学习内容

1. 相关概念（无监督学习、聚类的定义）
2. 性能度量(外部指标、内部指标)
3. 距离计算
4. 原型聚类
   K均值
   LVQ
   高斯混合聚类
5. 层次聚类
   AGNES
   自顶而下
6. 密度聚类
   DBSCAN
   其他密度聚类算法
7. 优缺点
8. sklearn参数详解

1.相关概念

- 无监督学习： 无监督学习是机器学习的一种方法，没有给定事先标记过的训练示例，自动对输入的数据进行分类或分群。无监督学习的主要运用包含：聚类分析、关系规则、维度缩减。它是监督式学习和强化学习等策略之外的一种选择。 一个常见的无监督学习是数据聚类。在人工神经网络中，生成对抗网络、自组织映射和适应性共振理论则是最常用的非监督式学习。
- 聚类： 聚类是一种无监督学习。聚类是把相似的对象通过静态分类的方法分成不同的组别或者更多的子集，这样让在同一个子集中的成员对象都有相似的一些属性，常见的包括在坐标系中更加短的空间距离等。

2.性能度量

在机器学习中我们都需要对任务进行评价以便于进行下一步的优化，聚类的性能度量主要有一下两种。

- 外部指标：是指把算法得到的划分结果跟某个外部的“参考模型”（如专家给出的划分结果）比较
- 内部指标：是指直接考察聚类结果，不利用任何参考模型的指标。

3.距离计算

在机器学习和数据挖掘中，我们经常需要知道个体间差异的大小，进而评价个体的相似性和类别。

- 欧式距离（2-norm距离）
- 曼哈顿距离（Manhattan distance, 1-norm距离）
- 切比雪夫距离
- 闵可夫斯基距离
- 余弦相似性
- 马氏距离

欧式距离:欧氏距离是最易于理解的一种距离计算方法，源自欧氏空间中两点间的距离公式。

$$d(x,y)=\sqrt{\Sigma_{k=1}^{n}(x_k-y_k)^2}$$

曼哈顿距离: 曼哈顿距离也称为街区距离，计算公式如下：

$$d(x,y)=\Sigma_{k=1}^{n}|x_k-y_k|$$

切比雪夫距离：

$$d(x,y)=\lim_{n\to\infty}(\Sigma_{k=1}^{n}(|x_k-y_k|)^r)^{\frac{1}{r}}=max_k(|x_k-y_k|)$$

闵可夫斯基距离:

$$d(x,y)=(\Sigma_{k=1}^{n}(|x_k-y_k|)^r)^{\frac{1}{r}}$$

式中，r是一个可变参数，根据参数r取值的不同，**闵可夫斯基距离可以表示一类距离**

r = 1时，为曼哈顿距离

r = 2时，为欧式距离

r →∞时，为切比雪夫距离

闵可夫斯基距离包括欧式距离、曼哈顿距离、切比雪夫距离都假设数据各维属性的量纲和分布（期望、方差）相同，因此适用于度量独立同分布的数据对象。

余弦距离:

余弦相似度公式定义如下：

$$cos(x,y)=\frac{xy}{|x|\,|y|}=\frac{\Sigma_{k=1}^{n}x_k y_k}{\sqrt{\Sigma_{k=1}^{n}x_k^2}\sqrt{\Sigma_{k=1}^{n}y_k^2}}$$

余弦相似度实际上是向量xx和yy夹角的余弦度量，可用来衡量两个向量方向的差异。如果余弦相似度为11，则xx和yy之间夹角为0°0°，两向量除模外可认为是相同的；如果预先相似度为00，则xx和yy之间夹角为90°90°，则认为两向量完全不同。在计算余弦距离时，将向量均规范化成具有长度11，因此不用考虑两个数据对象的量值。 余弦相似度常用来度量文本之间的相似性。文档可以用向量表示，向量的每个属性代表一个特定的词或术语在文档中出现的频率，尽管文档具有大量的属性，但每个文档向量都是稀疏的，具有相对较少的非零属性值。

马氏距离:

$$mahalanobis(x,y)=(x-y)\Sigma^{-1}(x-y)^T$$

式中，Σ−1Σ−1是数据协方差矩阵的逆。 前面的距离度量方法大都假设样本独立同分布、数据属性之间不相关。马氏距离考虑了数据属性之间的相关性，排除了属性间相关性的干扰，而且与量纲无关。若协方差矩阵是对角阵，则马氏距离变成了标准欧式距离；若协方差矩阵是单位矩阵，各个样本向量之间独立同分布，则变成欧式距离。

4.原型聚类

原型聚类亦称"基于原型的聚类" (prototype-based clustering)，此类算法假设聚类结构能通过一组原型刻画，在现实聚类任务中极为常用.通常情形下，算法先对原型进行初始化，然后对原型进行迭代更新求解.采用不同的原型表示、不同的求解方式，将产生不同的算法.

- K均值
- LVQ
- 高斯混合聚类

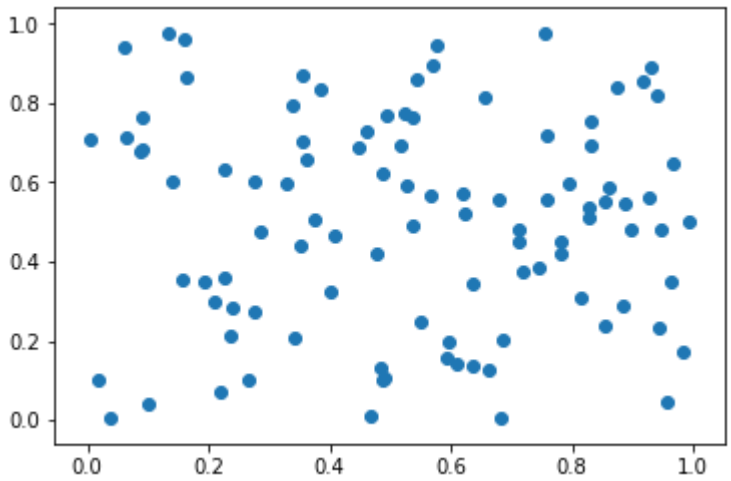
k均值聚类算法（k-means clustering algorithm）是一种迭代求解的聚类分析算法，其步骤是
创建 k 个点作为起始质心（通常是随机选择）
当任意一个点的簇分配结果发生改变时（不改变时算法结束）
    对数据集中的每个数据点
        对每个质心
            计算质心与数据点之间的距离
        将数据点分配到距其最近的簇
    对每一个簇, 计算簇中所有点的均值并将均值作为质心
聚类中心以及分配给它们的对象就代表一个聚类。

In [3]:

```
# 通过简单的例子来直接查看K均值聚类的效果
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline


# 聚类前
X = np.random.rand(100, 2)
plt.scatter(X[:, 0], X[:, 1], marker='o')
```

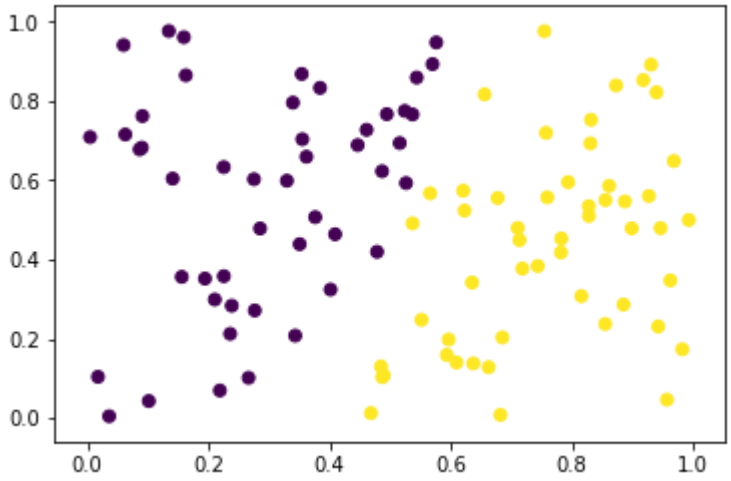
Out[3]:

```
<matplotlib.collections.PathCollection at 0x11c127240>
```



In [4]:

```
#聚类后
kmeans = KMeans(n_clusters=2).fit(X)
label_pred = kmeans.labels_
plt.scatter(X[:, 0], X[:, 1], c=label_pred)
plt.show()
```



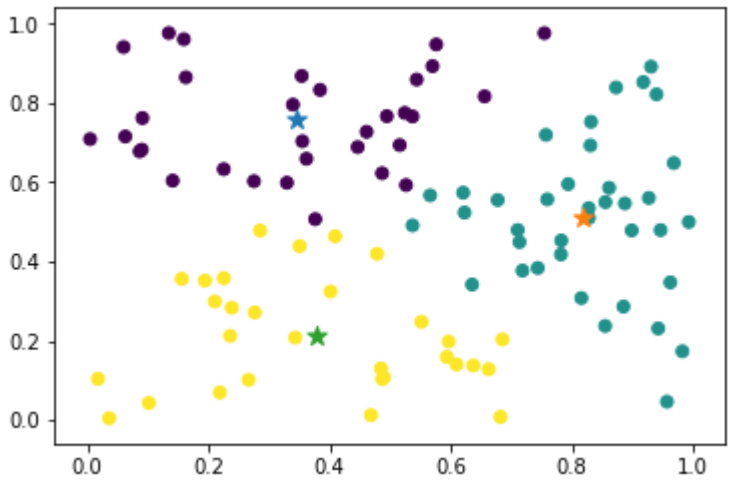
In [23]:

```
def distEclud(vecA, vecB):
    '''
    欧氏距离计算函数
    :param vecA:
    :param vecB:

    :return: float
    '''
    dist = 0.0
    # ========= show me your code ==================
    for d in range(vecA.shape[0]):  # d 即维数
        dist += (vecA[d]-vecB[d]) ** 2
    dist = np.sqrt(dist)
    # ========= show me your code ==================
    return dist


def randCent(dataMat, k):
    '''
    为给定数据集构建一个包含K个随机质心的集合,
    随机质心必须要在整个数据集的边界之内,这可以通过找到数据集每一维的最小和最大值来完成
    然后生成0到1.0之间的随机数并通过取值范围和最小值,以便确保随机点在数据的边界之内
    :param np.dataMat:
    :param k:

    :return: np.dataMat
    '''
    # 获取样本数与特征值
    m, n = np.shape(dataMat)
    # 初始化质心,创建以(k,n)个零填充的矩阵
    centroids = np.mat(np.zeros((k, n)))

    # ========= show me your code ==================
    # 循环遍历特征值
    import random as rd
    for j in range(k):
        for col in range(n):
            temp = rd.random()
            col_max = max(dataMat[:, col])
            col_min = min(dataMat[:, col])
            r_num = col_min + (col_max-col_min)*temp
            centroids[j, col] = r_num
    # ========= show me your code ==================

    # 返回质心
    return centroids.A  # matrix矩阵名.A代表将矩阵转化为array数组类型,值与行列不变


def kMeans(dataMat, k, distMeas=distEclud):
    '''
    创建K个质心,然后将每个点分配到最近的质心,再重新计算质心。
    这个过程重复数次,直到数据点的簇分配结果不再改变为止
    :param dataMat: 数据集
    :param k: 簇的数目
    :param distMeans: 计算距离
    :return:
    '''
    # 获取样本数和特征数
    m, n = np.shape(dataMat)
```

```
    # 初始化一个矩阵来存储每个点的簇分配结果
    # clusterAssment包含两个列:一列记录簇索引值,第二列存储误差(误差是指当前点到簇质心的距离,后面
    clusterAssment = np.mat(np.zeros((m, 2)))
    # 创建质心,随机K个质心
    centroids = randCent(dataMat, k)

    # 初始化标志变量,用于判断迭代是否继续,如果True,则继续迭代
    clusterChanged = True
    while clusterChanged:
        clusterChanged = False
        # 遍历所有数据找到距离每个点最近的质心,
        # 可以通过对每个点遍历所有质心并计算点到每个质心的距离来完成
        for i in range(m):
            minDist = float("inf")
            minIndex = -1
            for j in range(k):  #遍历每个质心, 每个点记录距离最近的那个质心
                # 计算距离是使用distMeas参数给出的距离公式,默认距离函数是distEclud
                distJI = distMeas(centroids[j, :], dataMat[i, :])  #第 j 个质心和第 i
                # 如果距离比minDist(最小距离)还小,更新minDist(最小距离)和最小质心的index(索
                if distJI < minDist:
                    minDist = distJI
                    minIndex = j

            # 如果任一点的簇分配结果发生改变,则更新clusterChanged标志
            # ========= show me your code ==================
            if clusterAssment[i, 0] != minIndex:
                clusterChanged = True
            # ========= show me your code ==================
            # 更新簇分配结果为最小质心的index(索引),minDist(最小距离)的平方
            clusterAssment[i, :] = minIndex, minDist ** 2


        # 遍历所有质心并更新它们的取值, 对每一个簇, 计算簇中所有点的均值并将均值作为质心
        # ========= show me your code ==================
        for j in range(k):
            temp = np.array([dataMat[i] for i in range(m) if clusterAssment[i, 0]==j
            for col in range(n):
                centroids[j][col] = np.mean(temp[:, col])
        # ========= show me your code ==================

    # 返回所有的类质心与点分配结果
    return centroids, clusterAssment
```

In [33]:

```
# 运行Kmeans，假设有k聚类中心
give_k = 3
center,label_pred = kMeans(X, k=give_k)

# 将标签转化成易绘图的形式
label = label_pred[:, 0].A.reshape(-1)

# 将结果可视化
plt.scatter(X[:, 0], X[:, 1], c=label)
for i in range(give_k):
    plt.scatter(center[i, 0], center[i, 1], marker="*", s = 100)
plt.show()
```



- 学习向量量化(Learning Vector Quantization,简称LVQ)

属于原型聚类，即试图找到一组原型向量来聚类，每个原型向量代表一个簇，将空间划分为若干个簇，从而对于任意的样本，可以将它划入到它距离最近的簇中，不同的是LVQ假设数据样本带有类别标记，因此可以利用这些类别标记来辅助聚类。

> 大致思想如下：
> 1. 统计样本的类别，假设一共有q类，初始化为原型向量的标记为{t1,t2,……,tq}。从样本中随机选取q个样本点位原型向量{p1, p2 ,……, pq}。初始化一个学习率a,a 取值范围(0,1)。
> 2. 从样本集中随机选取一个样本(x, y)，计算该样本与q个原型向量的距离（欧几里得距离），找到最小的那个原型向量p，判断样本的标记y与原型向量的标记t是不是一致。若一致则更新为p′ = p + a*(x-p)，否则更新为p′ = p − a*(x − p)。
> 3. 重复第2步直到满足停止条件。（如达到最大迭代次数）
> 4. 返回q个原型向量。

- 高斯混合聚类：

高斯混合聚类与k均值、LVQ用「原型向量」来刻画聚类结构不同，高斯混合聚类采用「概率模型」来表达聚类原型。相对于k均值聚类算法使用 k 个原型向量来表达聚类结构，高斯混合聚类使用 k 个高斯概率密度函数混合来表达聚类结构

相当于 k 个正态分布的线性组合。学习过程就是去估计K个高斯分布的概率密度Ø(y|θk)，以及每个高斯分布的权重α_k. 所谓聚类，就是对于某个样本yj，把该样本代入到k个高斯分布中求出属于每个类别的概率，然后选择概率值最高的那个类别作为它最终的归属 。

$$P(x_i|y_k) = \frac{1}{\sqrt{2\pi\sigma_{y_k}^2}} exp(-\frac{(x_i - \mu_{y_k})^2}{2\sigma_{y_k}^2})$$

于是迭代更新 k 个簇原型向量的工作转换为了迭代更新 k 个高斯概率密度函数的任务。每个高斯概率密度函数代表一个簇，当一个新的样本进来时，我们可以通过这 k 的函数的值来为新样本分类。

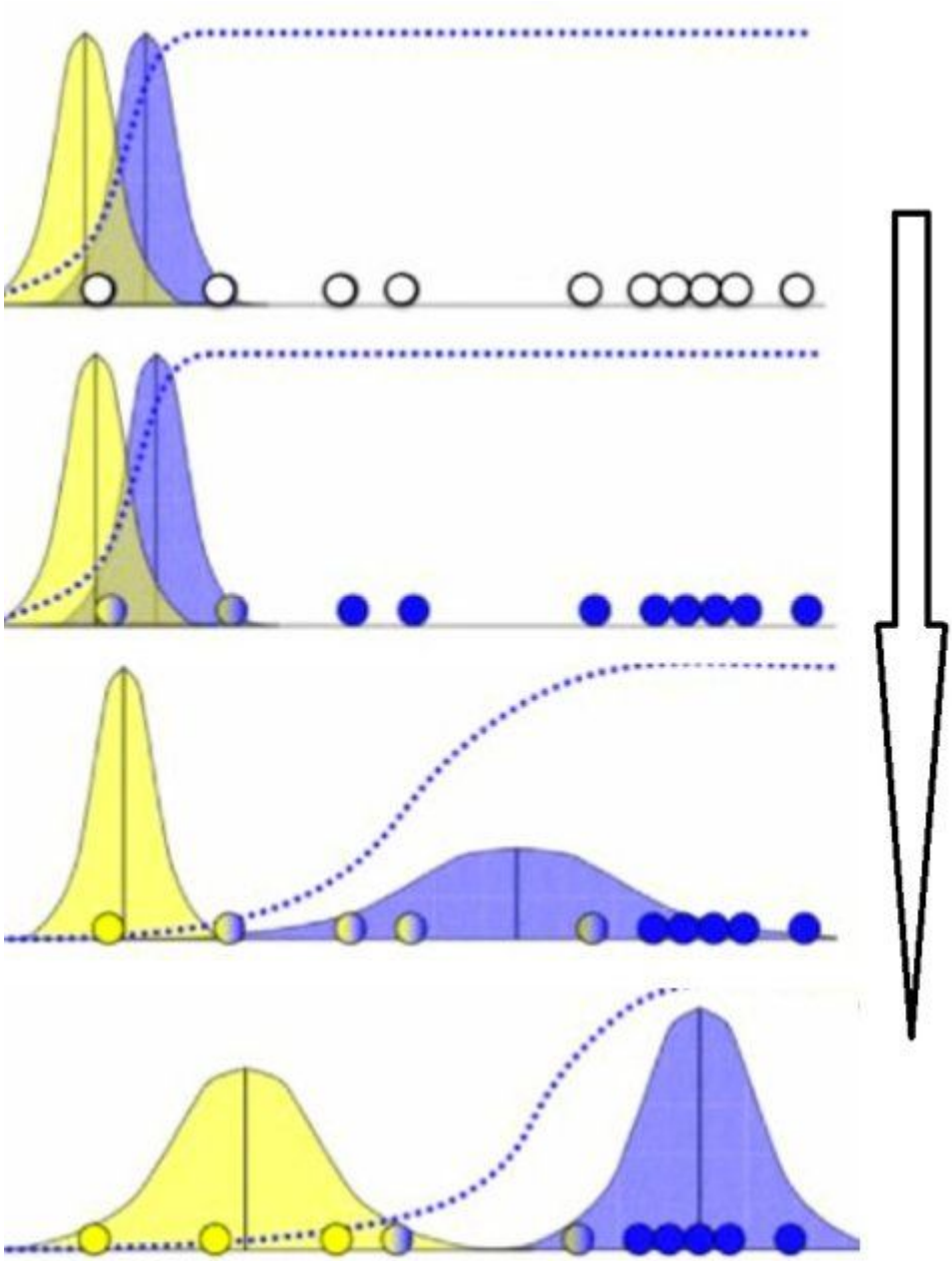
高斯混合聚类的步骤：首先假设样本集具有一些规律，包括可以以α参数作为比例分为k类且每类内符合高斯分布。然后根据贝叶斯原理利用极大似然法同时求出决定分类比例的 α 和决定类内高斯分布的 μ、Σ。最后将样本根据 α、μ、Σ 再次通过贝叶斯原理求出样本该分在哪个簇。

整个步骤下来，这种做法其实就是一种原型聚类：通过找到可以刻画样本的原型（α、μ、Σ 参数），迭代得到 α、μ、Σ 参数的最优解。

高斯混合模型聚类算法EM步骤如下：

1. 猜测有几个类别，既有几个高斯分布。
2. 针对每一个高斯分布，随机给其均值和方差进行赋值。
3. 针对每一个样本，计算其在各个高斯分布下的概率。
4. 针对每一个高斯分布，每一个样本对该高斯分布的贡献可以由其下的概率表示，如概率大则表示贡献大，反之亦然。这样把样本对该高斯分布的贡献作为权重来计算加权的均值和方差。之后替代其原本的均值和方差。
5. 重复3~4直到每一个高斯分布的均值和方差收敛。

参考链接：https://www.cnblogs.com/Luv-GEM/p/10851395.html (https://www.cnblogs.com/Luv-GEM/p/10851395.html) .下图可理解为，假设有 10 个身高样本数据，并不知道每个样本数据是来自男生还是女生。在这种情况下，如何将这 10 个身高数据聚成男女生两大类。

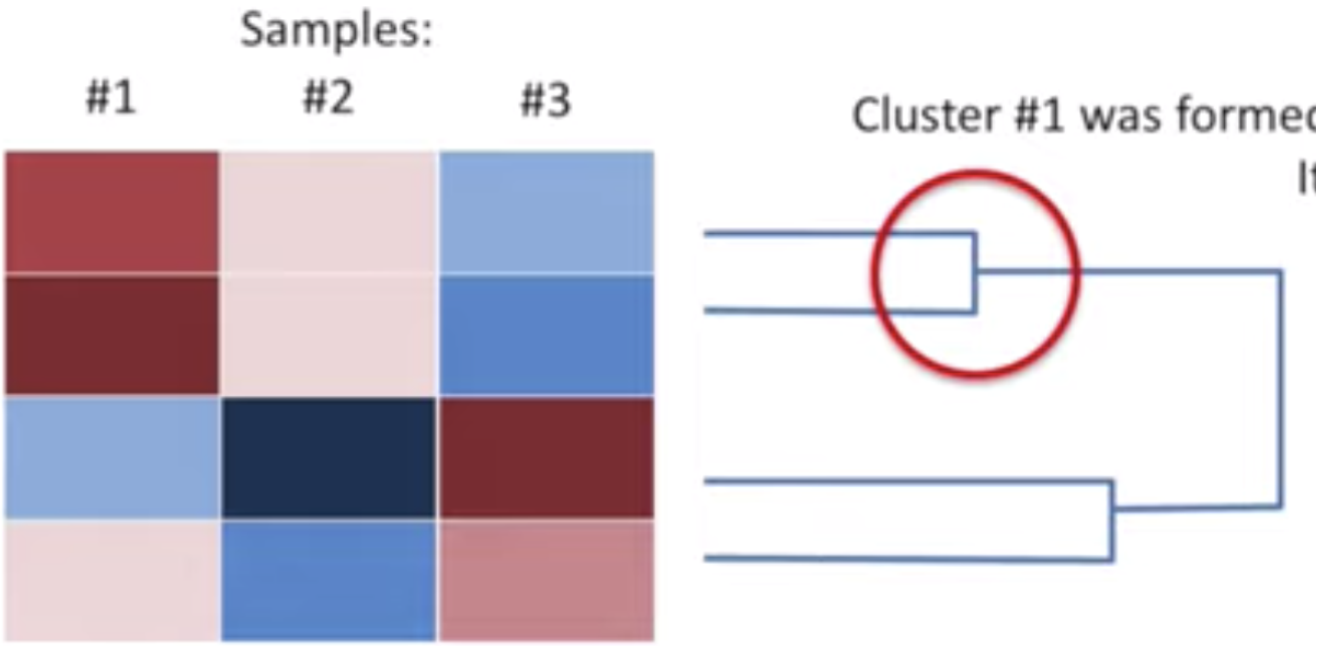


5.层次聚类

层次聚类(hierarchical clustering)基于 **「簇间的相似度」** 在不同层次上分析数据，从而形成树形的聚类结构，层次聚类一般有两种划分策略：自底向上的聚合（agglomerative）策略和自顶向下的分拆（divisive）策略

- AGNES
- 自顶而下

AGNES算法是自底向上的层次聚类算法。开始时将数据集中的每个样本初始化为一个簇，**「然后找到距离最近的两个簇，将他们合并，不断重复这个过程」**，直达到到预设的聚类数目为止。



簇间距离的计算可以有三种形式：

最小距离：$d_{min}(C_i, C_j) = \min_{p \in C_i, q \in C_j} |p - q|$.

最大距离：$d_{max}(C_i, C_j) = \max_{p \in C_i, q \in C_j} |p - q|$.

平均距离：$d_{avg}(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{p \in C_i} \sum_{q \in C_j} |p - q|$.

输入：样本集D={x1,x2,...,xm}D={x1,x2,...,xm}
聚类簇距离度量函数dd；
聚类簇数kk

过程：

```
1. for j=1,2,...,mj=1,2,...,m do
2.  Cj={xj}Cj={xj}
3. end for
4. for i=1,2,...,mi=1,2,...,m do
5.  for i=1,2,...,mi=1,2,...,m do
6.   M(i,j)=d(Ci,Cj)M(i,j)=d(Ci,Cj);
7.   M(j,i)=M(i,j)M(j,i)=M(i,j);
8.  end for
9. end for
10. 设置当前聚类簇个数：q=mq=m;
11. while q>kq>k do
12.  找出距离最近的两个聚类簇Ci*Ci*和Cj*Cj*;
13.  合并Ci*Ci*和Cj*Cj*：Ci*=Ci*∪Cj*Ci*=Ci*∪Cj*;
14.  for j=j*+1,j*+2,..,qj=j*+1,j*+2,..,q do
15.   将聚类簇CjCj重新编号为CjCj
16.  end for
17.  删除距离矩阵MM的第j*j*行和第j*j*列;
18.  for j=1,2,...,q−1j=1,2,...,q−1 do
19.   M(i,j)=d(Ci,Cj)M(i,j)=d(Ci,Cj);
20.   M(j,i)=M(i,j)M(j,i)=M(i,j);
21.  end for
22.  q=q−1q=q−1
23. end while
```

输出：簇划分:C={C1,C2,...,Ck}

自顶而下：把整个数据集视作一个簇，然后把一个簇分成几个簇，接着再分别把每一个簇分成更小的簇，如此反复下去，直到满足要求为止。

6.密度聚类

密度聚类假设聚类结构通过样本分布的紧密程度。此算法是基于密度的角度来考察样本之间的连接性，并基于连接性不断扩展聚类簇最后获得最终的结果。通过判断样本在区域空间内是否大于某个阈值来决定是否将其放到与之相近的样本中。

- DBSCAN
- 其他密度聚类算法

DBSCAN

e-邻域:对xj∈D,其∈邻域包含样本集D中与xj的距离不大于e的样本,即N(xj)= {xi∈D | dist(xi,xj)≤e};
核心对象(core object): 若xj的E-邻域至少包含MinPts个样本，即|Ne(xj)|≥MinPts,则xj是-一个核心对象;
密度直达(directly density- reachable):若xj位于xi的e-邻域中,且xi是核心对象,则称x;由xi密度直达;
密度可达(density. reachable): 对xi与xj,若存在样本序列P1,P2,... ,Pn,其中p1=xi,Pn=xj且pi+1由pi密度直达,则称xj由xi密度可达;
密度相连(density-conected): 对xi与xj,若存在xk使得xi与xj均由xk密度可达,则称xi与xj密度相连.

```
首先将数据集D中的所有对象标记为未处理状态
for（数据集D中每个对象p） do
    if （p已经归入某个簇或标记为噪声） then
         continue;
    else
         检查对象p的Eps邻域 NEps(p) ；
         if (NEps(p)包含的对象数小于MinPts) then
                  标记对象p为边界点或噪声点；
         else
                 标记对象p为核心点，并建立新簇C，并将p邻域内所有点加入C
                 for (NEps(p)中所有尚未被处理的对象q) do
                       检查其Eps邻域NEps(q)，若NEps(q)包含至少MinPts个对象，则将N
Eps(q)中未归入任何一个簇的对象加入C；
                 end for
        end if
    end if
 end for
```

优点

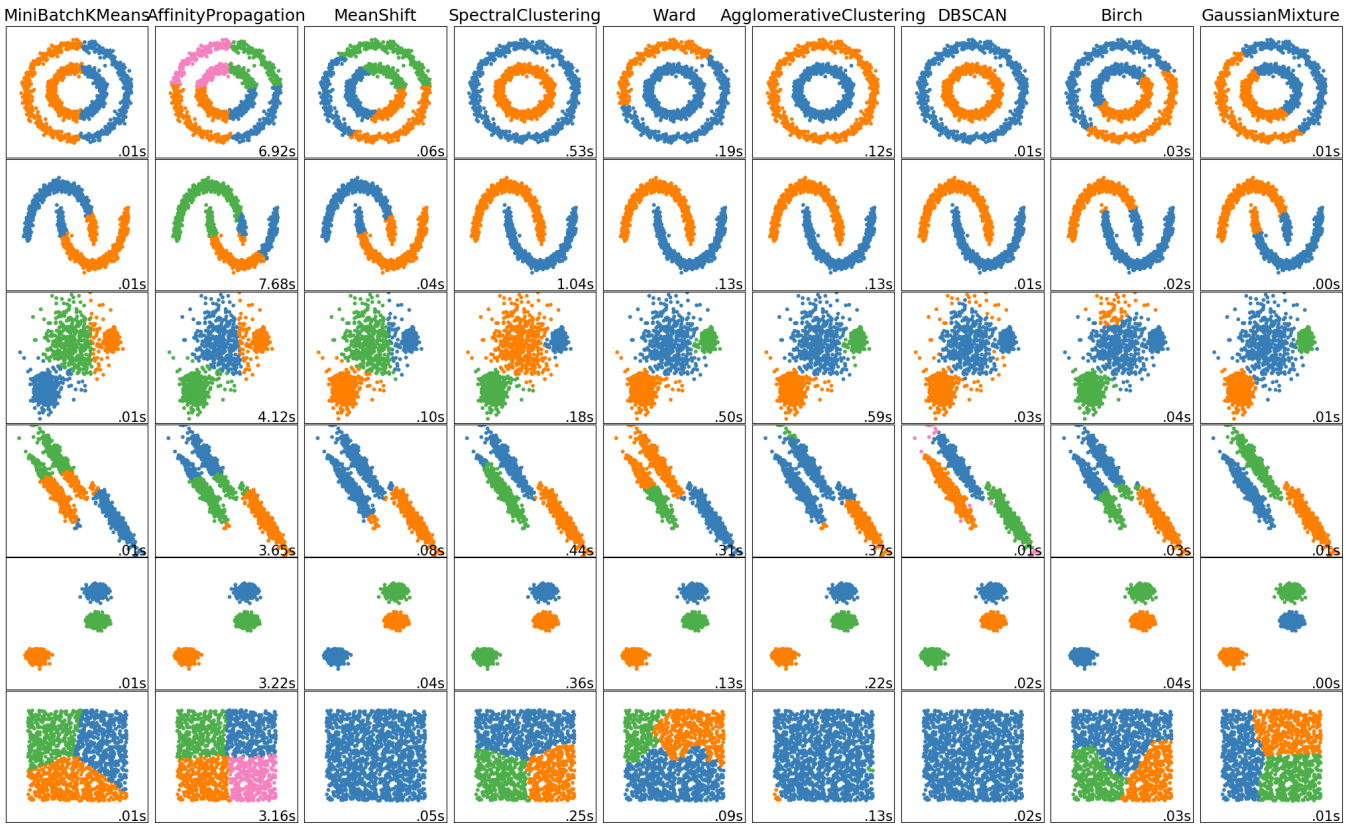
- 相比 K-平均算法，DBSCAN 不需要预先声明聚类数量。
- DBSCAN 可以找出任何形状的聚类，甚至能找出一个聚类，它包围但不连接另一个聚类，另外，由于 MinPts 参数，single-link effect （不同聚类以一点或极幼的线相连而被当成一个聚类）能有效地被避免。
- DBSCAN 能分辨噪音（局外点）。
- DBSCAN 只需两个参数，且对数据库内的点的次序几乎不敏感（两个聚类之间边缘的点有机会受次序的影响被分到不同的聚类，另外聚类的次序会受点的次序的影响）。
- DBSCAN 被设计成能配合可加速范围访问的数据库结构，例如 R*树。
- 如果对资料有足够的了解，可以选择适当的参数以获得最佳的分类。

缺点

- DBSCAN 不是完全决定性的：在两个聚类交界边缘的点会视乎它在数据库的次序决定加入哪个聚类，幸运地，这种情况并不常见，而且对整体的聚类结果影响不大——DBSCAN 对核心点和噪音都是决定性的。DBSCAN* 是一种变化了的算法，把交界点视为噪音，达到完全决定性的结果。
- DBSCAN 聚类分析的质素受函数 regionQuery(P,ε) 里所使用的度量影响，最常用的度量是欧几里得距离，尤其在高维度资料中，由于受所谓“维数灾难”影响，很难找出一个合适的 ε，但事实上所有使用欧几里得距离的算法都受维数灾难影响。
- 如果数据库里的点有不同的密度，而该差异很大，DBSCAN 将不能提供一个好的聚类结果，因为不能选择一个适用于所有聚类的 minPts-ε 参数组合。
- 如果没有对资料和比例的足够理解，将很难选择适合的 ε 参数。

7.优缺点

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-Means	number of clusters	Very large `n_samples`, medium `n_clusters` with MiniBatch code	General-purpose, even cluster size, flat geometry, not too many clusters	Distances between points
Affinity propagation	damping, sample preference	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	bandwidth	Not scalable with `n_samples`	Many clusters, uneven cluster size, non-flat geometry	Distances between points
Spectral clustering	number of clusters	Medium `n_samples`, small `n_clusters`	Few clusters, even cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Ward hierarchical clustering	number of clusters	Large `n_samples` and `n_clusters`	Many clusters, possibly connectivity constraints	Distances between points
Agglomerative clustering	number of clusters, linkage type, distance	Large `n_samples` and `n_clusters`	Many clusters, possibly connectivity constraints, non Euclidean distances	Any pairwise distance
DBSCAN	neighborhood size	Very large `n_samples`, medium `n_clusters`	Non-flat geometry, uneven cluster sizes	Distances between nearest points
Gaussian mixtures	many	Not scalable	Flat geometry, good for density estimation	Mahalanobis distances to centers
Birch	branching factor, threshold, optional global clusterer.	Large `n_clusters` and `n_samples`	Large dataset, outlier removal, data reduction.	Euclidean distance between points



8.sklearn参数详解

接下来就通过查看sklearn的参数去完成自己的例子 https://sklearn.apachecn.org/docs/0.21.3/22.html (https://sklearn.apachecn.org/docs/0.21.3/22.html)

参考文献： 西瓜书 维基百科 机器学习实战

https://www.zybuluo.com/rianusr/note/1199877 (https://www.zybuluo.com/rianusr/note/1199877)

http://ddrv.cn/a/66611 (http://ddrv.cn/a/66611)

https://blog.csdn.net/zhouxianen1987/article/details/68945844 (https://blog.csdn.net/zhouxianen1987/article/details/68945844)

http://ddrv.cn/a/66611 (http://ddrv.cn/a/66611)

https://zhuanlan.zhihu.com/p/29538307 (https://zhuanlan.zhihu.com/p/29538307)