DPC 实验报告

王昊然 PB20010382

2023 年 12 月 10 日

1 简介

基于论文"Clustering by fast search and find of density peaks”，具体数据集请见https://github.com/USTC-MLI-F23/Assignments/tree/main/assignment3

2 理论背景

2.1 DPC 的基本理论

DPC（Density Peak Clustering）算法是一种基于密度的聚类方法，其核心思想是通过识别数据点的局部密度峰值来确定聚类中心。算法的主要步骤可以总结如下：

1. **计算局部密度** ρ：对于每个数据点 i，其局部密度 ρ_i 通过以下公式计算：

$$\rho_i = \sum_j \chi(d_{ij} - d_c)$$

其中，d_{ij} 是点 i 和点 j 之间的距离，d_c 是一个预设的截断距离，$\chi(x)$ 是一个决策函数，当 $x < 0$ 时 $\chi(x) = 1$，否则 $\chi(x) = 0$。

2. **计算相对距离** δ：对于每个点 i，计算其到具有更高密度的最近点的最小距离 δ_i：

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

对于具有最高密度的点，δ 被定义为所有点对距离的最大值。

3. **选择聚类中心**：在 $\rho - \delta$ 图中，那些既具有较高局部密度又具有较大相对距离的点被选择为聚类中心。

4. **分配剩余点到聚类**：对于不是聚类中心的每个点 i，将其分配给距离最近的、密度更高的聚类中心。

DPC 算法的优点在于它不需要预先指定聚类数量，并且能够有效地处理具有复杂结构的数据集。然而，该算法的性能在一定程度上依赖于截断距离 d_c 的选择，且在决策图的解读上需要一定的经验和直觉。

2.1.1 注意事项

需考虑以下几个关键的注意事项：

- **截断距离 d_c 的选择**：截断距离 d_c 对于确定局部密度 ρ 至关重要。d_c 的选择需要根据数据集的特性和分布来调整，一个不恰当的 d_c 值可能导致误导性的聚类结果。
- **决策图的解读**：在 $\rho - \delta$ 图中识别聚类中心需要主观判断。这可能导致不同的分析者得出不同的聚类结果。因此，对决策图的解读需要谨慎，并考虑结合其他信息或方法来验证聚类的合理性。
- **数据规模和复杂性**：虽然 DPC 算法适用于各种类型的数据集，但在处理非常大规模或高度复杂的数据时，可能会面临计算效率和准确性的挑战。
- **离群点的处理**：DPC 算法可能对离群点敏感。在某些情况下，离群点可能会被错误地识别为聚类中心，或者影响到其他聚类中心的确定。
- **参数的敏感性**：除了截断距离 d_c，其他参数的选择（如邻域大小）也会影响算法的输出。合适的参数调整对于获得可靠的聚类结果是必要的。

3 实验结果

3.0.1 R15 数据集

聚类结果

3.0.2 D31 数据集

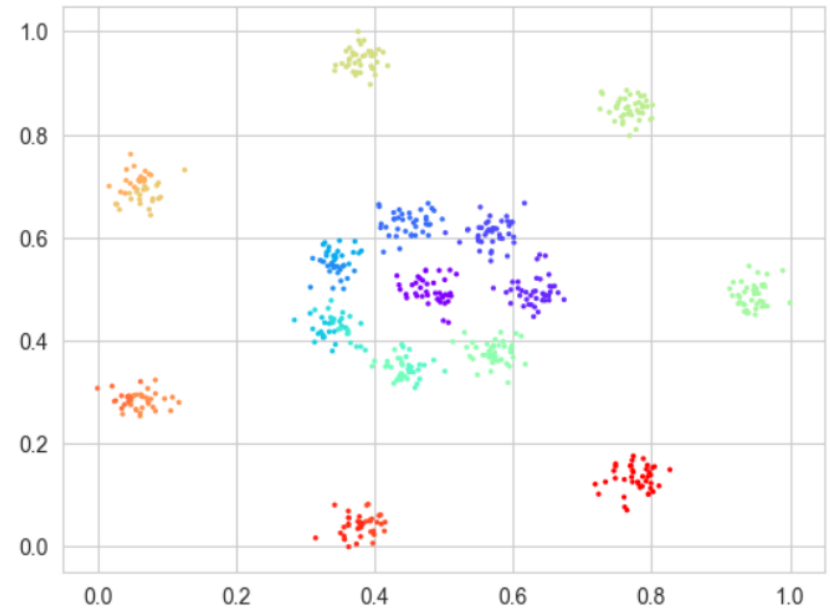
聚类结果

3.0.3 Aggregation 数据集

聚类结果

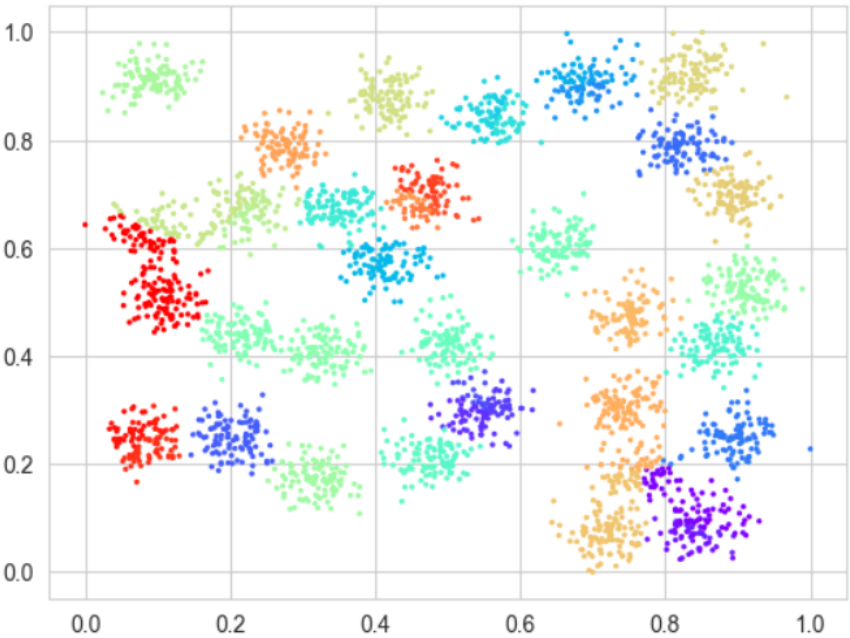
4 代码

```
# 2. 计算距离矩阵
@jit(nopython=True)
def Dist(df):
    dist=np.zeros((len(df),len(df)))
    for i in range(len(df)):
```




```
聚类的DBI index为=0.8206765309899083
```

图 1: R15




```
聚类的DBI index为=0.955500991703965
```

图 2: D31

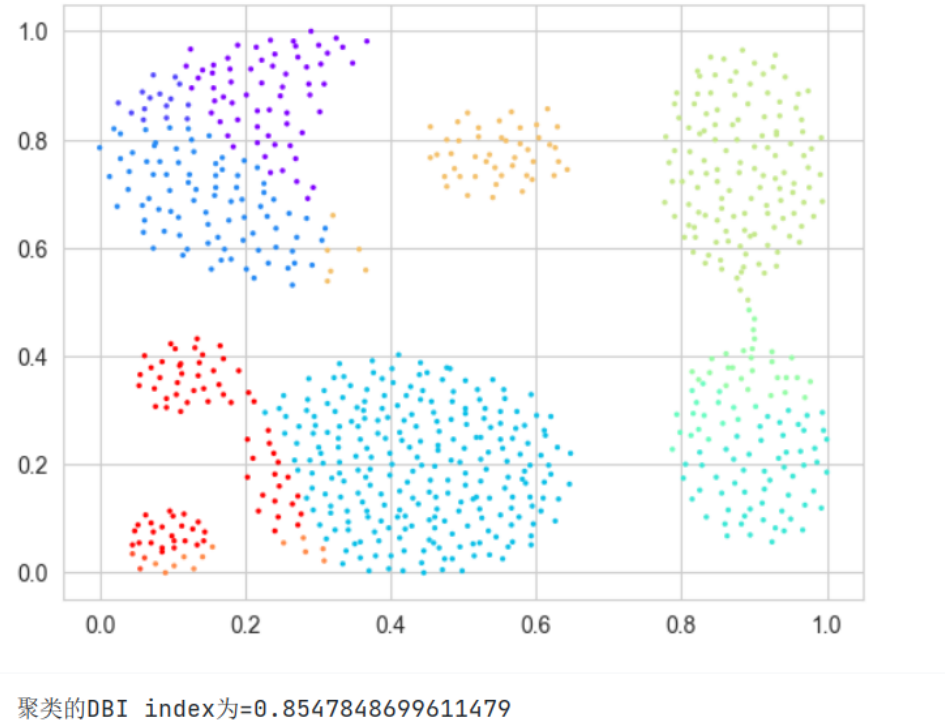


图 3: Aggregation

```
        for j in range(len(df)):
            dist[i,j]=np.sqrt(np.sum(np.square(df[i,:]-df[j,:])))
    return dist
# 3. 计算密度矩阵
@jit(nopython=True)
def Rho(dist,intercept=0.02):
    rho=np.zeros(len(dist))
    for i in range(len(dist)):
        for j in range(len(dist)):
            if dist[i,j]<intercept:
                rho[i]+=1
    return rho
# 4. 计算相对距离矩阵
def get_real_dist(dist,rho):
    rel_dist=np.zeros((len(dist),2))
    for i in range(len(rho)):
        #if 集合不空
        if dist[rho>rho[i],i].any():
            rel_dist[i,0]=np.min(dist[rho>rho[i],i])
            rel_dist[i,1]=np.where(dist[i,:]==rel_dist[i,0])[0][0]
```

```
        else:
            rel_dist[i,0]=np.max(dist[i,:])
            rel_dist[i,1]=np.where(dist[i,:]==rel_dist[i,0])[0][0]
    return rel_dist
# 5. 计算DBI
def get_info_mat(rho,dist,rel_dist):
    info=np.zeros((len(dist),2))
    info[:,0]=rho.copy()
    info[:,1]=rel_dist[:,0].copy()
    #rename np.array columns
    info_pd=pd.DataFrame(info,columns=['rho','rel_dist'])
    return info_pd
# 6. 为了选取中心点, 画出rho和rel_dist的散点图
def plot_info(info):
    info=np.array(info)
    plt.scatter(info[:,0],info[:,1],s=2)
    plt.xlabel('rho')
    plt.ylabel('rel_dist')
    plt.show()
# 7. 选取中心点
def get_center_index(info_pd,rho_threshold,rel_dist_threshold):
    center_index=np.where((info_pd['rho']>=rho_threshold)&(info_pd['rel_dist']>=rel_d
    return center_index
# 8. 聚类非中心点
def get_cluster_index(df,center_index,dist):
    center_dist=np.zeros((df.shape[0],len(center_index)))
    for i in range(df.shape[0]):
        for j in range(len(center_index)):
            center_dist[i,j]=dist[i,center_index[j]]
    cluster_index=np.zeros(df.shape[0])
    for i in range(df.shape[0]):
        cluster_index[i]=center_dist[i,:].argmin()
    return cluster_index
# 9. 画出聚类结果
def plot_cluster(df,cluster_index):
    plt.scatter(df[0],df[1],c=cluster_index,s=2,cmap='rainbow')#cmap='rainbow'表示彩虹
    # 'viridis', 'plasma' 'inferno' 'magma' 'cividis'
```

```
    plt.show()
# 10. 计算DBI
def get_dbi(df,cluster_index):
    dbi=davies_bouldin_score(df,cluster_index)
    print(f'聚类的DBI index为={dbi}')
```