数据挖掘第2次编程作业

> 学号：1951976
>
> 姓名：李林飞

a. 采用相对位置信息进行聚类分析

1. 核心思路

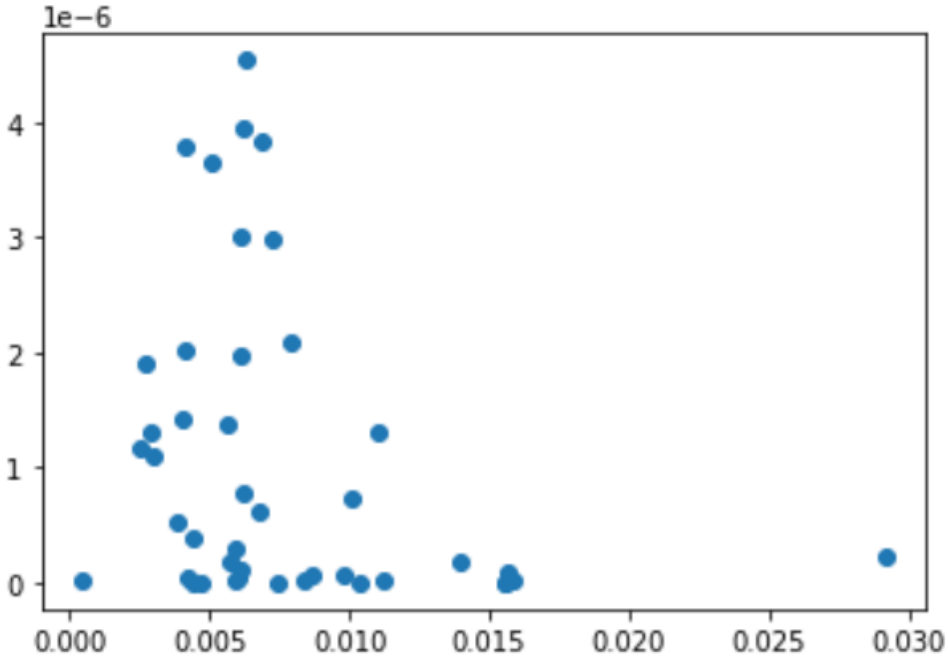
根据主基站将MR数据进行分组，计算相对距离。对于一个主基站分组，我选用相对距离的均值和方差作为特征进行聚类分析。然后采用手肘法计算最优的K值。

2. 实现过程

- 提取每一组的特征信息

```
# 选取所有组的特征信息
group_feature = []
for i in range(len(data)):
    dist = []
    for index, row in data[i][1].iterrows():
        d = np.sqrt(row['lo']**2 + row['la']**2)
        dist.append(d)
    group_feature.append([np.mean(dist), np.var(dist)])
print(group_feature[0])
# [0.011040551965685848, 1.308053782087067e-06]
```

每组特征点分布图



- **采用手肘法**计算最优K值。

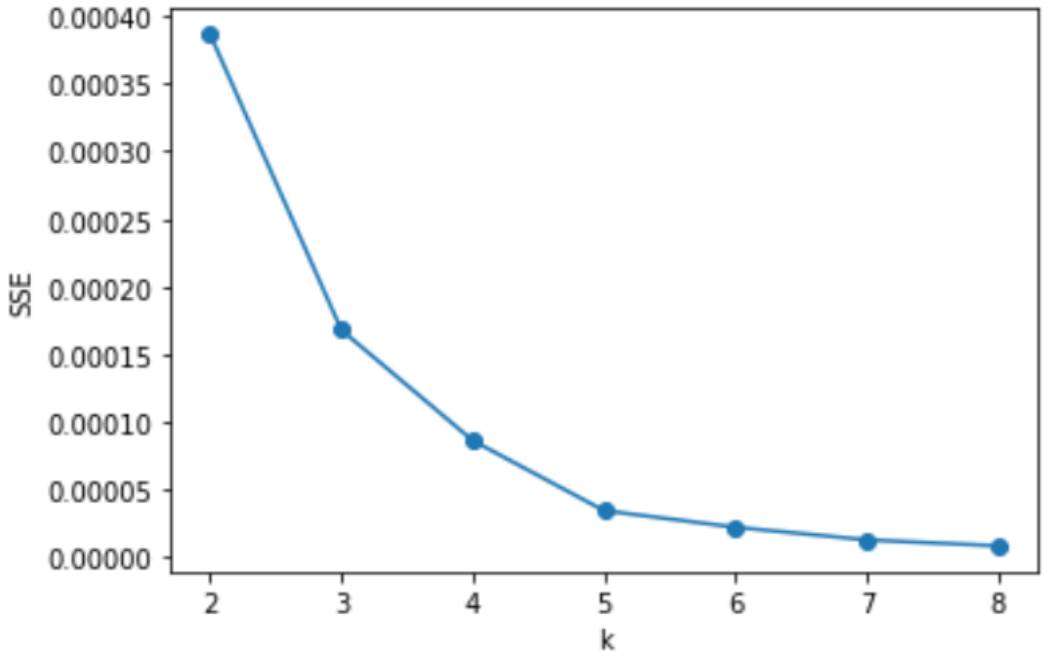
手肘法的核心指标是SSE(sum of the squared errors，误差平方和):

$$SSE = \sum_{i=1}^{k} \sum_{p \in C_i} |p - m_i|^2$$

其中，C_i是第i个簇，p是C_i中的样本点，m_i是C_i的质心（C_i中所有样本的均值），SSE是所有样本的聚类误差，代表了聚类效果的好坏。

手肘法的核心思想是：随着聚类数k的增大，样本划分会更加精细，每个簇的聚合程度会逐渐提高，那么误差平方和SSE自然会逐渐变小。并且，当k小于真实聚类数时，由于k的增大会大幅增加每个簇的聚合程度，故SSE的下降幅度会很大，而当k到达真实聚类数时，再增加k所得到的聚合程度回报会迅速变小，所以SSE的下降幅度会骤减，然后随着k值的继续增大而趋于平缓，也就是说SSE和k的关系图是一个手肘的形状，而这个肘部对应的k值就是数据的真实聚类数。当然，这也是该方法被称为手肘法的原因。

```
# 通过每一组的特征信息进行聚类
# 手肘法调优：
sse = []
for i in range(2,9):
    mid = KMeans(n_clusters=i, init='k-means++', n_init=10,
max_iter=200)
    mid.fit(X)
    sse.append(mid.inertia_) #inertia为簇内误差平方和

plt.plot(range(2,9), sse, marker='o')
plt.xlabel('k')
plt.ylabel('SSE')
plt.show()
```

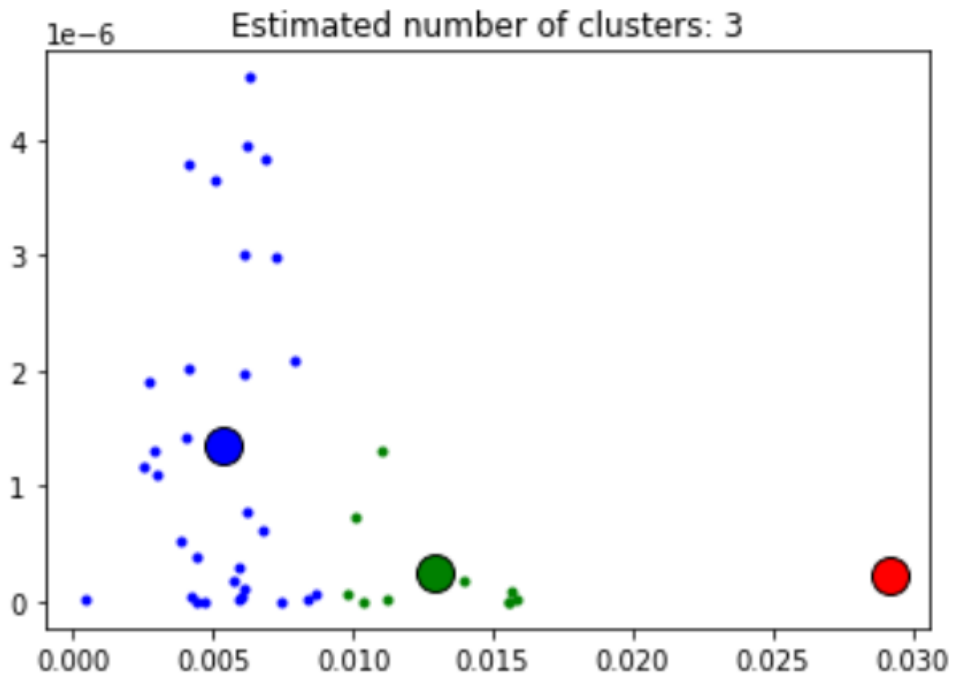


当聚类个数未达到最优个数k时，随着聚类个数的增加，SSE下降较快。达到最优个数以后，SSE下降缓慢。斜率最大处即为最优k值。上图显示了最优的聚类结果是3个。下面我们显示聚类的效果。

- 最优K值为3，展示聚类效果

```
# K=3的聚类效果    训练
model = KMeans(n_clusters=3, init='k-means++', n_init=10, max_iter=200)
model.fit(X)

labels = model.labels_  # 获取聚类标签
cluster_centers = model.cluster_centers_  # 获取聚类中心
inertia = model.inertia_  # 获取聚类准则的总和

labels_unique = np.unique(labels)
n_clusters_ = len(labels_unique)

plt.figure(1)
plt.clf()

colors = cycle("bgrcmykbgrcmykbgrcmykbgrcmyk")
for k, col in zip(range(n_clusters_), colors):
    my_members = labels == k
    cluster_center = cluster_centers[k]
    plt.plot(X[my_members, 0], X[my_members, 1], col + ".")
    plt.plot(
        cluster_center[0],
        cluster_center[1],
        "o",
        markerfacecolor=col,
        markeredgecolor="k",
        markersize=14,
    )
plt.title("Estimated number of clusters: %d" % n_clusters_)
```

```
29 plt.show()
```



聚类结果显示：红色点由于偏离太远而单独成为一组，这显然不太合理。这主要是由于相对距离都减小(0.001差距)。

由于k-means算法依赖于距离，因此必须保证每个维度的距离单位一致，所以对数据进行归一化处理

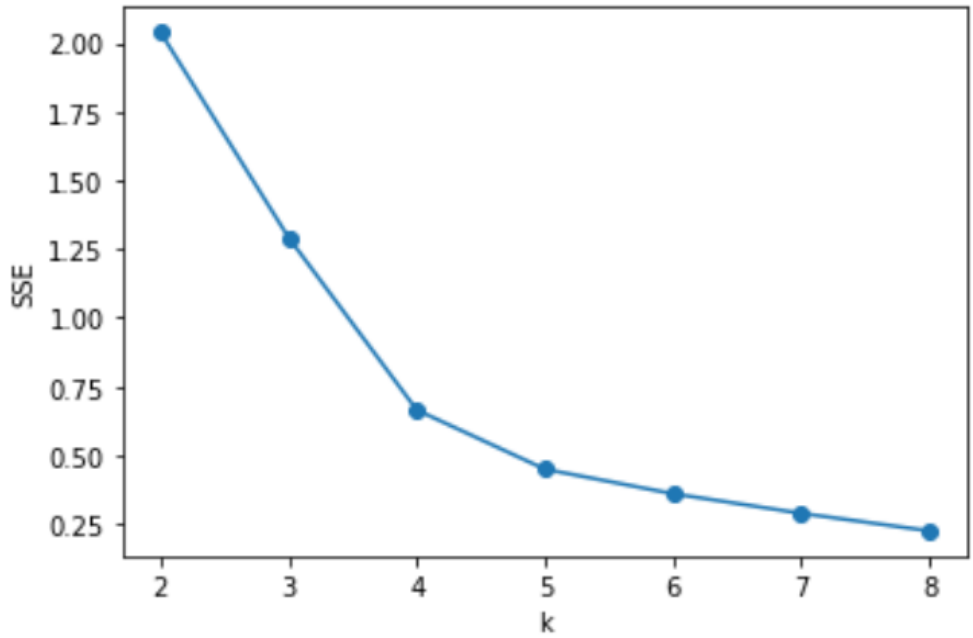
3. 优化：数据归一化处理

数据归一化：由于k-means算法依赖于距离，因此必须保证每个维度的距离单位一致。

- 归一化处理

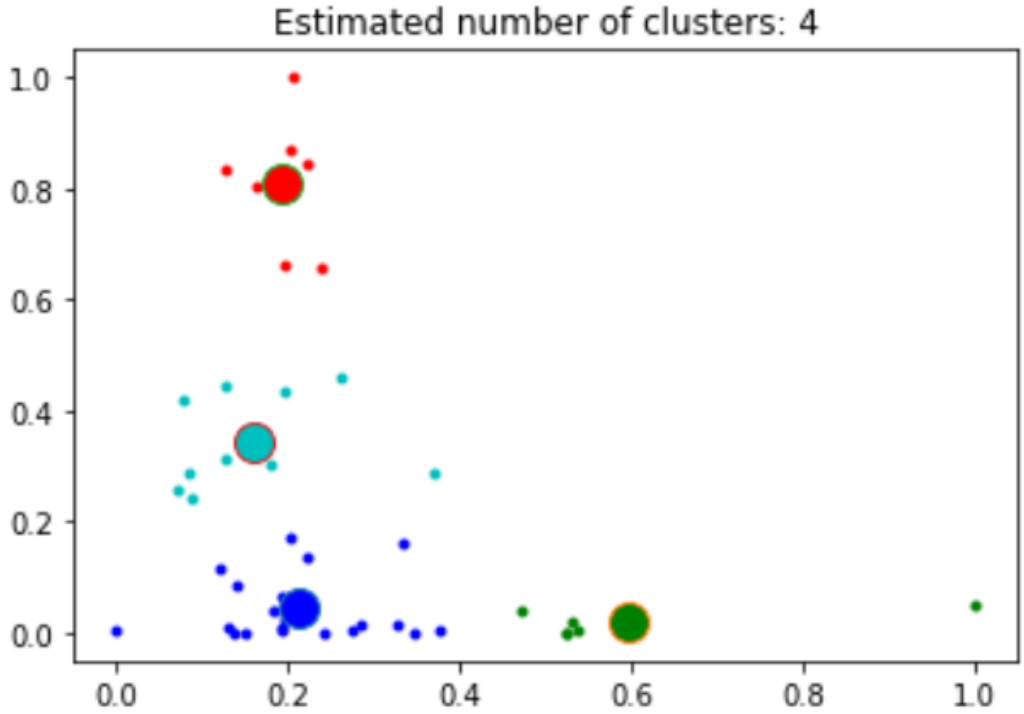
```
1 # 归一化
2 minmax = MinMaxScaler()
3 scaler_data = minmax.fit_transform(X)
```

- 归一化后使用手肘法寻找最优K值：此时最优K值是4



- 聚类效果：相比上面的结果更为合理，此时的簇内误差平方和也只有0.6644810230818553。

```
0.6644810230818553
```



b. 采用信号强度进行聚类分析

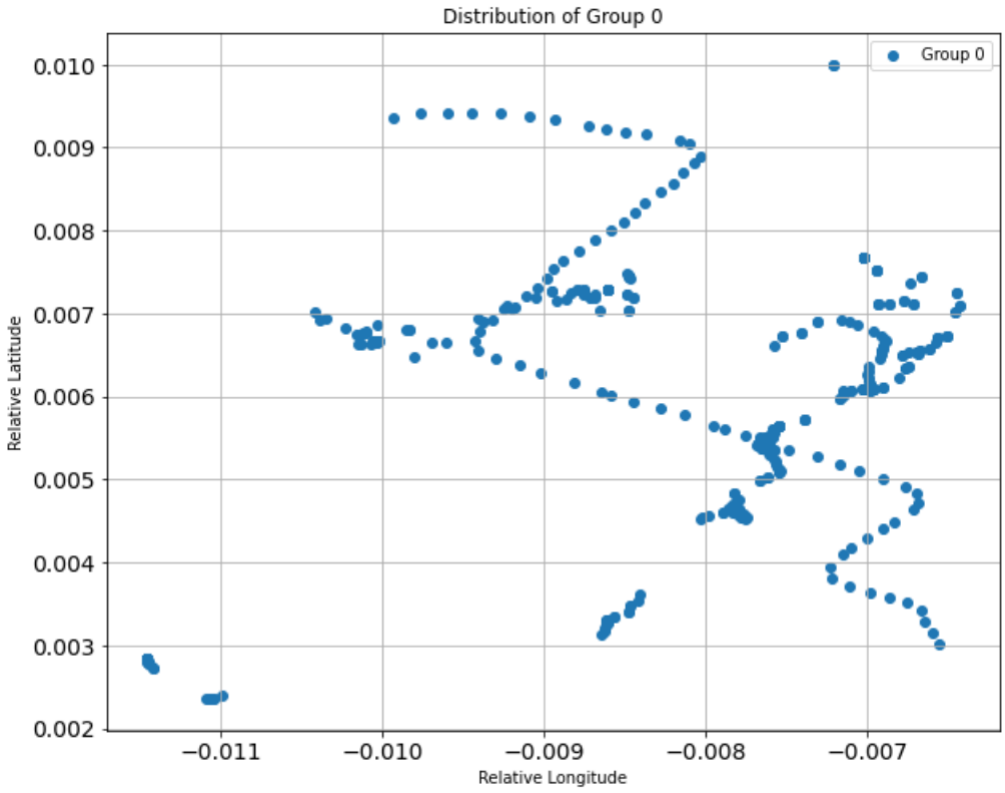
1. 核心思路

主要采用信号强度信息作为特征来表征一个主基站。这里采用根据相对位置对一个组里的MR数据划分栅格，然后统计每一个栅格中的两个主要特征：到主基站的相对距离的欧式距离和信号强度分布直方图。然后，将一个主机站中的所有栅格特征作为输入进行聚类，分为一类，主要是获取中心点的”坐标“。该坐标作为主基站的特征，然后再对主基站进行聚类分析。这样，一个主基站的两个特征中，都包含了相对距离和信号强度的信息。

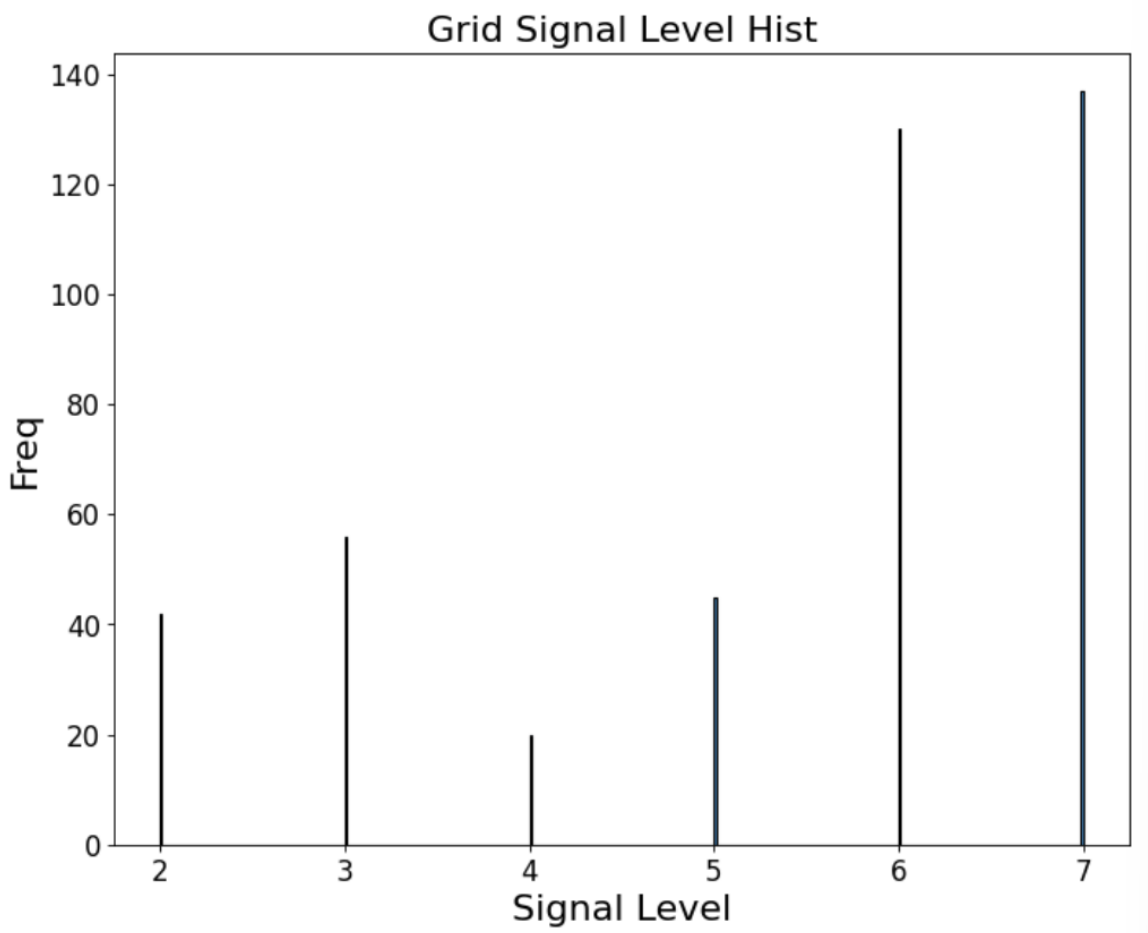
2. 实现过程

- 首先，将每一个主基站中的MR数据根据相对位置划分栅格

```
#计算栅格的经纬度增加量大小Lon和Lat
deltaLon = 0.001
deltaLat = 0.001
#lon方向是lonsnum个栅格
lonsnum = int((max_lo-min_lo)/deltaLon)+1
#lat方向是latsnum个栅格
latsnum = int((max_la-min_la)/deltaLat)+1

# 划分栅格
group_grid = []
for i in range(lonsnum):
    for j in range(latsnum):
        # 每一个栅格的范围
        lo_start =  i * deltaLon + (min_lo - deltaLon / 2)
        lo_end = (i+1) * deltaLon + (min_lo - deltaLon / 2)
        la_start = j * deltaLat + (min_la - deltaLat / 2)
        la_end = (j+1) * deltaLat + (min_la - deltaLat / 2)
        grid_data = []
        for index, row in g0.iterrows():
            if lo_start <= row['lo'] < lo_end and la_start <= row['la']
< la_end:
                grid_data.append(row)
        group_grid.append([i,j,grid_data])
```



- 统计每一个栅格上的信号强度分布：横坐标表示信号的等级，纵坐标表示在该栅格中出现此等级的次数。计算栅格中心点到主基站的距离和直方图的均值作为一个栅格的特征值。



- 计算所有主机站的栅格特征

```
import math
Group_Feature = []   # 每个主基站的栅格特征

#计算栅格的相对经纬度增加量大小Lon和Lat
deltaLon = 0.001
deltaLat = 0.001

for i in range(43):
    one_group_feature = []  # 一个主基站的栅格特征集
    g = data[i][1]   # 一个主基站的数据
    # 找到划分栅格的尺度
    min_lo = min(g.loc[:, 'lo'])
    max_lo = max(g.loc[:, 'lo'])
    min_la = min(g.loc[:, 'la'])
    max_la = max(g.loc[:, 'la'])
    #lon方向是lonsnum个栅格
    lonsnum = int((max_lo-min_lo)/deltaLon)+1
    #lat方向是latsnum个栅格
    latsnum = int((max_la-min_la)/deltaLat)+1
    # 对一个主基站划分栅格
    group_grid = []
    for i in range(lonsnum):
        for j in range(latsnum):
            # 每一个栅格的范围
            lo_start =  i * deltaLon + (min_lo - deltaLon / 2)
```

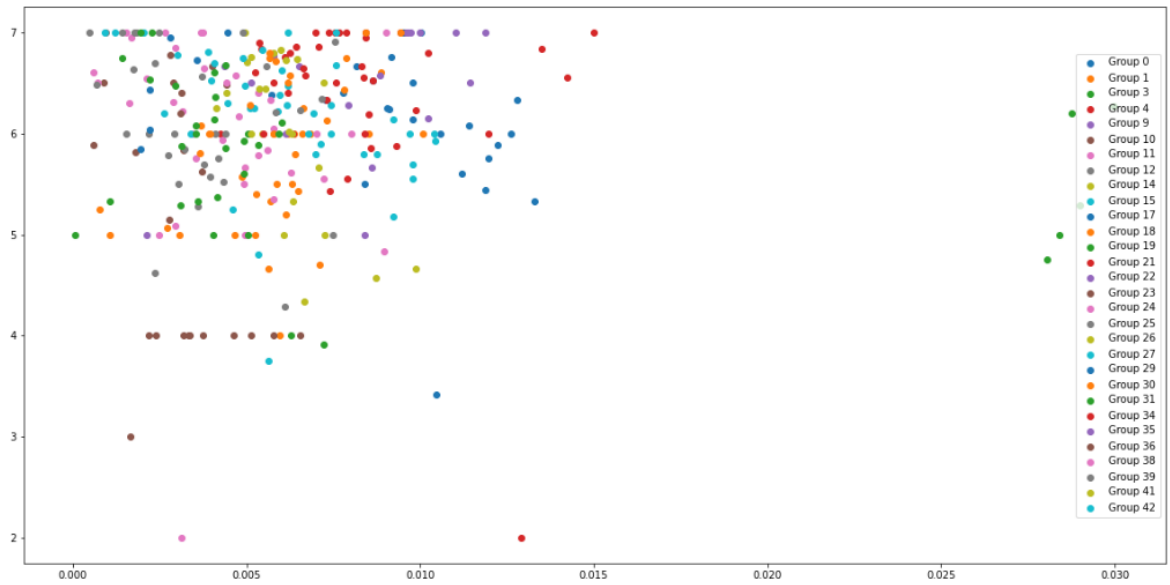
```
            lo_end = (i+1) * deltaLon + (min_lo - deltaLon / 2)
            la_start = j * deltaLat + (min_la - deltaLat / 2)
            la_end = (j+1) * deltaLat + (min_la - deltaLat / 2)
            # 计算栅格相对于主基站的中心位置  距离
            center_lo = (lo_start + lo_end) / 2
            center_la = (la_start + la_end) / 2
            center_dist = math.sqrt(center_lo**2 + center_la**2) # 栅格到达主基站的距离
            grid_data = []   # 栅格数据
            for index, row in g.iterrows():
                if lo_start <= row['lo'] < lo_end and la_start <= row['la'] < la_end:
                    grid_data.append(row)
            group_grid.append([i,j,center_dist,grid_data])  # ij栅格位置，栅格中心距离，mr数据

    # 提取主基站的每一个栅格的特征
    for k in range(1, len(group_grid)):
        # 统计每一个栅格中的信号强度数据
        Grid_Signal_Level = []
        for m in range(len(group_grid[k][3])):
            mr = group_grid[k][3][m]   # 该数据中，LinkNum表示邻近基站的数量，即比较信号强度时最高的顺序号
            Grid_Signal_Level.append(mr['LinkNum'])
        # 每一个栅格的直方图特征 均值和方差
        arr_mean = np.mean(Grid_Signal_Level)  # 均值可以表征该栅格的信号强度
#         arr_var = np.var(Grid_Signal_Level)    # 方差表征信号强度的分布情况
        if not math.isnan(arr_mean):  # 去除nan
            one_group_feature.append([group_grid[k][2], arr_mean])    # 主机站栅格信息列表
    Group_Feature.append({'lonsnum': lonsnum, 'latsnum': latsnum, 'grid': one_group_feature})  # 一个主基站的特征信息
```

- 展示一个主基站的特征信息

In [127]:
```
# 第一组主基站的特征数据
Group_Feature[0]  # 根据该数据可以确定主基站中栅格的分布情况以及每个栅格的特征数据
```

Out[127]:
```
{'lonsnum': 6,
 'latsnum': 8,
 'grid': [[0.01223502468614056, 5.888888888888889],
  [0.012783457633616546, 6.333333333333333],
  [0.011392639249553306, 6.083333333333333],
  [0.011979699039227267, 5.75],
  [0.013302447484221207, 5.333333333333333],
  [0.009095369650017275, 6.235294117647059],
  [0.009510347473697035, 7.0],
  [0.0100083799423511, 7.0],
  [0.010577742153713902, 6.0],
  [0.01120756838348338, 5.6],
  [0.01188825256589279, 5.444444444444445],
  [0.012611562515027893, 6.0],
  [0.008174481578084869, 6.666666666666667],
  [0.00863383513106714, 6.521739130434782],
  [0.009179546234452406, 6.754716981132075],
  [0.009797194959295725, 6.5],
  [0.010474062682190219, 3.4126984126984126],
  [0.007274513665561599, 6.333333333333333],
  [0.007787137411812036, 6.4],
  [0.008388114750672101, 5.5],
  [0.00905988019073338, 6.25],
  [0.009787869485768765, 6.136363636363637]]}
```

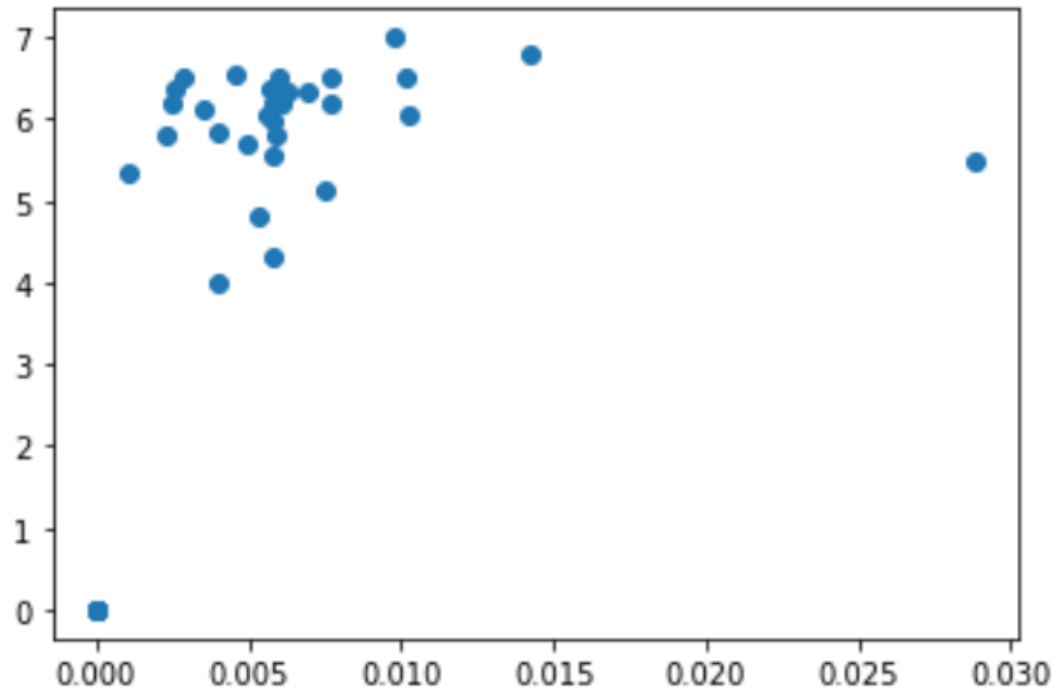
- 每一个主基站中栅格信息的分布：有的主基站中的栅格中不存在MR数据，所以计算为nan。所有为nan的单独作为一组。



- 计算每一个主基站中的栅格特征作为主基站的特征：主要是对栅格数据特征进行聚类分析，得到所有栅格的中心点坐标，作为主基站的特征信息。

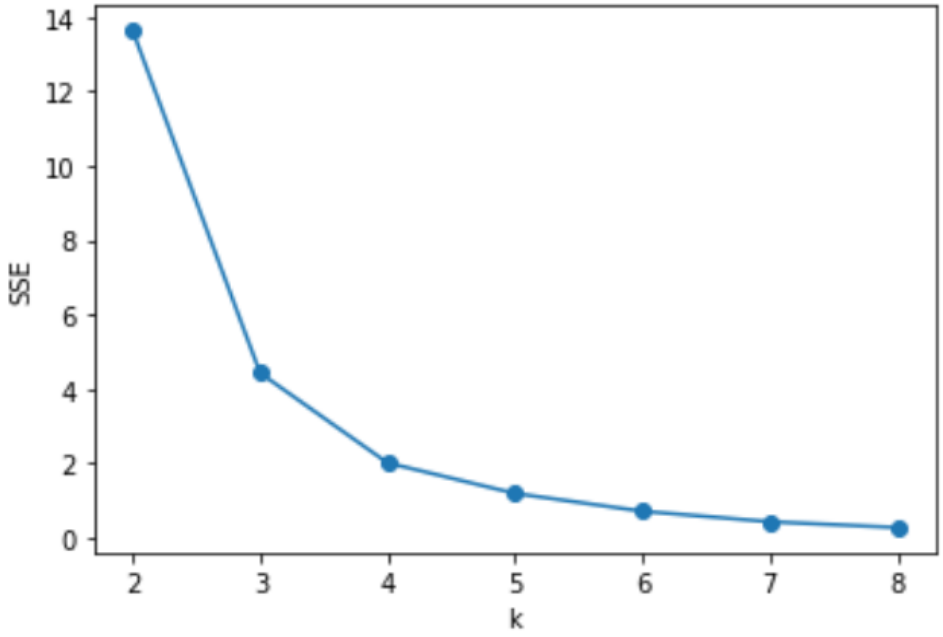
```
# 每一组中的栅格特征数据进行聚类获取聚合后的特征数据
group_clusters = []
for i in range(len(Group_Feature)):
    X = np.array(Group_Feature[i]['grid'])
    if len(X) is not 0:  # 主基站的栅格中不存在数据
        mid = KMeans(n_clusters=1, init='k-means++', n_init=10,
max_iter=200)   # 计算中心点,聚类为一类
        mid.fit(X)
        group_clusters.append([mid.cluster_centers_[0][0],
mid.cluster_centers_[0][1]])   # 簇的中心点
    else:
        group_clusters.append([0,0])
```

主基站的特征信息：由于经过多次特征提取，横纵坐标的物理意义已经不明显了，基本与MR数据看不出直接关联性。但纵坐标中隐藏着主基站的信号强度信息，横坐标中隐藏着主基站内部栅格分布信息。



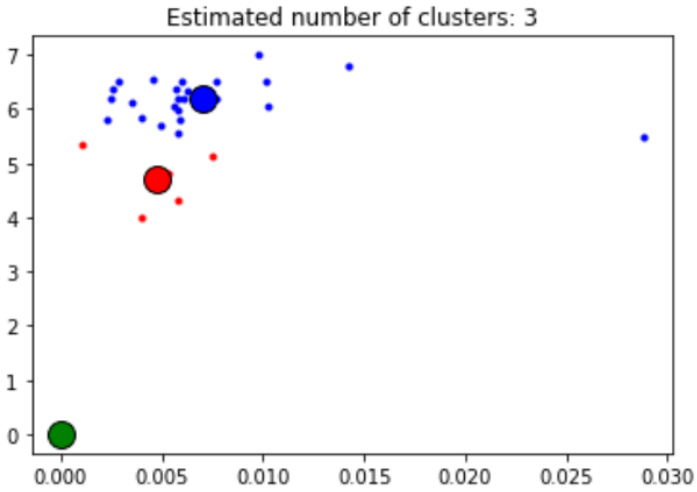
- 使用K-means进行聚类：使用归一化和不使用归一化的最优K值都是3。主要原因是该特征与距离关系不大，归一化影响较小。

手肘法寻找最优K值：



根据手肘法可知，最优的K为3,下面显示K为3的具体聚类结果。

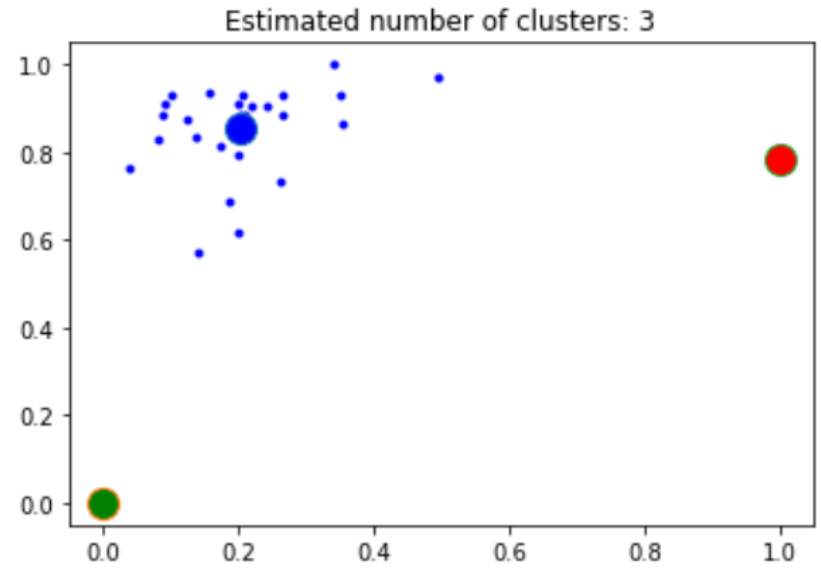
- 聚类结果展示：注意绿色的点！！



注意！！！图中绿色聚类结果中含有12组数据，因为我前面将没有提取出栅格中的组的数据默认为$[0,0]$。从分布上来看，与第一问中的最优K值相似，都是3。

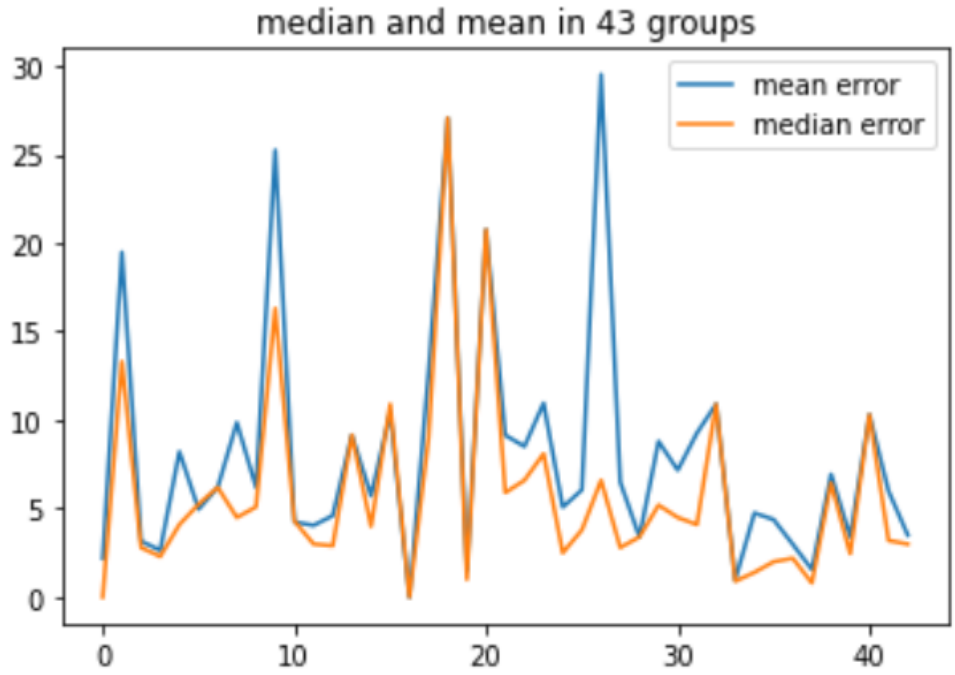
归一化后聚类结果：簇内误差平方和为0.5333879734579469。

0. 5333879734579469



c. 聚类定位模型

- **在第一次作业的b小题中，我们的定位模型是**：在完成每个分组的MR记录相对位置计算之后，针对每个分组构建一个对应的MR定位模型，不过该模型是以MR记录与主基站的相对位置作为标签。使用处理好的训练集用于训练模型，测试数据集用于测试统计，通过上述随机森林模型预测测试数据的相对位置。有趣的是，我第一次作业中人为-999的填充太随便了，然后分析后发现缺失的辅助基站数据很有可能是现实中根本不存在该基站，所有没有采取填充的方法。同时，为了解决缺失数据的问题，我在分组的基础上又根据缺失值进行分组，导致误差高的离谱。这里一个主要的原因就是，存在一些主基站分组中只有3条MR数据，而着3条MR数据中的缺失情况也可能不一样，这就导致可能训练时只有一条数据，然后进行预测，误差几乎就是随机值，所有最终的平均误差高达2000左右。在第2次作业中，我对第一次作业中进行了一些改动，采用-999的填充方案，并将所有属性作为特征，使用随机森林进行回归预测，所有组的平均误差和为6.2左右！！！



```
27组误差：  2.8 6.501481481481481 13.2
28组误差：  3.4 3.4 3.5
29组误差：  5.2 8.796 21.2
30组误差：  4.5 7.170588235294119 18.2
31组误差：  4.1 9.267605633802816 14.6
32组误差：  10.9 10.9 10.9
33组误差：  0.9 0.9666666666666668 1.3
34组误差：  1.4 4.752631578947369 20.0
35组误差：  2.0 4.385 13.2
36组误差：  2.2 2.991111111111111 6.8
37组误差：  0.8 1.563636363636364 2.2
38组误差：  6.45 6.94 15.2
39组误差：  2.45 3.407142857142857 8.2
40组误差：  10.3 10.3 12.8
41组误差：  3.2 5.989583333333333 13.8
42组误差：  3.0 3.505882352941176 6.2
所有组的平均误差：  6.584538653366584
```

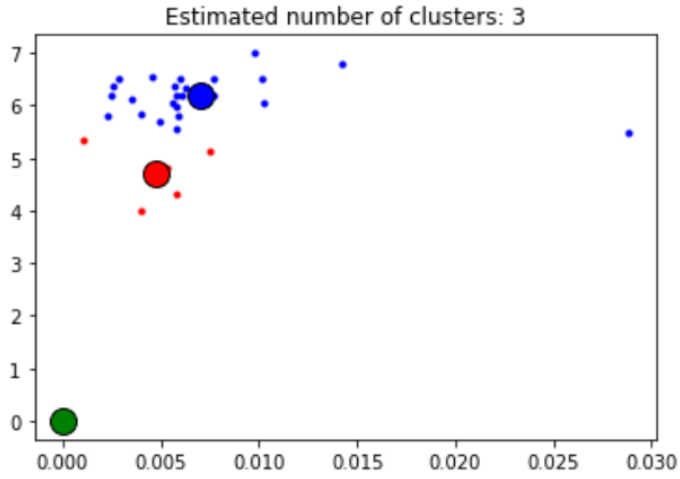
- **在此次作业的a和b两个小问中**：主要工作使使用每个主基站分组中的不同特征将主基站进行聚类。a小题中采用相对位置信息进行聚类，实验结果表明在采用原始数据聚类是最优K值为3，对数据进行归一化处理后的最优K值为4。在b题中，主要采用相对位置和信号强度结合的信息作为主基站特征来进行分类，实验结果表明无论数据是否进行归一化处理，最优K值都是3.

- **结合上面的分析，给出一个定位模型**：在第一次作业b小题的基础上，我们改变分组的方式，使分组训练的数据集分布的更加合理。具体为：通过a和b中的聚类结果对MR数据进行分簇，每一簇中含有不同的主基站中的MR数据。然后，对每一簇中的MR数据使用随机随林模型进行回归预测，作为对比，我们依然采用相对位置作为标签。这样，结合主基站的经纬度和预测的相对位置就可以实现定位。

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. 使用b题中聚类结果进行分组回归预测定位

- 分组信息：



注意：绿色的簇代表没有从栅格中提取到栅格的特征信息，我将这些组单独放在一个簇中，该簇中一共有12组主基站的数据。

- 组装分组数据集：根据b题中的分组方案进行分组回归预测。

组装数据集

In [43]:
```
# 经过聚类后的分组数据
X0 = pd.DataFrame(data[0][1])
X1 = pd.DataFrame(data[2][1])
X2 = pd.DataFrame(data[14][1])
for i in range(1,len(data)):
    if i != 0 and labels[i] == 0:
        X0.append(data[i][1])
    if i != 2 and labels[i] == 0:
        X1.append(data[i][1])
    if i != 14 and labels[i] == 0:
        X2.append(data[i][1])
```

In [47]:
```
print('X0:', X0.shape[0])
print('X1:', X1.shape[0])
print('X2:', X2.shape[0])
```

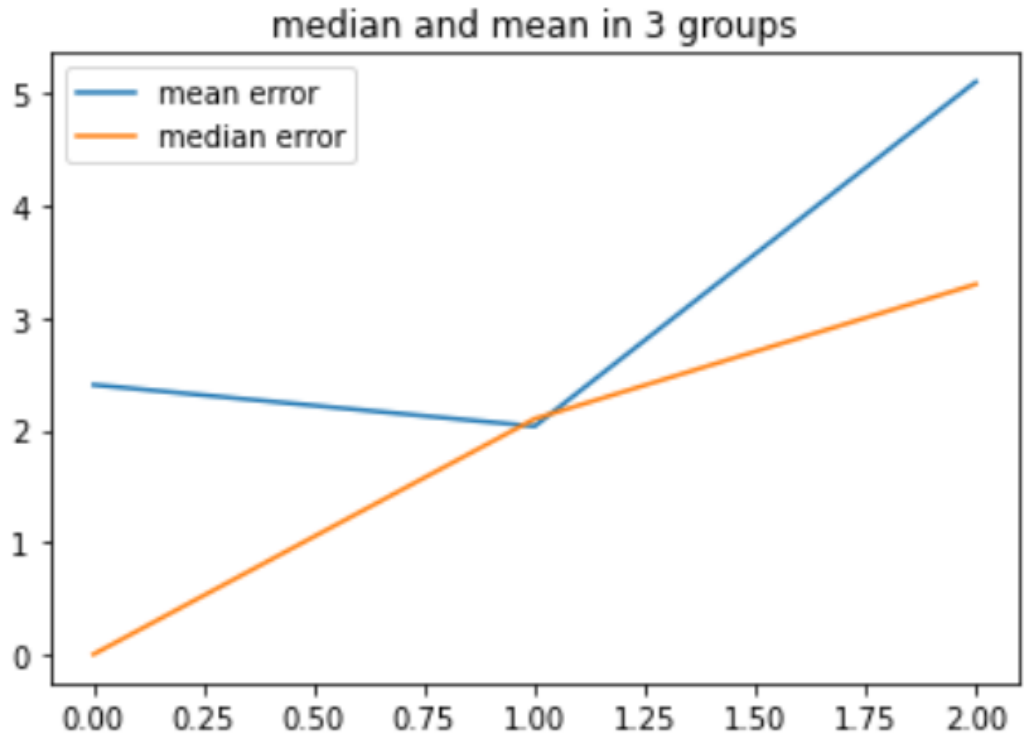
```
X0: 819
X1: 15
X2: 67
```

- 使用随机森林进行分组回归预测

```
from sklearn.model_selection import train_test_split  # 数据集划分
from sklearn.ensemble import RandomForestRegressor

# 按照主机站进行分组训练
data = [X0,X1,X2]
med = []  # 中位数
```

```
mea = [] # 均方误差
nin = [] # 分位数
loss = [] # 总误差

# 总共43组
for i in range(len(data)):
    group = data[i] # 一组
    pre = []
    # 以相对坐标作为预测值
    X, Xe, Y, Ye = train_test_split(
        group,
        group[['Longitude', 'Latitude', 'lo', 'la']],
        test_size=0.2,
        random_state=200)
    reg = RandomForestRegressor()
    reg.fit(X, Y[['lo', 'la']].values)
    pred = reg.predict(Xe)
    # 换回真实坐标计算误差
    for bias in pred:
        pre.append([bias[0] + group.iloc[0,-4], bias[1] +
group.iloc[0,-3]])
    err = [distance(p,t) for p, t in
zip(pre,Ye[['Longitude','Latitude']].values)]
    err = sorted(err)
    # 考虑每组样本不同
    loss.extend(err)
    med.append(np.median(err))
    mea.append(np.mean(err))
    print(str(i) +"组误差：", np.median(err), np.mean(err),
err[int(len(err)*0.9)])

# 所有组的误差
print("所有组的平均误差：", np.mean(loss))
```

```
0组误差：  0.0 2.4030487804878047 7.9
1组误差：  2.1 2.033333333333333 4.0
2组误差：  3.3 5.107142857142857 10.6
所有组的平均误差：  2.606077348066299
```

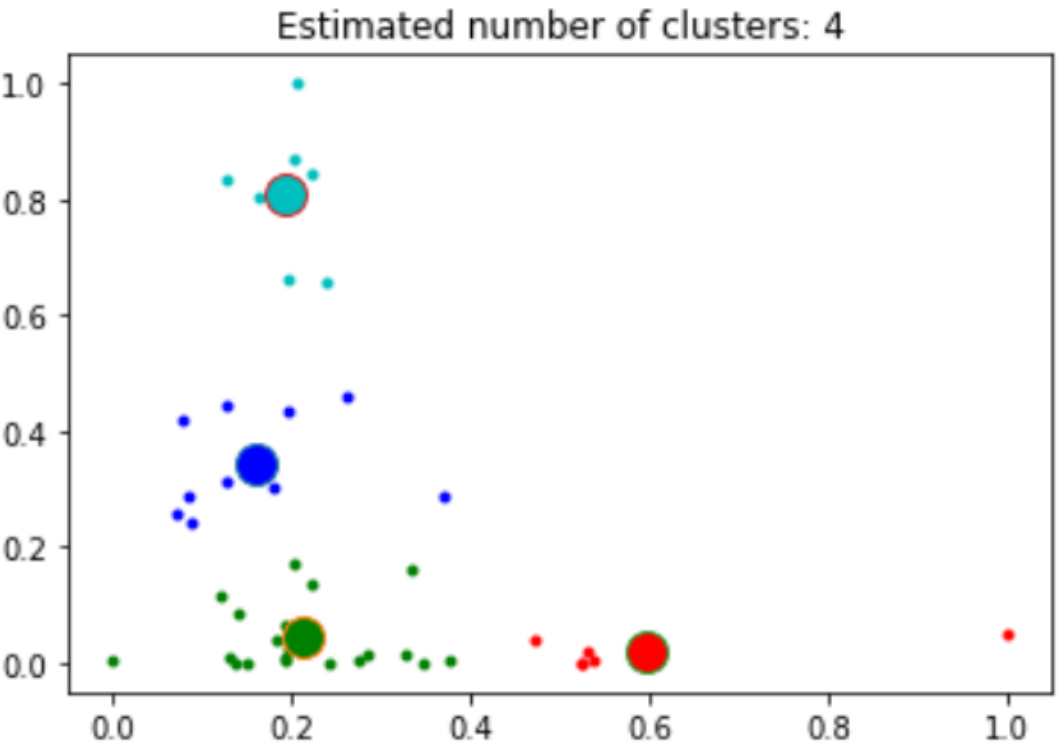


通过使用聚类分组数据集进行回归预测，所有组的平均误差从6.2降为2.6左右。

2. 使用a中相对位置聚类结果进行分组预测定位

在a中，我们聚类结果有两个最优的K值，原始数据上是3，但由于聚类本身对距离非常敏感，所以我将相对位置进行归一化处理后的最优K值为4.这里采用K值为4进行分组训练预测定位。

- 分组信息：



- 组装数据集

In [17]:
```
# 组装数据集
# 经过聚类后的分组数据
X0 = pd.DataFrame(data[0][1])
X1 = pd.DataFrame(data[5][1])
X2 = pd.DataFrame(data[2][1])
X3 = pd.DataFrame(data[1][1])
for i in range(1,len(data)):
    if i != 0 and labels[i] == 0:
        X0.append(data[i][1])
    if i != 5 and labels[i] == 0:
        X1.append(data[i][1])
    if i != 2 and labels[i] == 0:
        X2.append(data[i][1])
    if i != 1 and labels[i] == 0:
        X3.append(data[i][1])
```

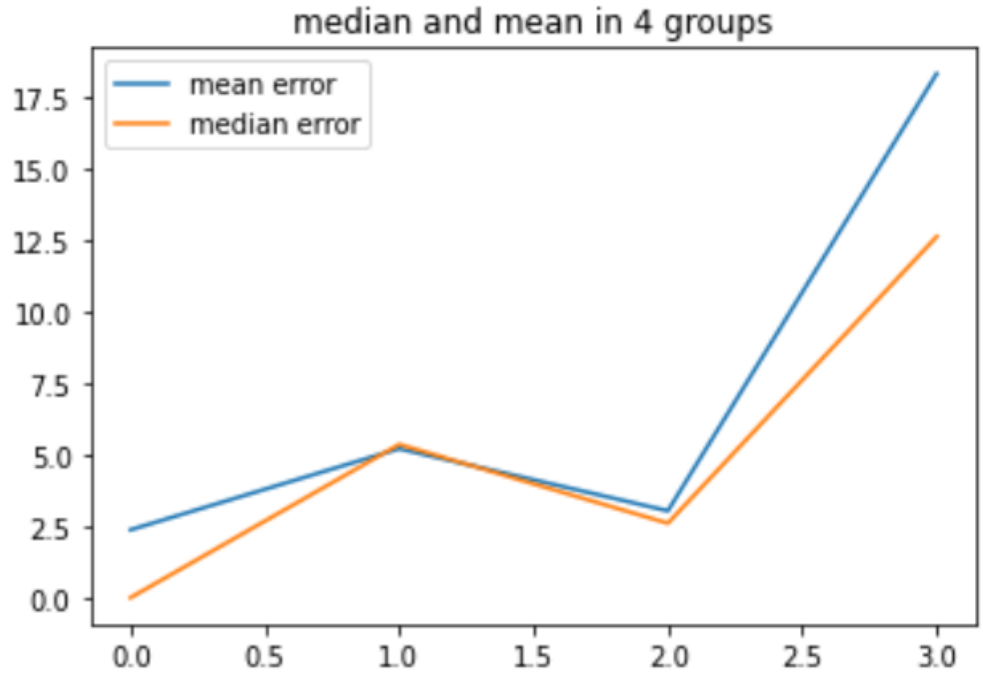
In [18]:
```
print('X0:', X0.shape[0])
print('X1:', X1.shape[0])
print('X2:', X2.shape[0])
print('X3:', X3.shape[0])
# 分的相对而言更加均匀
```

```
X0: 819
X1: 38
X2: 15
X3: 103
```

- 分组训练回归预测结果

```
# 所有组的误差
print("所有组的平均误差：", np.mean(loss))
```

```
0组误差：  0.0 2.3689024390243905 7.2
1组误差：  5.35 5.199999999999999 9.2
2组误差：  2.6 3.033333333333333 6.5
3组误差：  12.6 18.285714285714285 25.6
所有组的平均误差：  4.2
```



3. 定位模型总结与对比

截至目前，使用随机森林回归预测的定位模型一共做过4个，下面进行对比。

模型	是否填充	是否采用所有字段作为特征	回归算法	所有组的平均误差	解释
按照主基站分组训练-1	否	否	随机森林	2000+	主要原因是含有个别组MR数据较少，导致误差陡增
按照主基站分组训练-2	是	是	随机森林	6.58	
按照相对位置聚类结果进行分组训练	是	是	随机森林	4.2	
按照相对位置+信号强度聚类结果进行分组训练	是	是	随机森林	2.61	分组方案较好，可以把关系较大的MR数据分在同一簇中，且每个簇中的MR数据量相对较平衡