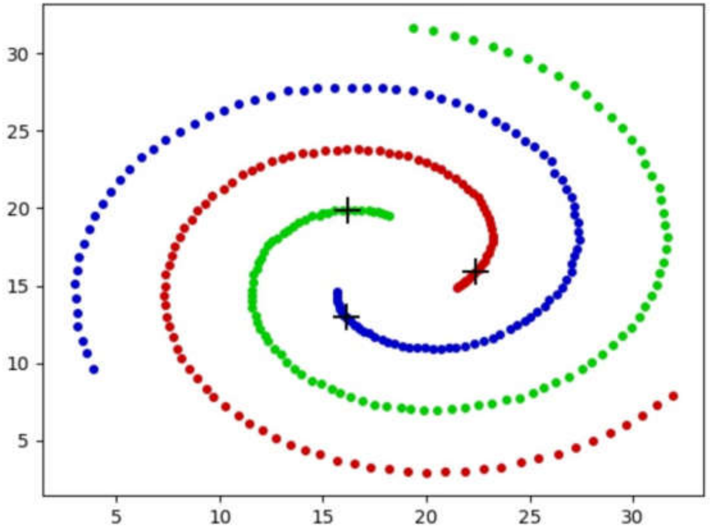
Python编程与人工智能实践



算法篇： 密度峰值聚类 (Density Peak)

于泓

鲁东大学

信息与电气工程学院

2021.3.20

密度峰值聚类（Density Peak）

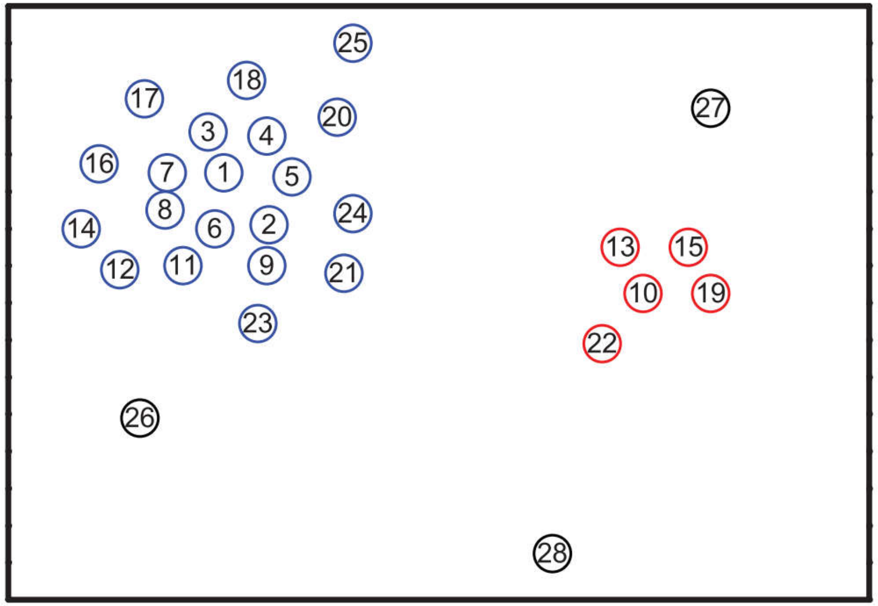
- 密度峰值聚类（DP）算法是一种不需要迭代的，可以一次性找到聚类中心的方法聚类方法。

- 基本思想：

  (1)　聚类中心的**密度**（Density）应当**比较大**

  (2) 聚类中心应当离比其**密度更大的点较远**

Rodriguez A , Laio A . Clustering by fast search and find of density peaks[J]. Science, 2014, 344(6191):1492.



点 1 密度最大是一个聚类中心

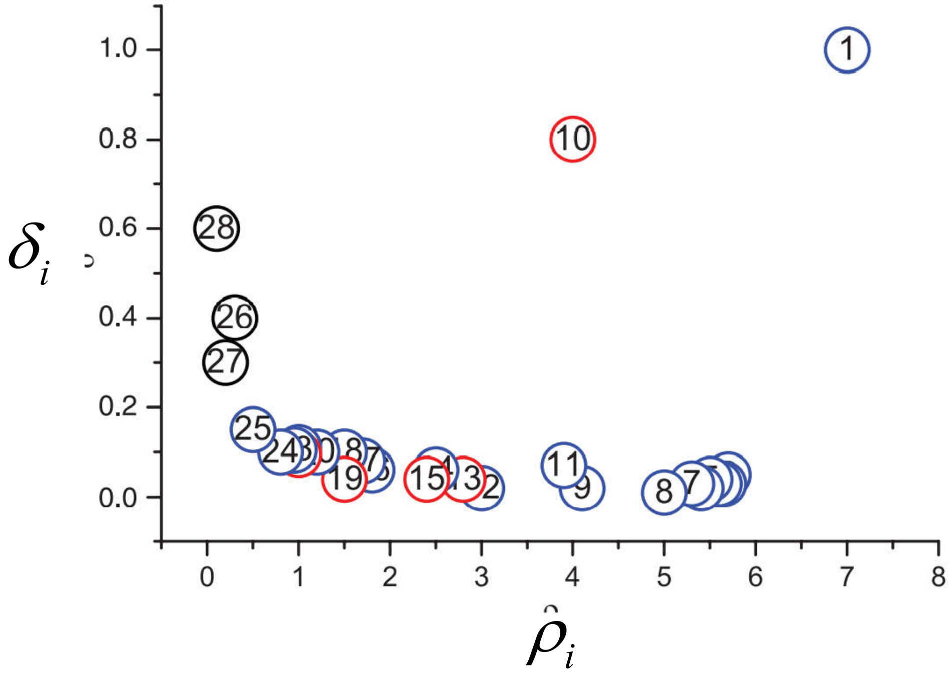
点2,6,4密度也比较大，但是距离比他们密度更大的点（点1）太近，
所以不是聚类中心

点10 密度较大，且离密度比它大的点（1,2,4,6）较远是聚类中心

在整个的算法中对数据中的每个点计算两个参数：

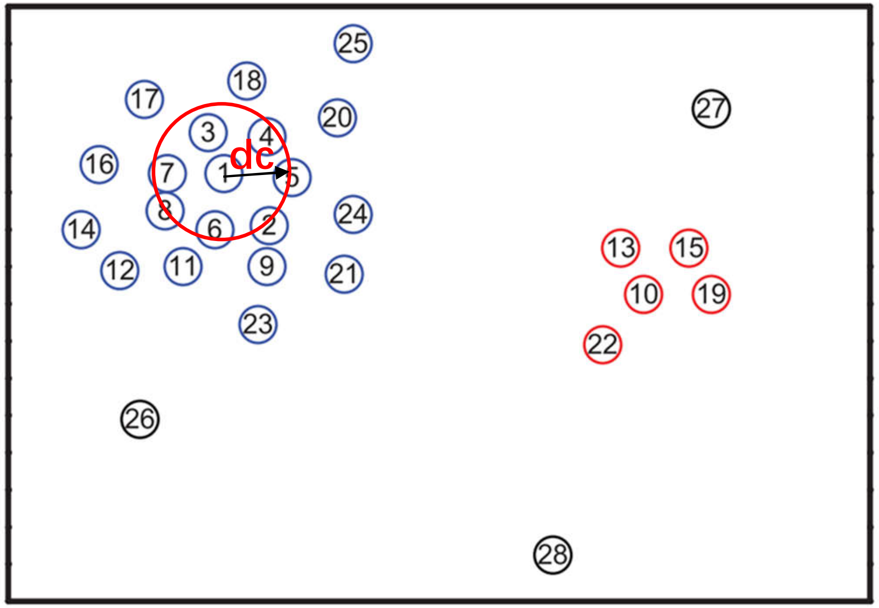
1、局部密度ρ_i

2、到密度比其大的点的最小距离 $\delta_i = \min_{j:\rho_j > \rho_i} (d_{ij})$ （**中心偏移距离**）



两者都大的点就是聚类中心点

局部密度求解方法



对每个点，以$\mathbf{d_c}$为半径画一个圆形区域，统计其中点的数目（**硬统计**）

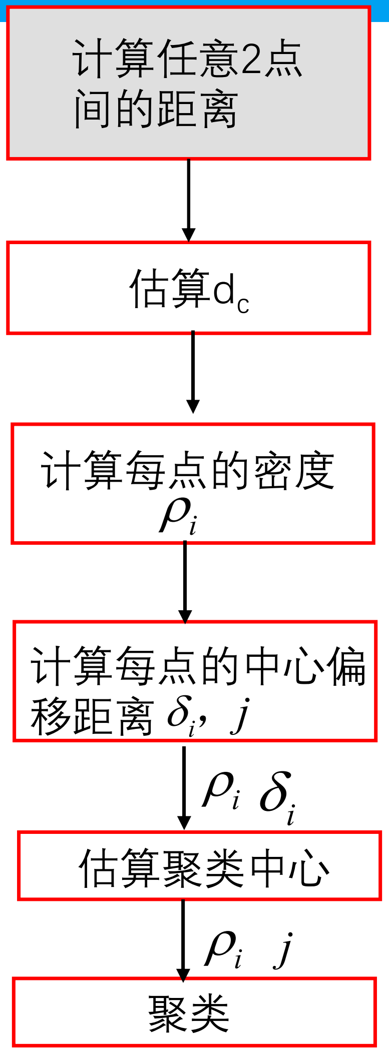
利用类高斯公式

$$\rho_i = \sum_{j=1}^{N} e^{-\left(\frac{d_{ij}}{d_c}\right)^2}$$ （**软统计**）

$\mathrm{d_c}$的求解：

落在$\mathbf{d_c}$**圆**区域内平均点数，占总点数的1%-2%

$$\delta_i = \min_{j:\rho_j>\rho_i}(d_{ij})$$



计算任意2点间的距离

估算d_c

计算每点的密度ρ_i

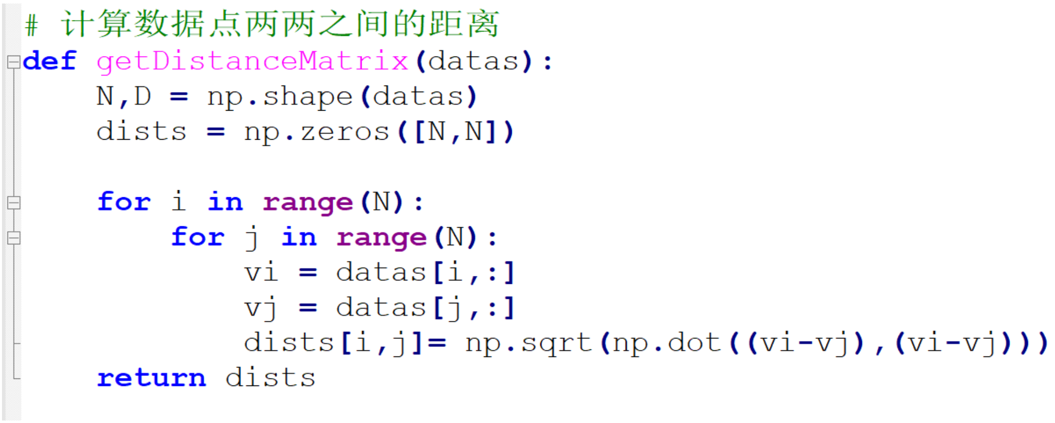
计算每点的中心偏移距离δ_i，j

ρ_i δ_i

估算聚类中心

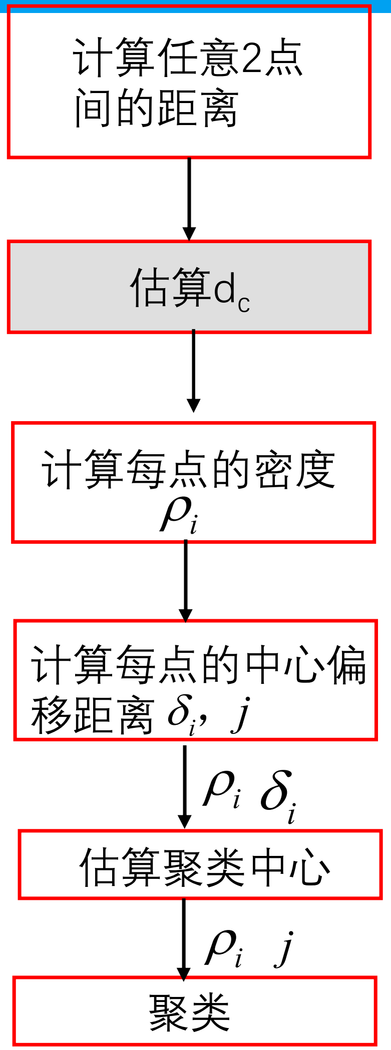
ρ_i j

聚类



```
# 计算数据点两两之间的距离
def getDistanceMatrix(datas):
    N,D = np.shape(datas)
    dists = np.zeros([N,N])

    for i in range(N):
        for j in range(N):
            vi = datas[i,:]
            vj = datas[j,:]
            dists[i,j]= np.sqrt(np.dot((vi-vj),(vi-vj)))
    return dists
```

$$\delta_i = \min_{j:\rho_j > \rho_i} (d_{ij})$$



计算任意2点间的距离

估算d_c

计算每点的密度 ρ_i

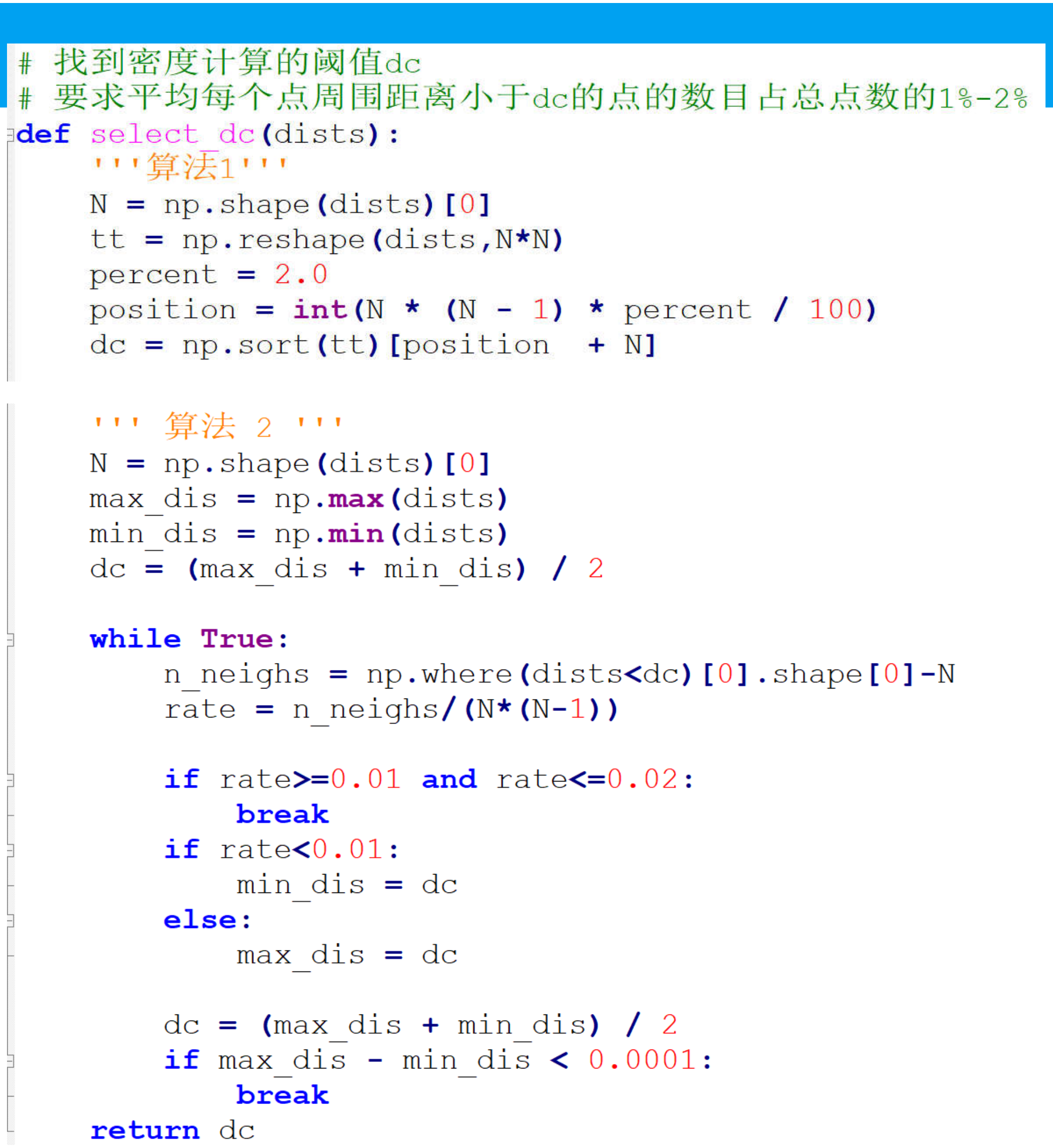
计算每点的中心偏移距离 δ_i, j

ρ_i δ_i

估算聚类中心

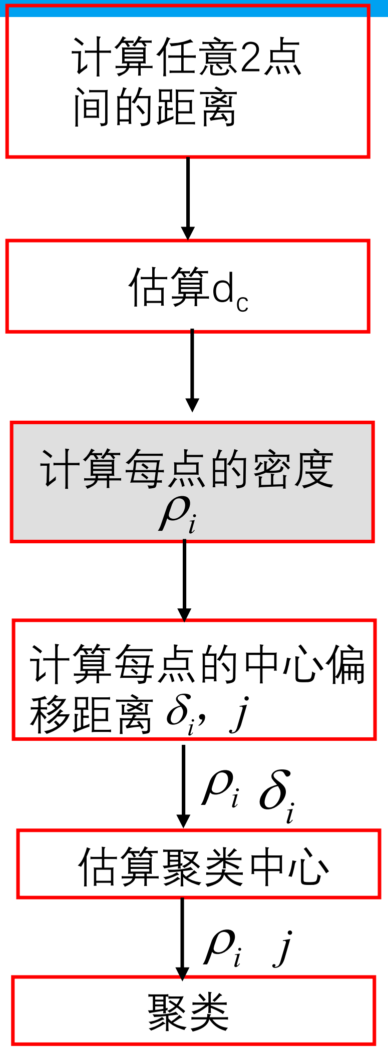
ρ_i j

聚类

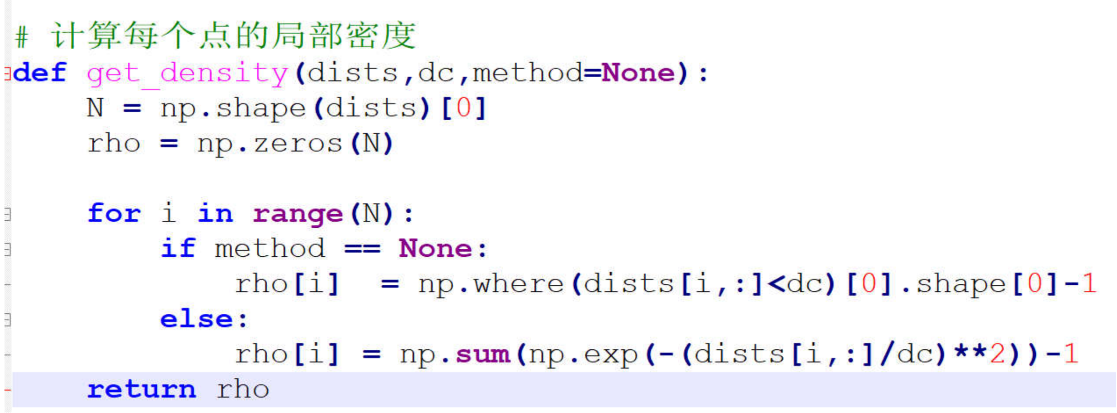


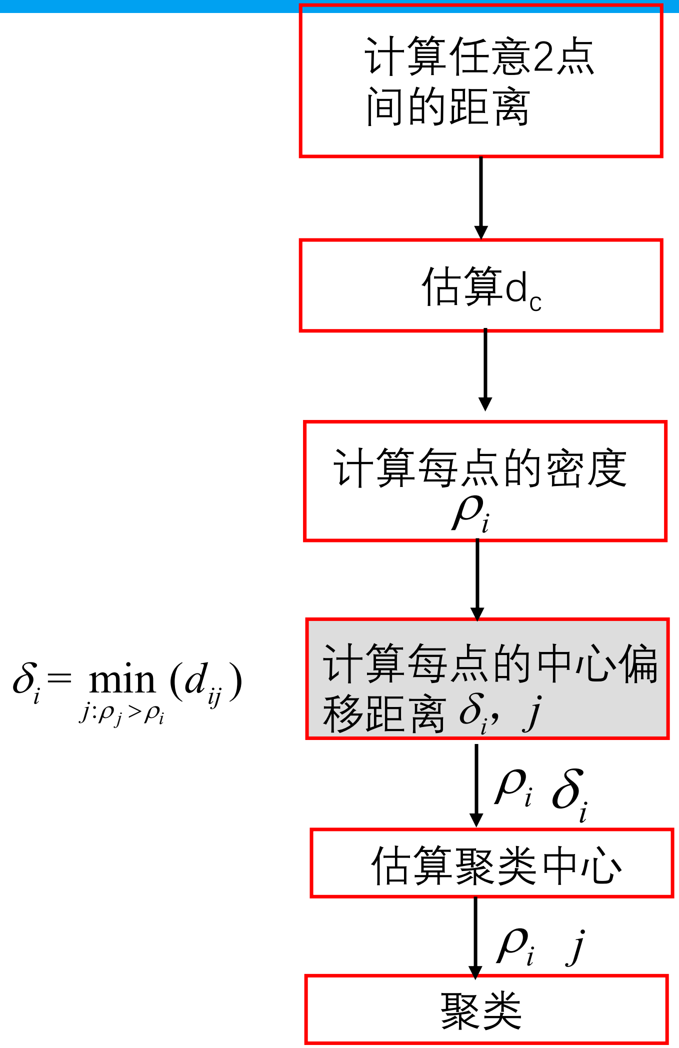
```
# 找到密度计算的阈值dc
# 要求平均每个点周围距离小于dc的点的数目占总点数的1%-2%
def select_dc(dists):
    '''算法1'''
    N = np.shape(dists)[0]
    tt = np.reshape(dists,N*N)
    percent = 2.0
    position = int(N * (N - 1) * percent / 100)
    dc = np.sort(tt)[position  + N]

    ''' 算法 2 '''
    N = np.shape(dists)[0]
    max_dis = np.max(dists)
    min_dis = np.min(dists)
    dc = (max_dis + min_dis) / 2

    while True:
        n_neighs = np.where(dists<dc)[0].shape[0]-N
        rate = n_neighs/(N*(N-1))

        if rate>=0.01 and rate<=0.02:
            break
        if rate<0.01:
            min_dis = dc
        else:
            max_dis = dc

        dc = (max_dis + min_dis) / 2
        if max_dis - min_dis < 0.0001:
            break
    return dc
```

利用二分法查找



$$\delta_i = \min_{j:\rho_j > \rho_i} (d_{ij})$$


```
# 计算每个点的局部密度
def get_density(dists,dc,method=None):
    N = np.shape(dists)[0]
    rho = np.zeros(N)

    for i in range(N):
        if method == None:
            rho[i]  = np.where(dists[i,:]<dc)[0].shape[0]-1
        else:
            rho[i] = np.sum(np.exp(-(dists[i,:]/dc)**2))-1
    return rho
```



计算任意2点间的距离

估算d_c

计算每点的密度 ρ_i

$$\delta_i = \min_{j:\rho_j>\rho_i}(d_{ij})$$

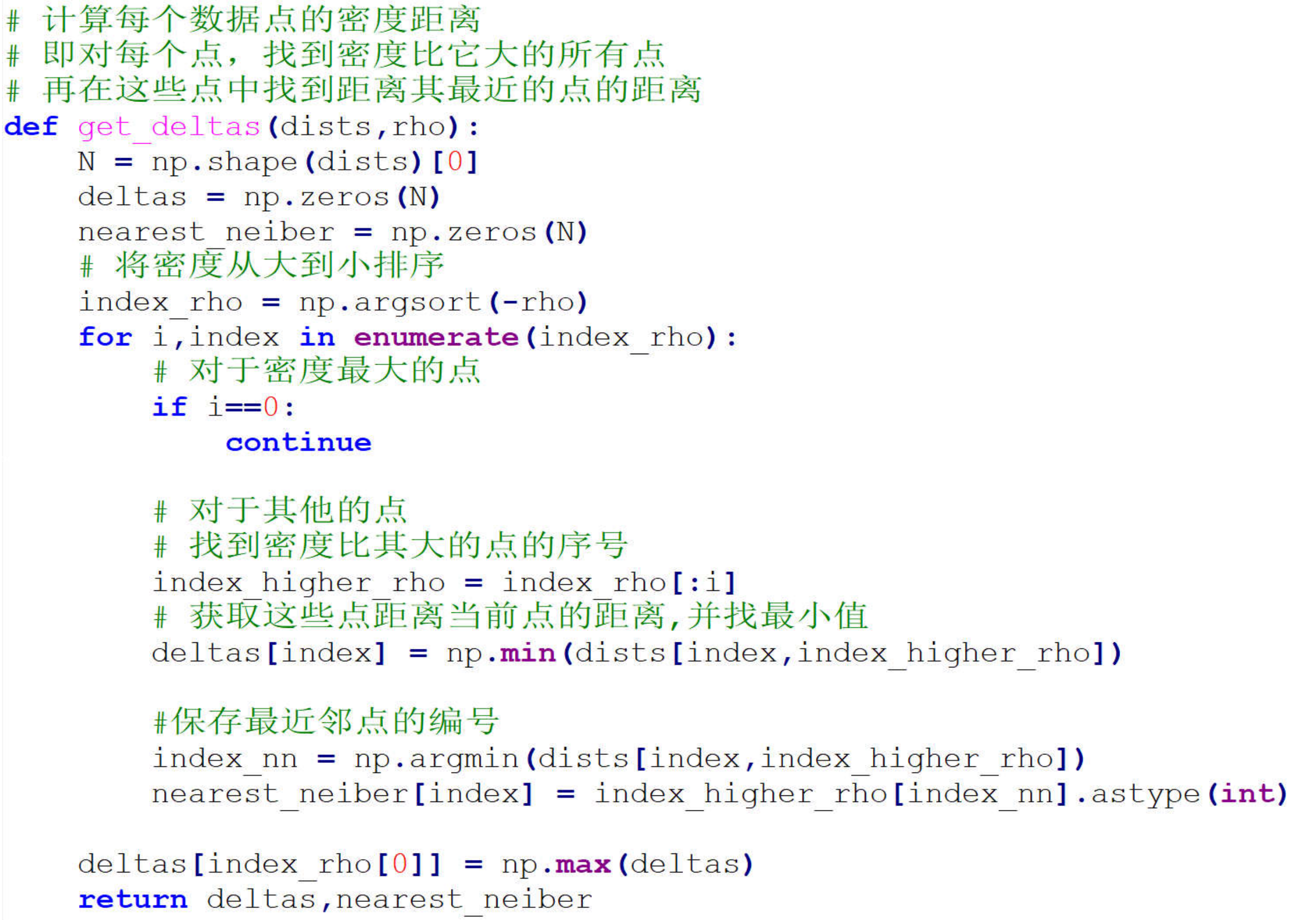
计算每点的中心偏移距离 δ_i，j

ρ_i δ_i

估算聚类中心

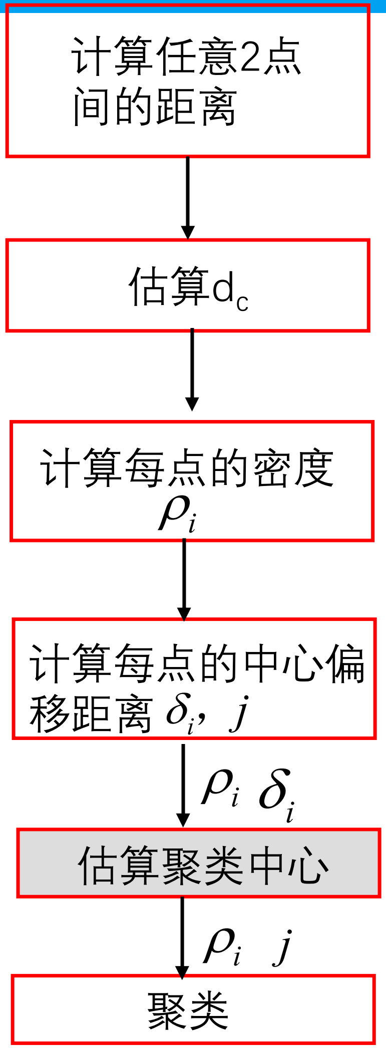
ρ_i j

聚类



```
# 计算每个数据点的密度距离
# 即对每个点，找到密度比它大的所有点
# 再在这些点中找到距离其最近的点的距离
def get_deltas(dists,rho):
    N = np.shape(dists)[0]
    deltas = np.zeros(N)
    nearest_neiber = np.zeros(N)
    # 将密度从大到小排序
    index_rho = np.argsort(-rho)
    for i,index in enumerate(index_rho):
        # 对于密度最大的点
        if i==0:
            continue

        # 对于其他的点
        # 找到密度比其大的点的序号
        index_higher_rho = index_rho[:i]
        # 获取这些点距离当前点的距离,并找最小值
        deltas[index] = np.min(dists[index,index_higher_rho])

        #保存最近邻点的编号
        index_nn = np.argmin(dists[index,index_higher_rho])
        nearest_neiber[index] = index_higher_rho[index_nn].astype(int)

    deltas[index_rho[0]] = np.max(deltas)
    return deltas,nearest_neiber
```

$$\delta_i = \min_{j:\rho_j>\rho_i}(d_{ij})$$



计算任意2点间的距离

估算d_c

计算每点的密度 ρ_i

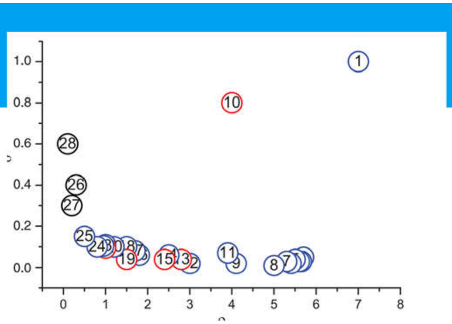
计算每点的中心偏移距离 δ_i，j

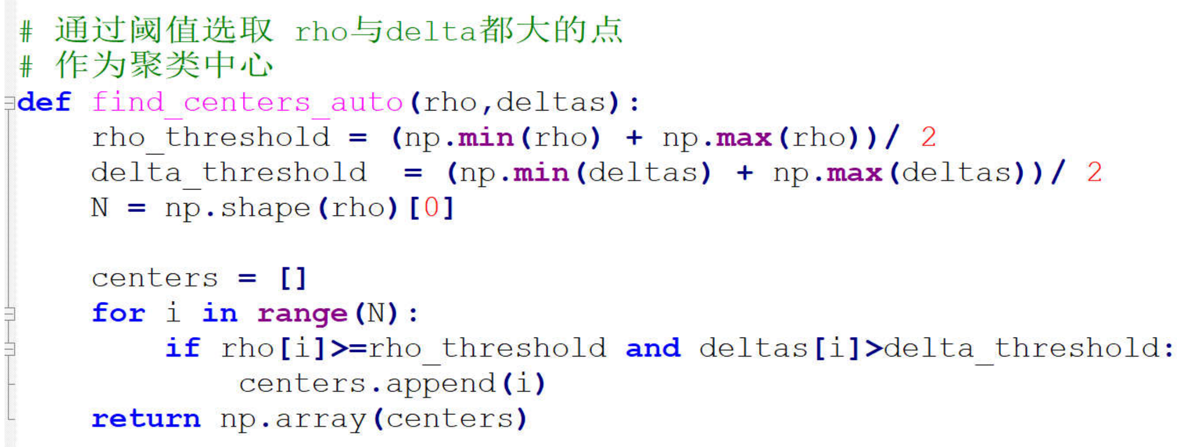
ρ_i δ_i

估算聚类中心

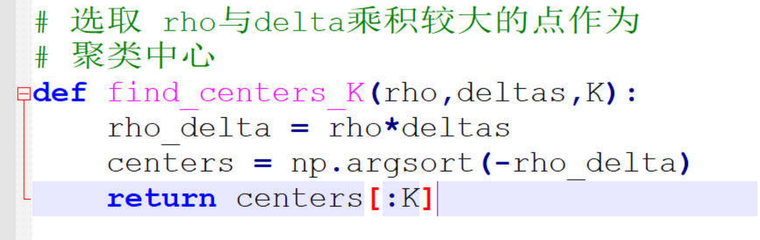
ρ_i j

聚类

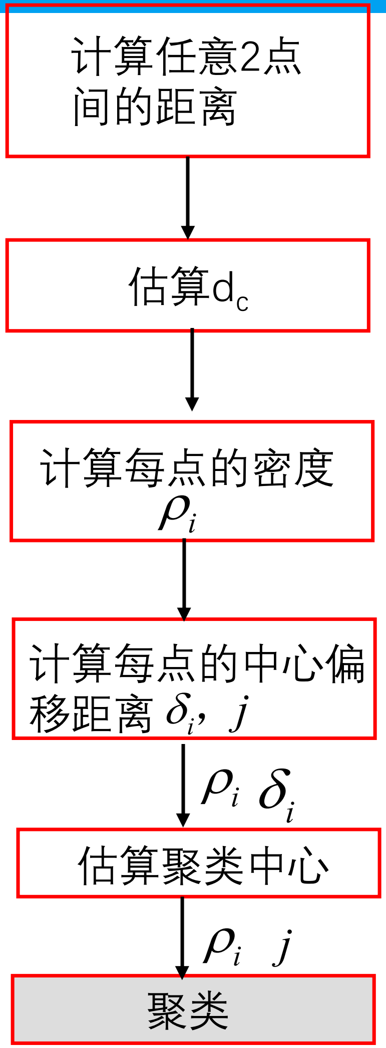




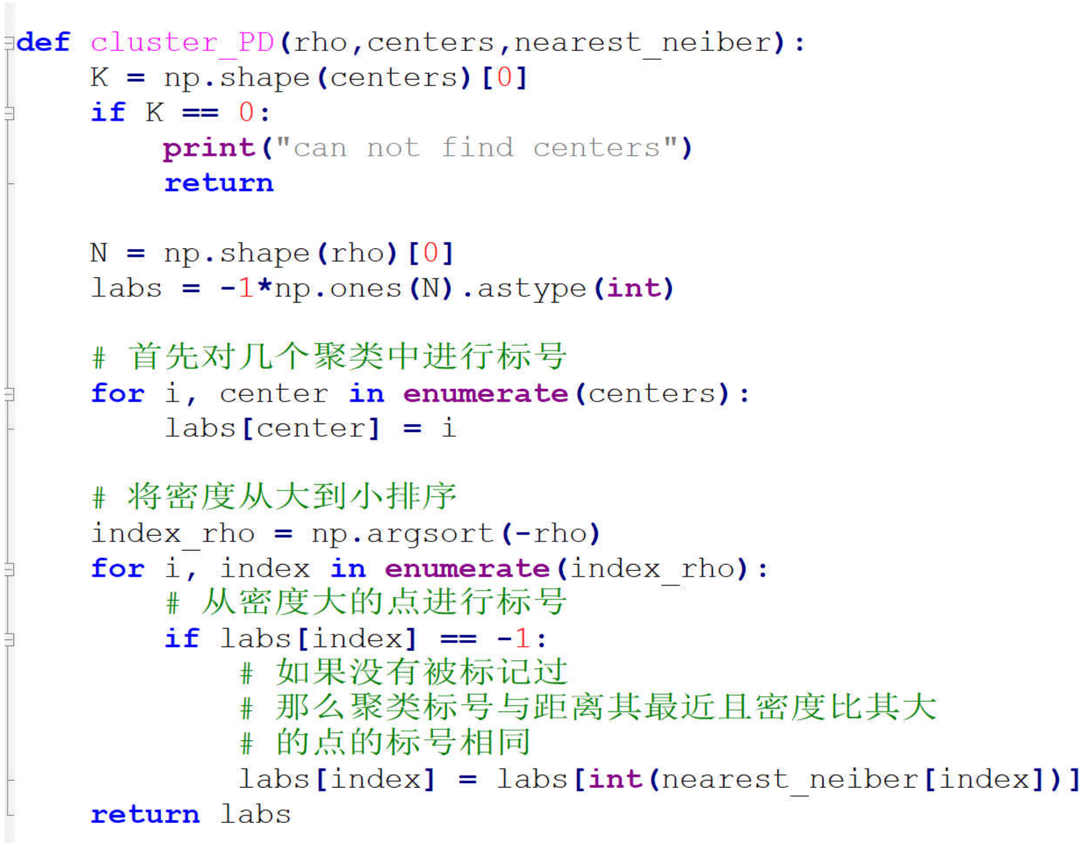
```
# 通过阈值选取 rho与delta都大的点
# 作为聚类中心
def find_centers_auto(rho,deltas):
    rho_threshold = (np.min(rho) + np.max(rho))/ 2
    delta_threshold  = (np.min(deltas) + np.max(deltas))/ 2
    N = np.shape(rho)[0]

    centers = []
    for i in range(N):
        if rho[i]>=rho_threshold and deltas[i]>delta_threshold:
            centers.append(i)
    return np.array(centers)
```



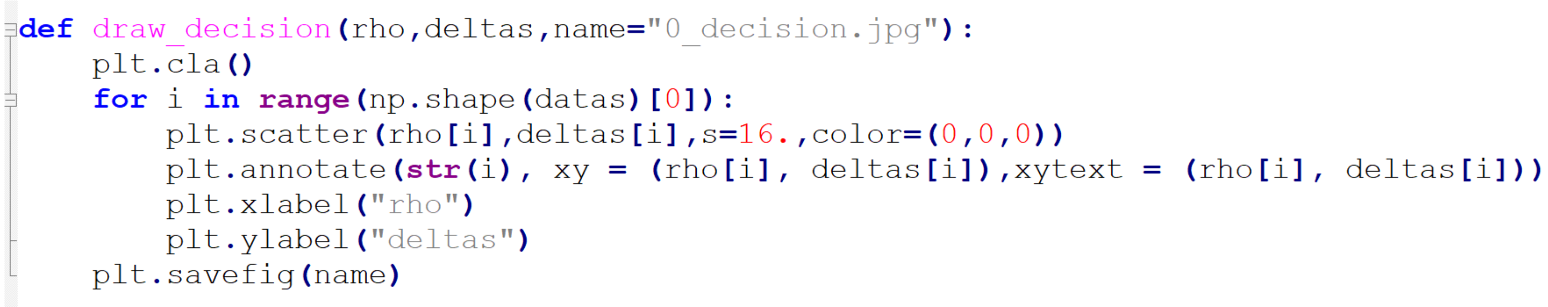
```
# 选取 rho与delta乘积较大的点作为
# 聚类中心
def find_centers_K(rho,deltas,K):
    rho_delta = rho*deltas
    centers = np.argsort(-rho_delta)
    return centers[:K]
```



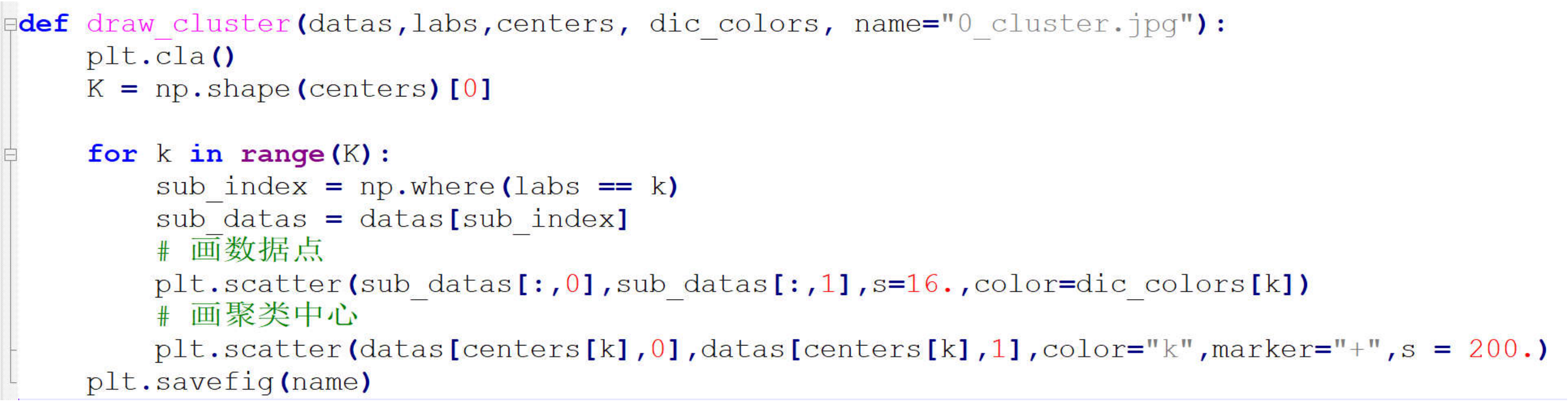
$$\delta_i = \min_{j:\rho_j>\rho_i}(d_{ij})$$



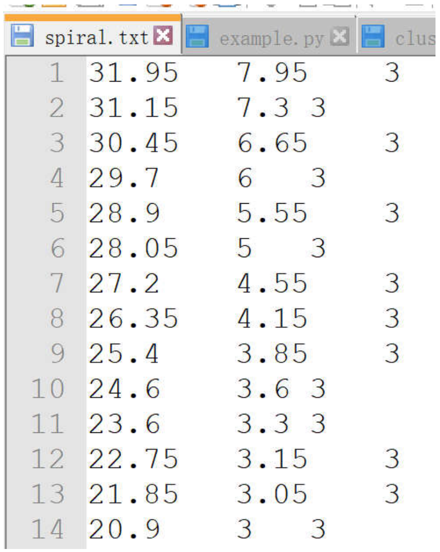
```python
def cluster_PD(rho,centers,nearest_neiber):
    K = np.shape(centers)[0]
    if K == 0:
        print("can not find centers")
        return

    N = np.shape(rho)[0]
    labs = -1*np.ones(N).astype(int)

    # 首先对几个聚类中进行标号
    for i, center in enumerate(centers):
        labs[center] = i

    # 将密度从大到小排序
    index_rho = np.argsort(-rho)
    for i, index in enumerate(index_rho):
        # 从密度大的点进行标号
        if labs[index] == -1:
            # 如果没有被标记过
            # 那么聚类标号与距离其最近且密度比其大
            # 的点的标号相同
            labs[index] = labs[int(nearest_neiber[index])]
    return labs
```



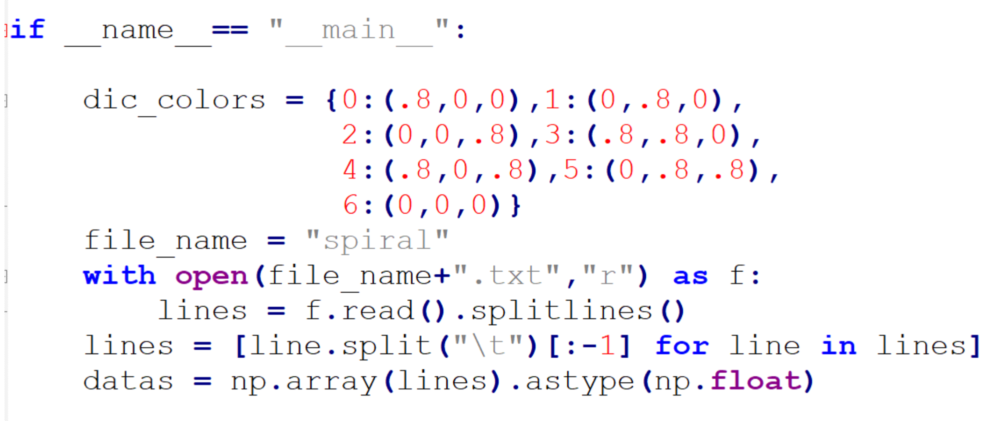
```python
def draw_decision(rho,deltas,name="0_decision.jpg"):
    plt.cla()
    for i in range(np.shape(datas)[0]):
        plt.scatter(rho[i],deltas[i],s=16.,color=(0,0,0))
        plt.annotate(str(i), xy = (rho[i], deltas[i]),xytext = (rho[i], deltas[i]))
        plt.xlabel("rho")
        plt.ylabel("deltas")
    plt.savefig(name)
```



```python
def draw_cluster(datas,labs,centers, dic_colors, name="0_cluster.jpg"):
    plt.cla()
    K = np.shape(centers)[0]

    for k in range(K):
        sub_index = np.where(labs == k)
        sub_datas = datas[sub_index]
        # 画数据点
        plt.scatter(sub_datas[:,0],sub_datas[:,1],s=16.,color=dic_colors[k])
        # 画聚类中心
        plt.scatter(datas[centers[k],0],datas[centers[k],1],color="k",marker="+",s = 200.)
    plt.savefig(name)
```

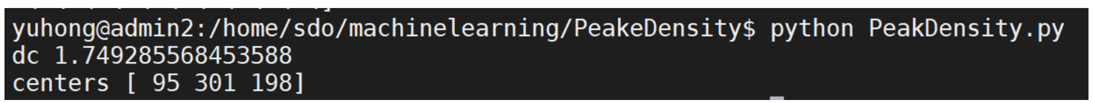


```
31.95    7.95    3
31.15    7.3  3
30.45    6.65    3
29.7     6    3
28.9     5.55    3
28.05    5    3
27.2     4.55    3
26.35    4.15    3
25.4     3.85    3
24.6     3.6  3
23.6     3.3  3
22.75    3.15    3
21.85    3.05    3
20.9     3    3
```

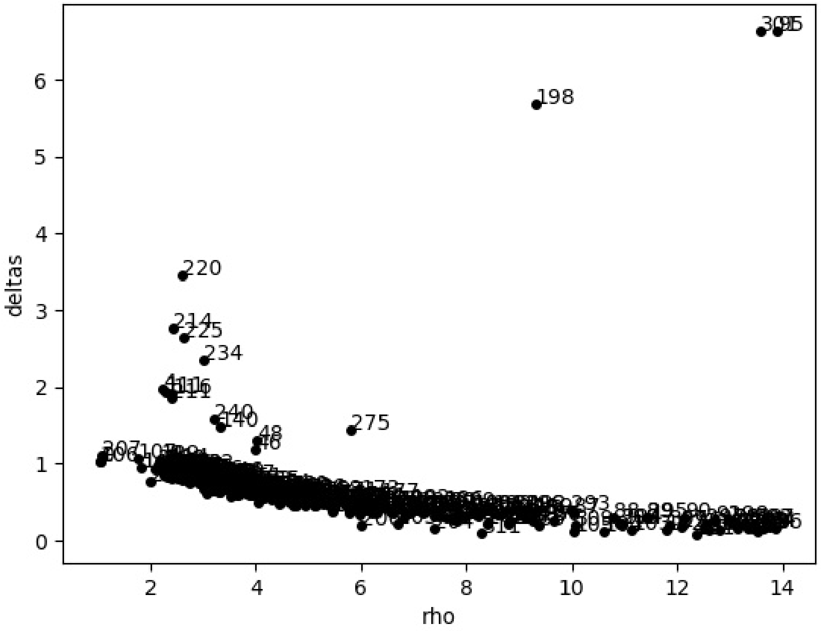


```python
if __name__ == "__main__":

    dic_colors = {0:(.8,0,0),1:(0,.8,0),
                  2:(0,0,.8),3:(.8,.8,0),
                  4:(.8,0,.8),5:(0,.8,.8),
                  6:(0,0,0)}
    file_name = "spiral"
    with open(file_name+".txt","r") as f:
        lines = f.read().splitlines()
    lines = [line.split("\t")[:-1] for line in lines]
    datas = np.array(lines).astype(np.float)
```

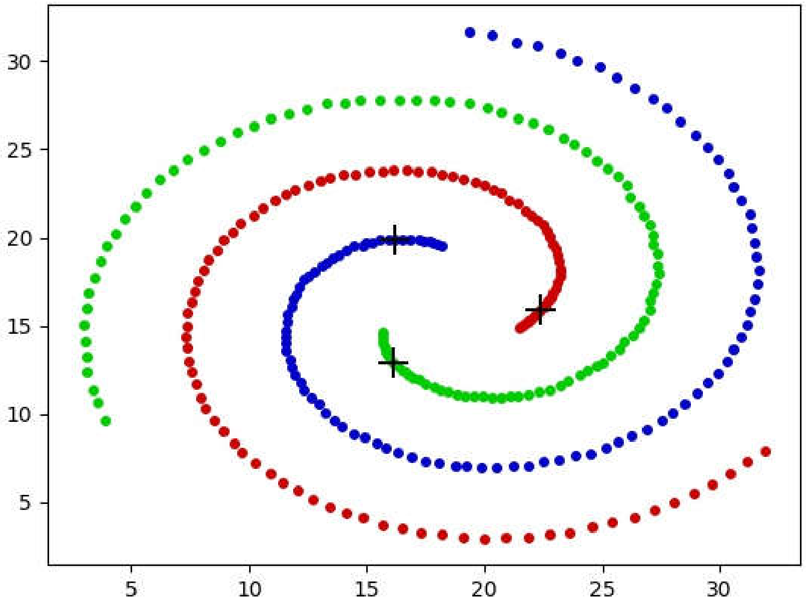
```
# 计算距离矩阵
dists = getDistanceMatrix(datas)
# 计算dc
dc = select_dc(dists)
print("dc",dc)
# 计算局部密度
rho = get_density(dists,dc,method="Gaussion")
# 计算密度距离
deltas, nearest_neiber= get_deltas(dists,rho)

# 绘制密度/距离分布图
draw_decision(rho,deltas,name=file_name+"_decision.jpg")

# 获取聚类中心点
centers = find_centers_K(rho,deltas,3)
# centers = find_centers_auto(rho,deltas)
print("centers",centers)

labs = cluster_PD(rho,centers,nearest_neiber)
draw_cluster(datas,labs,centers, dic_colors, name=file_name+"_cluster.jpg")
```



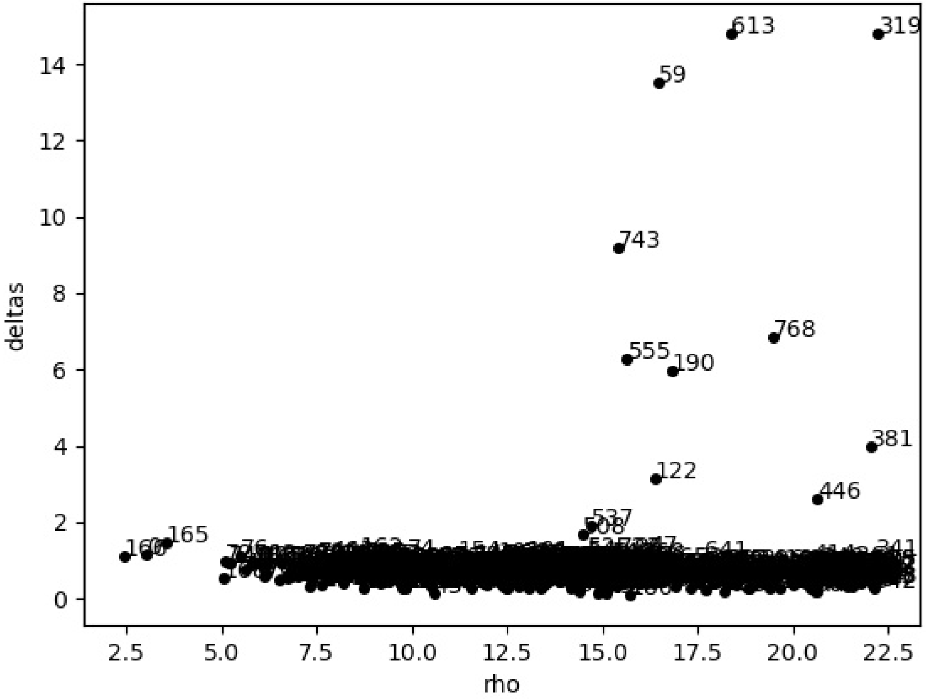
```
yuhong@admin2:/home/sdo/machinelearning/PeakeDensity$ python PeakDensity.py
dc 1.749285568453588
centers [ 95 301 198]
```

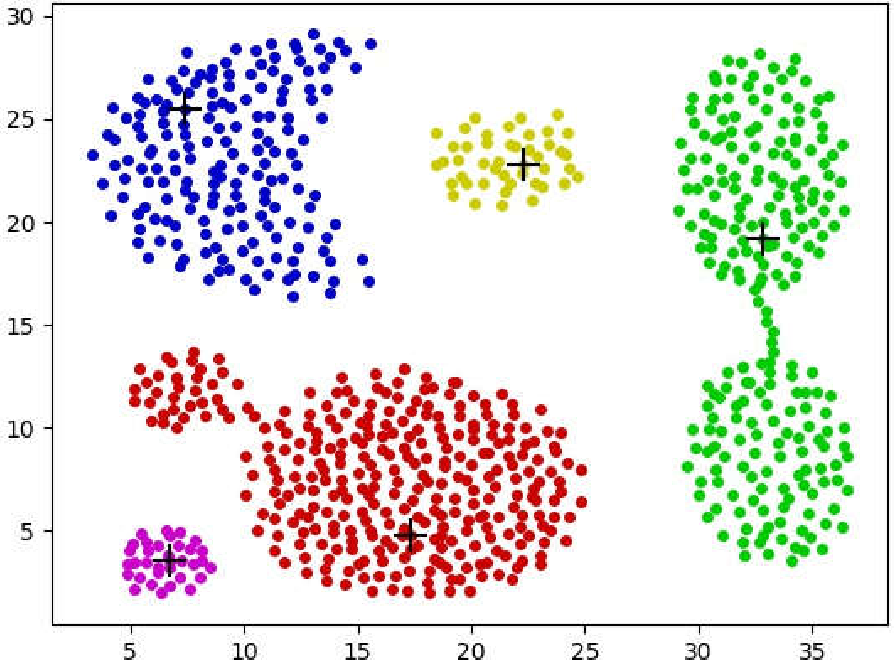


deltas
rho
0
1
2
3
4
5
6
2
4
6
8
10
12
14
198
220
214
225
234
240
140
48
46
275
207
206



5
10
15
20
25
30
5
10
15
20
25
30


613
319
59
743
768
555
190
381
122
446
537
508
165
160
deltas
rho
0
2
4
6
8
10
12
14
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5


30
25
20
15
10
5
5
10
15
20
25
30
35