DAM Assignment1-2

> 吴桐欣 1652677

Task 2

数据预处理

1.商家类别列表

在yelp官网上找到了官方提供的商家category list，以此为依据，匹配数据集中商家的类别。category list有1000+项。存在部分类别没更新到category list中，如：Canadian(new)，American(new)，但因数量较少，所以可以忽略不计。

2.用户地理位置

如果应用场景中要根据聚类对用户做推荐的话，地理位置也是一个需要考虑的因素，应尽量向用户推荐住址附近的商店。

但此处我们无法获得用户的住址，因此可根据用户访问过的商店，根据几个商店的经纬度，计算出几个商店的中心，假定为用户的住址，亦用经度和纬度表示。

3.用户兴趣爱好

用一个1000+维度的向量来记录一个用户所访问过的商店的类别。

遍历用户去过的商店以及商店的类别，将相应类别的维度加一。最后对整个向量做一个归一化处理，维度的值不再是用户访问某类商店的次数，而是表示次数。因为不同用户可能具有不同的活跃度，单纯地记录次数并不准确。

4.加权

前面既选取了地理位置又选取了类别维度，但因为经度、维度通常值会比较大，而类别维度因为做过归一化处理，都是小于1的值。因此将类别维度全部乘以10，以强化类别维度的影响。

5.PCA降维

> 参考资料: https://www.cnblogs.com/pinard/p/6243025.html

选取5000个用户，1000多个维度，这样一个矩阵含很多0的元素，毕竟一个用户不大可能访问过所有类别的商店。因此，使用PCA降维，去除对聚类帮助不大的维度。 代码如下：

```python
pca = PCA(n_components=26)
return pca.fit_transform(user_data)
```

原本尝试令`n_components='mle'`让其自动选取合适的维度，但实际跑得太慢了，半天出不了结果，就手动选择维度了。

explained variance ratio：其实排在15以后的维度的方差占比已经极小

```
▼ explained_variance_ratio_ = {ndarray} [9.75757506e-01 1.23
    min = {float64} 2.0381177537024937e-33
    max = {float64} 0.9757575063061418
  ▶ shape = {tuple} <class 'tuple'>: (1309,)
  ▶ dtype = {dtype} float64
    size = {int} 1309
  ▼ array = {NdArrayItemsContainer} <pydevd_plugins.exten
      0000 = {float64} 0.9757575063061418
      0001 = {float64} 0.012343075310291876
      0002 = {float64} 0.004635407574574938
      0003 = {float64} 0.001320610579873417
      0004 = {float64} 0.0009937708520803043
      0005 = {float64} 0.0008034192932446982
      0006 = {float64} 0.0006582966572570598
      0007 = {float64} 0.0005389209891027158
      0008 = {float64} 0.00044825120367695024
      0009 = {float64} 0.0003960428017216099
      0010 = {float64} 0.0002491655528711212
      0011 = {float64} 0.00021101351035851678
      0012 = {float64} 0.00017281010139828763
      0013 = {float64} 0.00013918285498452142
      0014 = {float64} 0.00012136739259069348
      0015 = {float64} 6.072896491848216e-05
      0016 = {float64} 5.6570887162468544e-05
      0017 = {float64} 4.809060086006413e-05
      0018 = {float64} 4.545230145764899e-05
      0019 = {float64} 3.770076075745677e-05
      0020 = {float64} 3.4389686729972115e-05
      0021 = {float64} 3.317908545490775e-05
      0022 = {float64} 2.9735835896980735e-05
```

explained variance：想更多地保留一些维度，所以最终选取了26个

```
▼ array = {NdArrayItemsContainer} <pydevd_plugins.
    0000 = {float64} 410.57253273246675
    0001 = {float64} 5.193634339579576
    0002 = {float64} 1.9504549192198346
    0003 = {float64} 0.5556774373015217
    0004 = {float64} 0.41815206448055575
    0005 = {float64} 0.3380572446963177
    0006 = {float64} 0.2769935400061871
    0007 = {float64} 0.2267634673662109
    0008 = {float64} 0.18861205863609337
    0009 = {float64} 0.16664416632454454
    0010 = {float64} 0.10484216770133155
    0011 = {float64} 0.0887888136435026
    0012 = {float64} 0.07271384596511482
    0013 = {float64} 0.05856440449047551
    0014 = {float64} 0.05106813675022096
    0015 = {float64} 0.025553116195018502
    0016 = {float64} 0.023803508834017672
    0017 = {float64} 0.0202352322868504
    0018 = {float64} 0.019125106809203003
    0019 = {float64} 0.01586346682458761
    0020 = {float64} 0.01447025586721011
    0021 = {float64} 0.013960867388597286
    0022 = {float64} 0.01251204052055729
    0023 = {float64} 0.012263167270083246
    0024 = {float64} 0.011719001477861465
    0025 = {float64} 0.011098659648853497
    0026 = {float64} 0.00891272802717292
    0027 = {float64} 0.00853764130799277
```

用户间距离定义

根据选取的维度，一个用户以一个元素均为数字的向量表示，用户间的距离即为欧式距离。

聚类效果评估

直接查看聚类效果

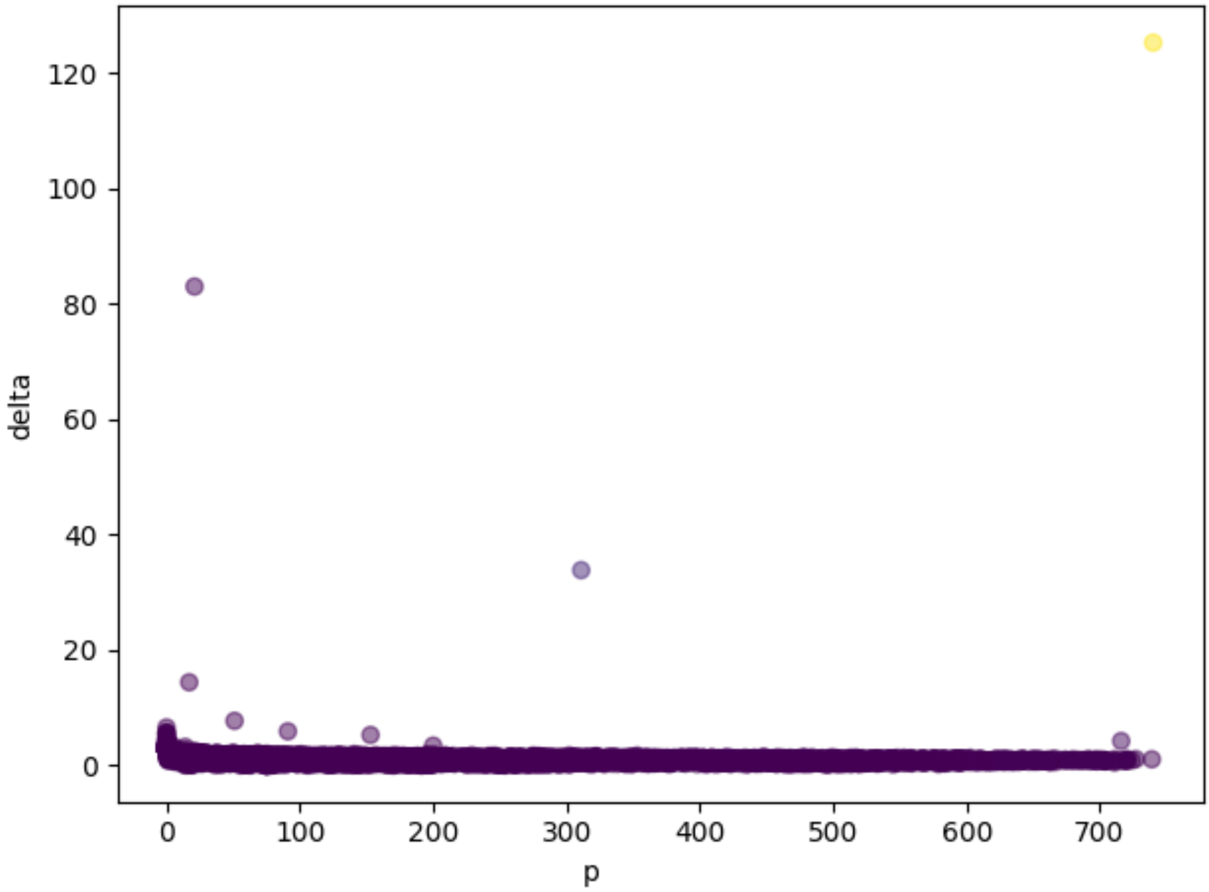
直接用sc系数或ch系数来判断聚类结果不够直观。所以，每次聚类后对聚类结果进行评判，除了使用sc系数和ch系数外，我还将每一个聚类的数据打印输出，直接根据数据来评判聚类的效果。

主要看以下信息：

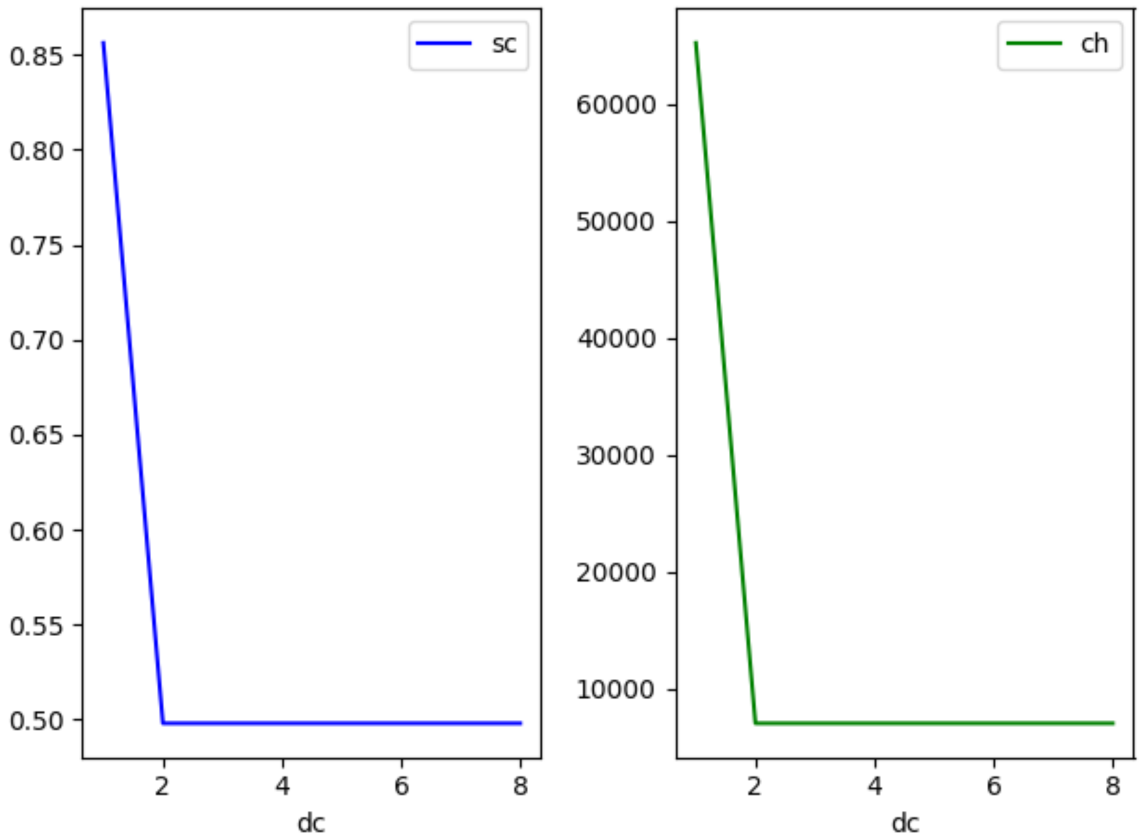
- 聚类的总个数：评判是否聚类过多或过少
- 每个聚类的大小(所含数据点的个数)：分类是否平均
- 每个聚类内部占比较高的维度的均值：分类是否合理
  - 对于每一个聚类，计算其内所有数据点、每个维度的均值，并计算每个维度均值占该类全部维度均值总和的比例，按照比例由大到小排序
  - 如果每个聚类排在前面的维度不属于同一维度或差值很大，则认为具有不同特征的数据被较好分类
  - 因为用户的维度代表了用户去某类商店的比例，这样计算有一定合理性，代表了不同类别用户偏好的商店类型

1.Density Peaks

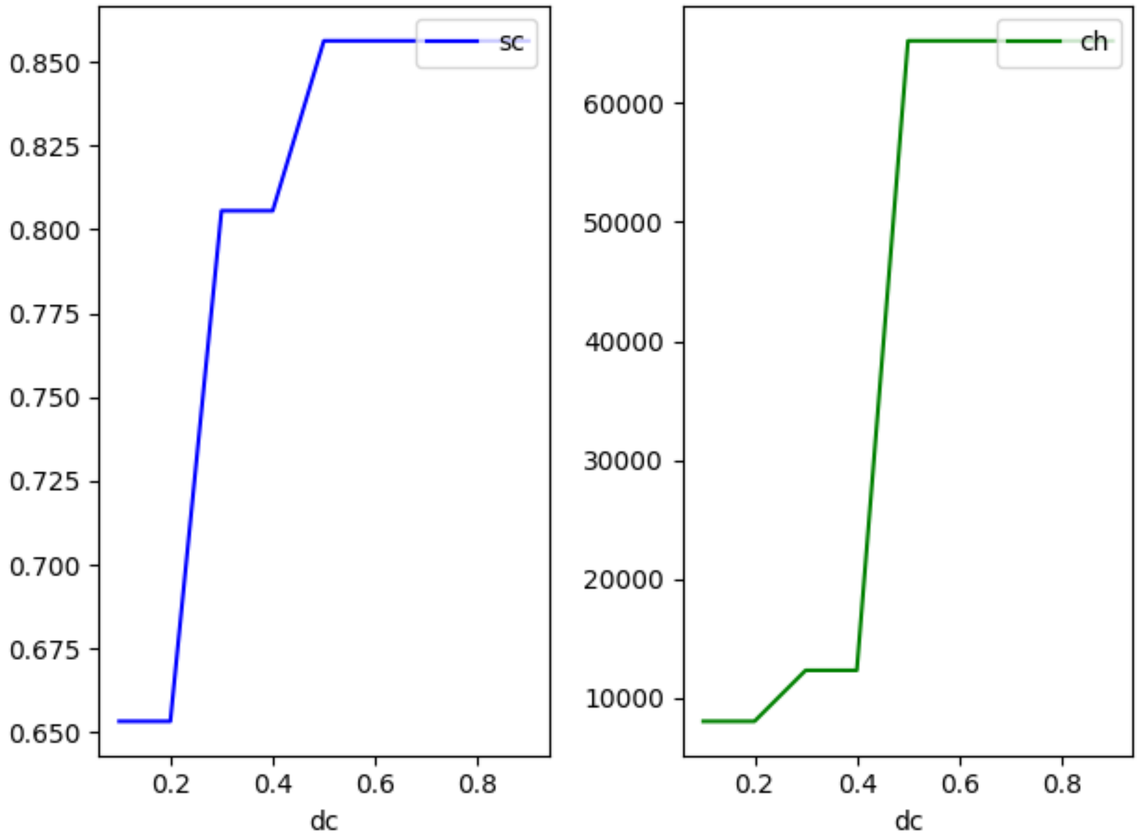
由decision graph可以确定聚类的个数



dc: 1 ~ 9



dc: 0.1 ~ 1



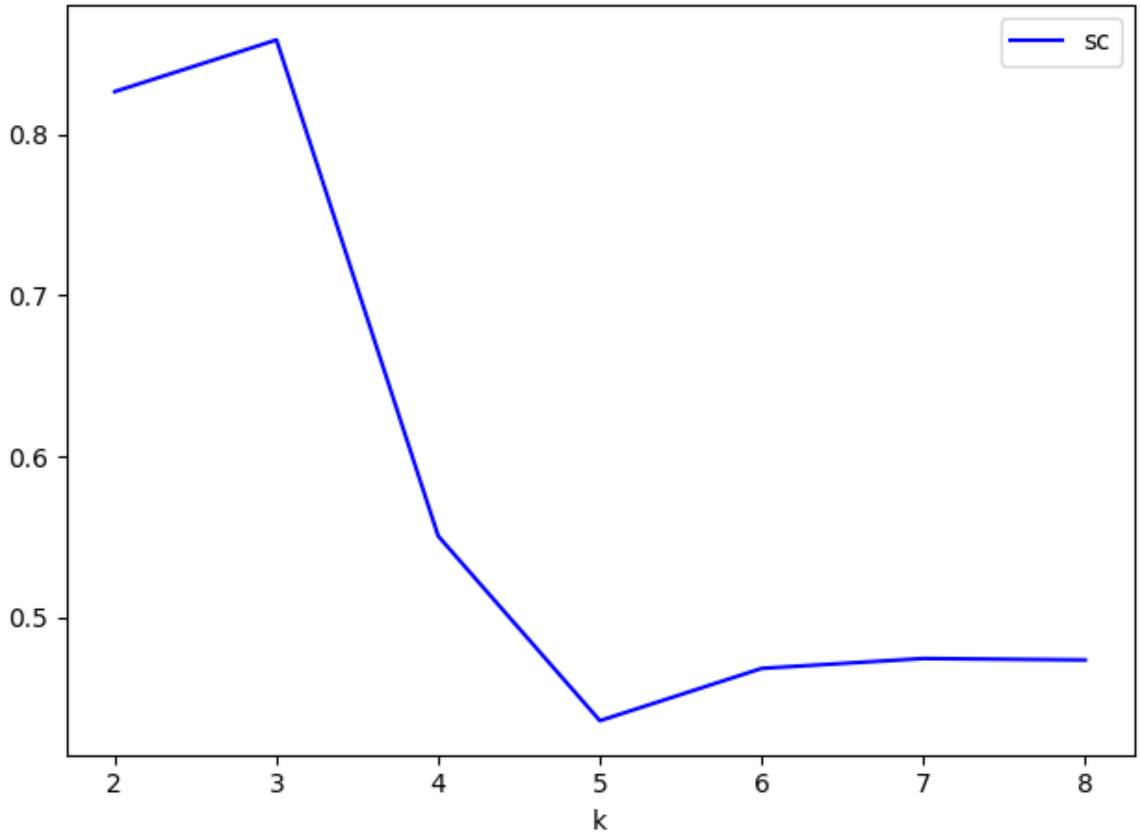
综上，dc 在0.5～1之间都能取得最好的效果

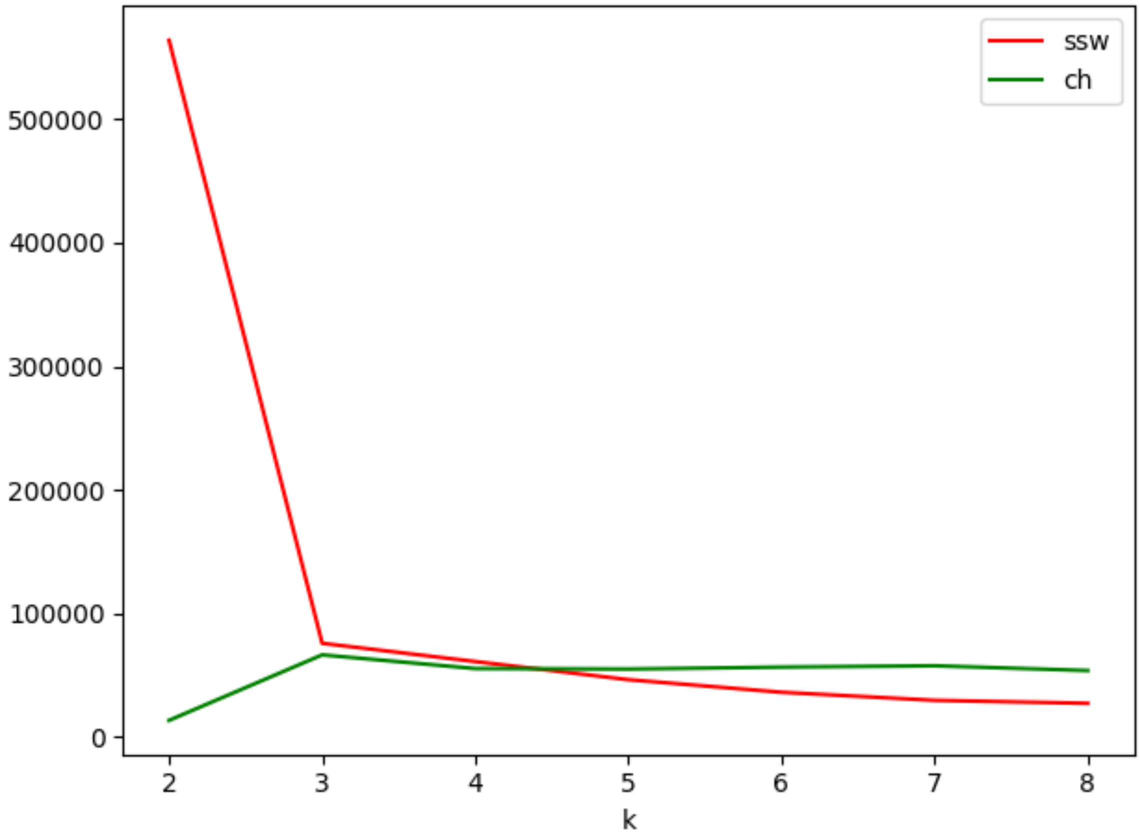
dc	clusters	sc	ch
0.8	3	0.8562917353463756	65202.36474134977

```
cluster: 0 size: 3414
[(1, 0.21800953852132424), (2, 0.06575485930926134), (6, 0.014118769416314509), (12, 0.0074
cluster: 1 size: 1514
[(0, 21.909480274657486), (5, 0.03565200066317886), (10, 0.010900640568301747), (14, 0.0105
cluster: 2 size: 72
[(0, 106.03192897089978), (1, 3.3307622434493327), (2, 1.4333826748971994), (6, 0.307984152

DENSITY PEAK
dc : 0.8
clusters : 3
*---------------------------------------------------*
| silhouette_score        | 0.8562917353463756     |
| calinski_harabaz_score  | 65202.36474134977      |
*---------------------------------------------------*
```

2.K means

k: 2 ~ 8





综上，k=3 时能取得最好的效果

clusters	sc	ch	k_means_inertia
3	0.8586452557676696	66611.76929088337	75985.05315044575

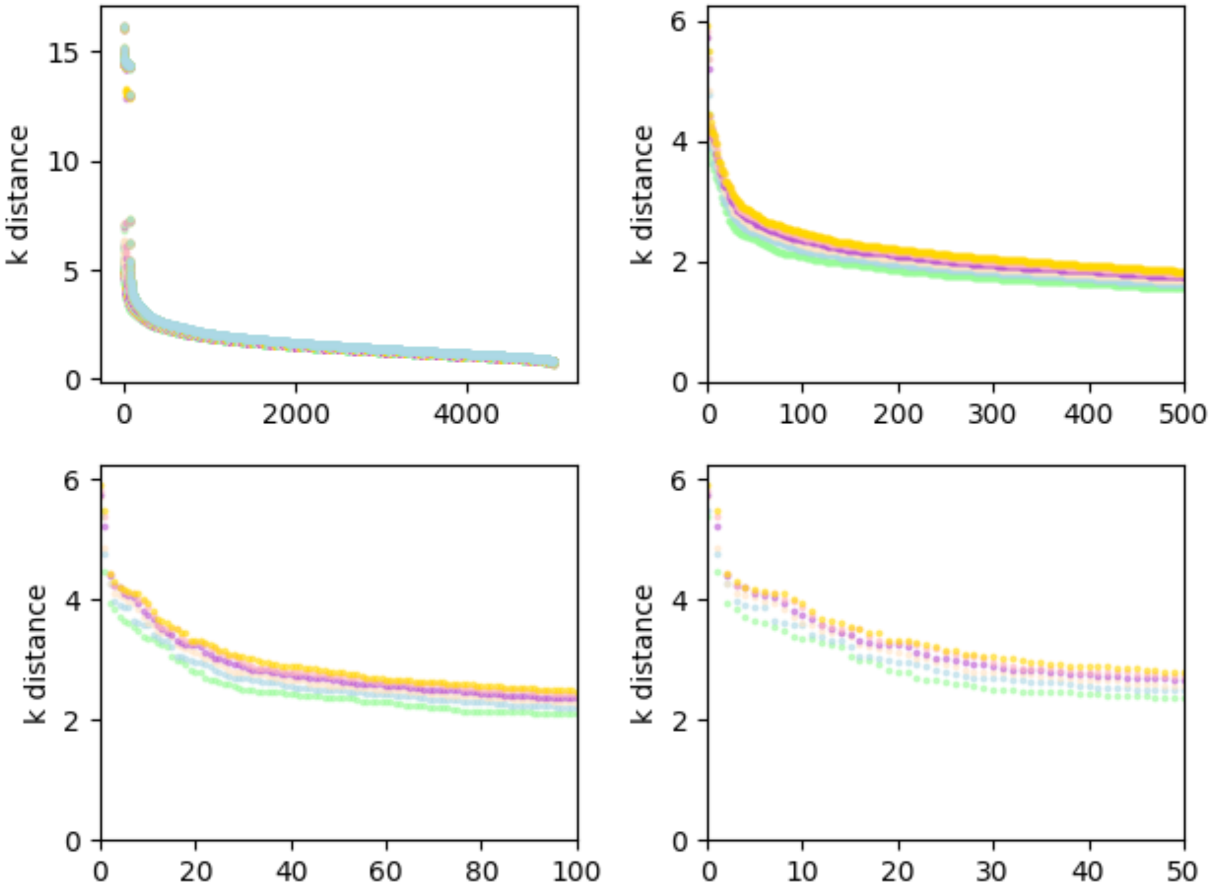
```
cluster: 0 size: 3414
[(1, 0.21800953852132424), (2, 0.06575485930926134), (6, 0.014118769416314509), (12, 0.0074
cluster: 1 size: 1514
[(0, 21.909480274657486), (5, 0.03565200066317886), (10, 0.010900640568301747), (14, 0.0105
cluster: 2 size: 72
[(0, 106.03192897089978), (1, 3.3307622434493327), (2, 1.4333826748971994), (6, 0.307984152

K MEANS
k : 3
clusters : 3
*--------------------------------------------------*
| silhouette_score        | 0.8562917353463756      |
| calinski_harabaz_score  | 65202.36474134977       |
*--------------------------------------------------*
```

3.DBSCAN

调参过程

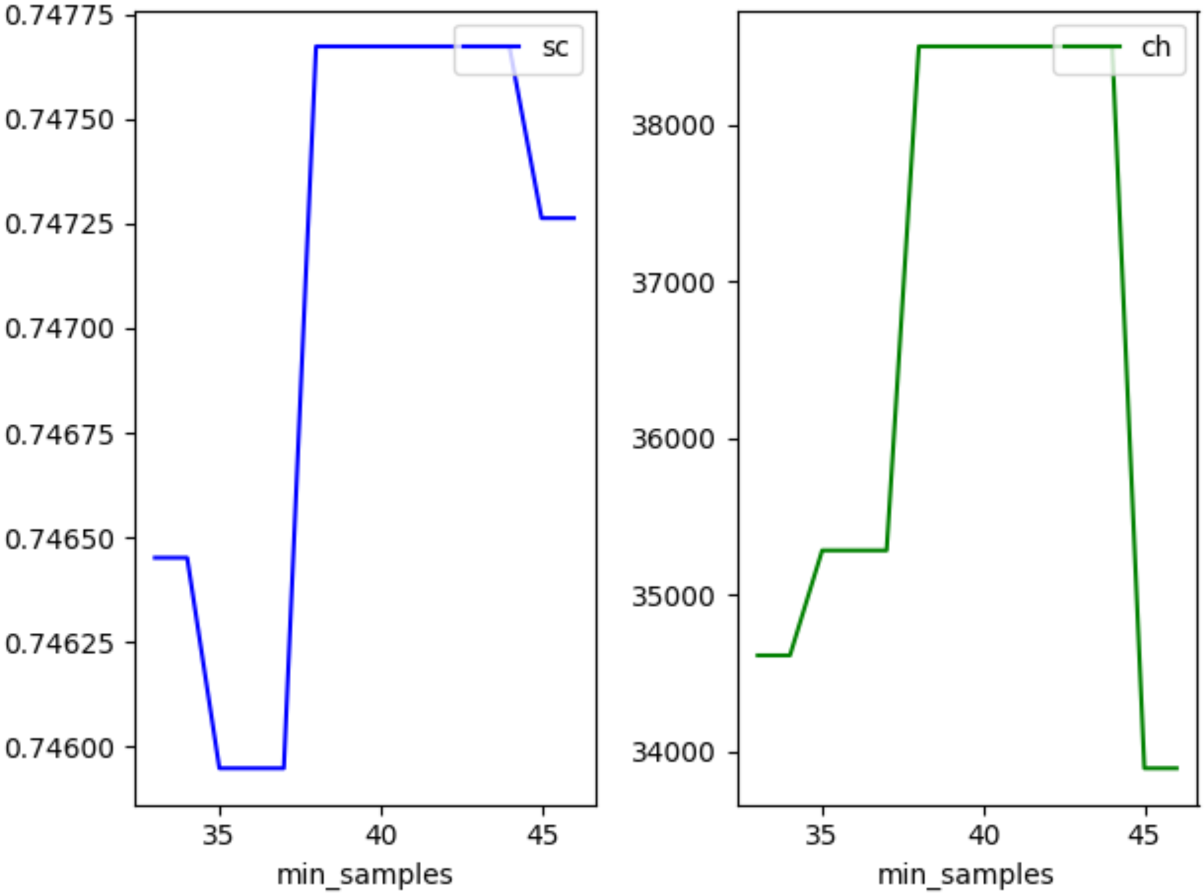
> 参考资料：https://en.wikipedia.org/wiki/DBSCAN
>
> **minPts**
>
> As a rule of thumb, a minimum *minPts* can be derived from the number of dimensions *D* in the data set, as ***minPts* ≥ *D* + 1**. As a rule of thumb, ***minPts* = 2·*dim*** can be used, but it may be necessary to choose larger values for very large data, for noisy data or for data that contains many duplicates.
>
> **eps(ε)**
>
> The value for ε can then be chosen by using a **k-distance graph**, plotting the distance to the ***k* = *minPts*-1** nearest neighbor ordered from the largest to the smallest value. Good values of ε are where this plot **shows an "elbow"**.

1. 绘制 k-distance graph



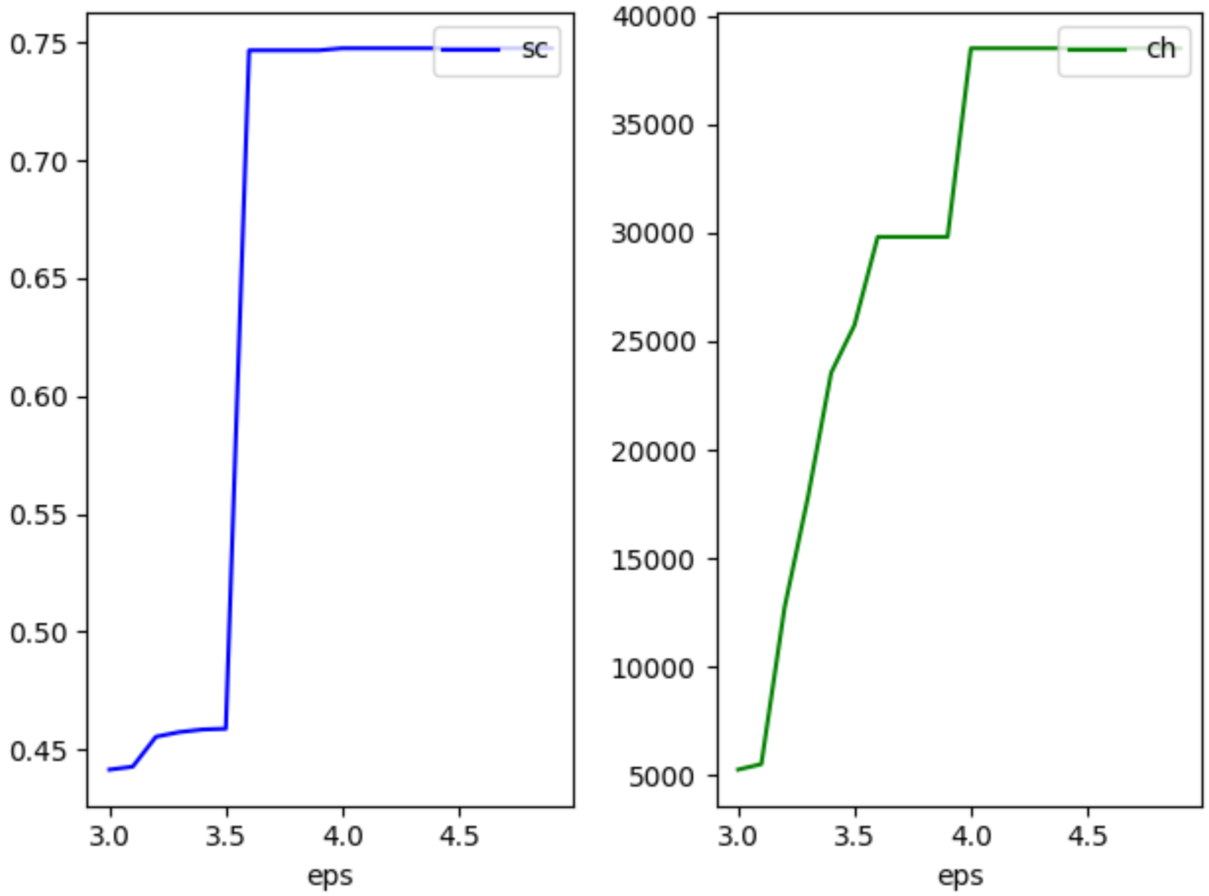
- 上图中，逐渐减小横轴显示的限制值，使拐点更清晰
- 不同颜色的线表示不同的 k 值，取值范围为 26 ～ 51
- 根据上图，初步选择 eps = 4

2. 找合适的 minPt 值



- 在 eps = 4 的条件下，从 26～52 范围中找合适的 minPt 值
- 可知，minPt 范围在 38～43 内具有一样好的效果

3. 验证 eps 值



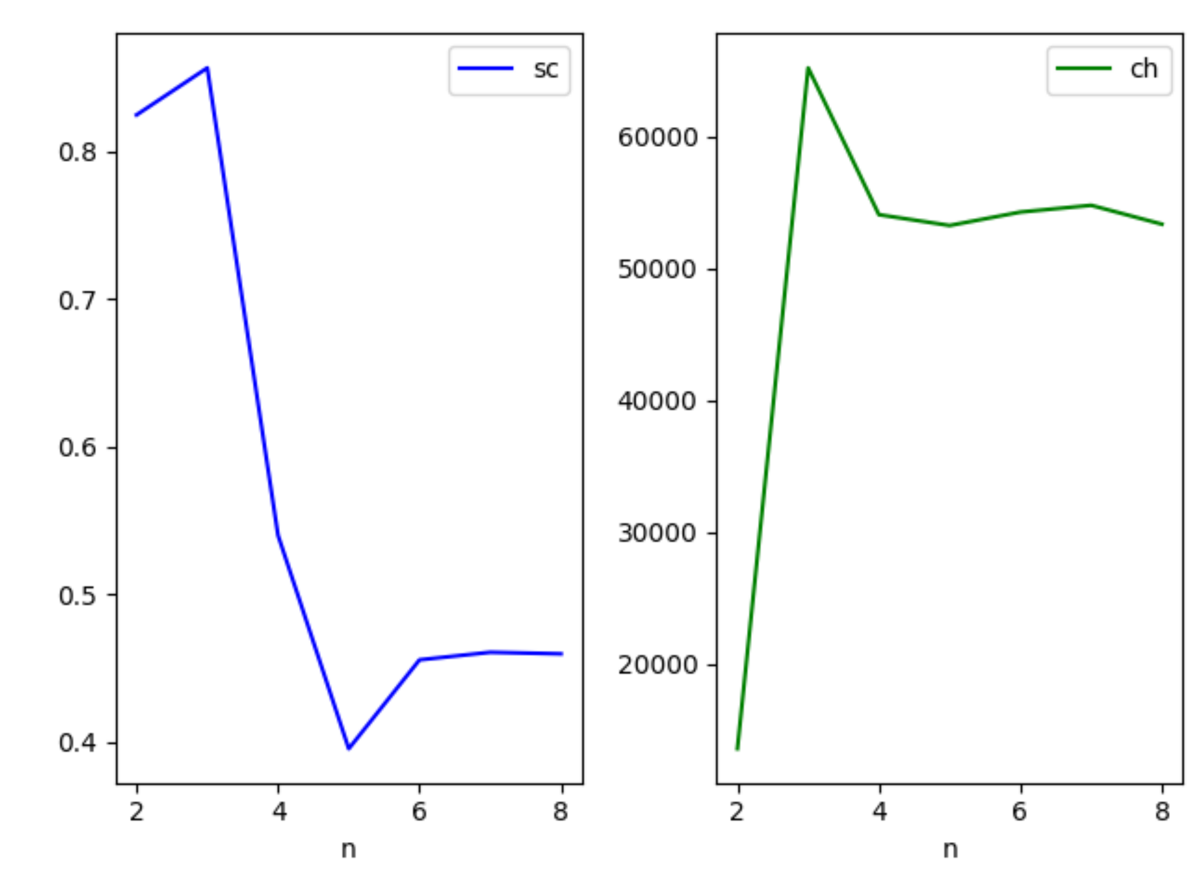
- 在 minPt = 38 的条件下，从 3.0～5.0 范围中找合适的 eps 值
- 可知 eps = 4 时确实能取得最好的效果

eps	min_samples	clusters	sc	ch
4	38	5	0.7476730775596092	38497.82240915197

```
cluster: 0 size: 73
[(0, 104.88235469254421), (1, 3.2972868066438394), (2, 1.465423665732804), (6, 0.2693448634647577), (12, 0
cluster: 1 size: 964
[(0, 22.42205624515312), (5, 0.05166540968270762), (14, 0.017066860947697948), (15, 0.011642202199397678),
cluster: 2 size: 3414
[(1, 0.21800953852132424), (2, 0.06575485930926134), (6, 0.014118769416314509), (12, 0.007404495038041982)
cluster: 3 size: 141
[(0, 13.48552051756061), (2, 0.17780487665151046), (4, 0.07479242835049006), (11, 0.04601493272760417), (1
cluster: 4 size: 261
[(0, 20.39268670548766), (1, 5.4507923928356075), (6, 0.1416931183040556), (3, 0.11619022809481075), (4, 0
cluster: 5 size: 147
[(0, 29.319920311187644), (4, 0.16094667675626328), (20, 0.07623124578889733), (8, 0.07621618636996394), (

DB SCAN
eps : 4.0
min_samples : 38
clusters : 5
*-----------------------------------------------*
| silhouette_score        | 0.7476730775596092     |
| calinski_harabaz_score | 38497.82240915197      |
*-----------------------------------------------*
```

4.EM

n_clusters: 2 ~ 8



综上，n_clusters=3 时有最好的效果

clusters	sc	ch
3	0.8562917353463756	65202.36474134977

```
cluster: 0 size: 3414
[(1, 0.21800953852132424), (2, 0.06575485930926134), (6, 0.014118769416314509), (12, 0.007
cluster: 1 size: 72
[(0, 106.03192897089978), (1, 3.3307622434493327), (2, 1.4333826748971994), (6, 0.30798415
cluster: 2 size: 1514
[(0, 21.909480274657486), (5, 0.03565200066317886), (10, 0.010900640568301747), (14, 0.010

EM
n : 3
clusters : 3
*--------------------------------------------------*
| silhouette_score        | 0.8562917353463756     |
| calinski_harabaz_score | 65202.36474134977      |
*--------------------------------------------------*
```

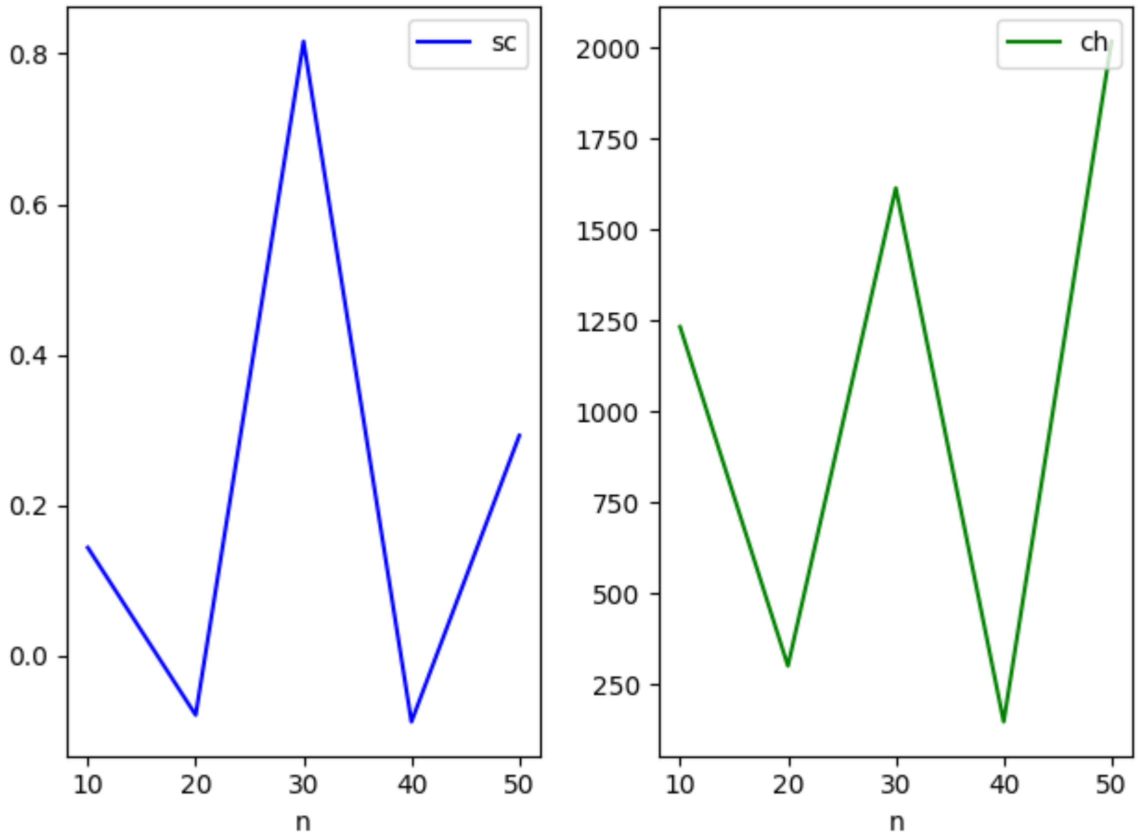
5.Spectral

使用K邻近法('nearest_neighbors')计算相似矩阵，因为默认的高斯函数'rbf'跑很久也跑不出来。

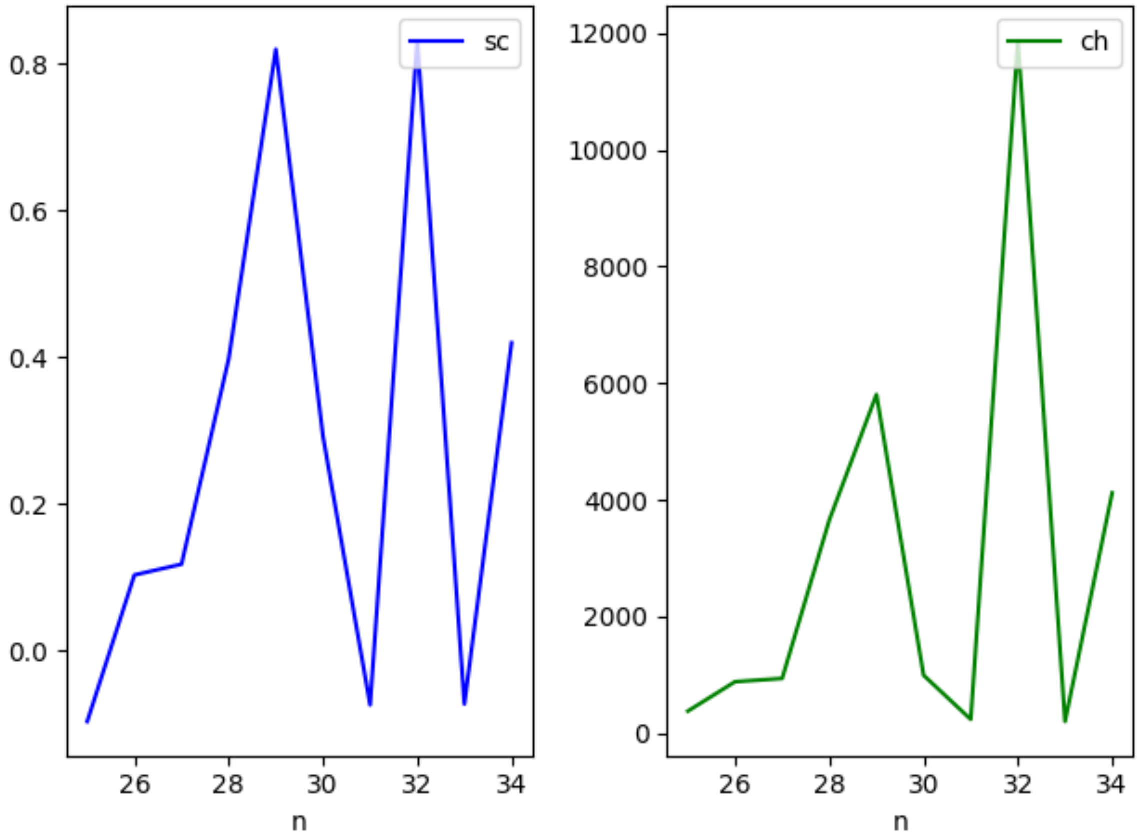
n_cluster 根据前面的经验选择3，根据前面 DBSCAN 算法的经验，在 10～50 范围内选择 nearest_neighbors 。

```python
task = SpectralClustering(n_clusters=3, affinity='nearest_neighbors', n_neighbors=32)
```

n_neighbors: 10 ~ 50



n_neighbors: 25 ~ 35



选取最好的效果如下：

n_neighbors	clusters	sc	ch
32	3	0.8309303645547059	11874.757638816136

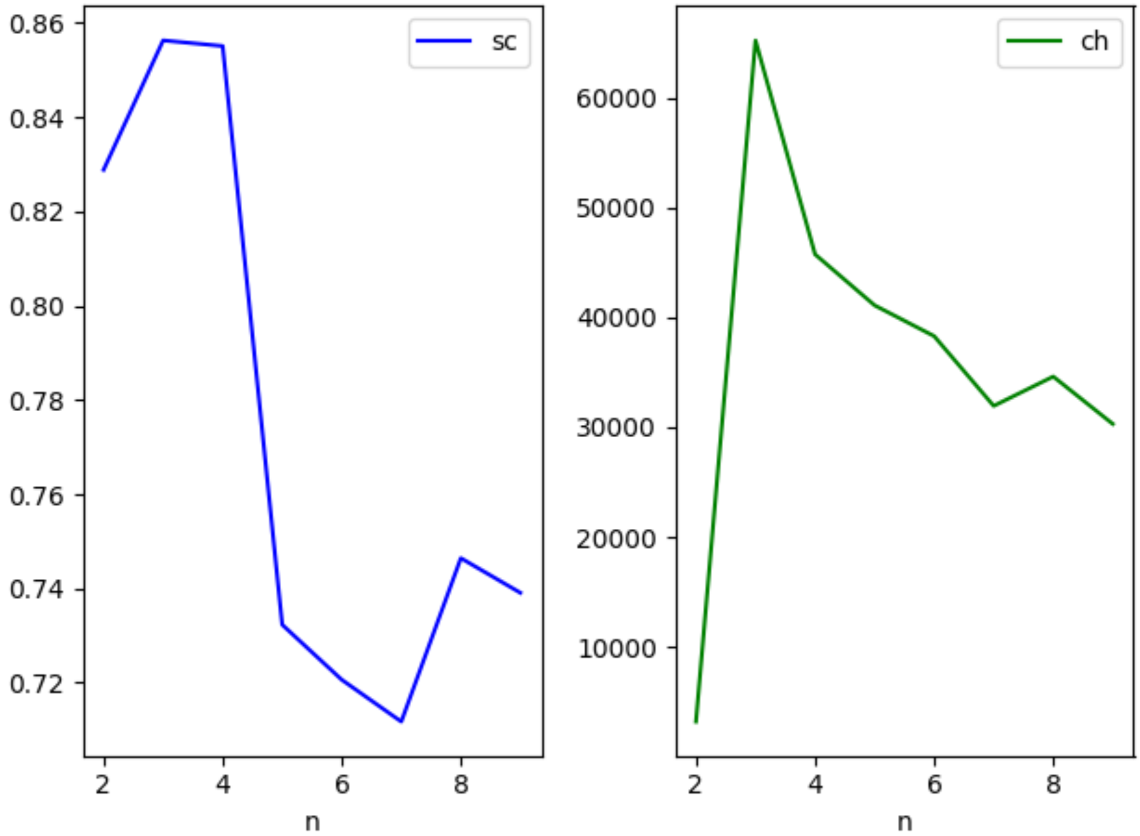
```
cluster: 0 size: 1551
[(0, 24.043861881103098), (5, 0.030244992730469437), (10, 0.009234722789809439), (4, 0.0089
cluster: 1 size: 3414
[(1, 0.21800953852132424), (2, 0.06575485930926134), (6, 0.014118769416314509), (12, 0.0074
cluster: 2 size: 35
[(0, 100.37777840415409), (2, 2.1290909637674504), (3, 0.4176266403411672), (12, 0.13445577

SPECTRAL
n : 32
clusters : [0 1 2]
*-----------------------------------------------------*
| silhouette_score        | 0.8309303645547059      |
| calinski_harabaz_score  | 11874.757638816136      |
*-----------------------------------------------------*
```

但是，其实同样的参数，每次用spectral算法算出来的结果也不一样，甚至可能差很多。可能是由切图的初始随机性导致的。

> 参考资料：https://www.cnblogs.com/pinard/p/6221564.html

6.Hierarchy

n_clusters: 2 ~ 9



综上，n_clusters=3 时有最好的效果

criterion	t	clusters	sc	ch
maxclust	3	3	0.8562917353463756	65202.36474134977

```
cluster: 0 size: 72
[(0, 106.03192897089978), (1, 3.3307622434493327), (2, 1.4333826748971994), (6, 0.30798415
cluster: 1 size: 3414
[(1, 0.21800953852132424), (2, 0.06575485930926134), (6, 0.014118769416314509), (12, 0.007
cluster: 2 size: 1514
[(0, 21.909480274657486), (5, 0.03565200066317886), (10, 0.010900640568301747), (14, 0.010

HIERARCHY
n : 3
clusters : [1 2 3]
*---------------------------------------------------*
| silhouette_score        | 0.8562917353463756      |
| calinski_harabaz_score  | 65202.36474134977       |
*---------------------------------------------------*
```

7.算法之间比较

algorithm	clusters	sc	ch	time(s)
Density Peaks	3	0.8562917353463756	65202.36474134977	24.94
K means	3	0.8586452557676696	66611.76929088337	2.19
DBSCAN	5	0.7476730775596092	38497.82240915197	3.50
EM	3	0.8562917353463756	65202.36474134977	2.75
Spectral	3	0.8309303645547059	11874.757638816136	9.60
Hierarchy	3	0.8562917353463756	65202.36474134977	2.21

调参

- 所有算法中，只有 DBSCAN 和 Spectral 算法是调参比较麻烦的，需要两个参数来回调，而且很难调出最合适的参数
- Density Peaks 也稍微有点麻烦，主要是dc距离值的精度对结果也有一定影响，一般需要精确到小数点后一位或两位，但可以通过一点点缩小范围。此外，由算法自动选择peak点还是不够准确，有时候会只选出一个、两个peak点，需要人根据 decision graph 手动设置peak点个数
- K means、EM、Hierarchy 调参的感觉都差不多，因为其参数就是聚类个数，只要对在 2～20 内的整数值进行遍历尝试即可(通常 2～9 就足够)，并且这几个算法出来的结果也都比较稳定

效果

- 从上表可以看出，对于这个数据集，K means 具有最好的聚类效果。Density Peaks、EM、Hierarchy 的效果都差不多，聚类结果是基本一致的

性能

- K means 因为其算法简单，有明显的性能优势。而谱聚类 Spectral 因为其算法复杂，要计算相似矩阵、再切图等，所以耗时较长。而 Density Peaks 可能因为是自己写的算法，没有经过很好的优化，所以时间明显地长了