ML HW06 – Kernel K-means and Spectral Clustering

309553048_多工所_陳柏丞

- **Code Explanation：**

1. Read image：

```
path = "ML_HW06/image1.png"
img = cv2.imread(path)

row, col, channel = img.shape
```

2. Spatial & Color RBF kernel：

$$k(x, x') = e^{-\gamma_s \|S(x)-S(x')\|^2} \times e^{-\gamma_c \|C(x)-C(x')\|^2}$$

```
# Spatial info
SpatialInfo = [[i, j] for i in range(row) for j in range(row)]
SpatialInfo = np.array(SpatialInfo)

# Spatial Kernel
SpatialKernel = (-1/10000 * pdist(SpatialInfo, 'sqeuclidean'))

#Img info
ImgInfo = img.reshape(row*col, channel)

#Img Kernel
ImgKernel = (-1/10000 * pdist(ImgInfo, 'sqeuclidean'))

NewKernel = np.exp(SpatialKernel + ImgKernel)
```

3. Gram matrix：

$$W(A, B) := \sum_{i \in A, j \in B} w_{ij}.$$

```
# Gram matrix
GramMatrix = squareform(NewKernel)
```

4. Initialize k-means by random or k-means++：

```
def GetMeanIndex(k, mode=False):
    mean_index = []

    # Random k-means
    for i in range(k):
        index = random.randint(0, row*col-1)
        while index in mean_index:
            index = random.randint(0, row*col-1)
        mean_index.append(index)

    if mode:
        # k-means++
        mean_index[0] = random.randint(0, row*col-1)

        for _k in range(1, k):
            distance = np.zeros((row*col, ))
            classify = np.zeros((row*col, ))
            closest = np.zeros((k, ))
            for p in range(row*col):
                for i in range(_k):
                    closest[i] = np.linalg.norm(GramMatrix[p] - GramMatrix[mean_index[i]])
                classify[p] = closest.argmin()

                distance[p] = np.linalg.norm(GramMatrix[p] - GramMatrix[mean_index[int(classify[p])]])

            dis_sum = np.sum(distance)
            val = random.uniform(0, dis_sum)
            count = 0
            while val < 0:
                val -= distance[count]
                count += 1
            mean_index[_k] = count

    return mean_index
```

5. E step in k-means：

```
def Estep(GramMatrix, mean, k):
    classify = np.zeros((row*col, ))
    distance = np.zeros((k, ))
    for i in range(row*col):
        for j in range(k):
            distance[j] = np.linalg.norm(GramMatrix[i] - mean[j])
        classify[i] = distance.argmin()

    return classify
```

6. M step in k-means：

```
def Mstep(GramMatrix, classify, k):
    new_mean = np.zeros((k, row*col))
    for i in range(k):
        cluster = GramMatrix[classify == i]
        new_mean[i] = cluster.sum(axis = 0)
        if len(cluster) > 0:
            new_mean[i] /= len(cluster)

    return new_mean
```

7. K-means & generate gif array：

```
gif = np.zeros((1, row, col))

# Update k-means
flag = True
while flag:
    classify = Estep(GramMatrix, mean, k)
    new_mean = Mstep(GramMatrix, classify, k)

    gif = np.vstack((gif, classify.reshape(1, row, col)))
    print(np.linalg.norm(new_mean - mean))

    if np.linalg.norm(new_mean - mean) < 1:
        flag = False
    mean = new_mean
```

8. Colorize & generate gif：

```
ColorMap = [
    [  0,   0,   0],
    [255,   0,   0],
    [255, 255,   0],
    [  0, 255,   0],
    [ 65, 255, 225],
    [160,  32, 240]
]

colorful_gif = []
for count in range(gif.shape[0]):
    colorful_gif.append([])
    for i in range(gif.shape[1]):
        for j in range(gif.shape[2]):
            color = ColorMap[int(gif[count][i][j])]
            colorful_gif[count].append(color)

colorful_gif = np.array(colorful_gif).reshape(gif.shape[0], row, col, 3)

imageio.mimsave("kmeans.gif", colorful_gif, 'GIF', duration=0.5)
```

9. Degree matrix：

$$d_i = \sum_{j=1}^{n} w_{ij}.$$

```
row_sum = np.sum(GramMatrix, axis=1)
Degree = np.diag(row_sum)
```

10. Ratio cut：

$$L = D - W.$$

```
UnnormalizedLap = Degree - GramMatrix
```

11. Normalized cut：

$$L_{\text{sym}} := D^{-1/2} L D^{-1/2}$$

```
InvSqrtDegree = np.linalg.inv(np.sqrt(Degree))
NormalizedLap = InvSqrtDegree @ UnnormalizedLap @ InvSqrtDegree
```

12. Get k eigenvectors & generate T matrix & do k-means (same as above)：

```
# eig = linalg.eigh(NormalizedLap)
eig = linalg.eigh(UnnormalizedLap)

U = eig[1][:, :k]
norm_U = np.linalg.norm(U, axis=1)

T = U
# T = np.array([U[i]/norm_U[i] for i in range(len(U))])

Tkernel = np.exp((-1/100000 * pdist(T, 'sqeuclidean')))
T_GramMatrix = squareform(Tkernel)
T_mean = T_GramMatrix[mean_index]
```

13. Examine whether the data points within the same cluster do have the same coordinates in the eigenspace of graph Laplacian or not & plotting：

```
if k == 2:
    gif_reshape = gif.reshape(gif.shape[0], row*col)
    cluster1_x = []
    cluster1_y = []
    cluster2_x = []
    cluster2_y = []

    for i in range(row*col):
        if gif_reshape[-1][i] == 1:
            cluster1_x.append(U[i][0])
            cluster1_y.append(U[i][1])
        else:
            cluster2_x.append(U[i][0])
            cluster2_y.append(U[i][1])

    plt.scatter(cluster1_x, cluster1_y, color='r')
    plt.scatter(cluster2_x, cluster2_y)
    plt.show()
```

- **Experiment setting and results：**

1. K-means & normalized cut & ratio cut：
   spatial gamma=1/10000, color gamma=1/10000, k=2

   image1：
   k-means. Normalize. ratio

   image2：
   k-means. Normalize. ratio

2. K-means & normalized cut & ratio cut：
   spatial gamma=1/10000, color gamma=1/10000, k=3

   image1：
   k-means. Normalize. Ratio

   image2：
   k-means. Normalize. Ratio

   spatial gamma=1/10000, color gamma=1/10000, k=4

   image1：
   k-means. Normalize. Ratio

   image2：

k-means. Normalize. Ratio



3. K-means++：

spatial gamma=1/10000, color gamma=1/10000

image1：

k=2. k=3. k=4



image2：

k=2. k=3. k=4



image1(normalized cut)：

k=2. k=3. k=4



image2(normalized cut)：

k=2. k=3. k=4



image1(ratio cut)：

k=2. k=3. k=4



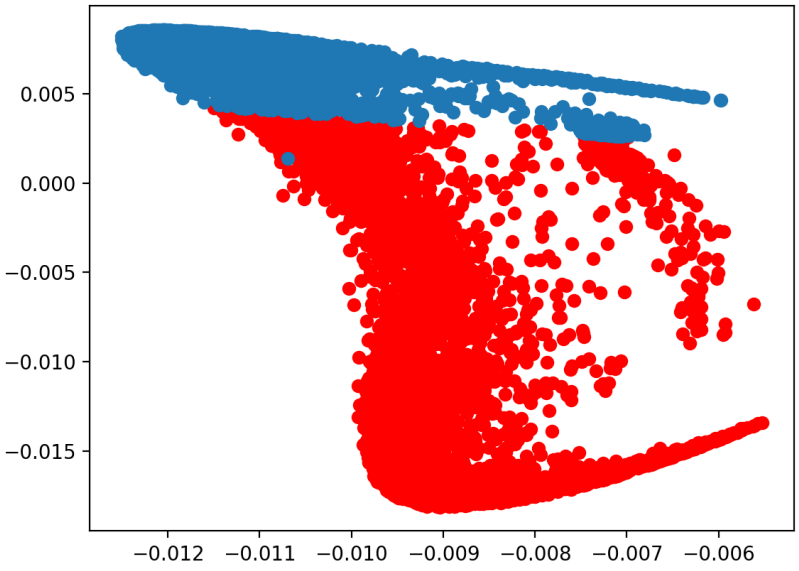
image2(ratio cut)：

k=2. k=3. k=4

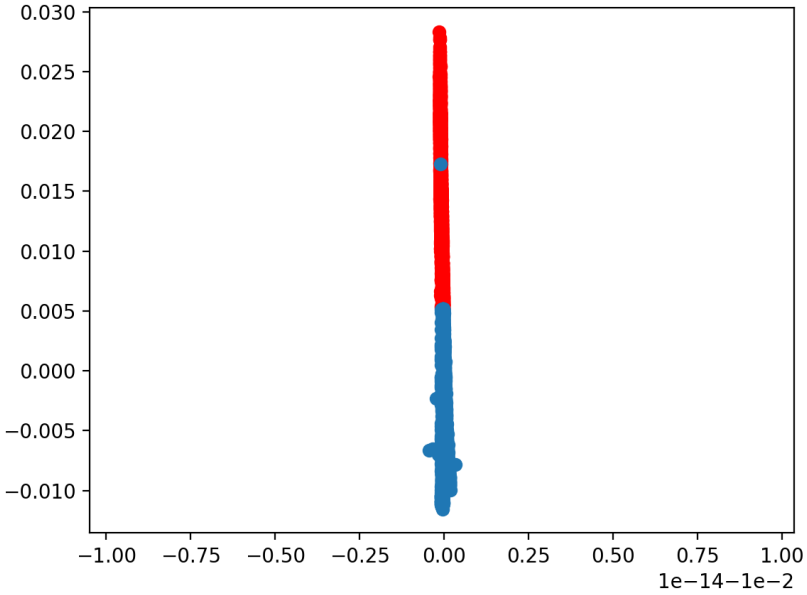


4. Plotting：

   normalized cut：



   ratio cut：



- **Observations and discussion：**

1. In spatial, color gamma=1/10000 and k=2, the results of three situations are the same. However, when k is larger than 2, the result of ratio cut would be worst.

Among the three situations, ratio cut and normalized cut are slower than origin k-means because of inverse of matrix which is 10000x10000.

2. After k-means++, the count of iterations is smaller than performance without k-means. Then, I found that no matter which k is, the results of ratio cut and normalized cut are always clustering in 2 categories. Perhaps, I will adjust the values of spatial gamma and color gamma. When k=3, result of ratio cut is worst among all.

3. Above all, I tried to set different spatial gamma and color gamma.

   image1：
   spatial gamma=1/10000, color gamma=1/100, k=3

   spatial gamma=1/10000, color gamma=1/1000, k=3

   spatial gamma=1/100, color gamma=1/10000, k=3

   spatial gamma=1/1000, color gamma=1/10000, k=3

Summary, spatial gamma and color gamma would lead to different clustering, and I thought spatial RBF kernel had great influence in k-means clustering.