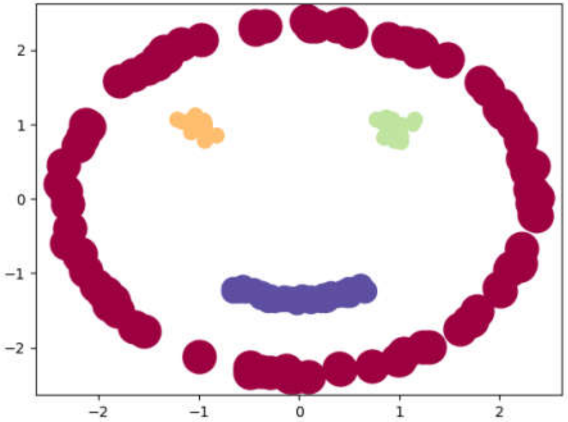
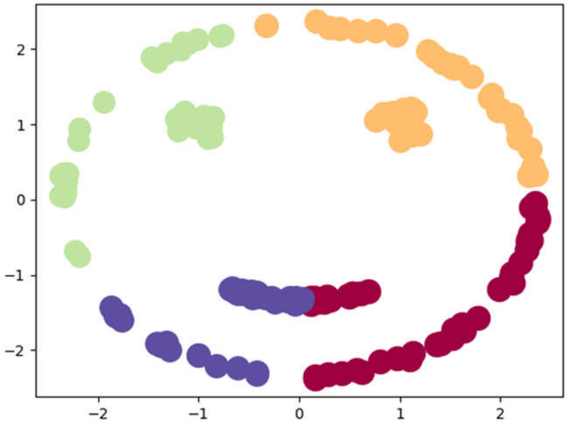
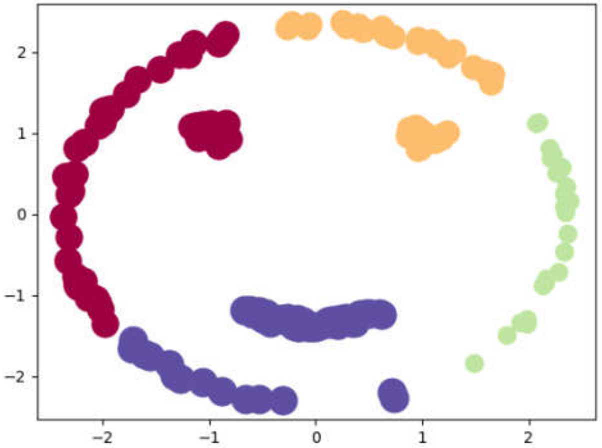
Python编程与人工智能实践

算法篇： AGENS

（Agglomerative Nesting （凝聚层次聚类））





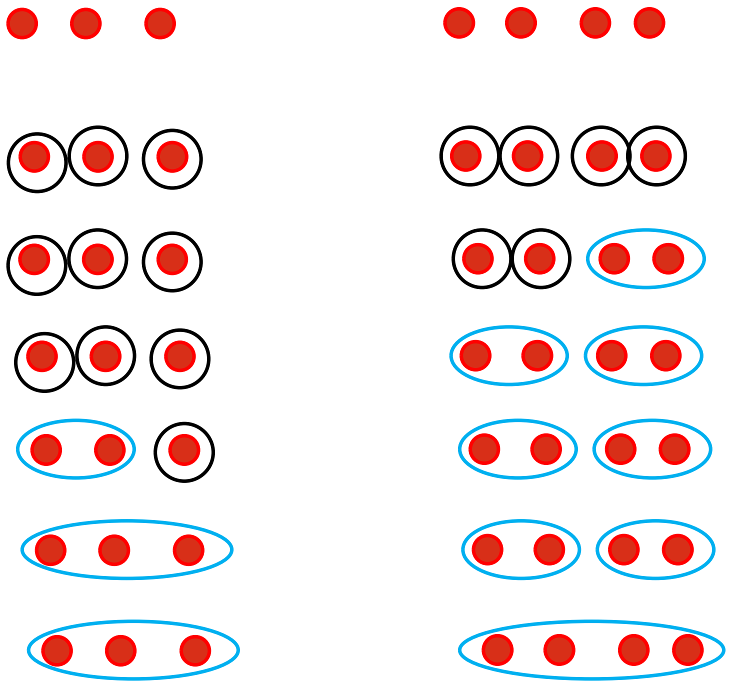


于泓
鲁东大学
信息与电气工程学院
2022.6.27

AGENS

- AGENS是一种自底向上的聚类策略，首先将每个对象作为一个簇，然后将距离较近的簇进行合并，通过不断的迭代，合并越来越大的簇，直到某个终结条件被满足。

AGNES（自底向上凝聚算法）
算法的具体步骤：
输入：包含N个样本的数据集，最终聚类的数目K
（1）将每个样本作为一个**簇**
（2）计算所有**簇，两两之间的距离，将距离最近的两个簇进行合并**
生成新的簇
（3）重复步骤（2）直到最终聚类数目达到K



主要问题：
如何计算两个簇 之间的距离

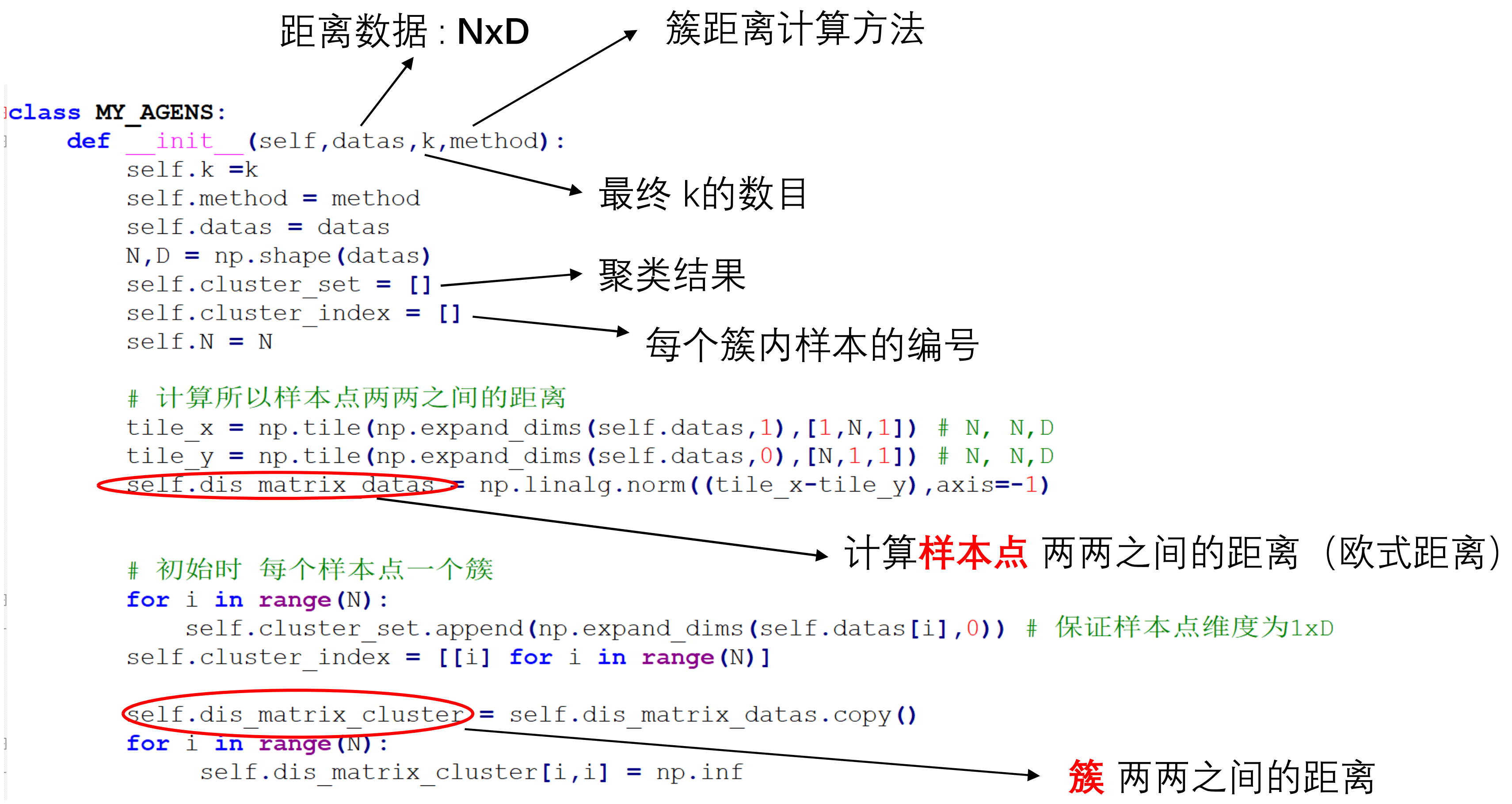
假设 簇 A [2,5,7]
簇 B [3,8]

single-linkage $D_{AB}=\min(d_{23},d_{53},d_{73},d_{28},d_{58},d_{78})$

complete-linkage $D_{AB}=\max(d_{23},d_{53},d_{73},d_{28},d_{58},d_{78})$

average-linkage $D_{AB}=\mathrm{mean}(d_{23},d_{53},d_{73},d_{28},d_{58},d_{78})$

d_{ij} 欧式距离、L1、， ……



```
class MY_AGENS:
    def __init__(self,datas,k,method):
        self.k =k
        self.method = method
        self.datas = datas
        N,D = np.shape(datas)
        self.cluster_set = []
        self.cluster_index = []
        self.N = N

        # 计算所以样本点两两之间的距离
        tile_x = np.tile(np.expand_dims(self.datas,1),[1,N,1]) # N, N,D
        tile_y = np.tile(np.expand_dims(self.datas,0),[N,1,1]) # N, N,D
        self.dis_matrix_datas = np.linalg.norm((tile_x-tile_y),axis=-1)

        # 初始时 每个样本点一个簇
        for i in range(N):
            self.cluster_set.append(np.expand_dims(self.datas[i],0)) # 保证样本点维度为1xD
        self.cluster_index = [[i] for i in range(N)]

        self.dis_matrix_cluster = self.dis_matrix_datas.copy()
        for i in range(N):
             self.dis_matrix_cluster[i,i] = np.inf
```

距离数据：NxD

簇距离计算方法

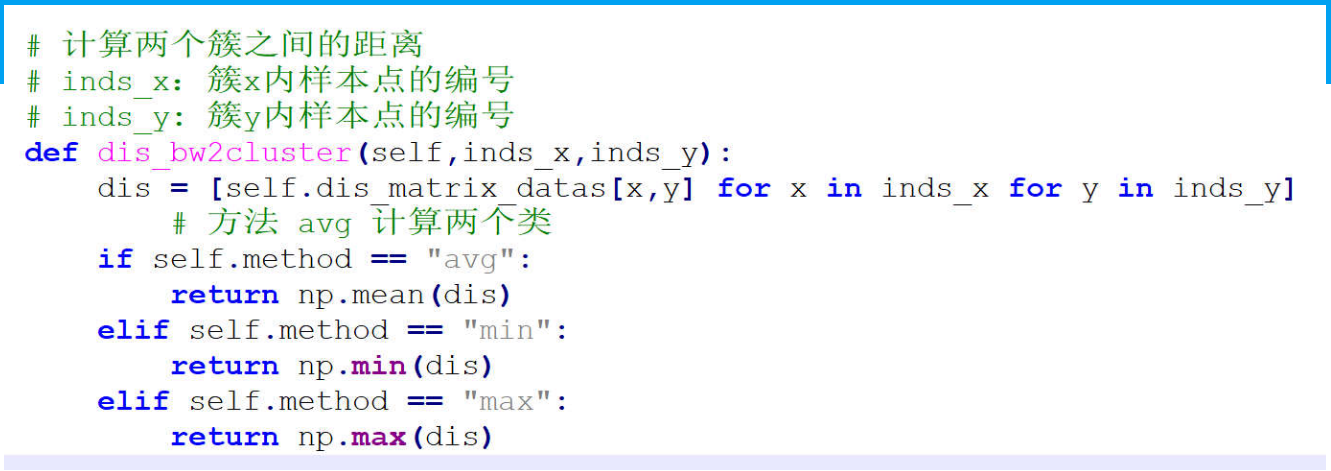
最终 k的数目

聚类结果

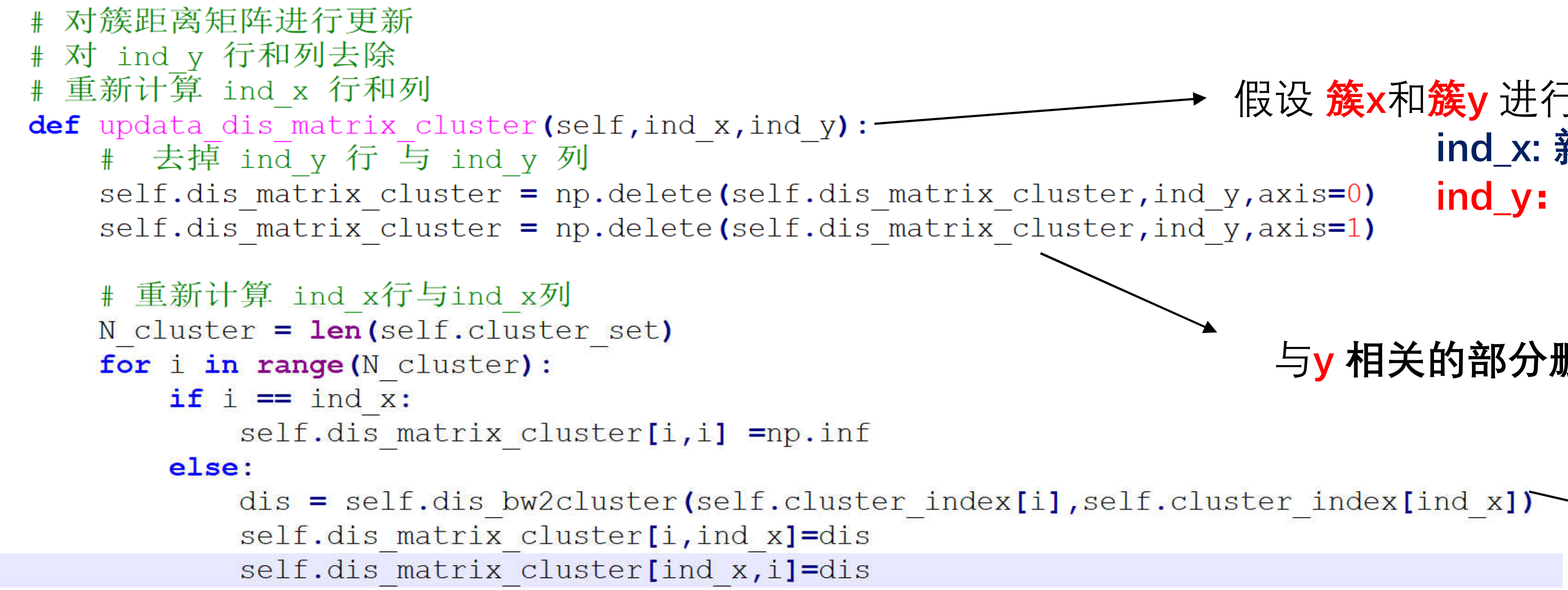
每个簇内样本的编号

计算**样本点** 两两之间的距离（欧式距离）

簇 两两之间的距离



```
# 计算两个簇之间的距离
# inds_x: 簇x内样本点的编号
# inds_y: 簇y内样本点的编号
def dis_bw2cluster(self,inds_x,inds_y):
    dis = [self.dis_matrix_datas[x,y] for x in inds_x for y in inds_y]
        # 方法 avg 计算两个类
    if self.method == "avg":
        return np.mean(dis)
    elif self.method == "min":
        return np.min(dis)
    elif self.method == "max":
        return np.max(dis)
```



```
# 对簇距离矩阵进行更新
# 对 ind_y 行和列去除
# 重新计算 ind_x 行和列
def updata_dis_matrix_cluster(self,ind_x,ind_y):
    #  去掉 ind_y 行 与 ind_y 列
    self.dis_matrix_cluster = np.delete(self.dis_matrix_cluster,ind_y,axis=0)
    self.dis_matrix_cluster = np.delete(self.dis_matrix_cluster,ind_y,axis=1)

    # 重新计算 ind_x行与ind_x列
    N_cluster = len(self.cluster_set)
    for i in range(N_cluster):
        if i == ind_x:
            self.dis_matrix_cluster[i,i] =np.inf
        else:
            dis = self.dis_bw2cluster(self.cluster_index[i],self.cluster_index[ind_x])
            self.dis_matrix_cluster[i,ind_x]=dis
            self.dis_matrix_cluster[ind_x,i]=dis
```

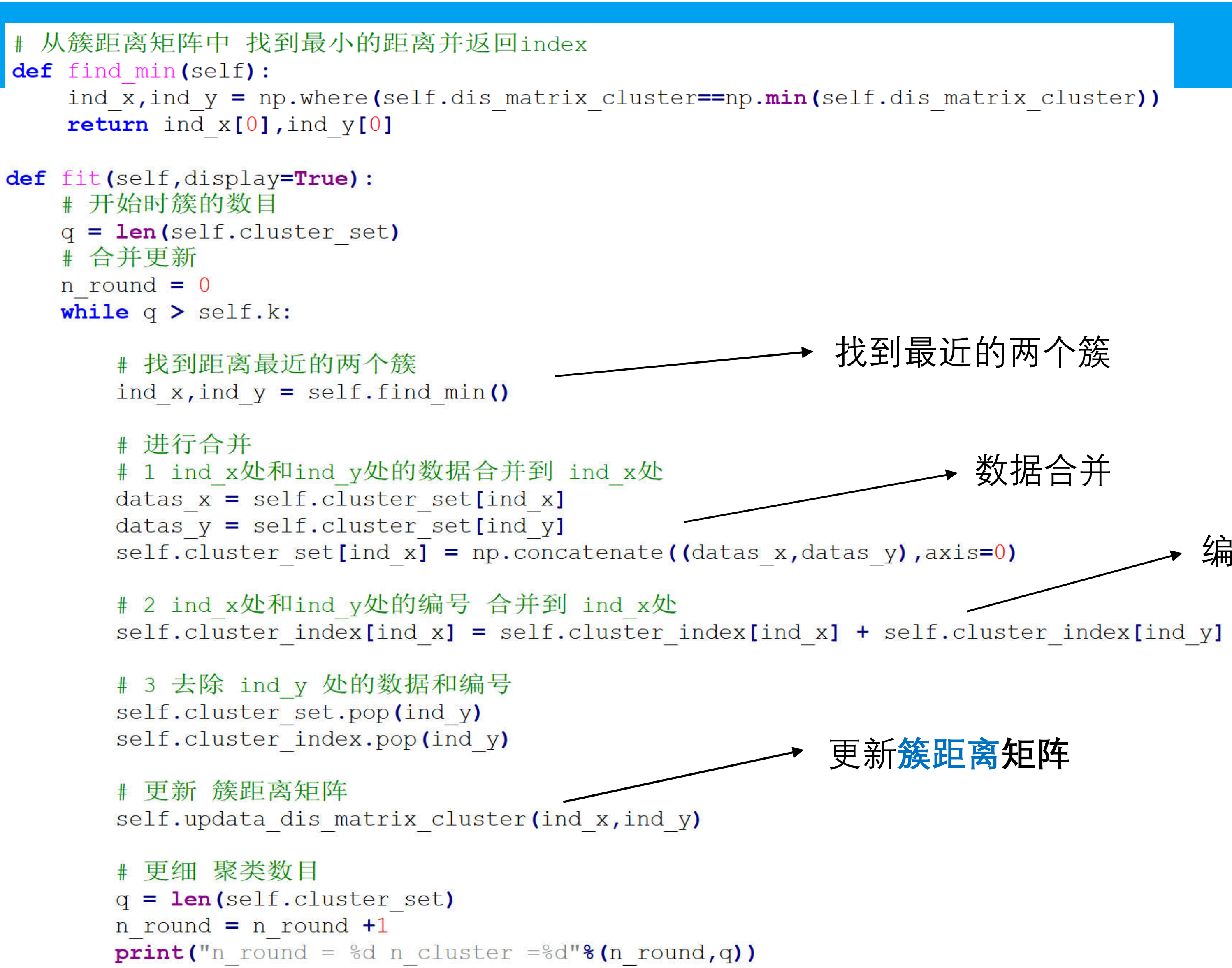
假设 簇x和簇y 进行合并生成新簇x

ind_x: 新簇x的编号

ind_y： 簇y的编号

与y 相关的部分删除

与x相关的部分
进行更新



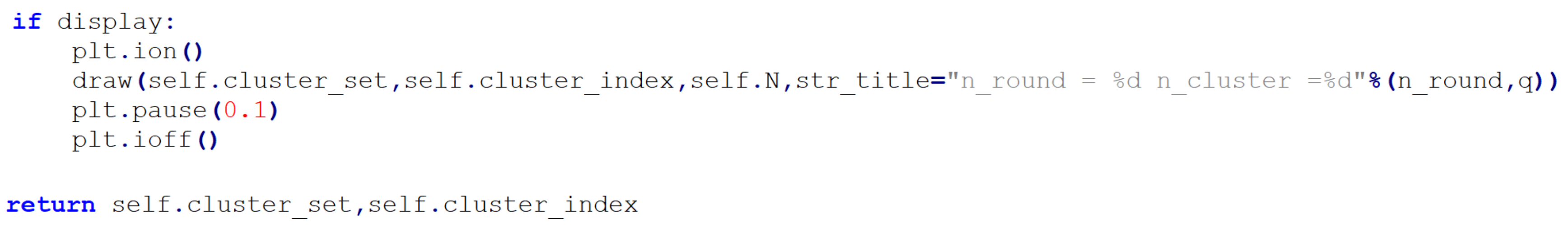
```python
# 从簇距离矩阵中 找到最小的距离并返回index
def find_min(self):
    ind_x,ind_y = np.where(self.dis_matrix_cluster==np.min(self.dis_matrix_cluster))
    return ind_x[0],ind_y[0]

def fit(self,display=True):
    # 开始时簇的数目
    q = len(self.cluster_set)
    # 合并更新
    n_round = 0
    while q > self.k:

        # 找到距离最近的两个簇
        ind_x,ind_y = self.find_min()

        # 进行合并
        # 1 ind_x处和ind_y处的数据合并到 ind_x处
        datas_x = self.cluster_set[ind_x]
        datas_y = self.cluster_set[ind_y]
        self.cluster_set[ind_x] = np.concatenate((datas_x,datas_y),axis=0)

        # 2 ind_x处和ind_y处的编号 合并到 ind_x处
        self.cluster_index[ind_x] = self.cluster_index[ind_x] + self.cluster_index[ind_y]

        # 3 去除 ind_y 处的数据和编号
        self.cluster_set.pop(ind_y)
        self.cluster_index.pop(ind_y)

        # 更新 簇距离矩阵
        self.updata_dis_matrix_cluster(ind_x,ind_y)

        # 更细 聚类数目
        q = len(self.cluster_set)
        n_round = n_round +1
        print("n_round = %d n_cluster =%d"%(n_round,q))
```

找到最近的两个簇

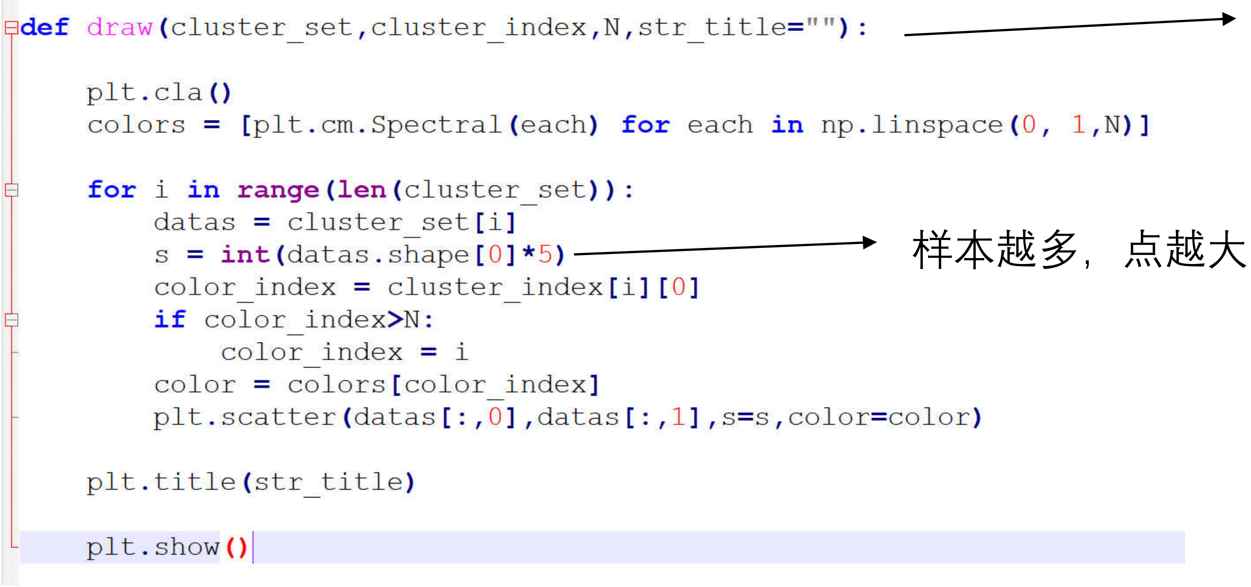
数据合并

编号合并

更新簇距离矩阵



```
if display:
    plt.ion()
    draw(self.cluster_set,self.cluster_index,self.N,str_title="n_round = %d n_cluster =%d"%(n_round,q))
    plt.pause(0.1)
    plt.ioff()

return self.cluster_set,self.cluster_index
```

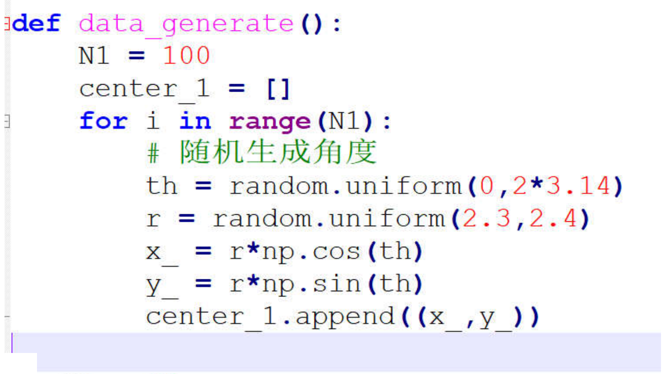


```
def draw(cluster_set,cluster_index,N,str_title=""):

    plt.cla()
    colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1,N)]

    for i in range(len(cluster_set)):
        datas = cluster_set[i]
        s = int(datas.shape[0]*5)
        color_index = cluster_index[i][0]
        if color_index>N:
            color_index = i
        color = colors[color_index]
        plt.scatter(datas[:,0],datas[:,1],s=s,color=color)

    plt.title(str_title)

    plt.show()
```

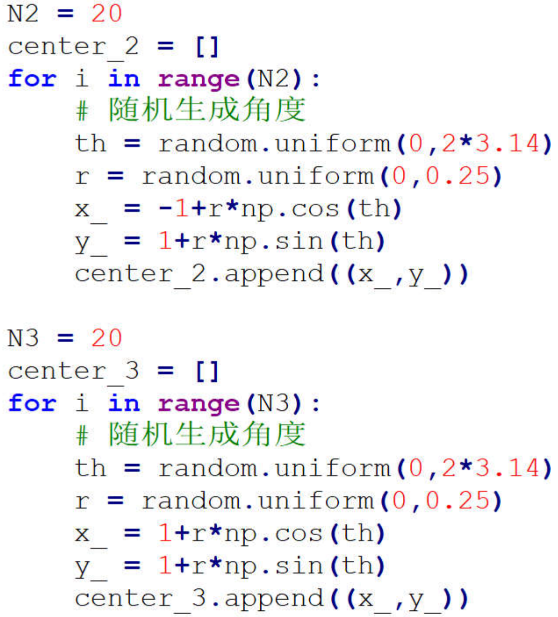
绘图部分

样本越多，点越大

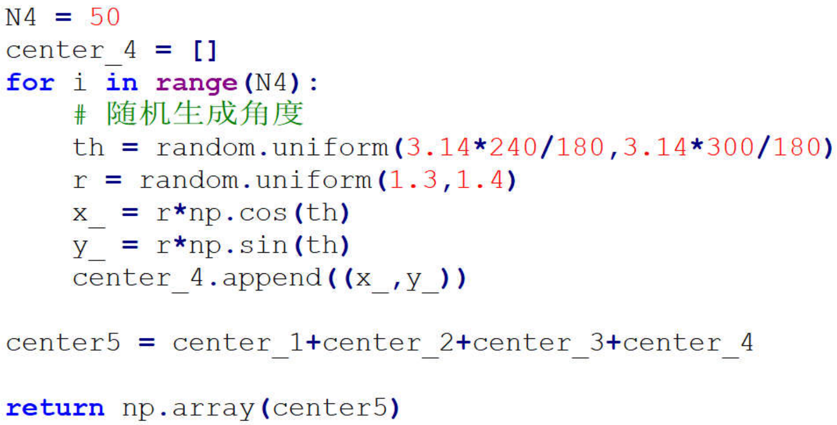
数据生成部分



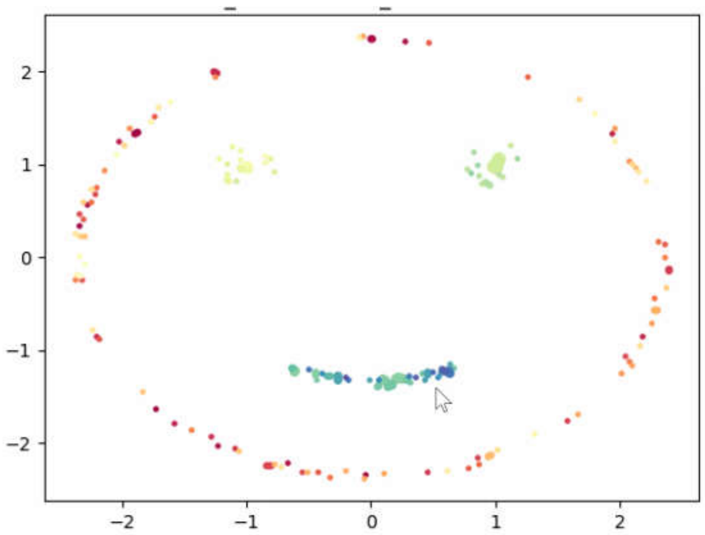
```
def data_generate():
    N1 = 100
    center_1 = []
    for i in range(N1):
        # 随机生成角度
        th = random.uniform(0,2*3.14)
        r = random.uniform(2.3,2.4)
        x_ = r*np.cos(th)
        y_ = r*np.sin(th)
        center_1.append((x_,y_))
```

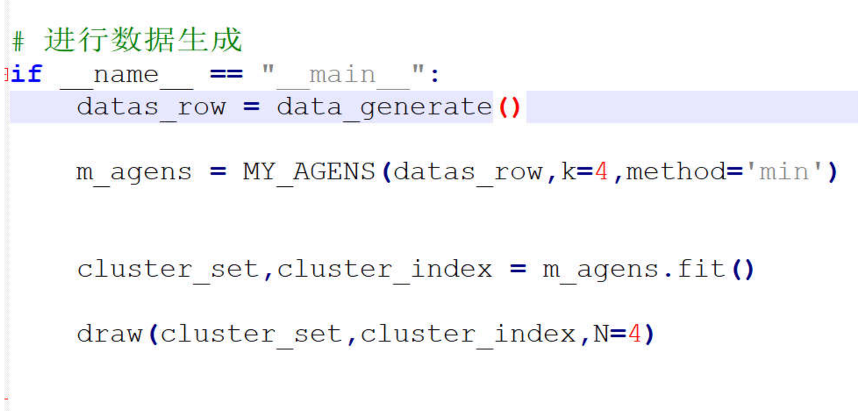


```
    N2 = 20
    center_2 = []
    for i in range(N2):
        # 随机生成角度
        th = random.uniform(0,2*3.14)
        r = random.uniform(0,0.25)
        x_ = -1+r*np.cos(th)
        y_ = 1+r*np.sin(th)
        center_2.append((x_,y_))

    N3 = 20
    center_3 = []
    for i in range(N3):
        # 随机生成角度
        th = random.uniform(0,2*3.14)
        r = random.uniform(0,0.25)
        x_ = 1+r*np.cos(th)
        y_ = 1+r*np.sin(th)
        center_3.append((x_,y_))
```



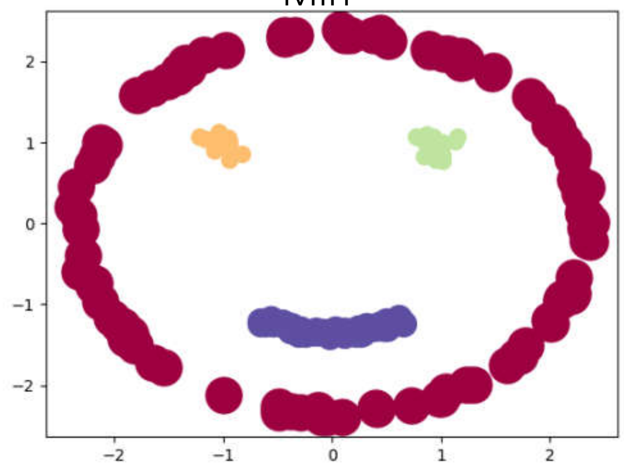
```
    N4 = 50
    center_4 = []
    for i in range(N4):
        # 随机生成角度
        th = random.uniform(3.14*240/180,3.14*300/180)
        r = random.uniform(1.3,1.4)
        x_ = r*np.cos(th)
        y_ = r*np.sin(th)
        center_4.append((x_,y_))

    center5 = center_1+center_2+center_3+center_4

    return np.array(center5)
```



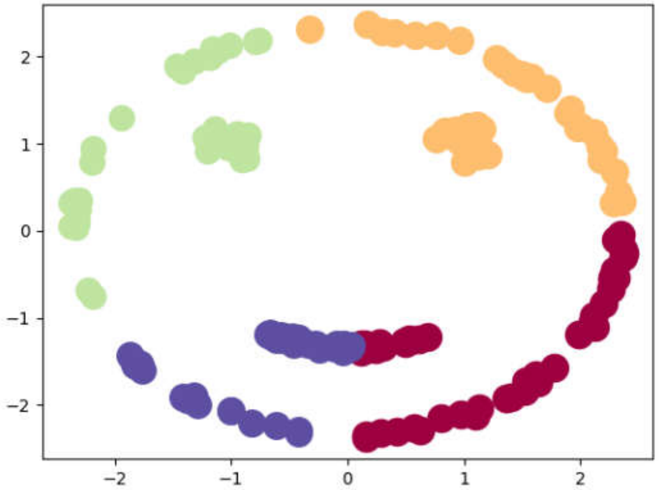


```
# 进行数据生成
if __name__ == "__main__":
    datas_row = data_generate()

    m_agens = MY_AGENS(datas_row,k=4,method='min')


    cluster_set,cluster_index = m_agens.fit()

    draw(cluster_set,cluster_index,N=4)
```

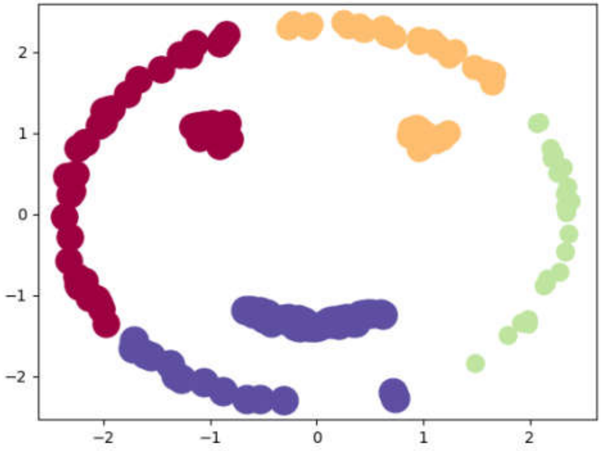
Min



Max



Avg


2
1
0
−1
−2
−2
−1
0
1
2