MR定位建模——聚类

1953871 邓泉

1. **获取数据**

读取2G\_data.csv的MR数据与2G\_gongcan.csv的工参表数据，进行匹配工参，使得在MR数据的基础上增添了邻接基站的GPS经纬度位置。

1. **数据集转换**
   1. **相对位置计算 + 异常值处理**

对于每一条MR数据，根据id\_combine= RNCID + CellID从多个邻接基站中标定唯一的主基站，新增两列属性m\_Longitude、m\_Latgitude用于标注主基站的GPS坐标。

计算MR记录的GPS相对坐标，用新增的属性r\_Longitude、r\_Latgitude来记录；将各个邻接基站的GPS坐标转换成以主基站为参考系的相对坐标（替代原值）。

**缺失值处理：**

* 对于在工参表中无匹配项的主基站，工参匹配时其GPS坐标为缺失值NaN，用工参表中已有的所有基站的GPS坐标平均值代替
* 用一个绝对值足够大的负数-999填充剩余的NaN
  1. **分组**

按照每个MR记录对应的主基站对 MR 记录进行分组，使得每组 MR 记录都有相同的主基站，总的分组个数即为主基站个数。

**结果显示，主基站一共有43个。**

****

1. **Question a)**
   1. **问题重述**

MR样本先是按照主基站分组，比如有N个主基站，MR分组就有N个分组，聚类就是对N个分组聚类，可以取得K个簇，每个簇里面包括多个分组。换言之，聚类的粒度是主基站对应的分组group，而不是单个的MR样本记录。

设计和实现 K-Means 聚类方法，使得同簇内分组相对位置之间的相似度高、 跨簇分组相对位置之间的相似度低，要求计算一个最优K值及其聚类结果。（kmeans聚类的时候，都有确定一个输入的值，但是这个K不一定导致最优的聚类质量，所以需要选定一个最好的值，使得聚类质量最好。）

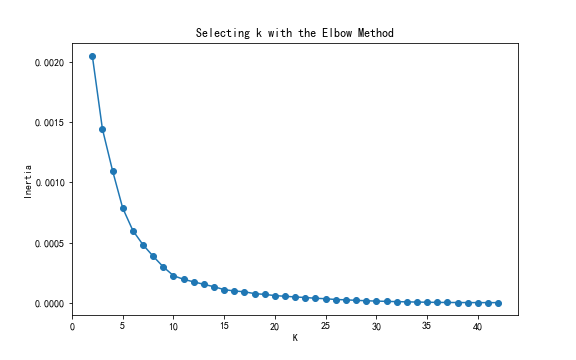
* 1. **算法设计**

提取每个分组group下所有MR样本数据的**经纬度均值和方差**，构成四维向量 用于刻画每一个分组，用K-Means方法进行聚类。

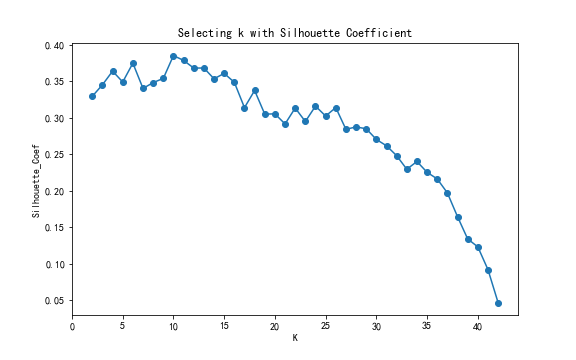
使用**The Elbow Method肘部法**和**The Silhouette Coefficient轮廓系数法**确定最优K值。

* 1. **结果**

**The Elbow Method肘部法：**

****

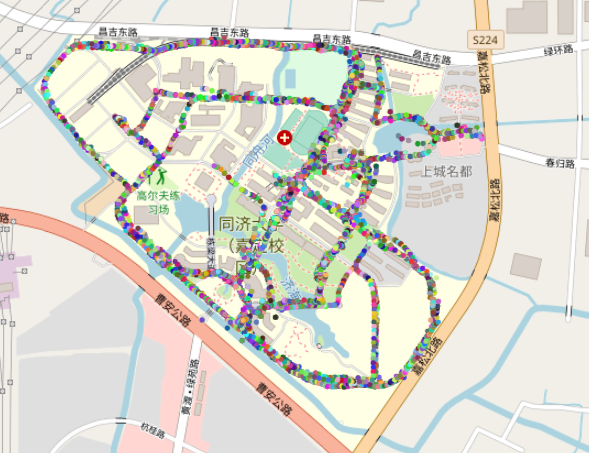
**The Silhouette Coefficient轮廓系数法：**



根据上列两图，选择**K = 10**为最优K值。其聚类结果为：



投影到**地图**实现**可视化**，不同颜色标识不同簇：



[**..\result data\cluster\Question a)\10 - cluster - tj\_map.html**](file:///D:\学习资料\大三下\数据分析与挖掘（饶卫雄）\hw1+hw2\result%20data\cluster\Question%20a)\10%20-%20cluster%20-%20tj_map.html)

* 1. **关键代码**

**聚类 + 肘部法/轮廓系数法**

|  |
| --- |
| 1. from sklearn.cluster import KMeans 2. from sklearn.metrics import davies\_bouldin\_score, silhouette\_score, calinski\_harabasz\_score 3. a\_SSE, a\_DBI, a\_Silhouette, a\_Calinski = [], [], [], [] 4. a\_df = pd.DataFrame(columns= 5. ['RNCID','CellID','mean\_lon', 'mean\_lat', 'std\_lon', 'std\_lat']) 6. for index, RNC\_Cell in enumerate(mbs.values): 7. *# 筛选出该主基站（分组Group）下的MR样本* 8. id\_combine = RNC\_Cell[0] 9. RNCID = RNC\_Cell[1] 10. CellID = RNC\_Cell[2] 11. MR\_sample = df[df["RNCID1"] == RNCID] 12. MR\_sample = MR\_sample[MR\_sample["CellID1"] == CellID] 13. *# 计算这些MR样本的坐标均值、方差* 14. mean\_lon = np.mean(MR\_sample["r\_Longitude"]) 15. mean\_lat = np.mean(MR\_sample["r\_Latitude"]) 16. std\_lon = np.std(MR\_sample["r\_Longitude"]) 17. std\_lat = np.std(MR\_sample["r\_Latitude"]) 18. a\_df.loc[len(a\_df.index)] = [RNCID, CellID, mean\_lon, mean\_lat, std\_lon, std\_lat] 19. a\_X = a\_df[['mean\_lon', 'mean\_lat', 'std\_lon', 'std\_lat']] 20. for k in range(2, len(a\_X)): 21. kmeans = KMeans(n\_clusters = k, random\_state=0).fit(a\_X) 22. a\_Silhouette.append(silhouette\_score(a\_X, kmeans.labels\_, metric='euclidean')) 23. a\_SSE.append(kmeans.inertia\_) 24. a\_DBI.append(davies\_bouldin\_score(a\_X, kmeans.labels\_)) 25. a\_Calinski.append(calinski\_harabasz\_score(a\_X, kmeans.labels\_)) 26. *#     print("-----")* 27. *# 绘图 - 手肘法* 28. c = [i for i in range(2, len(a\_X))] 29. x\_ticks = [i for i in c] 30. fig = plt.figure(figsize=(8,5)) 31. plt.plot(c,a\_SSE,'o-') 32. *# plt.xticks(x\_ticks)* 33. plt.xlabel("K") 34. plt.ylabel("Inertia") 35. plt.title('Selecting k with the Elbow Method') 36. plt.show() 37. fig.figure.savefig(savepath + 'Question a)/Elbow Method') *# 保存* 38. fig = plt.figure(figsize=(8,5)) 39. plt.plot(c,a\_Silhouette,'o-') 40. *# plt.xticks(x\_ticks)* 41. plt.xlabel("K") 42. plt.ylabel("Silhouette\_Coef") 43. plt.title('Selecting k with Silhouette Coefficient') 44. plt.show() 45. fig.figure.savefig(savepath + 'Question a)/Silhouette Coefficient') *# 保存* |

**聚类结果地图可视化**

|  |
| --- |
| 1. *# 随机⽣成颜⾊的RGB代码* 2. import random 3. def RandomColor(): 4. colorArr = ['1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'] 5. color = "" 6. for i in range(6): 7. color += colorArr[random.randint(0, len(colorArr)-1)] 8. return "#"+color 9. def ClusterVisual(all\_MR): 10. *# 绘制地图，一个簇一种颜色* 11. import folium 12. *# location 经纬度，list 或者 tuple 格式，顺序为 latitude(纬度), longitude(经度)* 13. *# zoom\_start 缩放值，默认为 10，值越大比例尺越小，地图放大级别越大* 14. tj\_map = folium.Map(location=[31.286363, 121.214786],zoom\_start=15) *# 显示地图* 15. *#     tj\_map = folium.Map(location=[list(Y\_test['Latitude'])[0], list(Y\_test['Longitude'])[0]], zoom\_start=15) # 显示地图* 16. *# 添加标记* 17. for cluster in range(len(all\_MR)): 18. cluster\_MR = all\_MR[cluster] 19. for i in range(len(cluster\_MR)): 20. lat = cluster\_MR.iloc[i, :]['Latitude'] 21. lon = cluster\_MR.iloc[i, :]['Longitude'] 22. folium.Circle( 23. radius=1, 24. location=[lat, lon], 25. popup='Laurelhurst Park', 26. color=RandomColor(), 27. fill=False 28. ).add\_to(tj\_map) 29. tj\_map.save(savepath + 'Question a)/' + str(K) + ' - cluster' + ' - tj\_map.html') 30. return tj\_map |

1. **Question b)**
   1. **问题重述**

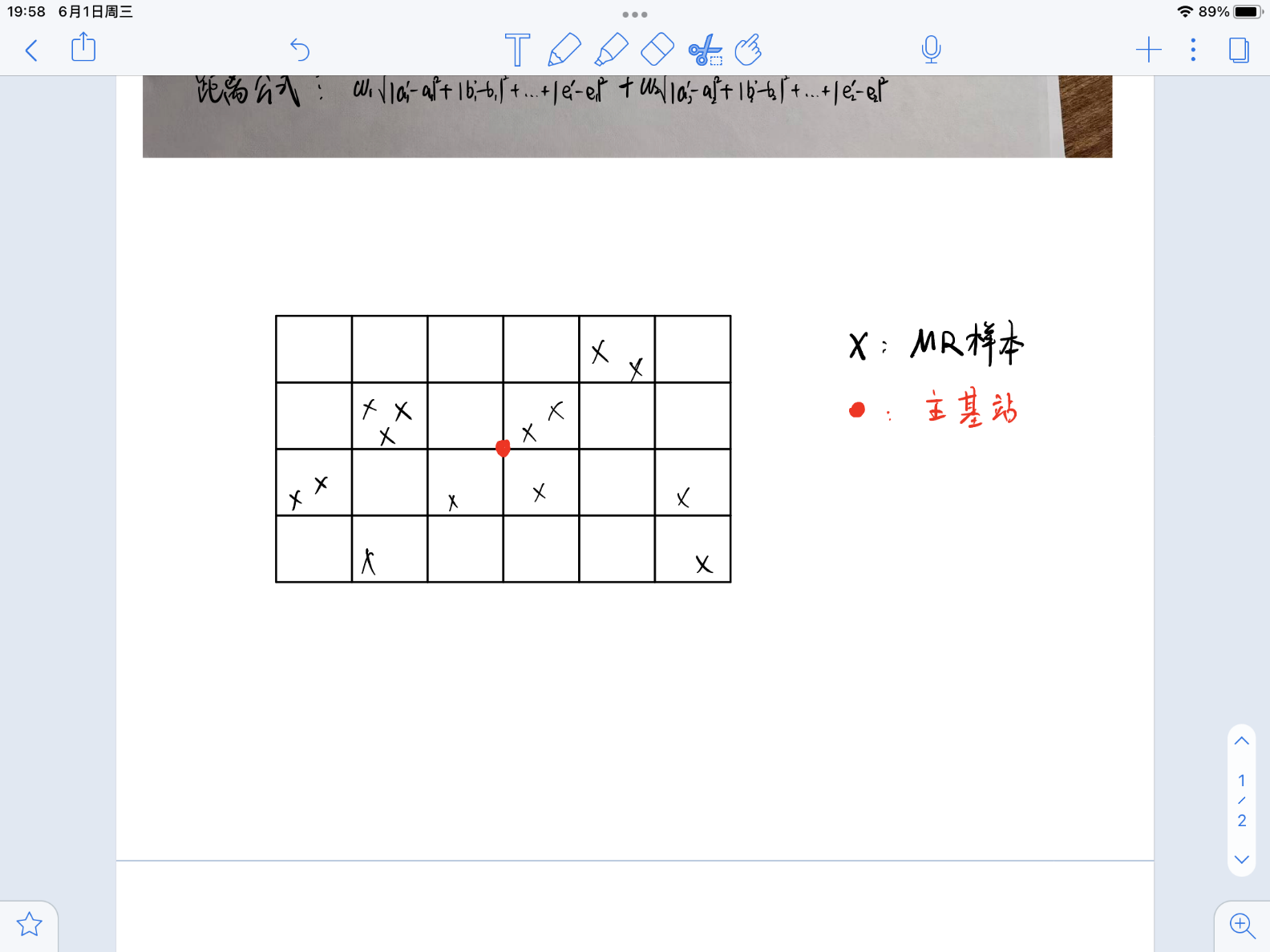
针对每个主基站分组后的 MR 数据，将分组中MR 数据进行简化处理：对每个 MR 样本中的基站编号替换为对应的顺序号，比如主服务器基站对应的顺序号为 1，次服务基站顺序号为 2…，然后在处理后的分组根据信号强度值进行聚类，要求设计一个合理的分组信号强度的距离计算公式，使得在该距离计算公式基础之上的聚类结果，尽可能与步骤 a)聚类结果接近。

* 1. **算法设计**

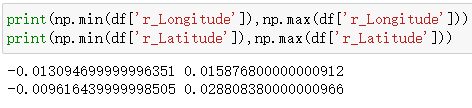
Step 1: 将MR样本坐标投影到主基站的相对坐标系

在步骤a)中已完成。

Step 2: 在该主基站的坐标空间划分栅格Grids



**设计一个分组Group的覆盖范围、单个栅格大小。**



一个Group的覆盖范围是一个矩形**，**上述代码计算了所有MR样本的相对GPS坐标的边界值。以主基站在矩形正中心——即坐标(0,0)，设定Group的相对坐标范围是：

**(±0.02, ±0.03) - (Longitude方向,Latitude方向)**

使得以最小的代价保证一个分组Group的范围覆盖改分组下所有MR样本。

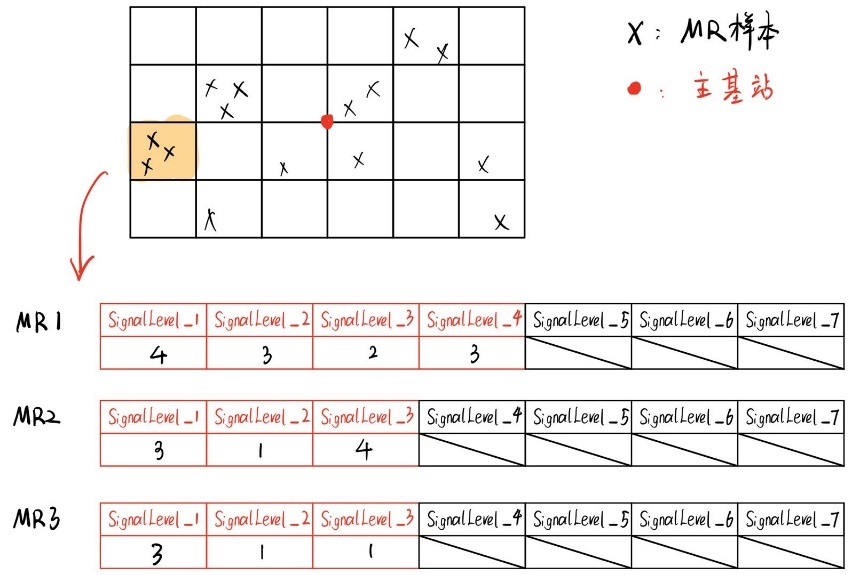
单个栅格Grid大小：

尝试了(0.01, 0.01)，(0.005, 0.005)，（0.005，0.01），（0.01，0.005），最终采用**（0.005，0.01）**，栅格数量是 。

Step 3: 在每个栅格grid中，计算信号强度指纹特征。

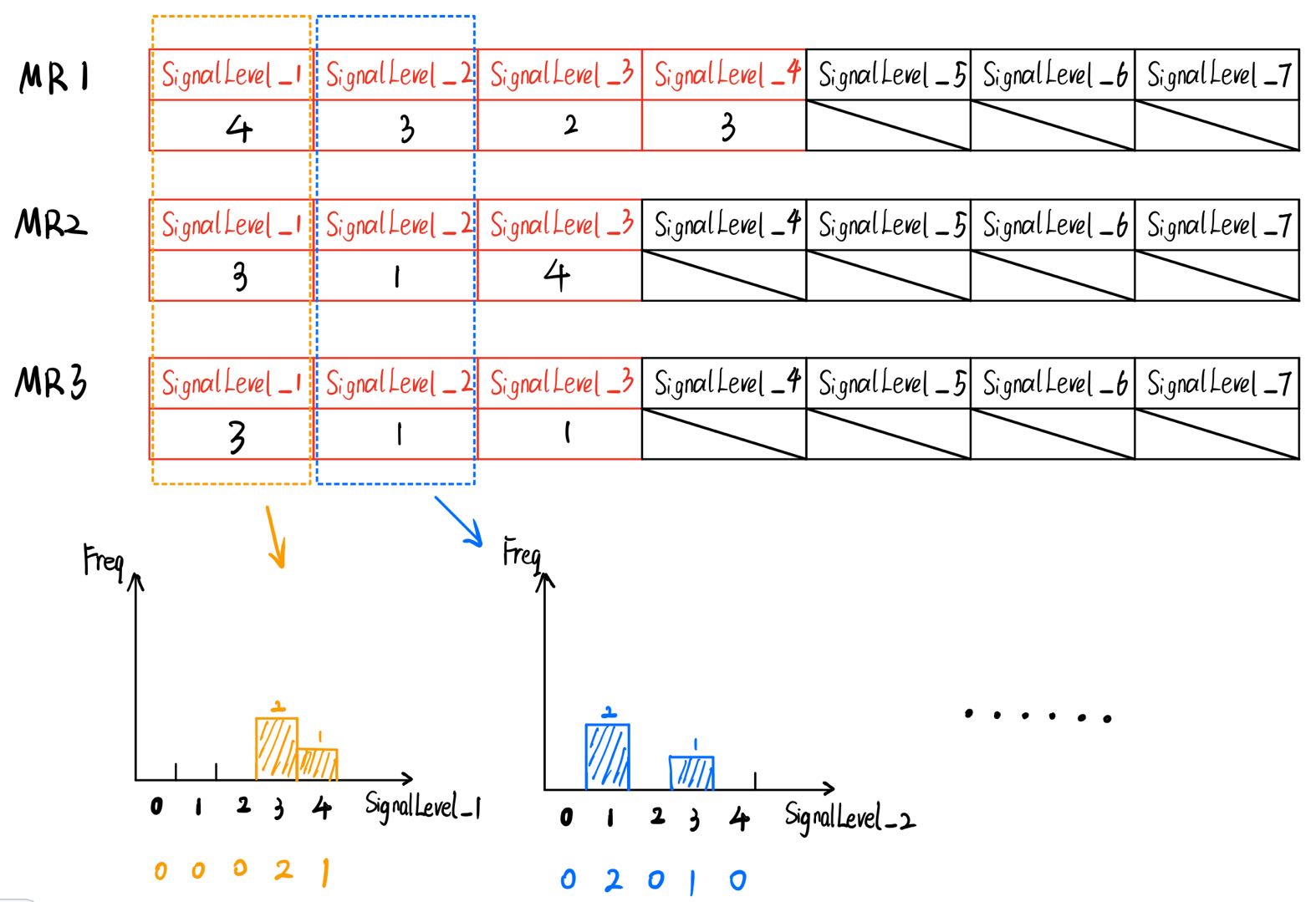
**信号强度指纹计算方法**：

对一个分组Group划分栅格Grids后，每一个栅格内可能包含多条MR样本数据，而一条MR样本有1到7个基站的信号强度数据（信号强度等级：SignalLevel\_1 ~ SignalLevel\_7，1~7表示服务器基站的顺序号），但每条MR样本的最高基站顺序号可能不相等。



在一个栅格Grid中，针对**某一顺序号的服务器基站**，可以作出一个相应的直方图，其横坐标为信号强度等级**SignalLevel**的值（取值范围：0~4，整数），纵坐标就是该栅格Grid中所有MR样本对应该顺序基站各信号强度等级的**频次**。

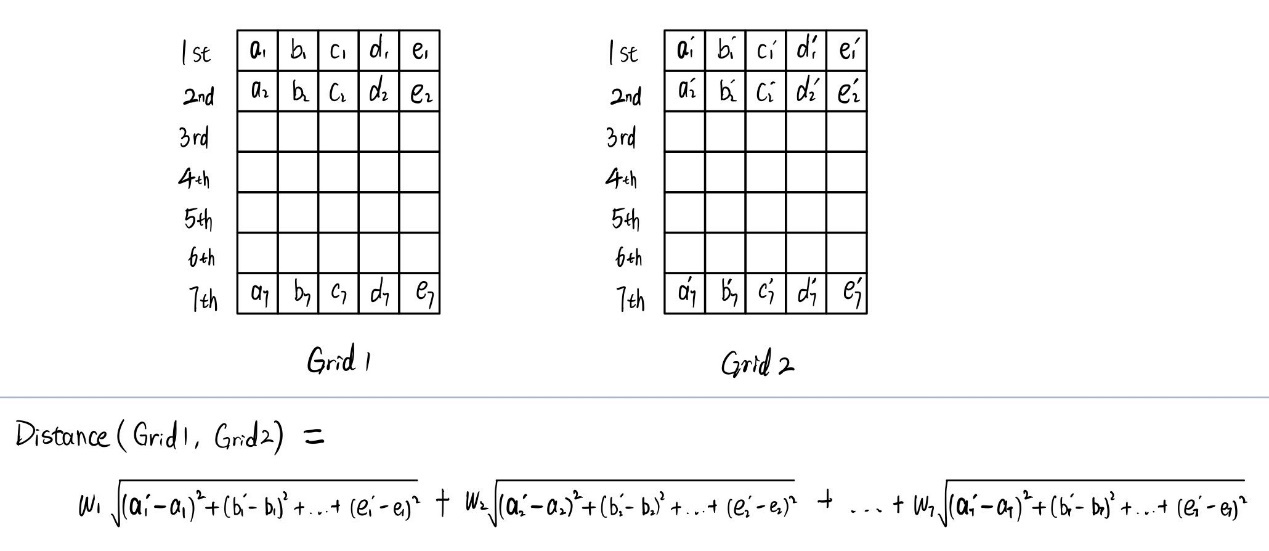
而实际上，这里的直方图都可以用一段**长度为5的顺序数字串**表征，数值便是直方图中的对应频次。



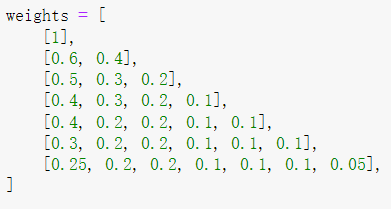
服务器基站的顺序号最高是7，因此一个栅格Grid就可以对应7个直方图，也就是7段**长度为5的顺序数字串**，便可以用一个  **的二维数组**来表征一个Grid。假设一个分组Group被划分成了 个栅格Grids，那么一个group便可用 个 的二维数组来表征，这其中实际上内涵了分组中所有MR样本数据的**信号强度特征**和**分布特征**，形象地称它为分组的**“指纹”**特征。

Step 4: 计算分组Group之间的距离（非相似度）

首先为**栅格Grid**之间设计**距离公式**：

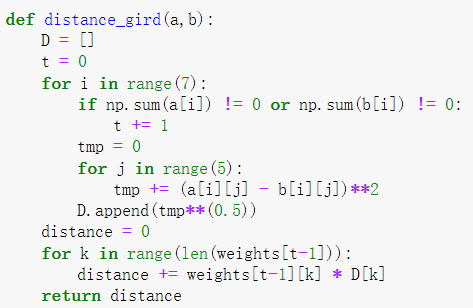


权值 的设计如下：

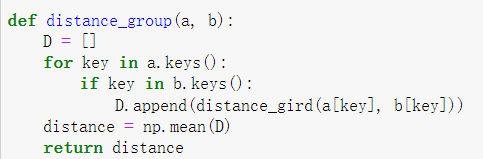


**设计准则**：

1. 总权值 = 1
2. 设两个栅格Grid中所有MR样本的最高顺序号为t，则选取weights[t-1]作为距离计算的权值向量
3. 基站顺序号越高，其在距离计算中的权值理应越大（至少不能更小）。例如主基站的权值就大于其他顺序号基站的权值。



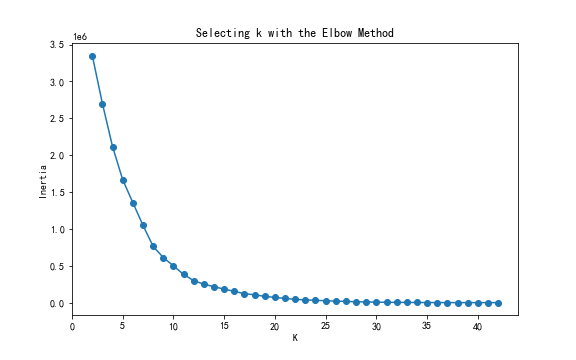
计算两个Group之间所有对应位置栅格的距离，取均值作为**分组Group**间的**距离。**

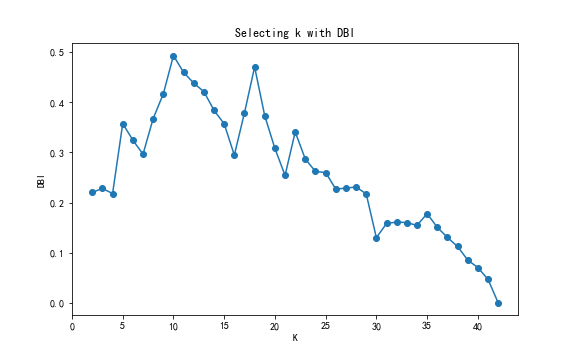


Step 5: 聚类（K-Means聚类 & 层次聚类）

这一步分别尝试了**K-Means聚类**和**层次聚类**。

* 1. **结果**
* **K-Means聚类**

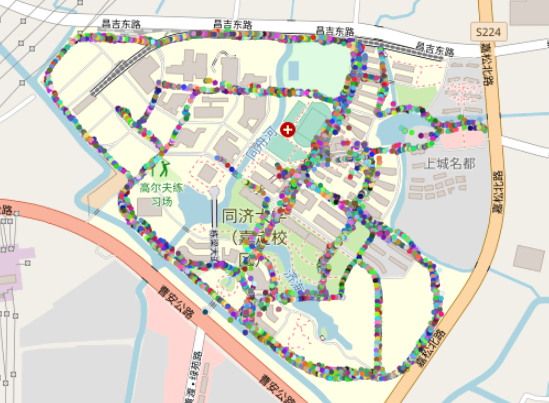




根据上图，选择**K = 10**为最优K值，这与步骤a)中结论一致。其聚类结果为：



可视化：

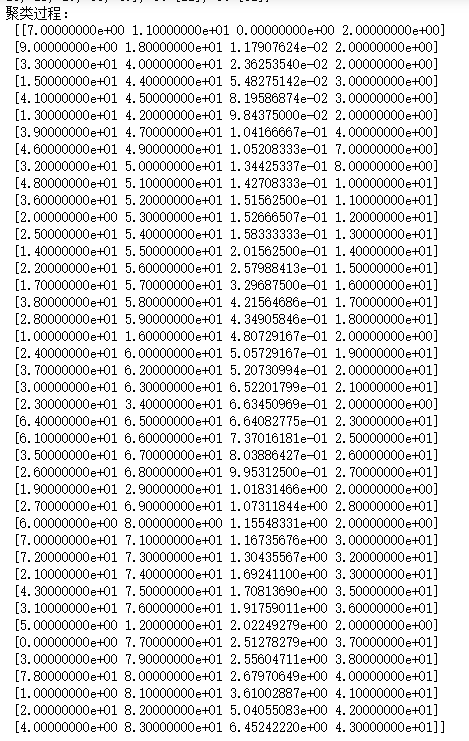
****

* **层次聚类**

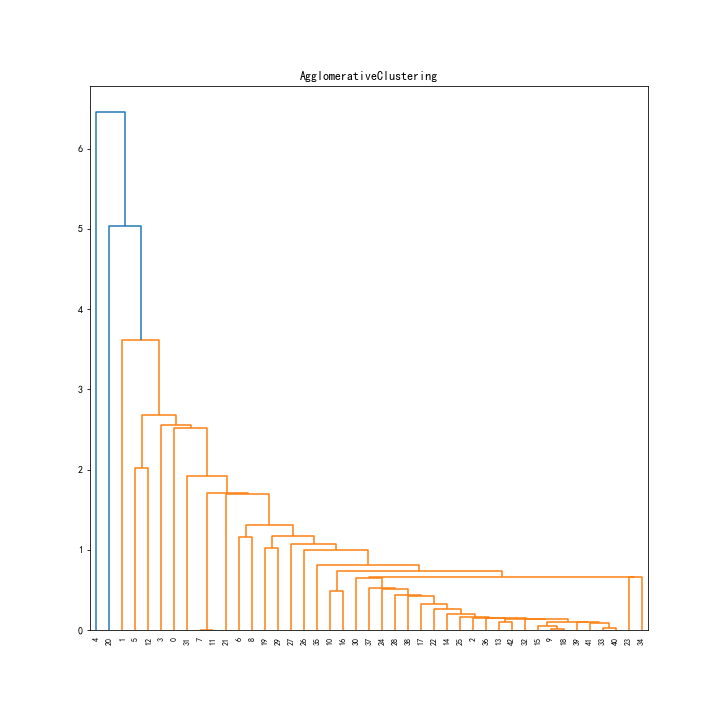
实际上，层次聚类结果与K-Means聚类非常接近。聚类结果（**K=10**）为：



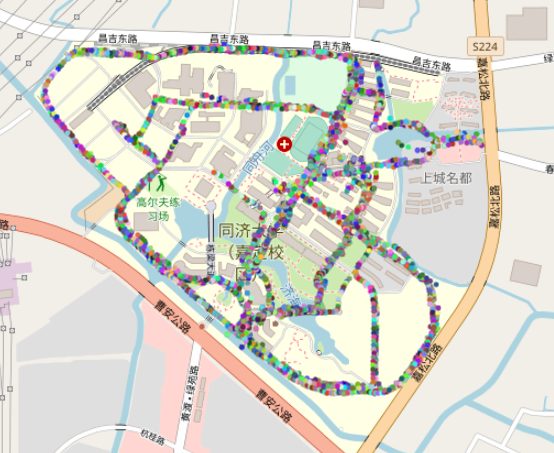
聚类过程：



将层级聚类结果以树状图表示出来：



可视化：



* 1. **关键代码**

**划分栅格**

|  |
| --- |
| 1. class RoadGrid: 2. def \_\_init\_\_(self, label, length, width): 3. xl, xr, yb, yt = -0.02, +0.02, -0.03, +0.03 4. lenNum = int((xr-xl)/length) 5. widNum = int((yt-yb)/width) 6. grid\_dict = {} 7. for m in range(lenNum): 8. for n in range(widNum): 9. if (m, n) not in grid\_dict: 10. grid\_dict[(m, n)] = np.zeros(shape=(7, 5)) 12. for index, mr in label.iterrows(): 13. *# 投影到主基站的相对坐标系* 14. lon = mr["r\_Longitude"] 15. lat = mr["r\_Latitude"] 16. x = int((lon - xl) / length) 17. y = int((lat - yb) / width) 18. for i in range(7): 19. if mr["SignalLevel\_" + str(i+1)] == -999 or mr["SignalLevel\_" + str(i+1)] == 6188: 20. continue 21. j = int(mr["SignalLevel\_" + str(i+1)]) 22. grid\_dict[(x, y)][i][j] += 1 23. self.Grids = grid\_dict 24. # 存放所有Group的Grids 25. AllGrids = [] 26. for index, RNC\_Cell in enumerate(mbs.values): 27. *# 筛选出该主基站（分组Group）下的MR样本* 28. id\_combine = RNC\_Cell[0] 29. RNCID = RNC\_Cell[1] 30. CellID = RNC\_Cell[2] 31. MR\_sample = df[df["RNCID1"] == RNCID] 32. MR\_sample = MR\_sample[MR\_sample["CellID1"] == CellID] 33. Grids = RoadGrid(MR\_sample, 0.01, 0.01).Grids 34. AllGrids.append(Grids) |

**层级聚类中计算距离矩阵**

|  |
| --- |
| 1. dist\_matrix = np.mat(np.zeros((len(AllGrids), len(AllGrids)))) 2. for i in range(len(AllGrids)): 3. for j in range(i, len(AllGrids)): 4. dist\_matrix[i, j] = distance\_group(AllGrids[i], AllGrids[j]) 5. dist\_matrix[j, i] = dist\_matrix[i, j] 6. dist\_matrix = np.nan\_to\_num(dist\_matrix) |

**显示聚类结果（以K-Means为例）**

|  |
| --- |
| 1. def showAgglo(K): 2. Agglo = AgglomerativeClustering(n\_clusters=K, affinity='precomputed', linkage='single').fit(dist\_matrix) 3. clusters = {} 4. for i in range(len(kmeans.labels\_)): 5. if kmeans.labels\_[i] not in clusters.keys(): 6. clusters[kmeans.labels\_[i]] = [] 7. clusters[kmeans.labels\_[i]].append(i) 8. # 调整距离矩阵的形状 9. dist\_matrix\_form = dist.squareform(dist\_matrix) 10. # linkage方法用于计算两个聚类簇s和t之间的距离d(s,t) 11. *# 层次聚类编码为一个linkage矩阵。* 12. Z = linkage(dist\_matrix\_form, 'single') 13. print("聚类过程：\n", Z) 14. # 将层级聚类结果以树状图表示出来 15. fig = plt.figure(figsize=(10, 10)) 16. *#     plt.xlabel("K")* 17. *#     plt.ylabel("DBI")* 18. plt.title('AgglomerativeClustering') 19. dn = dendrogram(Z) 20. plt.show() 21. fig.figure.savefig(savepath + 'Question b)/' + 'AgglomerativeClustering') 22. all\_MR = [] 23. for cluster\_groups in clusters.items(): 24. *# 该簇下所有MR样本* 25. cluster\_MR = pd.DataFrame(columns=df.columns.tolist()) 26. id\_cluster = cluster\_groups[0] 27. groups = cluster\_groups[1] 28. for group\_idx in groups: 29. RNCID = int(mbs.iloc[group\_idx,:]['RNCID1']) 30. CellID = int(mbs.iloc[group\_idx,:]['CellID1']) 31. group\_MR = df[df["RNCID1"] == RNCID] 32. group\_MR = group\_MR[group\_MR["CellID1"] == CellID] 33. cluster\_MR = cluster\_MR.append(group\_MR) 34. *#     print(cluster\_MR)* 35. all\_MR.append(cluster\_MR) 36. return ClusterVisual(all\_MR) |

1. **Question c)**
   1. **问题重述**

根据上述 b)聚类结果，将同簇的 MR 样本构建一个对应的定位模型（参考作业 1b），对比本次定位测试定位误差与作业 1b，并分析导致差异的原因。

* 1. **算法设计**

**动机：**

在hw1的步骤b)中，定位模型的构建过程是: 按照每个 MR 记录对应的主基站对 MR 记录进行分组，并将每个分组的MR记录位置投影到主基站的相对坐标系中，以相对坐标作为标签，**针对每个分组**构建一个对应的MR定位模型。

实验结果表明，分组后一些分组Group的数据较少（数据点过于稀疏），导致针对单个分组构建的定位模型误差较大，从而增大全局误差，影响定位效果。

**基于聚类的定位模型：**

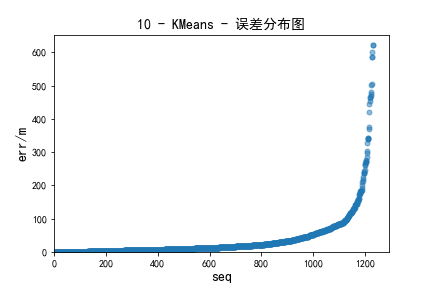
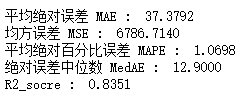
结合上述分析，为了增大单个回归定位模型的训练数据量，基于hw2 步骤b)完成的工作——在按照主基站分组后，以分组Group为粒度进行聚类，得到的结果中每一簇中含有不同的Group中的MR数据，然后依然是以相对坐标（以每个分组的主基站位置为参照点）作为标签，针对聚类之后的每个簇下多个分组的数据构建一个对应的MR定位模型。

**备注:**

为什么不结合栅格来进行定位呢?实际上，这个问题困扰了很长时间。严格来讲，无论怎么定位，最后定位的粒度应该都是一条MR的数据或者的一个尺寸不大的栅格。如果采用组中分栅格来进行定位，整体流程应该是:给定一条MR的数据，首先我们需要预测属于哪一个簇（然后簇的特征我们并没有分析的指标)，在簇中，我们需要预测属于哪一个主基站，进一步，预测属于哪一个栅格。很显然，在这个层次聚类的过程中，人为提取的特征往往会导致大量信息丢失。在某种意义上，人为提取的特征(MR、栅格、主基站、簇)并没有较强的实际物理意义，即内部可能并没有因果关系等。虽然刚才的过程是典型的层次聚类，但过程已经丢失大部分可用的信息，并不实际

* 1. **结果**
* **K-Means聚类**

**误差分析（全局）**

 ****

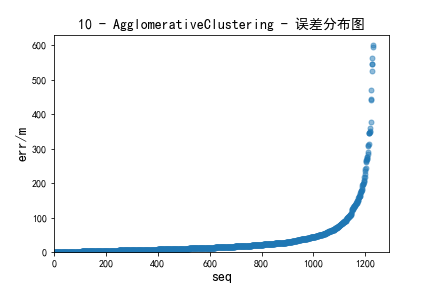
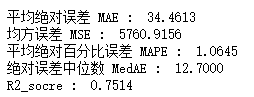
**地图可视化预测效果（全局）**

****

[**..\result data\cluster\Question c)\10 - KMeans - tj\_map.html**](../result%20data/cluster/Question%20c)/10%20-%20KMeans%20-%20tj_map.html)

* **层次聚类**

**误差分析（全局）**

 ****

**地图可视化预测效果（全局）**

****

[**..\result data\cluster\Question c)\10 - AgglomerativeClustering - tj\_map.html**](../result%20data/cluster/Question%20c)/10%20-%20AgglomerativeClustering%20-%20tj_map.html)

* 1. **关键代码**

**构建单个簇的定位模型**

|  |
| --- |
| 1. def LocateModel(df): 2. *# 划分数据集* 3. X = df[feature\_names] 4. Y = df[class\_names] 5. from sklearn.model\_selection import train\_test\_split 6. X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=200) 7. *# 特征工程* 8. *# 特征预处理：归一化* 9. from sklearn.preprocessing import MinMaxScaler 10. scaler = MinMaxScaler() 11. X\_train = scaler.fit\_transform(X\_train) 12. X\_test = scaler.transform(X\_test) 13. *# 模型训练* 14. *# 随机森林回归* 15. from sklearn.ensemble import RandomForestRegressor 16. estimator = RandomForestRegressor() 17. estimator.fit(X\_train, Y\_train[['r\_Longitude', 'r\_Latitude']]) 18. *# 误差计算* 19. *# 预测的相对经纬度坐标* 20. pred\_rloc = estimator.predict(X\_test) 21. *# 预测的绝对经纬度坐标(还原为原始位置)* 22. pred\_loc = (pred\_rloc + Y\_test[['m\_Longitude', 'm\_Latitude']]).values 23. *# 误差* 24. err = [distance(p\_loc,t\_loc) for p\_loc, t\_loc in zip(pred\_loc, Y\_test[['Longitude','Latitude']].values)] 25. return Y\_test, pred\_loc, err |

**对每一个簇构建定位模型 + 显示结果**

|  |
| --- |
| 1. # 对每一个簇构建定位模型 2. err\_all = [] 3. pred\_loc\_all = [] 4. Y\_test\_all = pd.DataFrame() 5. all\_MR = [] 7. for cluster\_groups in clusters.items(): 8. # 该簇下所有MR样本 9. cluster\_MR = pd.DataFrame(columns=df.columns.tolist()) 10. id\_cluster = cluster\_groups[0] 11. groups = cluster\_groups[1] 12. for group\_idx in groups: 13. RNCID = int(mbs.iloc[group\_idx,:]['RNCID1']) 14. CellID = int(mbs.iloc[group\_idx,:]['CellID1']) 15. group\_MR = df[df["RNCID1"] == RNCID] 16. group\_MR = group\_MR[group\_MR["CellID1"] == CellID] 17. cluster\_MR = cluster\_MR.append(group\_MR) 18. #     print(cluster\_MR) 19. all\_MR.append(cluster\_MR) 21. # 针对单个簇训练模型 22. Y\_test, pred\_loc, err = LocateModel(cluster\_MR) 23. # 误差分析 24. ErrorAnalysis(K, err, str(K) + ' - ' + model + '\_%d'% id\_cluster + ' - 误差分布图') 25. # 地图可视化 26. MapVisual(K, pred\_loc, Y\_test, str(K) + ' - ' + model + '\_%d'% id\_cluster + ' - tj\_map.html') 27. # 加入到全局list 28. pred\_loc\_all.extend(pred\_loc) 29. Y\_test\_all = pd.concat([Y\_test\_all, Y\_test], axis=0) 30. err\_all.extend(err) # 在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表) 31. # 误差分析（全局） 32. ErrorAnalysis(K, err\_all, str(K) + ' - ' + model + ' - 误差分布图') 33. # 地图可视化定位效果（全局） 34. MapVisual(K, pred\_loc\_all, Y\_test\_all, str(K) + ' - ' + model + ' - tj\_map.html') |

* 1. **对比：1b定位模型**

将本方法与hw1步骤b)针对每个分组的定位模型进行对比分析。

从模型构建的过程上分析，2c-基于聚类的定位模型相较于1b-基于分组的定位模型，优势在于**单个回归模型的数据量得到了提升**，从这方面看能有效改进单个模型的定位精度；但存在一个问题是，一个分组MR样本的相对坐标都出于相同的主基站参考系中，而在**聚类后的一个簇中，多个分组的MR样本的相对坐标分别处于不同的主基站参考系中**，这给回归预测带来了新的困难。

从下列评价指标的结果来看，两个模型的定位效果相当，可能是上述分析的优势和劣势分别对定位精度的**影响基本抵消**。

**对比分析：**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **MAE** | **MSE** | **MAPE** | **MedAE** | **R2** |
| **2c 基于聚类的定位模型** | 34.4613 | 5760.9156 | 1.0645 | 12.7000 | 0.7514 |
| **1b 基于分组的定位模型** | 34.2806 | 6304.3304 | 1.038 | 13.0000 | 0.7577 |