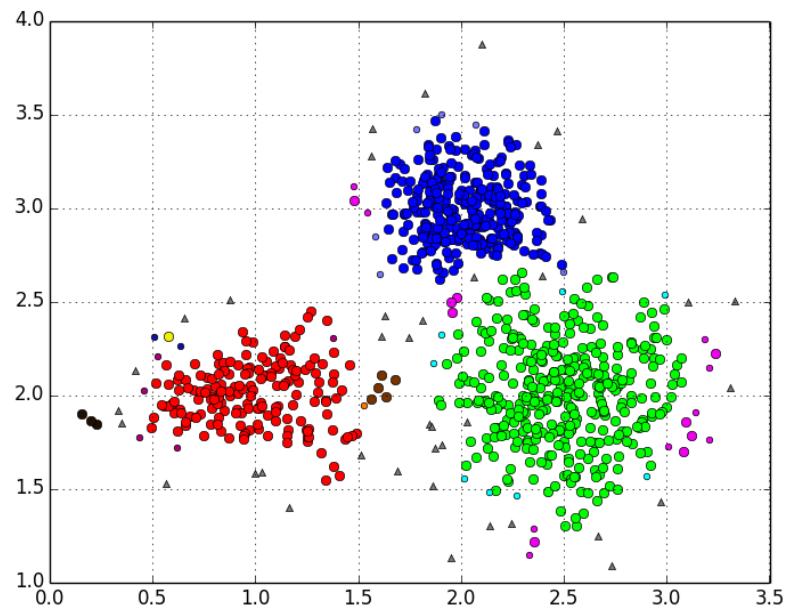
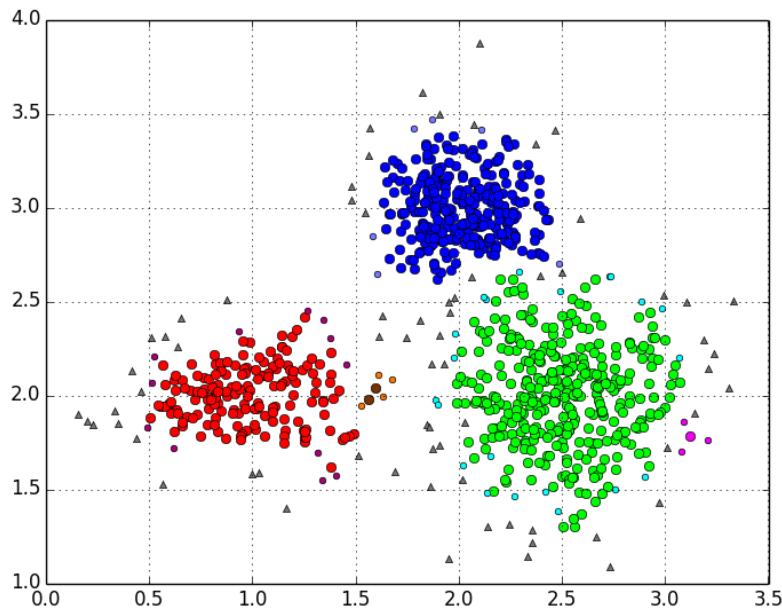
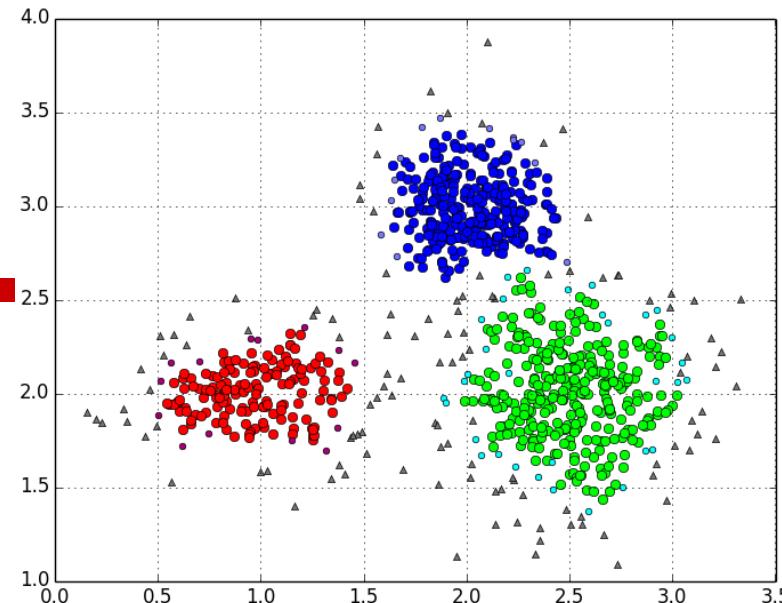


25

30 | 40



参数： $r=0.1$



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 ρ_i

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

- 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 δ_i
 - 简称“高密距离”（注：该称呼不具代表性）。

高局部密度点距离

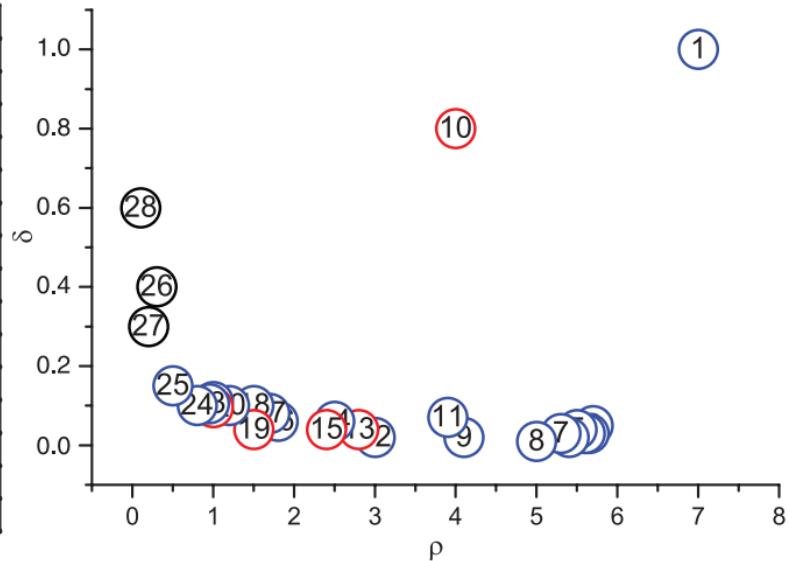
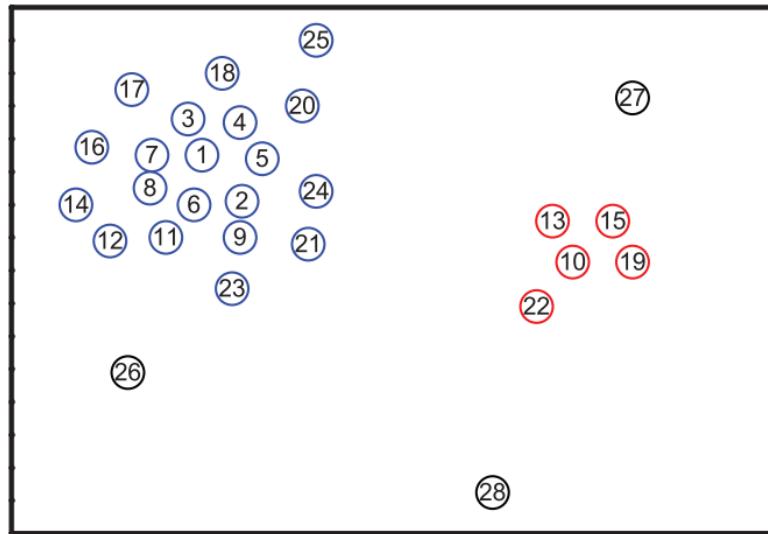
- 高局部密度点距离 $\delta_i = \min_{j: \rho_j > \rho_i} (d_{ij})$
- 在密度高于对象*i*的所有对象中，到对象*i*最近的距离，即高局部密度点距离。
- 对于密度最大的对象，设置 $\delta_i = \max(d_{ij})$ (即：该问题中的无穷大)。
 - 只有那些密度是局部或者全局最大的点才会有远大于正常值的高局部密度点距离。

簇中心的识别

- 那些有着比较大的局部密度 p_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 p_i 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。

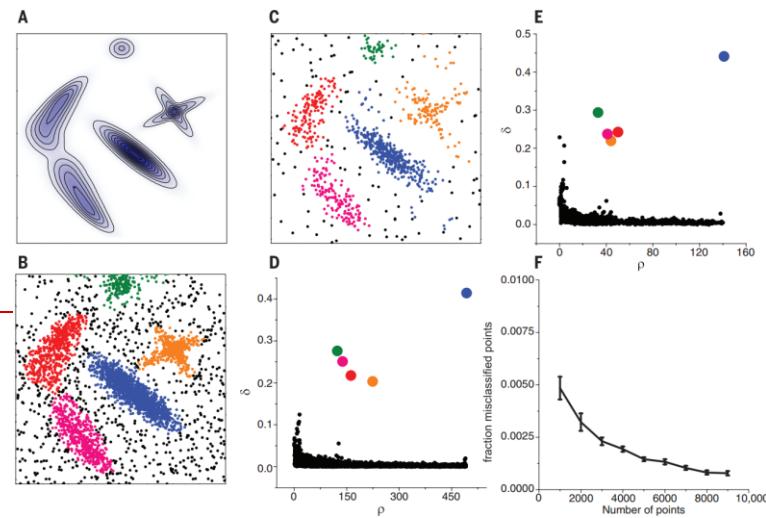
密度最大值聚类过程

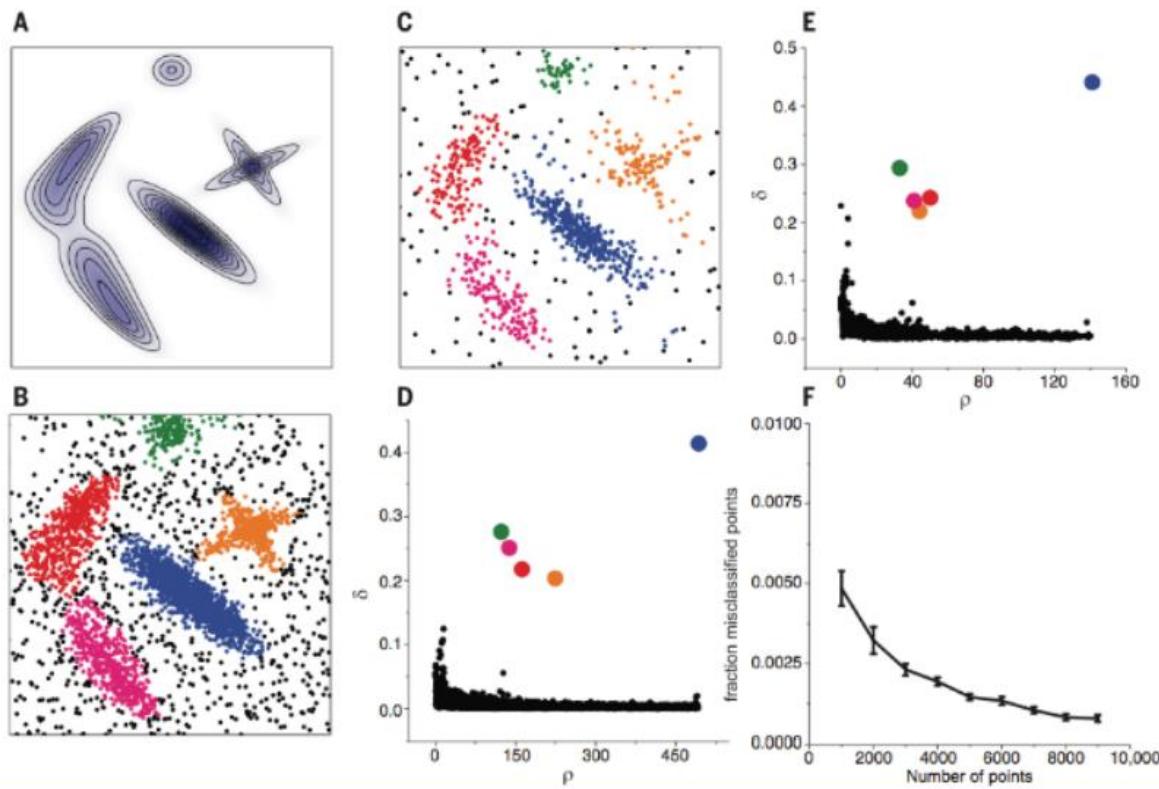
- 左图是所有点在二维空间的分布，右图是以 ρ 为横坐标，以 δ 为纵坐标绘制的决策图。可以看到，1和10两个点的 ρ_i 和 δ_i 都比较大，作为簇的中心点。26、27、28三个点的 δ_i 也比较大，但是 ρ_i 较小，所以是异常点。



边界和噪声的重认识

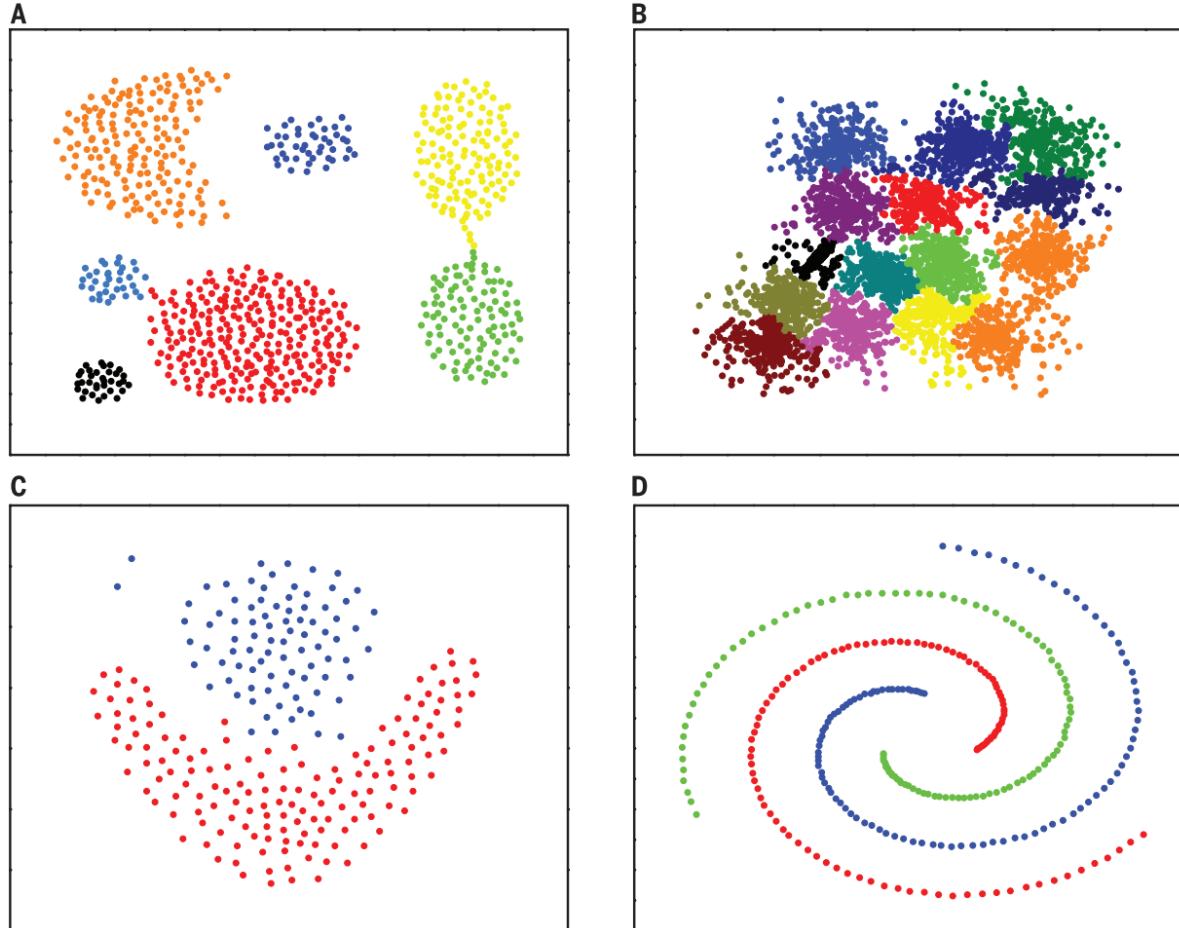
- 在聚类分析中，通常需要确定每个点划分给某个簇的可靠性：
- 在该算法中，可以首先为每个簇定义一个边界区域 (border region)，亦即划分给该簇但是距离其他簇的点的距离小于 d_c 的点的集合。然后为每个簇找到其边界区域的局部密度最大的点，令其局部密度为 ρ_h 。
- 该簇中所有局部密度大于 ρ_h 的点被认为是簇核心的一部分(亦即将该点划分给该类簇的可靠性很大)，其余的点被认为是该类簇的光晕(halo)，亦即可以认为是噪声。
 - 注：关于可靠性问题，在EM算法中仍然会有相关涉及。





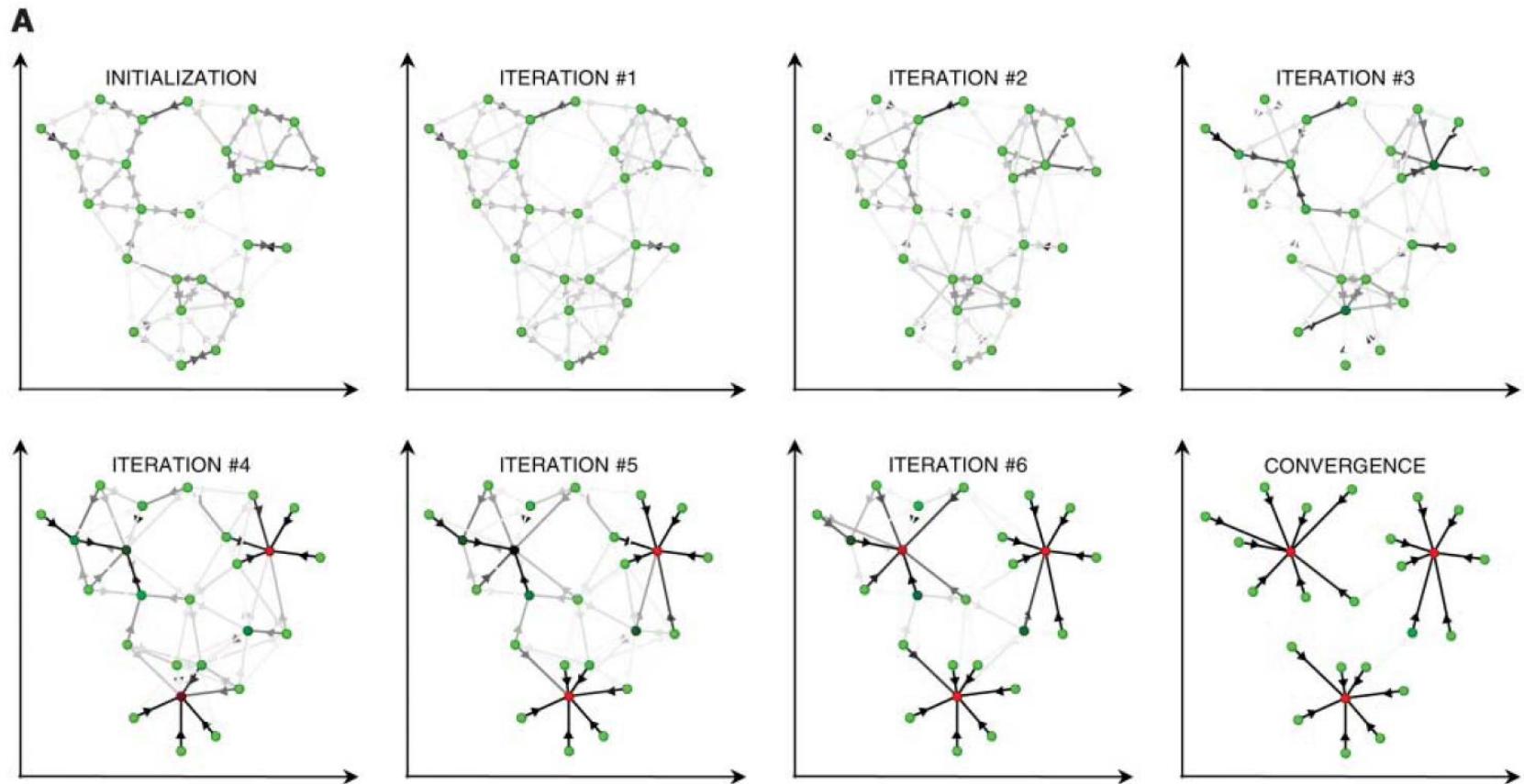
A图为生成数据的概率分布, B, C二图为分别从该分布中生成了4000, 1000个点. D, E分别是B, C两组数据的决策图(decision tree), 可以看到两组数据都只有五个点有比较大的 ρ_i 和很大的 δ_i . 这些点作为类簇的中心, 在确定了类簇的中心之后, 每个点被划分到各个类簇(彩色点), 或者是划分到类簇光晕(黑色点). F图展示的是随着抽样点数量的增多, 聚类的错误率在逐渐下降, 说明该算法是鲁棒的.

不同数据下密度最大值聚类的效果



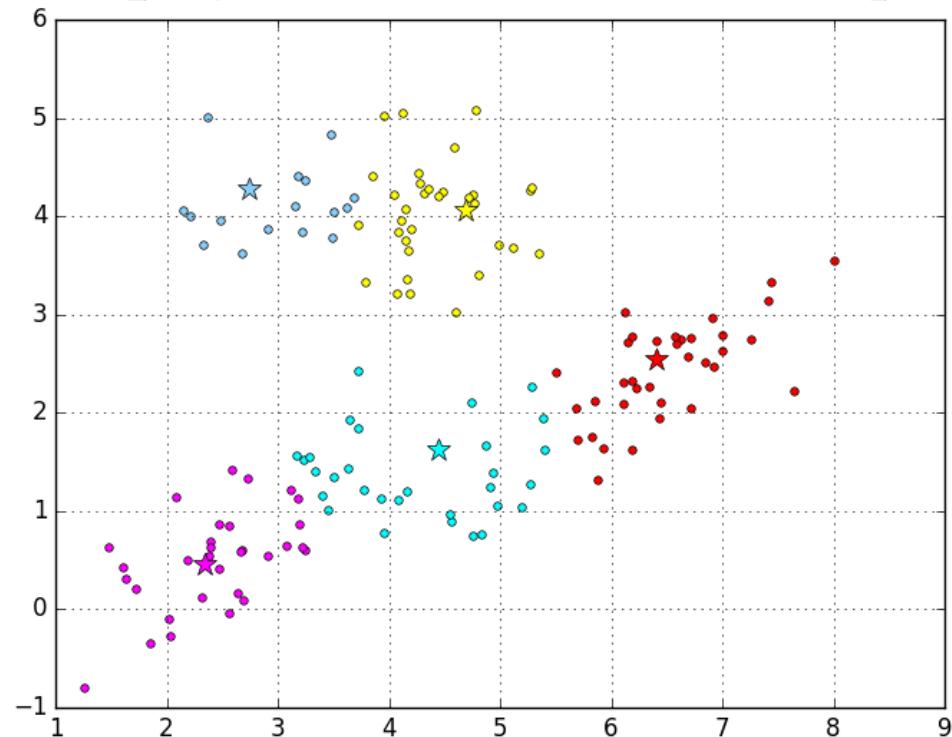
$$r(i,k) \leftarrow s(i,k) - \max_{k' \text{ s.t. } k' \neq k} \{a(i,k') + s(i,k')\}$$

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

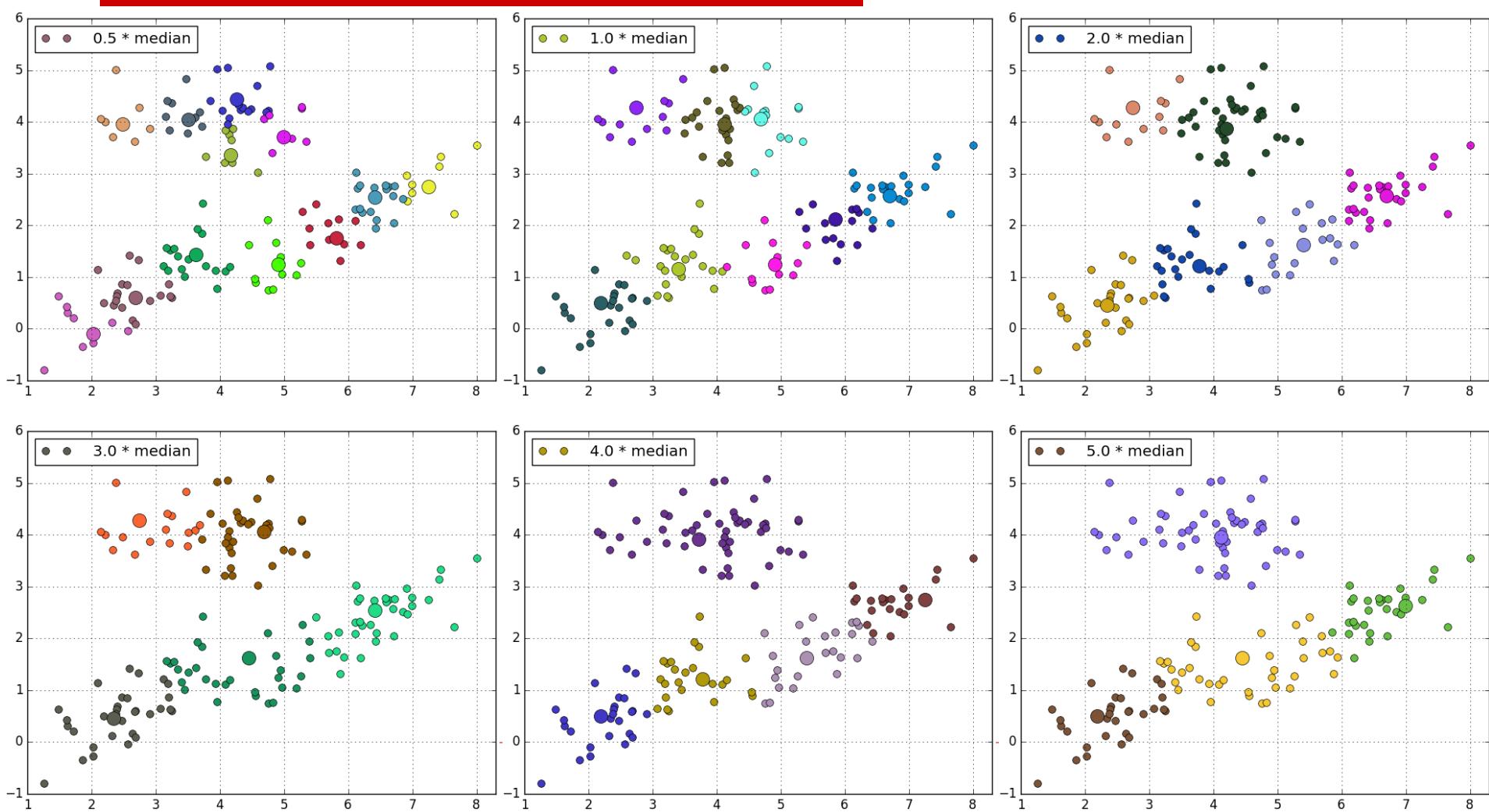
cluster



```
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 λ_1, λ_2 对应的特征向量分别是 μ_1, μ_2 ；
 - $\lambda_1, \lambda_2, \mu_1, \mu_2$ 都是实数或是实向量。

$$\begin{aligned} & \left\{ \begin{array}{l} A\mu_1 = \lambda_1\mu_1 \\ A\mu_2 = \lambda_2\mu_2 \end{array} \right. \Rightarrow \mu_1^T A\mu_2 = \mu_1^T \underline{\lambda_2\mu_2} \\ & \Rightarrow (A^T \mu_1)^T \mu_2 = \lambda_2 \mu_1^T \mu_2 \Rightarrow (\underline{A\mu_1})^T \mu_2 = \lambda_2 \mu_1^T \mu_2 \\ & \Rightarrow (\underline{\lambda_1\mu_1})^T \mu_2 = \lambda_2 \mu_1^T \mu_2 \\ & \Rightarrow \lambda_1 \mu_1^T \mu_2 = \lambda_2 \mu_1^T \mu_2 \\ & \xrightarrow{\lambda_1 \neq \lambda_2} \mu_1^T \mu_2 = 0 \end{aligned}$$

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： $(A^T A)$ 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, \dots, X_n ，记任意两个点之间的相似度(“距离”的减函数)为 $s_{ij} = \langle x_i, x_j \rangle$ ，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 $W = (w_{ij})_{i,j=1,\dots,n}$
- 顶点的度 $d_i \rightarrow$ 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

$$\mathbb{1}_A = (f_1, \dots, f_n)' \in \mathbb{R}^n$$

$$f_i = 1 \text{ if } v_i \in A$$

$$f_i = 0 \text{ otherwise}$$

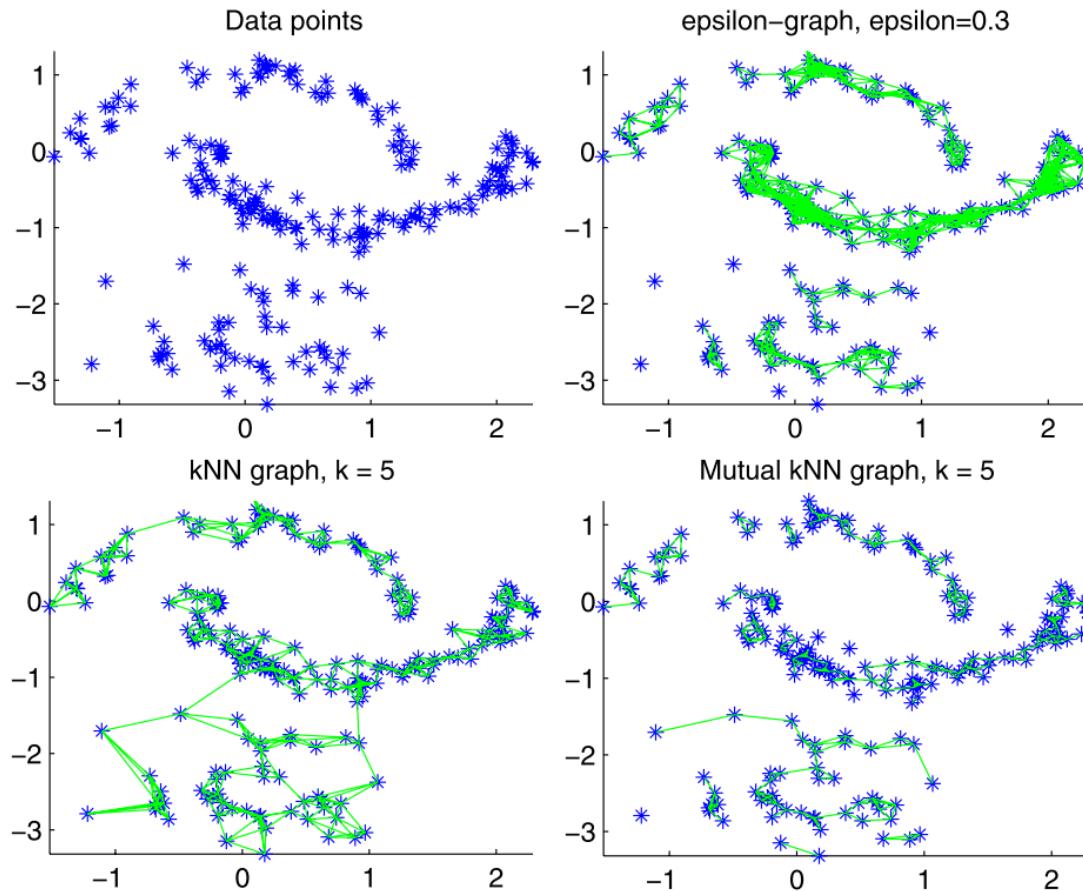
□ A和B是图G的不相交子图，则定义子图的连接权：

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

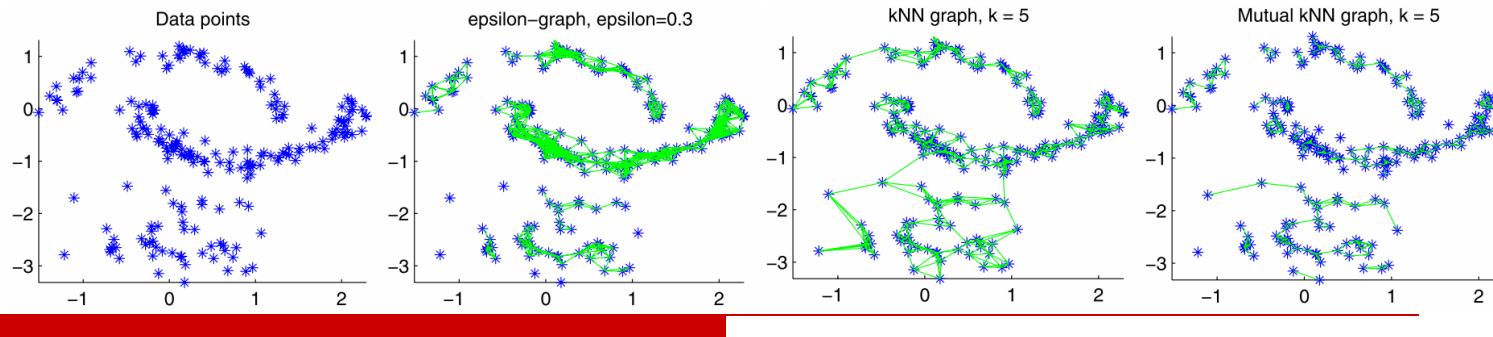
相似度图G的建立方法

- 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 v_i 的k最近邻包含 v_j , v_j 的k最近邻不一定包含 v_i : 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



权值比较



- ϵ 近邻图： $\epsilon=0.3$ ，“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： $L = D - W$

$$\begin{aligned} f^T L f &= f^T D f - f^T W f = \sum_{i=1}^n d_i f_i^2 - \sum_{i,j=1}^n f_i f_j w_{ij} \\ &= \frac{1}{2} \left(\sum_{i=1}^n d_i f_i^2 - 2 \sum_{i,j=1}^n f_i f_j w_{ij} + \sum_{j=1}^n d_j f_j^2 \right) \\ &= \frac{1}{2} \sum_{i,j=1}^n w_{ij} (f_i - f_j)^2 \end{aligned}$$

□ 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 $n \times n$ 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, \dots, u_k ;
 - 将k个列向量 u_1, u_2, \dots, u_k 组成矩阵U, $U \in R^{n \times k}$;
 - 对于 $i=1, 2, \dots, n$, 令 $y_i \in R^k$ 是U的第i行的向量;
 - 使用k-means算法将点 $(y_i)_{i=1, 2, \dots, n}$ 聚类成簇 C_1, C_2, \dots, C_k ;
 - 输出簇 A_1, A_2, \dots, A_k , 其中, $A_i = \{j | y_j \in C_i\}$

谱聚类算法：随机游走拉普拉斯矩阵

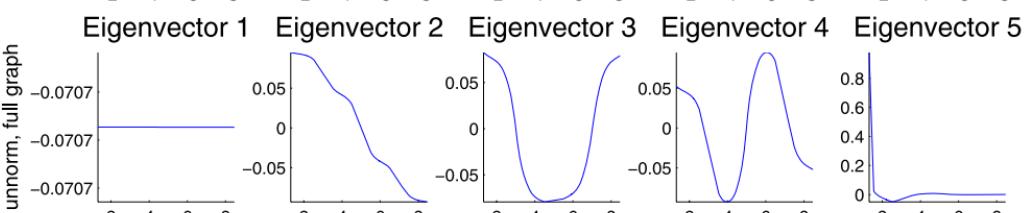
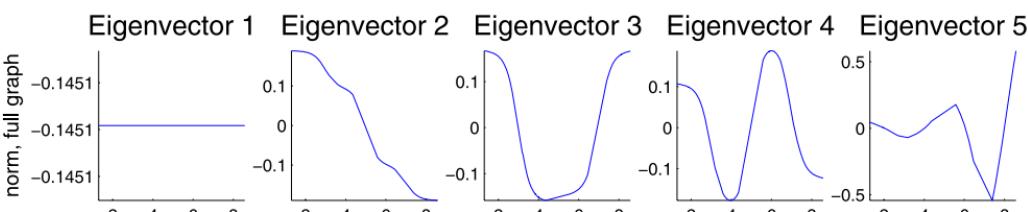
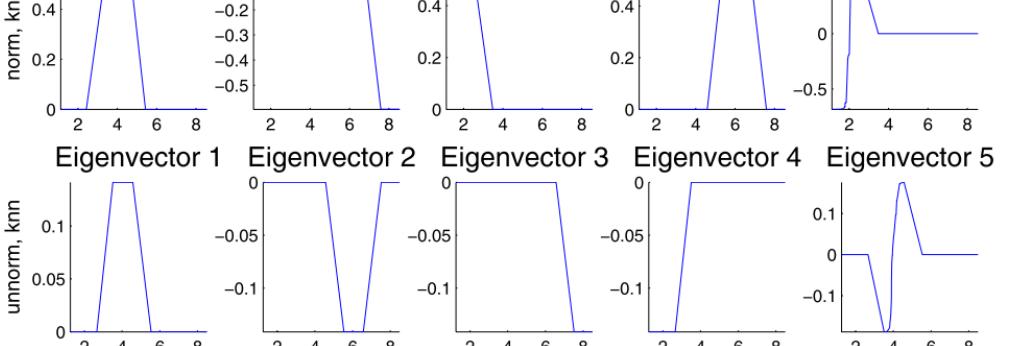
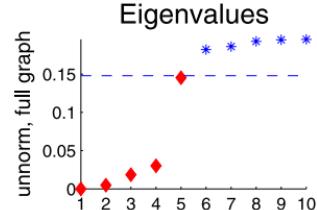
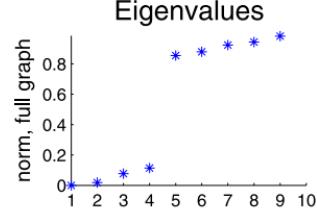
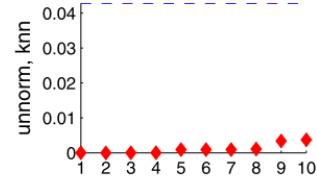
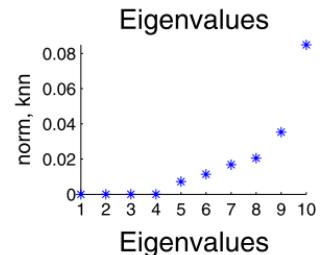
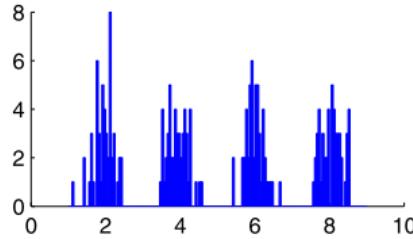
- 输入：n个点 $\{p_i\}$, 簇的数目k
 - 计算 $n \times n$ 的相似度矩阵W和度矩阵D;
 - 计算正则拉普拉斯矩阵 $L_{rw} = D^{-1}(D - W)$;
 - 计算 L_{rw} 的前k个特征向量 u_1, u_2, \dots, u_k ;
 - 将k个列向量 u_1, u_2, \dots, u_k 组成矩阵U, $U \in R^{n \times k}$;
 - 对于 $i=1, 2, \dots, n$, 令 $y_i \in R^k$ 是U的第i行的向量;
 - 使用k-means算法将点 $(y_i)_{i=1, 2, \dots, n}$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

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 $n \times n$ 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, \dots, u_k ;
 - 将k个列向量 u_1, u_2, \dots, u_k 组成矩阵U, $U \in R^{n \times k}$;
 - 对于 $i=1, 2, \dots, n$, 令 $y_i \in R^k$ 是U的第i行的向量;
 - 对于 $i=1, 2, \dots, n$, 将 $y_i \in R^k$ 依次单位化, 使得 $|y_i|=1$;
 - 使用k-means算法将点 $(y_i)_{i=1, 2, \dots, n}$ 聚类成簇 C_1, C_2, \dots, C_k ;
 - 输出簇 A_1, A_2, \dots, A_k , 其中, $A_i = \{j | y_j \in C_i\}$

一个实例

Histogram of the sample



Code

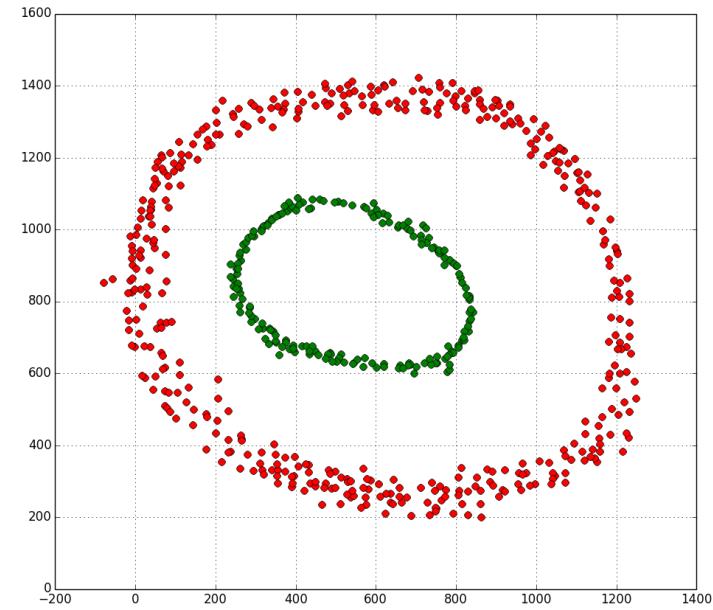
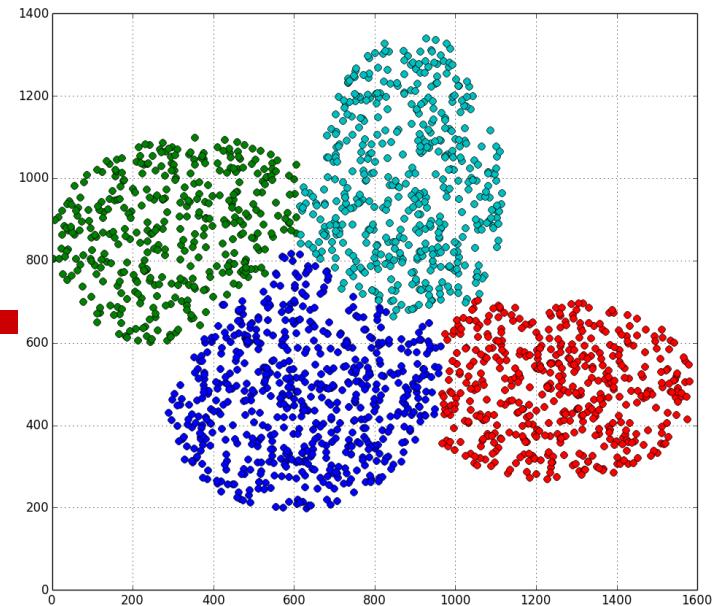
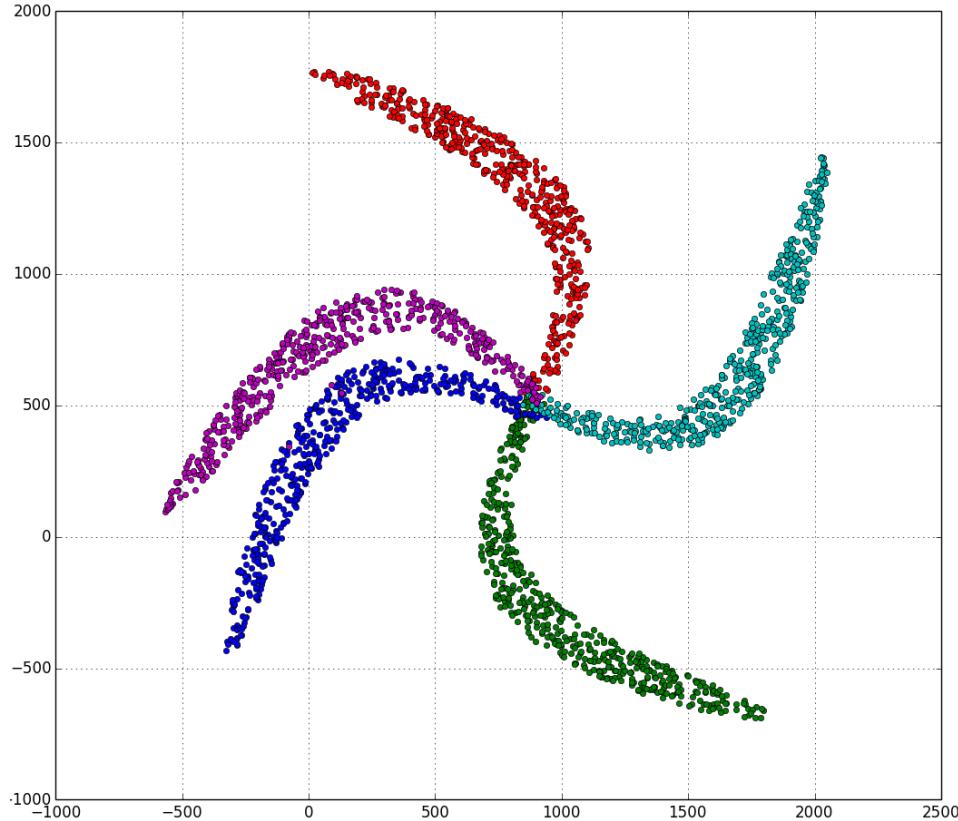
```
def spectral_cluster(data):
    lm = laplace_matrix(data)
    eg_values, eg_vectors = linalg.eig(lm)
    idx = eg_values.argsort()
    eg_vectors = eg_vectors[:, idx]

    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)
```

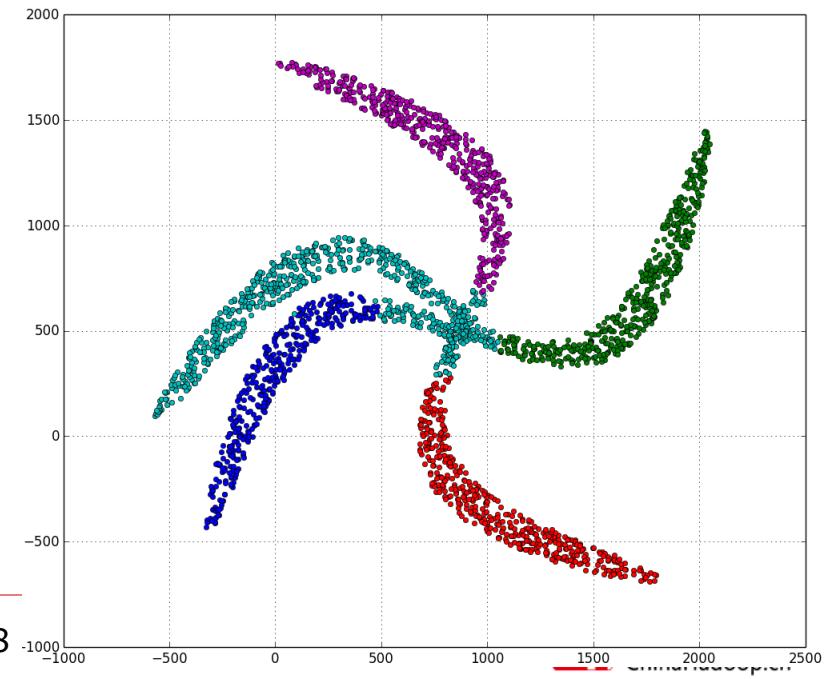
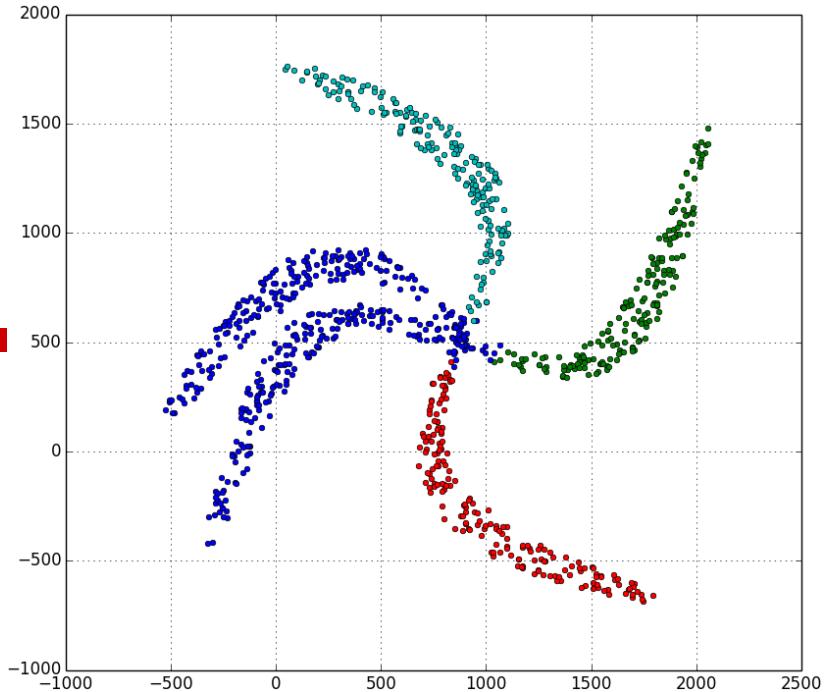
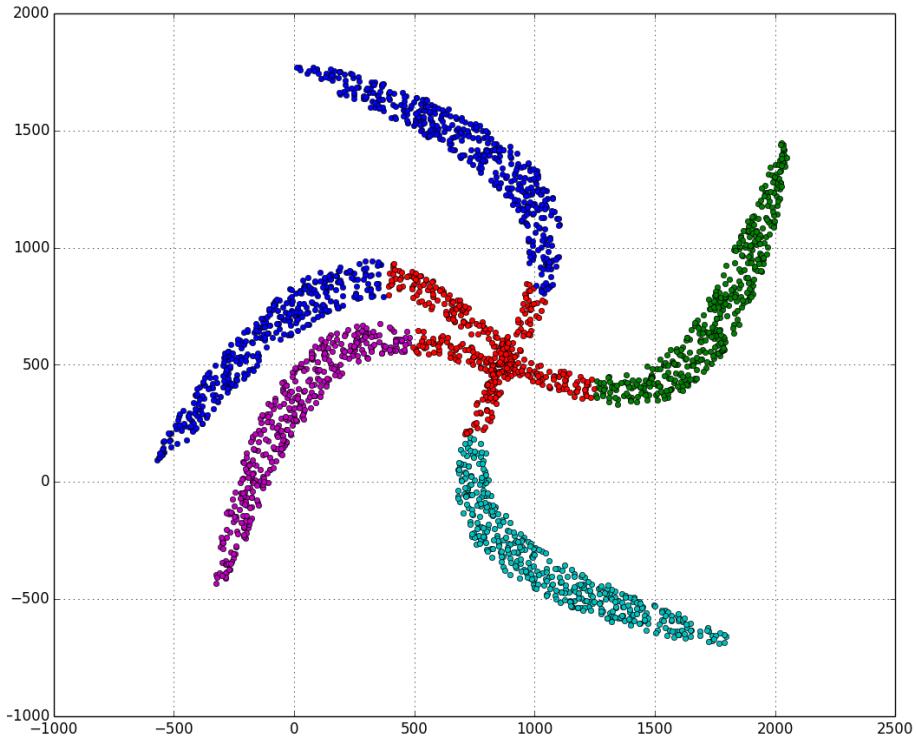
```
def laplace_matrix(data):
    m = len(data)
    w = [[0 for x in range(m)] for y in range(m)]
    for i in range(m):
        w[i] = [0 for x in range(m)]
    nearest = [0 for x in range(neighbor)]

    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



聚类失败的情况



进一步思考

- 谱聚类中的K如何确定? $k^* = \arg \max | \lambda_{k+1} - \lambda_k |$
- 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 v_i 跳转到顶点 v_j 的概率正比于边的权值 w_{ij}

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

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

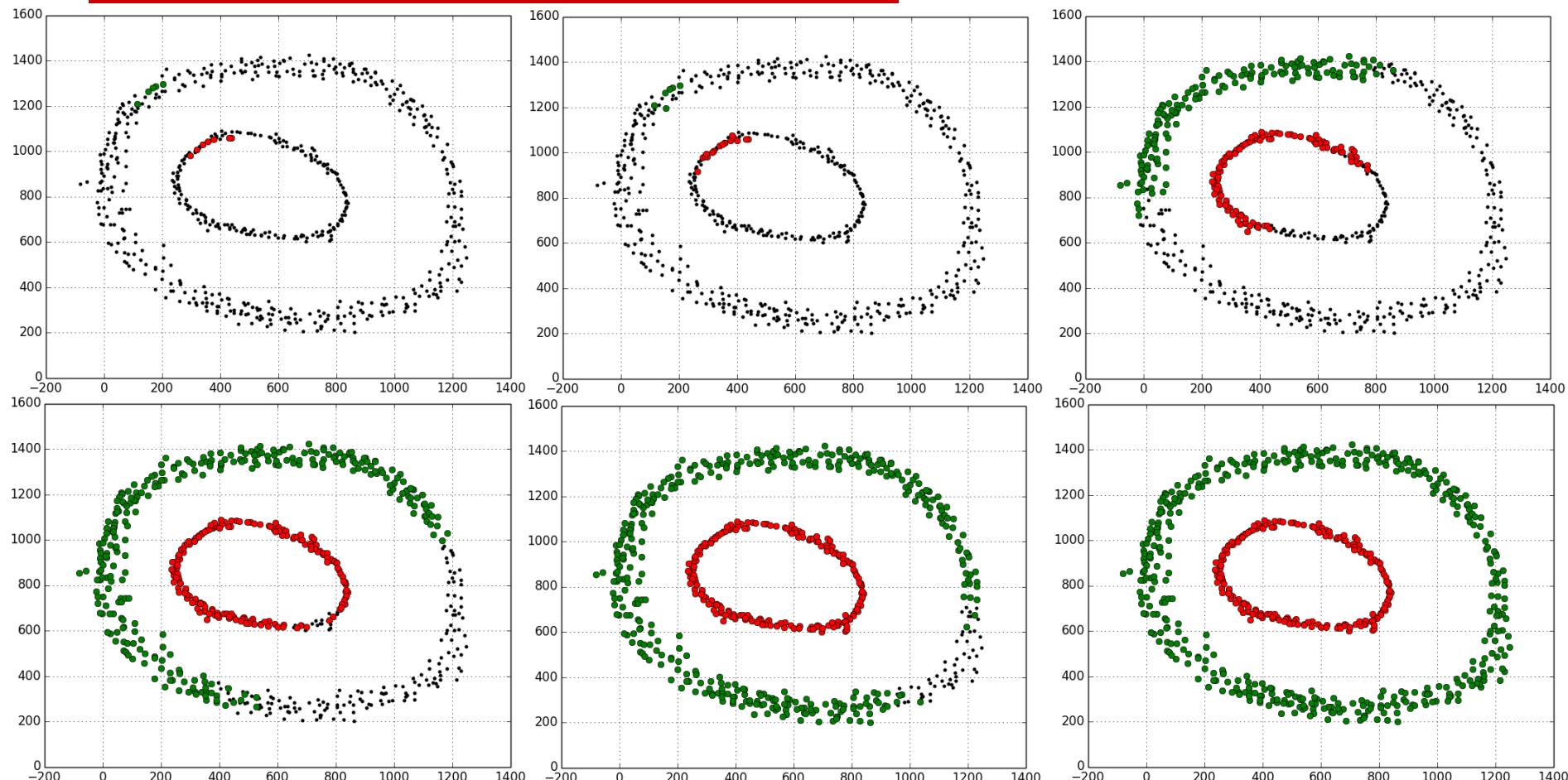
def calc_label(p, i):
    n = len(p[i])
    k = random.random()      #  $k \in [0, 1]$ 
    r = n-1
    for j in range(n):
        if p[i][j] > k:
            r = j
            break
    return r
```

```
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)]

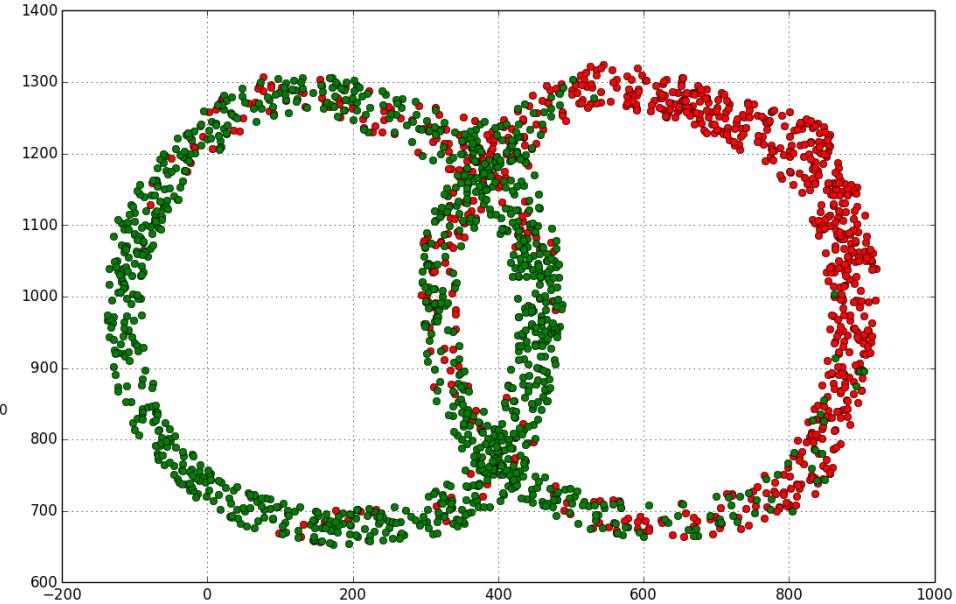
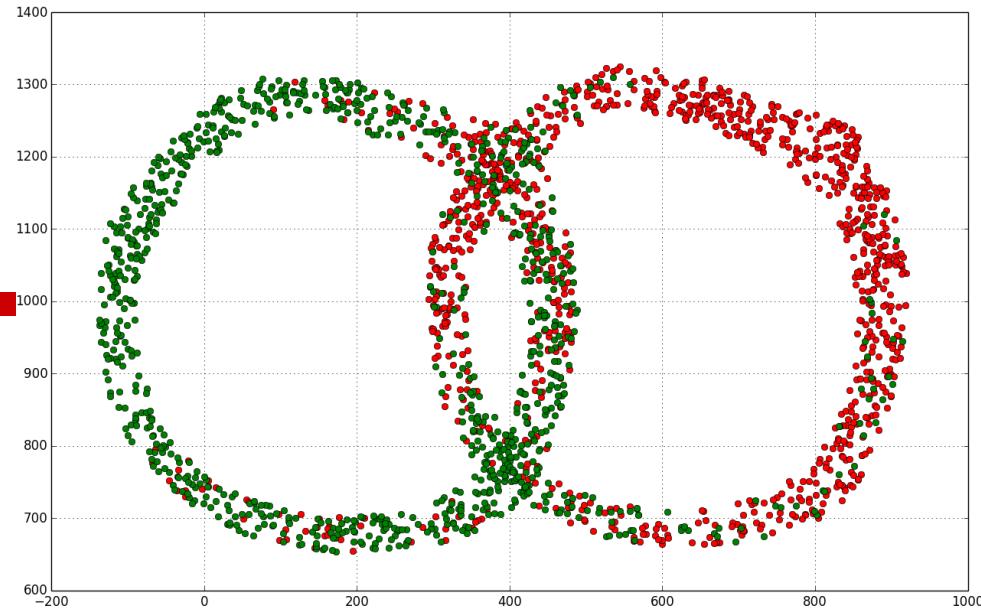
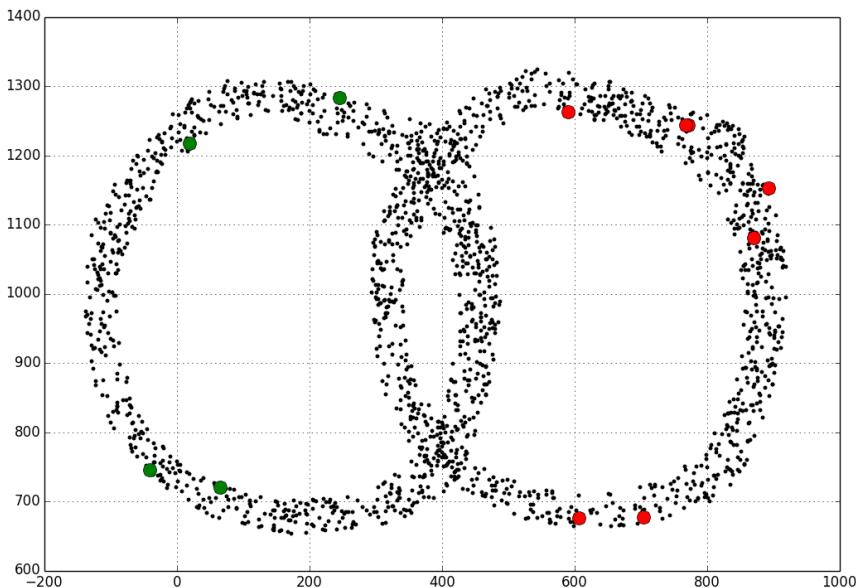
    for i in range(m):
        zero_list(nearest)
        for j in range(i+1, m):
            p[i][j] = similar(data, i, j)
            if not is_neighbor(p[i][j], nearest):
                p[i][j] = 0
            p[j][i] = p[i][j]      #对称
        p[i][i] = 0
    for i in range(m):
        s = 0
        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
```

初始	1	10
20	30	40

标签传递过程



带宽/邻域影响



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
- Andrew Rosenberg, Julia Hirschberg, *V-Measure: A conditional entropy-based external cluster evaluation measure*, 2007.
- W. M. Rand. *Objective criteria for the evaluation of clustering methods*. Journal of the American Statistical Association. 1971
- Nguyen Xuan Vinh, Julien Epps, James Bailey, *Information theoretic measures for clusterings comparison*, ICML 2009
- Peter J. Rousseeuw, *Silhouettes: a Graphical Aid to the Interpretation and Validation of Cluster Analysis*. Computational and Applied Mathematics 20: 53–65, 1987
- https://en.wikipedia.org/wiki/Rand_index
- https://en.wikipedia.org/wiki/Adjusted_mutual_information

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
 - 大数据分析挖掘

wenda.chinahadoop.cn/explore/

小象问答 搜索标题、用户 全站内容搜索 提问 首页 动态 发现 话题 通知

全部 招聘求职 机器学习 大数据平台技术 DCon 大数据行业应用 NoSQL数据库 数据科学 江湖救急

发现 最新 推荐 热门 等待回复

yarn运行时一直重复这个info...好像没找到资源，应该从哪里检查呢？
yarn fish 回复了问题 • 2人关注 • 1个回复 • 6次浏览 • 2016-05-18 13:51

两种不同的相关推荐列表
机器学习 Eric_Jiang 回复了问题 • 2人关注 • 1个回复 • 6次浏览 • 2016-05-18 13:29

如何在Linux下配java的JDK?
linux wangxialei 回复了问题 • 1人关注 • 10个回复 • 47次浏览 • 2016-05-18 12:04

sqoop把mysql数据导入Hbase报如图错误
sqoop fish 回复了问题 • 3人关注 • 3个回复 • 26次浏览 • 2016-05-18 11:56

泛化误差公式推导
机器学习 visio 回复了问题 • 4人关注 • 1个回复 • 22次浏览 • 2016-05-18 10:24

kafkaOffsetMonitor打开页面以后无法显示内容？
kafka fish 回复了问题 • 4人关注 • 2个回复 • 8次浏览 • 2016-05-18 09:36

markdown公式编辑\$符号不起作用
markdown masterwzh 回复了问题 • 3人关注 • 1个回复 • 13次浏览 • 2016-05-18 08:40

hadoop-2.6.2-src源码编译成功之后找不到native下如图一所示文件，执行图三所示搜索命令也没有找到，进入源码编译之后的目录如图二！这个文件找不到怎么解决呢？是编译没产生？
maven @CrazyChao 回复了问题 • 3人关注 • 4个回复 • 40次浏览 • 2016-05-17 21:36

opentsdb安装时出现72个warning，是正常的么？
opentsdb fish 回复了问题 • 3人关注 • 5个回复 • 49次浏览 • 2016-05-17 18:53

关于在线广告和个性化推荐区别的一点浅见
计算机广告 wayaya 回复了问题 • 4人关注 • 7个回复 • 108次浏览 • 2016-05-17 18:26

专题
招聘求职
大数据行业应用
数据科学
系统与编程
云计算技术

热门话题 更多 >
机器学习 193个问题, 86人关注
spark 200个问题, 91人关注
算法 55个问题, 65人关注
linux 179个问题, 47人关注
hbase 224个问题, 62人关注

热门用户 更多 >
gongfc 17个问题, 0次赞同
Hagrid 55个问题, 3次赞同
yanglei 55个问题, 12次赞同
天热不下雨 48个问题, 0次赞同
hiveman 19个问题, 1次赞同

感谢大家！

恳请大家批评指正！