309551101 郭育麟

HW06

a. Code with detailed explanations

Part1:

In this part , we need to make GIF images to show the clustering procedure of mine kernel k-means and spectral clustering(Ratio cut and Normalized cut) . And for every part's spectral clustering , I also plot the eigenspace of graph Laplacian (Part4) . The reason why using ratio cut and normalized cut is avoid extremely solution , for example , there are only one datapoint in the subgraph , and all the others datapoint are in another subgraph , it can avoid this situation by using ratio cut or normalized cut .

Kernel k means

Following the steps

I :

load the image .

```
def imread(img_path):
    """
    param img_path
    return a flatten image array (H*W,C)
    """
    image = cv2.imread(img_path)
    H,W,C = image.shape
    image_flatten = np.zeros((W*H,C))
    image_flatten = image.reshape(W*H,C)
    return image_flatten , H , W
```

This function will load picture and return image(flatten) and the image's height and width

II :

choose parameters "k" which represent "k-clustering" and the initialization of k-means clustering used in kernel k-means (In this part I choose "random" , other option is left to part3) and gamma_spatial and gamma_color.

III :

Calculate kernel function

```python
def precomputed_kernel(X,gamma_spatial,gamma_color):
    """
    kernel function: k(x,x')= exp(-r_s*||S(x)-S(x')||**2)* exp(-r_c*||C(x)-C(x')||**2)
    X: (H*W=10000,rgb=3) array
    """
    n = len(X)

    S = np.zeros((n,2))
    for i in range(n):
        S[i] = [i//100,i%100]
    spatial = np.exp(-gamma_spatial*pdist(S,'sqeuclidean'))
    spatial = squareform(spatial)

    color = np.exp(-gamma_color*pdist(X,'sqeuclidean'))
    color = squareform(color)

    answer_kernel = spatial * color
    return answer_kernel
```

The kernel function is $\mathrm{k}(\mathrm{x},\mathrm{x}') = e^{-r_S||S(x)-S(x')||^2} * e^{-r_C||C(x)-C(x')||^2}$

IV :

Used k-means algorithm .

```python
def initial_mean(X,k,initType):
    """
    X : ( H*W , 3 features)
    k : k klusters
    initType : 'random' , 'pick' , 'k_means_plusplus'
    Cluster : (k,3)

    """
    Cluster = np.zeros((k,X.shape[1]))

    if initType == 'k_means_plus_plus':

        #randomly choose one to be a cluster_mean
        Cluster[0] = X[np.random.randint(low=0,high=X.shape[0],size=1),:]

        #choose another k-1 cluster_mean
        for c in range(1,k):
            Dist_matrix = np.zeros((len(X),c))
            for i in range(len(X)):
                for j in range(c):
                    Dist_matrix[i,j] = np.sqrt(np.sum((X[i]-Cluster[j])**2))
            #這邊應該要用先對橫向找到最小值(計算所有點到其最近的質心的距離)
            #使用輪盤法找到下一個質心
            #https://zhuanlan.zhihu.com/p/32375430
            Dist_min=np.min(Dist_matrix,axis=1)
            sum=np.sum(Dist_min)*np.random.rand()
            for i in range(len(X)):
                sum-=Dist_min[i]
                if sum<=0:
                    Cluster[c]=X[i]
                    break
    if initType == 'random_nor':
        X_mean = np.mean(X,axis=0)
        X_std  = np.std(X,axis =0)
        for i in range(X.shape[1]):
            Cluster[:,i] = np.random.normal(X_mean[i],X_std[i],size = k)

    if initType =='random':
        random_pick=np.random.randint(low=0,high=X.shape[0],size=k)
        Cluster=X[random_pick,:]

    return Cluster
```

```python
colormap= np.random.choice(range(256),size=(100,3))
def visualize(X,k,H,W,colormap):
    """
    """
    colors = colormap[:k,:]
    res = np.zeros((H,W,3))
    for h in range(H):
        for w in range(W):
            res[h,w,:] = colors[X[h*W+w]]

    return res.astype(np.uint8)
```

```python
def k_means(X,k,H,W,initType='random',gifPath='default.gif'):
    """
    Want to do k klusters
    X : ( H*W , 3 features )
    k : k klusters

    """
    EPS = 1e-9
    Mean = initial_mean(X,k,initType)
    #Classes of each Xi
    C = np.zeros(len(X),dtype = np.uint8)
    segments = []

    diff = 1e9
    count = 1
    while diff>EPS:
        # E-step
        for i in range(len(X)):
            dist = []
            for j in range(k):
                dist.append(np.sqrt(np.sum((X[i]-Mean[j])**2)))
            C[i] = np.argmin(dist)

        #M-step
        New_Mean = np.zeros(Mean.shape)
        for i in range(k):
            belong = np.argwhere(C==i).reshape(-1)
            for j in belong:
                New_Mean[i] = New_Mean[i] + X[j]
            if len(belong)>0:
                New_Mean[i] = New_Mean[i]/len(belong)
        diff = np.sum((New_Mean - Mean)**2)
        Mean = New_Mean

        #visualize
        segment = visualize(C,k,H,W,colormap)
        segments.append(segment)
        print('iteration {}'.format(count))
        for i in range(k):
            print('k={}: {}'.format(i+1 , np.count_nonzero(C==i)))
        print('diff{}'.format(diff))


        plt.clf()
        plt.imshow(cv2.cvtColor(segment, cv2.COLOR_BGR2RGB))
        plt.pause(0.001)
        print('------------------------------------------------------------')

        count =count+1
    return C , segments
```

First , we need to use initial_mean function to calculate initial k clustering location . In this part , I use random to create k clustering location . Then we need to calculate every pairs distance and find the minimum distance to classified category . For example , given a datapoint , we need to compute the distance between this datapoint and the current k clustering center , and choose the smallest distance to represent that this datapoint is classified the specific category among k . This step will continue n times , n represent n datapoints .

Second , we need to update new k clustering location . we find the same category datapoint in the previous step and add them and also do normalization so that we can get new k clustering location and difference between new clustering location and old clustering location .

Third , we use visualize function to color the same category datapoint in the current clustering state and return it . So that we could use plt.show to plot the picture . Furthermore , after I use visualize function , I append it to a list to make GIF .

V:

Make GIF images

```
def save_gif(segments,gif_path):
    for i in range(len(segments)):
        segments[i] = segments[i].transpose(1,0,2)
    write_gif(segments,gif_path , fps = 2)
```

In this function , I put segments(every output from function visualize) and convert to RGB so that can use write_gif (from array2gif import write_gif) to make GIF .

Main:

```
if __name__ == '__main__':
    img_path = 'image1.png'
    image_flatten , H , W = imread(img_path)       I

    gamma_spatial = 0.001
    gamma_color   = 0.001

    k = 2  # k clusters
    #k_means_initType='k_means_plus_plus'
    #k_means_initType='random_nor'
    k_means_initType ='random'                      II
    gif_dir = './GIF'
    gif_path=os.path.join("GIF/%s_%sClusters_%s_kmeans.gif"%(img_path.split('.')[0],k,k_means_initType))
    if not os.path.isdir(gif_dir):
        os.mkdir(gif_dir)
    print(gif_path)
    kernel = precomputed_kernel(image_flatten,gamma_spatial,gamma_color)  III
    belongings , segments = k_means(kernel,k,H,W,initType=k_means_initType,gifPath=gif_path)  IV
    save_gif(segments,gif_path)   V
```

See result [link]

Spectral clustering – Ratio cut

Unnormalized Laplacian L = D-W serve in the approximation of the minimization of RatioCut

Following the steps

I:

Same as k-means algorithm I [link]

II:

Same as k-means algorithm II [link]

III:

Same as k-means algorithm III [link]

IV:

Compute Laplacian matrix

$\text{Graph Laplacian} = \mathrm{D} - \mathrm{W}$

D can seen as a degree matrix

Then use np.linalg.eig for eigenvalue decomposition for Laplacian matrix

V:

Sorting the eigenvalue and get the 2nd and 3rd (1st eigenvalue is 0 represent fully connected) eigenvalue and its corresponding eigenvector . Use these eigenvector to execute k_means funtcion .

(k-means algorithm is same as k-means algorithm IV [link])

VI:

Make GIF images and plot eigenvector

Make GIF images is same as k-mean algorithm V [link]

```python
def plot_eigenvector_3(x,y,z,C):
    """
    x y z datapoint array
    C belonging class
    """
    fig = plt.figure()
    ax  = fig.add_subplot(111,projection='3d')
    markers=['o','^','s']
    for marker , i in zip(markers,np.arange(3)):
        ax.scatter(x[C==i],y[C==i],z[C==i],marker=marker)
    ax.set_xlabel('eigenvector 1st dim')
    ax.set_ylabel('eigenvector 2st dim')
    ax.set_zlabel('eigenvector 3rd dim')
    plt.show()
#-----------------------------------------------------------
def plot_eigenvector_2(x,y,C):
    fig = plt.figure()
    markers=['o','^']
    for marker , i in zip(markers,np.arange(2)):
        plt.scatter(x[C==i],y[C==i],marker=marker)
    plt.xlabel('eigenvector 1st dim')
    plt.ylabel('eigenvector 2st dim')
    plt.show()
```

In plot_eigenvector_3 , x , y , z represent eigenvector of the graph Laplacian , because I want to do 3-clustering , so pick x,y,z . if I only want to do 2-clustering , only pick x, y .

Main:

```python
if __name__=='__main__':
    img_path = 'image2.png'
    image_flatten , H , W = imread(img_path)                I

    gamma_spatial = 0.001
    gamma_color   = 0.001

    k = 2  # k clusters
    k_means_initType = 'k_means_plus_plus'
    k_means_initType='random_nor'
    k_means_initType='random'                                II

    gif_dir = './GIF'
    gif_path=os.path.join("GIF/%s_%sClusters_%s_ratio.gif"%(img_path.split('.')[0],k,k_means_initType))
    if not os.path.isdir(gif_dir):
        os.mkdir(gif_dir)
    print(gif_path)

    WW = precomputed_kernel(image_flatten,gamma_spatial,gamma_color)    III
    D = np.diag(np.sum(WW,axis=1))
    L = D-WW

    #"""
    eigenvalue , eigenvector = np.linalg.eig(L)
    np.save('{}_eigenvalue_{:.3f}_{:.3f}_ratio.npy'.format(img_path.split('.')[0],gamma_spatial,gamma_color),eigenvalue)
    np.save('{}_eigenvector_{:.3f}_{:.3f}_ratio.npy'.format(img_path.split('.')[0],gamma_spatial,gamma_color),eigenvector)    IV
    #"""

    eigenvalue = np.load('{}_eigenvalue_{:.3f}_{:.3f}_ratio.npy'.format(img_path.split('.')[0],gamma_spatial,gamma_color))
    eigenvector = np.load('{}_eigenvector_{:.3f}_{:.3f}_ratio.npy'.format(img_path.split('.')[0],gamma_spatial,gamma_color))
    sort_index = np.argsort(eigenvalue)
                                                             V
    HH = eigenvector[:,sort_index[1:k+1]]

    belonging,segments=k_means(HH,k,H,W,initType=k_means_initType,gifPath=gif_path)
    save_gif(segments,gif_path)
    if k==3:
        plot_eigenvector_3(HH[:,0],HH[:,1],HH[:,2],belonging)        VI
    if k==2:
        plot_eigenvector_2(HH[:,0],HH[:,1],belonging)
```

See result [link]

Spectral clustering – Normalized cut

Normalized Laplacian $D^{-\frac{1}{2}}LD^{-\frac{1}{2}}$ serve in the approximation of the minimization of Normalized Cut .

This is very similar as ratio cut , the main difference between ratio cut and normalized cut is $L = D^{-\frac{1}{2}}LD^{-\frac{1}{2}}$, this means normalization . And also , the each row eigenvector also do normalization . Others are same as ratio cut .

Main:

```
f __name__ =='__main__':
    img_path = 'image1.png'
    image_flatten , H , W = imread(img_path)

    gamma_spatial = 0.001
    gamma_color   = 0.001

    k = 2  # k clusters
    #k_means_initType = 'k_means_plus_plus'
    #k_means_initType='random_nor'
    k_means_initType='random'

    gif_dir = './GIF'
    gif_path=os.path.join("GIF/%s_%sClusters_%s_Normalized.gif"%(img_path.split('.')[0],k,k_means_initType))
    if not os.path.isdir(gif_dir):
        os.mkdir(gif_dir)
    print(gif_path)

    #"""
    WW = precomputed_kernel(image_flatten,gamma_spatial,gamma_color)
    D = np.diag(np.sum(WW,axis=1))
    L = D-WW
    D_inverse_square= np.diag(1/np.diag(np.sqrt(D)))
    L = np.dot(np.dot(D_inverse_square,L),D_inverse_square)


    eigenvalue , eigenvector = np.linalg.eig(L)
    np.save('{}_eigenvalue_{:.3f}_{:.3f}_normalized.npy'.format(img_path.split('.')[0],gamma_spatial,gamma_color),eigenvalue
    np.save('{}_eigenvector_{:.3f}_{:.3f}_normalized.npy'.format(img_path.split('.')[0],gamma_spatial,gamma_color),eigenvect
    print("finish")
    #"""

    eigenvalue = np.load('{}_eigenvalue_{:.3f}_{:.3f}_normalized.npy'.format(img_path.split('.')[0],gamma_spatial,gamma_colo
    eigenvector = np.load('{}_eigenvector_{:.3f}_{:.3f}_normalized.npy'.format(img_path.split('.')[0],gamma_spatial,gamma_co
    sort_index = np.argsort(eigenvalue)

    HH = eigenvector[:,sort_index[1:k+1]]

    sums = np.sqrt(np.sum(np.square(HH),axis=1)).reshape(-1,1)
    HH = HH/sums

    belonging,segments=k_means(HH,k,H,W,initType=k_means_initType)
    save_gif(segments,gif_path)
    if k==3:
        plot_eigenvector_3(HH[:,0],HH[:,1],HH[:,2],belonging)
    if k==2:
        plot_eigenvector_2(HH[:,0],HH[:,1],belonging)
```

See result [link]

Part2:

Try more clusters

Only change parameters “k”

K-means algorithm [link]

Spectral clustering ratio cut [link]

Spectral clustering normalized cut [link]

Part3:

Try different initial of kernel k-means method

There are two extra method to initialize the kernel k-means

(1) Random-normalized

```
if initType == 'random_nor':
    X_mean = np.mean(X,axis=0)
    X_std  = np.std(X,axis =0)
    for i in range(X.shape[1]):
        Cluster[:,i] = np.random.normal(X_mean[i],X_std[i],size = k)
```

In this random_normalized , I calculate the whole data point mean and variance , and random generate k data point as a clustering location by given mean and variance , I believe that this could convergence faster .

(2) k-means++

```
if initType == 'k_means_plus_plus':

    #randomly choose one to be a cluster_mean
    Cluster[0] = X[np.random.randint(low=0,high=X.shape[0],size=1),:]

    #choose another k-1 cluster_mean
    for c in range(1,k):
        Dist_matrix = np.zeros((len(X),c))
        for i in range(len(X)):
            for j in range(c):
                Dist_matrix[i,j] = np.sqrt(np.sum((X[i]-Cluster[j])**2))
        #這邊應該要用先對橫向找到最小值(計算所有點到其最近的質心的距離)
        #使用輪盤法找到下一個質心
        #https://zhuanlan.zhihu.com/p/32375430
        Dist_min = np.min(Dist_matrix,axis=1)
        sum = np.sum(Dist_min)*np.random.rand()
        for i in range(len(X)):
            sum = sum - Dist_min[i]
            if sum<=0:
                Cluster[c] = X[i]
                break
```

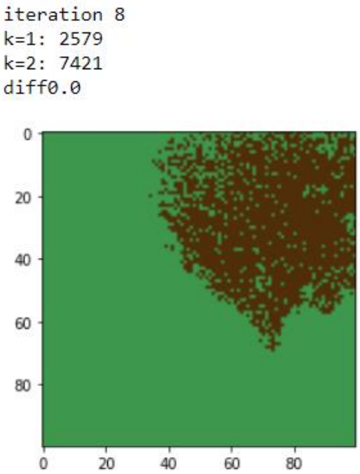
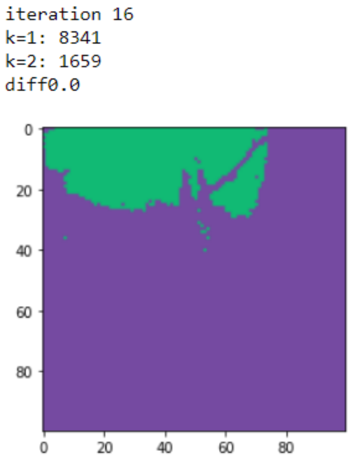
In k-means++ , random choose a datapoint as a clustering location center , and compute every datapoint to its distance , then if have more than 2 clustering center , find the minimum distance between a datapoint to some clustering center . Last , using roulette wheel section , choose next clustering center , this for-loop will continue k (the number of clustering) times .

Result [link]

b. Experiments settings and results & discussion

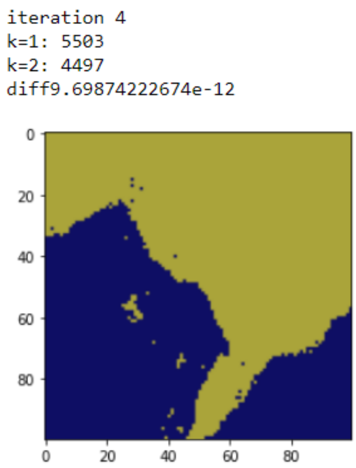
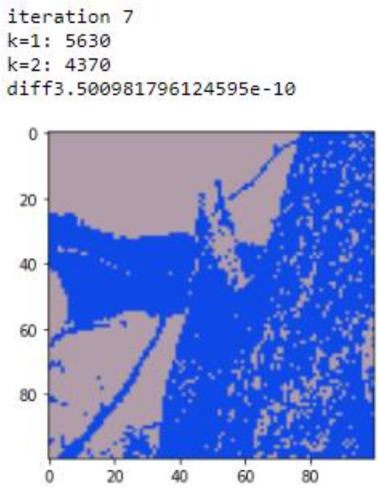
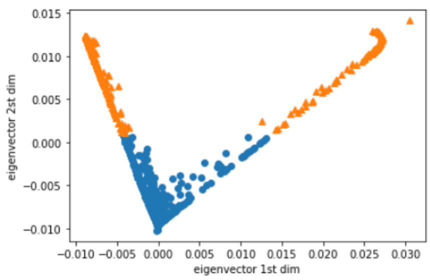
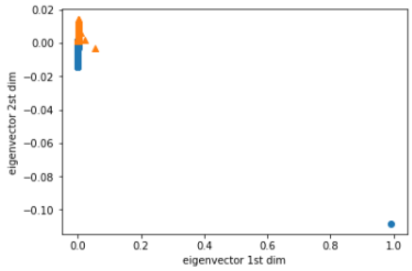
Part1

K-means algorithm

	Image1	Image2
Initial		
Result		
GIF	“image1_2Clusters_random_kmeans”	image2_2Clusters_random_kmeans

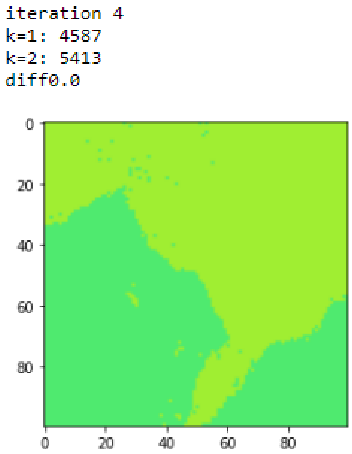
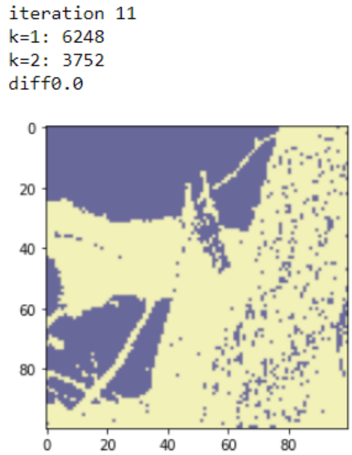
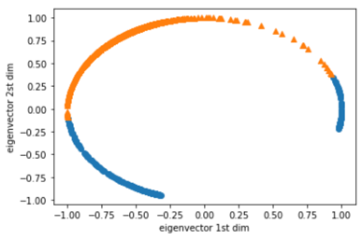
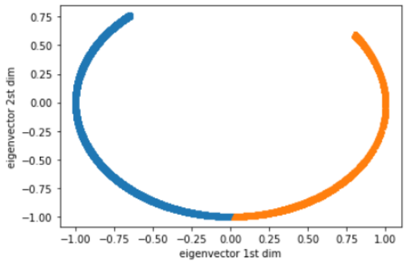
We can see that the result is not ideal , it can not present the initial graph feature , so I think 2-clustering is not enough , maybe more clustering .

Spectral clustering ratio cut

	Image1	Image2
Result		
Eigenvector		
GIF	image1_2Clusters_random_ratio	image2_2Clusters_random_ratio

We can see that this result is better than k-means algorithm , it can see approximately contour . And I think the eigenvector of graph Laplacian can have the same coordinates within the same cluster , it separates two category from a certain threshold .

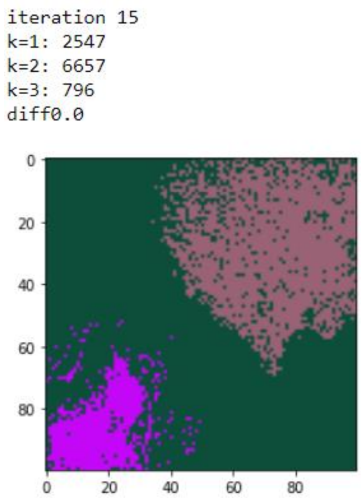
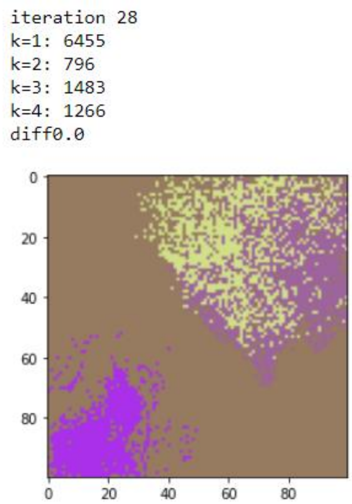
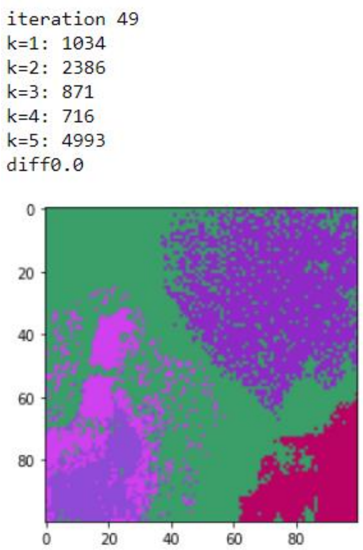
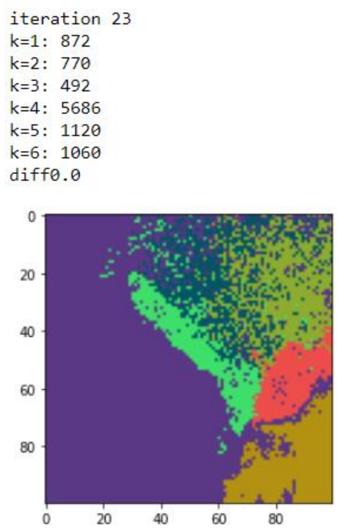
Spectral clustering normalized cut

	Image1	Image2
Result		
Eigenvector		
GIF	image1_2Clusters_random_Normalized	image2_2Clusters_random_Normalized

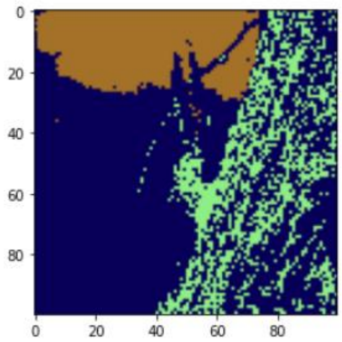
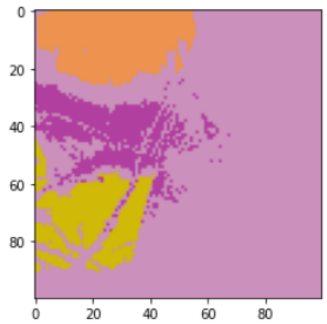
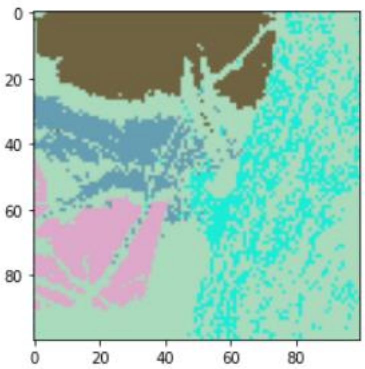
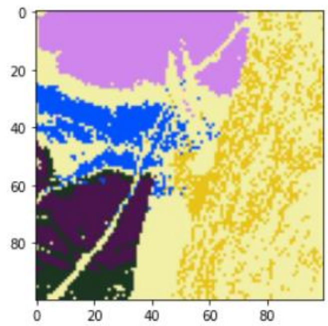
And by using normalized cut , the result seems like ratio cut . However , the eigenvector can see the difference better than ratio cut .

Part2:

K-means algorithm

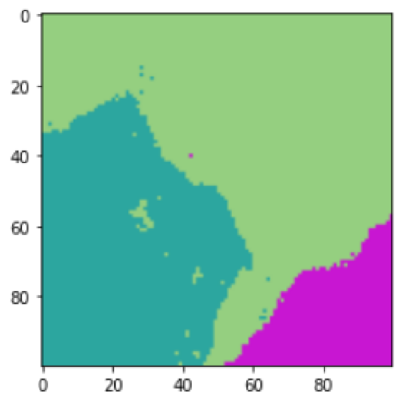
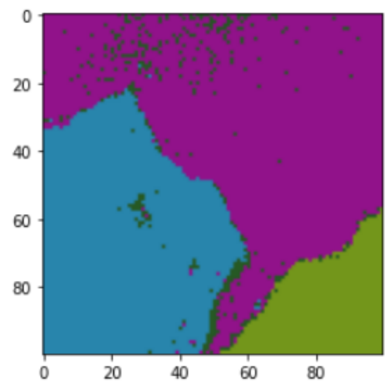
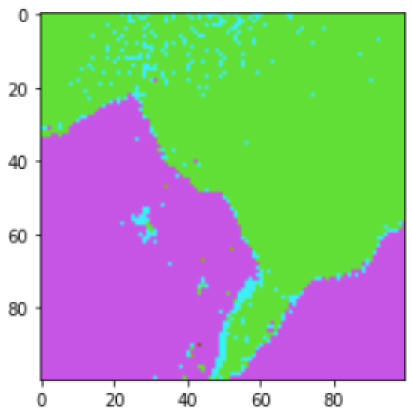
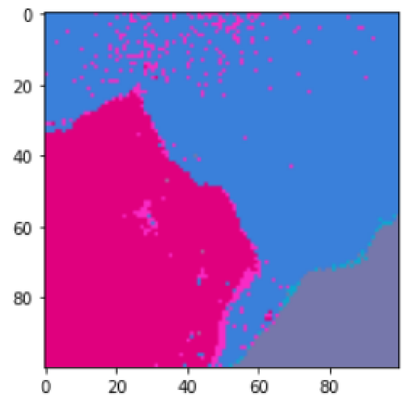
<table>
<tr><td>Image 1</td><td>K=3</td><td>K=4</td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td></td><td>image1_3Clusters_random_kmeans</td><td>image1_4Clusters_random_kmeans</td></tr>
<tr><td></td><td>K=5</td><td>K=6</td></tr>
<tr><td>Result</td><td></td><td></td></tr>
<tr><td></td><td>image1_5Clusters_random_kmeans</td><td>image1_6Clusters_random_kmeans</td></tr>
</table>

This result shows that maybe should not use more clustering . The more clustering I use , the graph is more complexity . So it might use 3-clustering in this case .

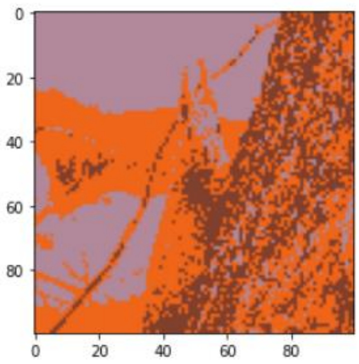
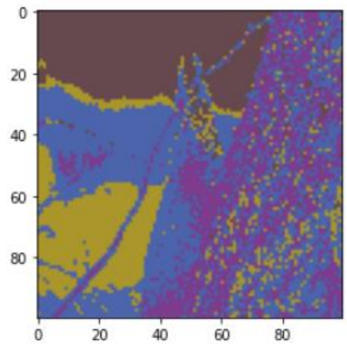
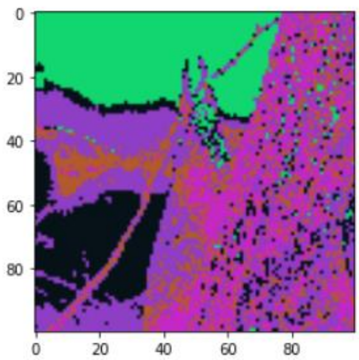
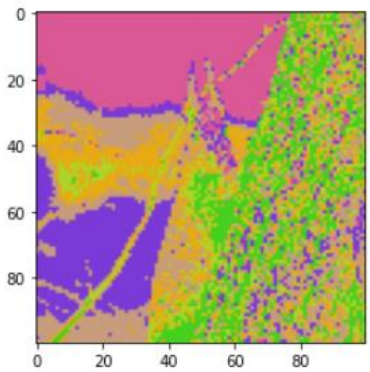
Image 2	K=3	K=4
	iteration 29 k=1: 1706 k=2: 1659 k=3: 6635 diff0.0 	------------------------------------ iteration 12 k=1: 1134 k=2: 6992 k=3: 992 k=4: 882 diff0.0 
	image2_3Clusters_random_kmeans	image2_4Clusters_random_kmeans
	K=5	K=6
	iteration 17 k=1: 1648 k=2: 880 k=3: 870 k=4: 5230 k=5: 1372 diff0.0 	iteration 41 k=1: 1346 k=2: 4730 k=3: 605 k=4: 820 k=5: 1645 k=6: 854 diff0.0 
	image2_5Clusters_random_kmeans	image2_6Clusters_random_kmeans

This result shows the same situation to image1 , I think that maybe use 3-clusering is the best choice .

Spectral clustering ratio cut

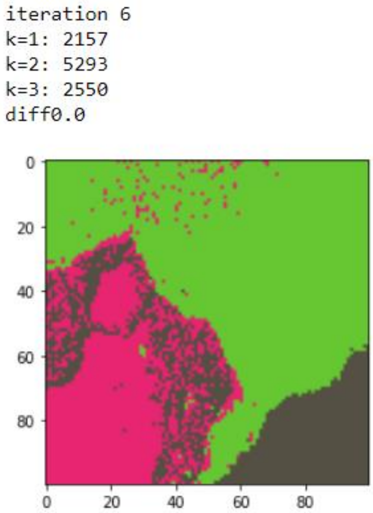
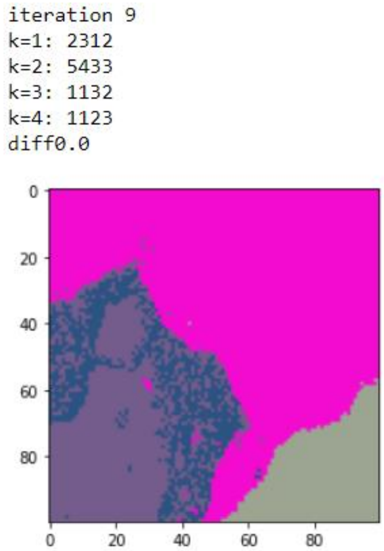
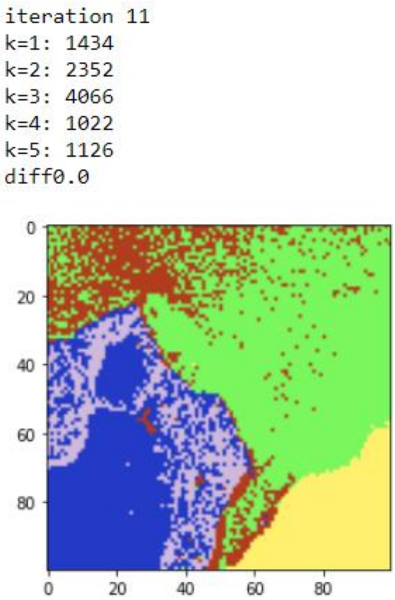
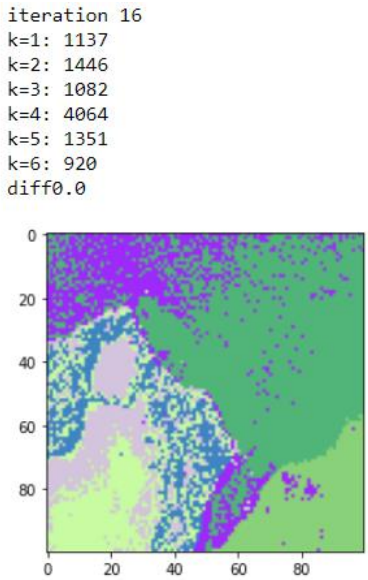
Image1	K=3	K=4
	iteration 17 k=1: 3389 k=2: 1108 k=3: 5503 diff2.3022506544282442e-10 	iteration 28 k=1: 1106 k=2: 407 k=3: 5115 k=4: 3372 diff3.287412006563854e-10 
	image1_3Clusters_random_ratio	image1_4Clusters_random_ratio
	K=5	K=6
	iteration 33 k=1: 4 k=2: 370 k=3: 5163 k=4: 4462 k=5: 1 diff5.916810580618924e-11 	iteration 23 k=1: 3369 k=2: 1094 k=3: 374 k=4: 3 k=5: 39 k=6: 5121 diff7.207795971572542e-10 
	image1_5Clusters_random_ratio	image1_6Clusters_random_ratio

By applying ratio cut , the result shows that it do the better performance than k-means . However , it still has the same problem , the more clustering I use , the graph is more complexity . And can see that when I choose k=5 , some clustering only has digits numbers , it reveal that these clustering subgraph are useless .

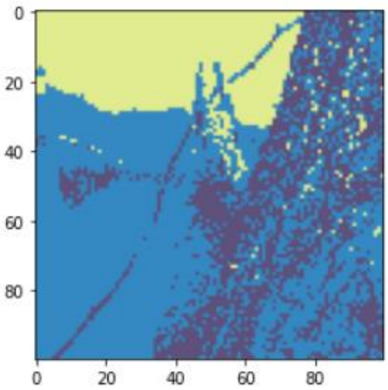
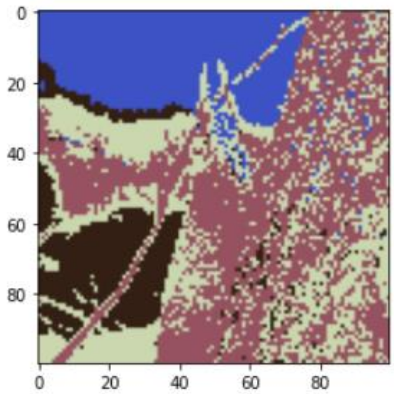
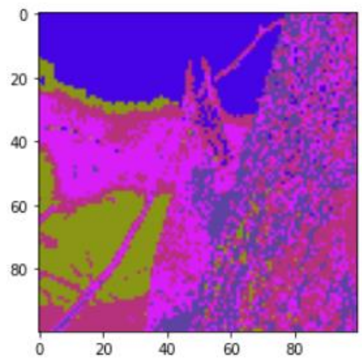
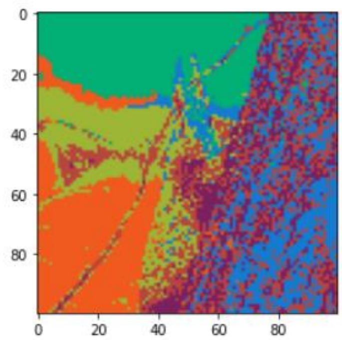
Image2	K=3	K=4
	iteration 6 k=1: 4157 k=2: 2276 k=3: 3567 diff2.964926943531498e-10  	iteration 10 k=1: 2230 k=2: 1795 k=3: 2097 k=4: 3878 diff7.74620594700798e-10 
	image2_3Clusters_random_ratio	image2_4Clusters_random_ratio
	K=5	K=6
	iteration 20 k=1: 1732 k=2: 3012 k=3: 1446 k=4: 1723 k=5: 2087 diff6.095161476098875e-10  	iteration 37 k=1: 1622 k=2: 2156 k=3: 1480 k=4: 991 k=5: 2082 k=6: 1669 diff8.488762897355564e-10 
	image2_5Clusters_random_ratio	image2_6Clusters_random_ratio

Also it has not the situation like image1 in more clustering , but it still make the picture more unclear .

Spectral clustering normalized cut

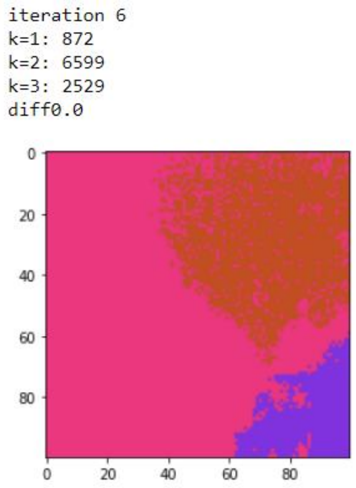
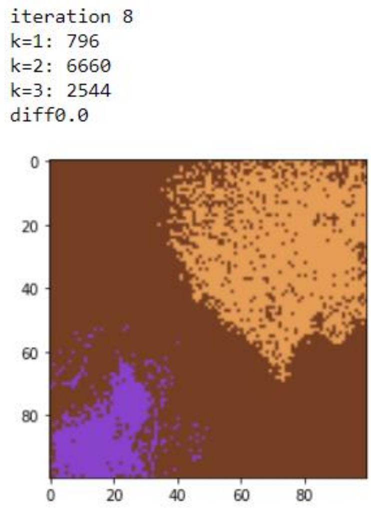
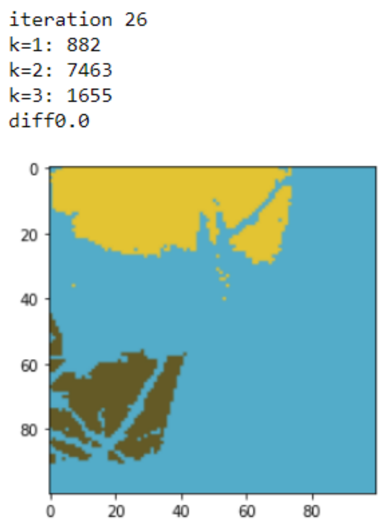
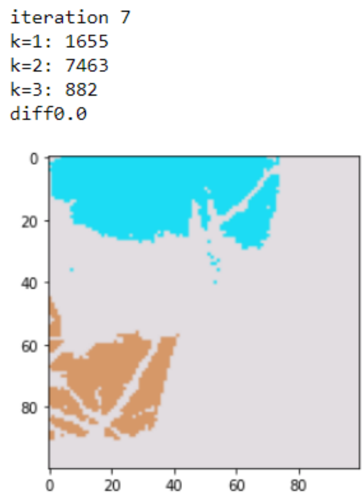
<table>
<tr><td>Image 1</td><td>K=3</td><td>K=4</td></tr>
<tr><td></td><td>
</td><td>
</td></tr>
<tr><td></td><td>image1_3Clusters_random_Normalized</td><td>image1_4Clusters_random_Normalized</td></tr>
<tr><td></td><td>K=5</td><td>K=6</td></tr>
<tr><td></td><td>
</td><td>
</td></tr>
<tr><td></td><td>image1_5Clusters_random_Normalized</td><td>image1_6Clusters_random_Normalized</td></tr>
</table>

By testing with many clustering and many method , I can conclude that the best choice is k = 3

Image 2	K=3	K=4
	iteration 12 k=1: 2128 k=2: 2510 k=3: 5362 diff0.0 	iteration 14 k=1: 2965 k=2: 3664 k=3: 1369 k=4: 2002 diff0.0 
	Image2_3Clusters_random_Normalized	Image2_4Clusters_random_Normalized
	K=5	K=6
	iteration 11 k=1: 2597 k=2: 1299 k=3: 1604 k=4: 1976 k=5: 2524 diff0.0 	iteration 17 k=1: 1305 k=2: 1947 k=3: 1674 k=4: 1892 k=5: 1448 k=6: 1734 diff0.0 
	Image2_5Clusters_random_Normalized	Image2_6Clusters_random_Normalized

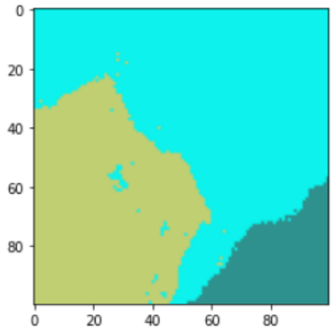
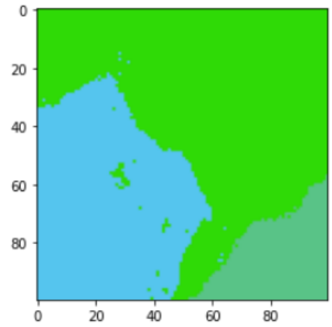
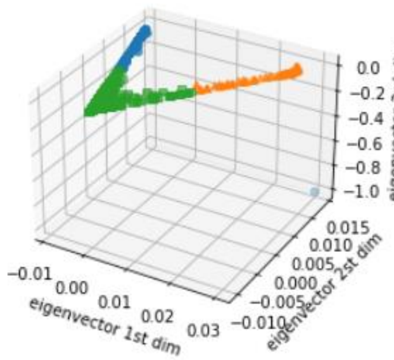
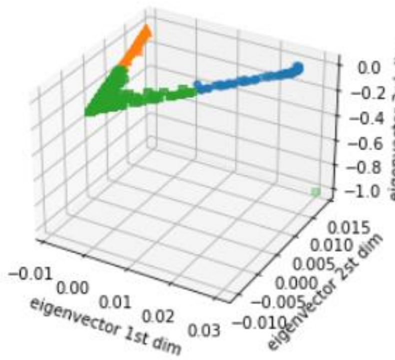
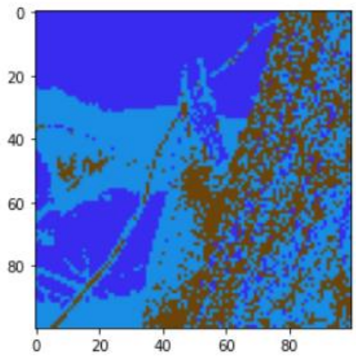
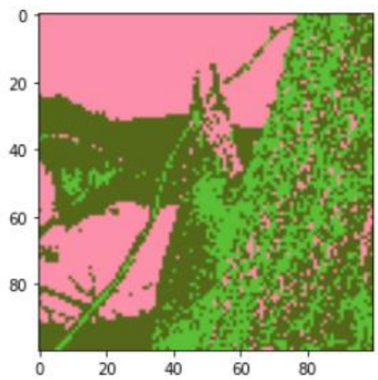
Part3:

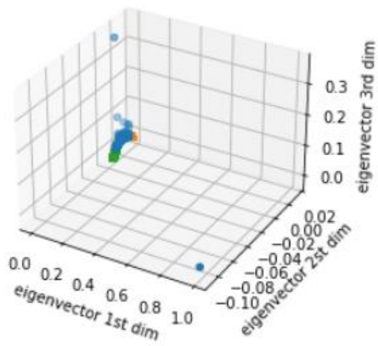
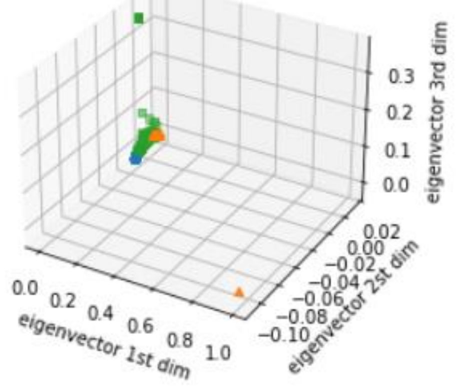
k-means

k = 3	Random-normalized	k-means++
Image 1		
	image1_3Clusters_random_nor_kmeans	image1_3Clusters_k_means_plus_plus_kmeans
Image 2		
	image2_3Clusters_random_nor_kmeans	image2_3Clusters_k_means_plus_plus_kmeans

The google says that it can faster convergence or benefit to convergence by adopting different kernel k-means , but in my practice , the k-means++ convergence speed is similar as original method , I think that the reason is the original picture is not too big , so the convergence speed is almost the same . And in ranomd_normalized , the convergence speed maybe get worse than original , furthermore , It sometimes got a worst performance .

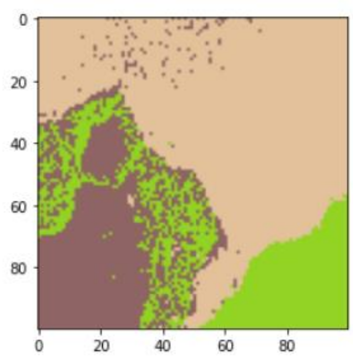
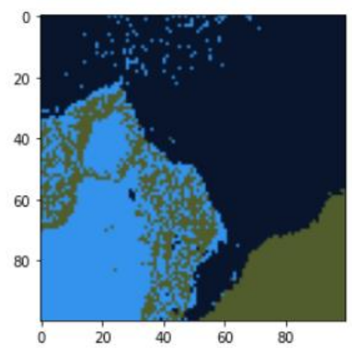
Spectral clustering ratio cut

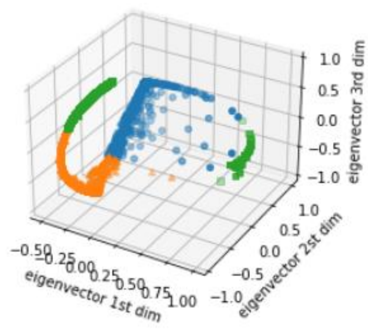
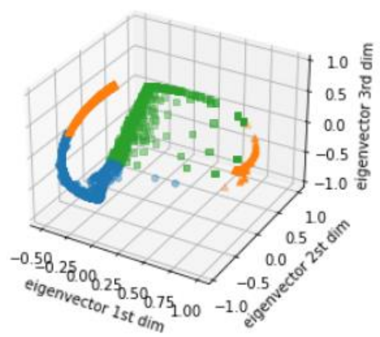
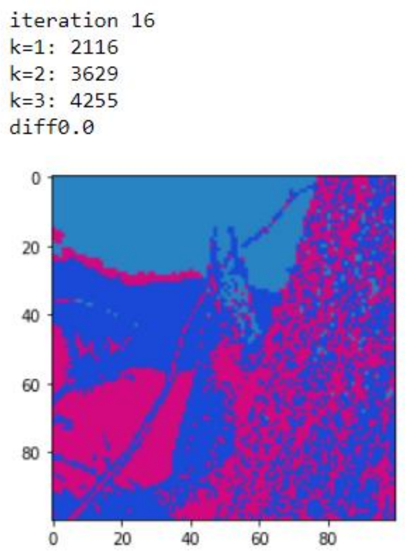
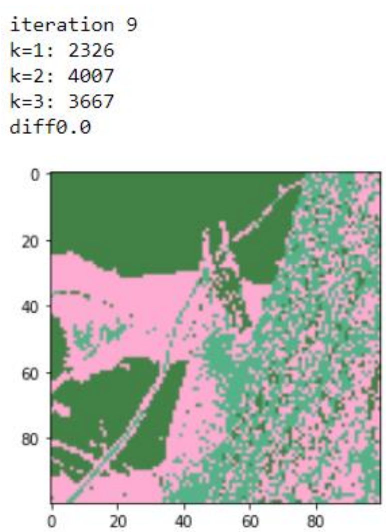
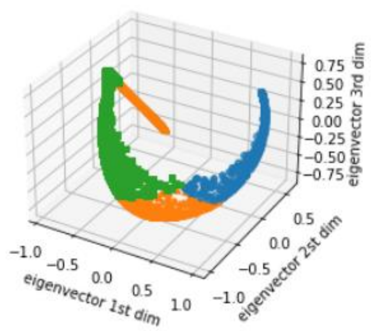
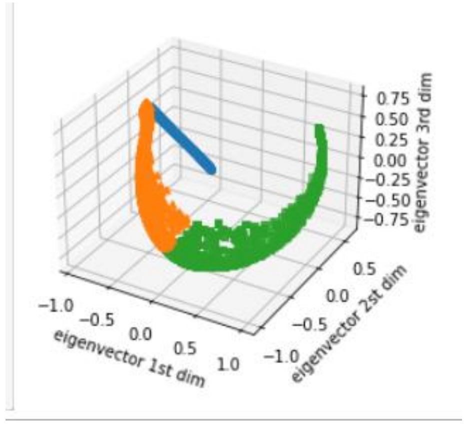
K=3	Random-normalized	k-means++
Image 1	iteration 13 k=1: 3389 k=2: 1107 k=3: 5504 diff9.441723994657584e-10 	iteration 2 k=1: 1107 k=2: 3389 k=3: 5504 diff0.0 
		
	image1_3Clusters_random_nor_ratio	image1_3Clusters_k_means_plus_plus_ratio
Image 2	iteration 8 k=1: 4158 k=2: 3566 k=3: 2276 diff7.529069797136024e-10 	iteration 10 k=1: 2268 k=2: 3569 k=3: 4163 diff4.854526511589266e-10 

		
	image2_3Clusters_random_nor_ratio	image2_3Clusters_k_means_plus_plus_ratio

In the previous , I conclude that k=3 is the perfect choice . Image1's the data points within the same cluster have the same coordinates in the eigenspace of graph Laplacian . However , the performance is not better by observing the graph in Iamge2 . The whole datapoint are all together , it can not easy to observe.

Spectral clustering Normalized cut:

K=3	Random-normalized	k-means++
Image1	iteration 11 k=1: 5293 k=2: 2550 k=3: 2157 diff0.0 	iteration 6 k=1: 2550 k=2: 2157 k=3: 5293 diff0.0 

<table>
<tr><td></td><td></td><td></td></tr>
<tr><td></td><td>image1_3Clusters_random_nor_Normalized</td><td>image1_3Clusters_k_means_plus_plus_Normalized</td></tr>
<tr><td>Image2</td><td>
</td><td>
</td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td></td><td>image2_3Clusters_random_nor_Normalized</td><td>image2_3Clusters_k_means_plus_plus_Normalized</td></tr>
<tr><td></td><td></td><td></td></tr>
</table>

By applying normalized cut , the image1 still separate perfectly , furthermore , the image2 not like the previous graph , it can separate perfectly too , this could easy to observe and let the result be more precise .

c. Observations and discussion

The result shows that the spectral clustering get a better performance than k-means algorithm . And using spectral clustering can reduce dimension from n to k dimension , this could reduce the burden and let computational efficiency . And how to choose and be found . For example , when slowly increase k and find that the eigenvalue suddenly bigger , maybe it should choose k-1 category as clustering .