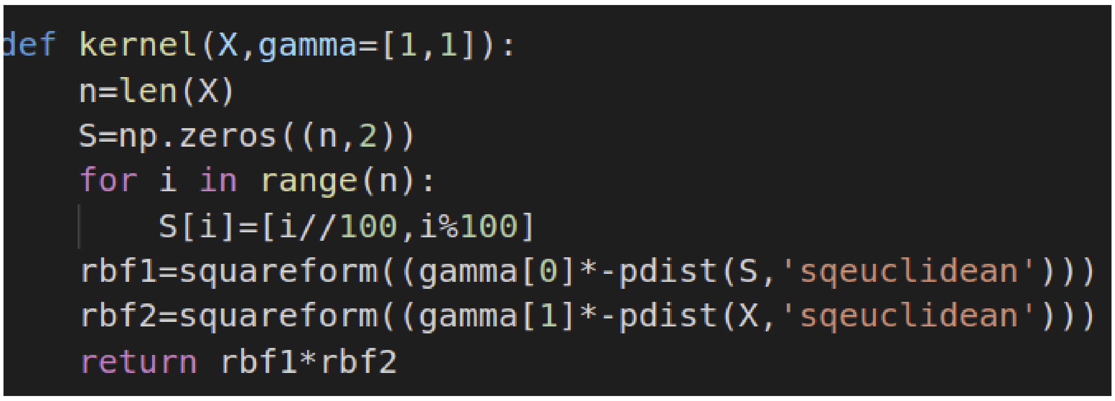
HW6_report

a.code with detailed explanations

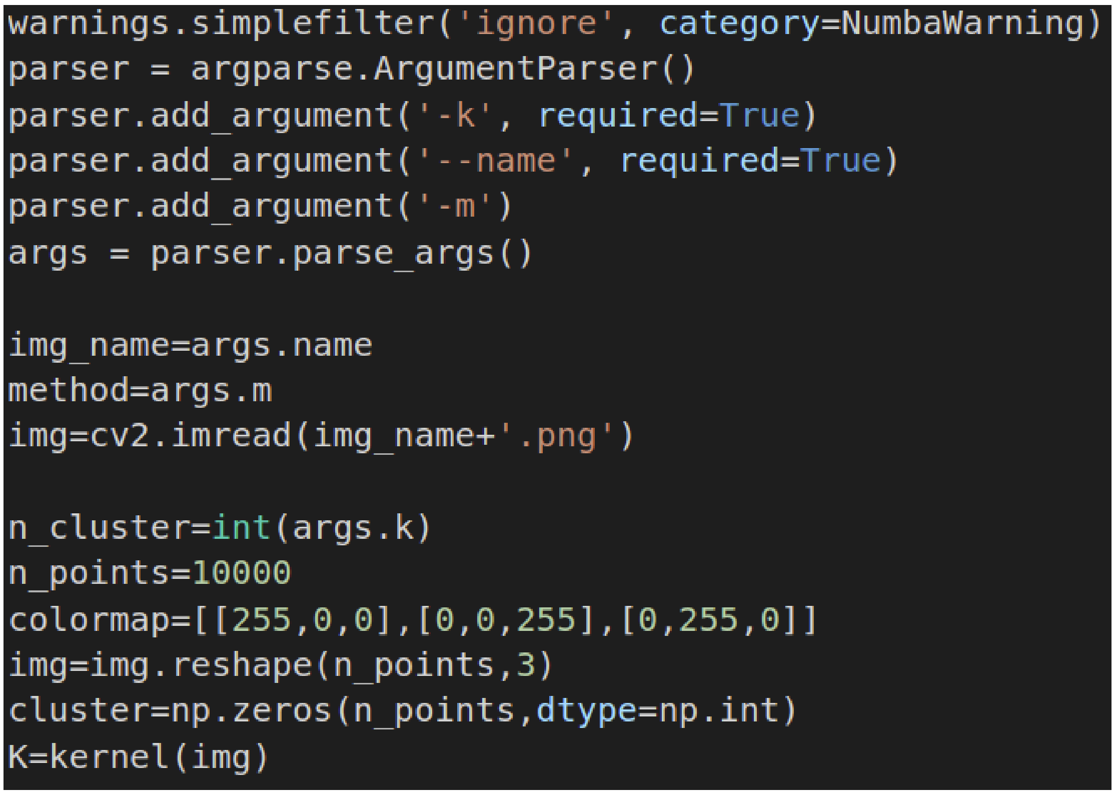
- **Part1 & Part2**

kernel k-means



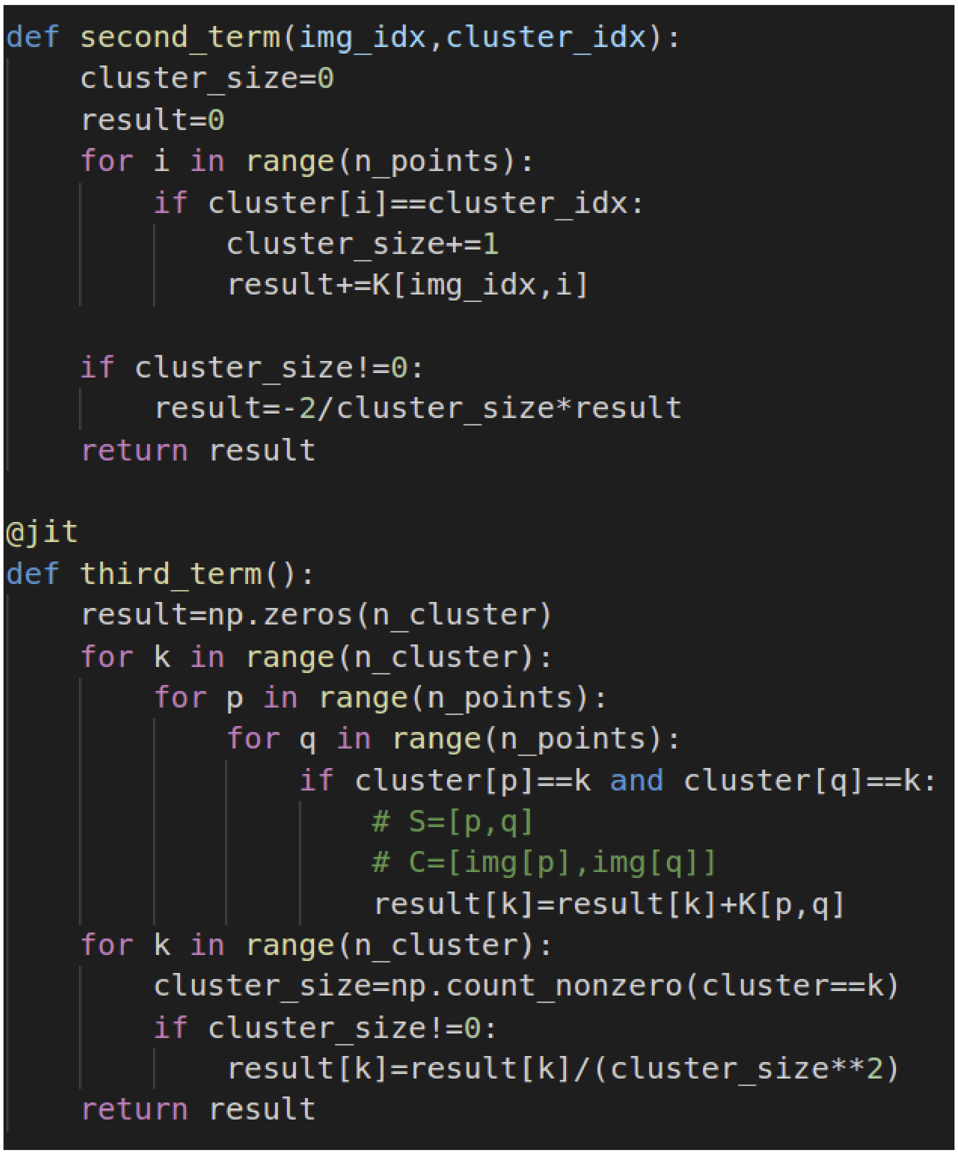
```python
def kernel(X,gamma=[1,1]):
    n=len(X)
    S=np.zeros((n,2))
    for i in range(n):
        S[i]=[i//100,i%100]
    rbf1=squareform((gamma[0]*-pdist(S,'sqeuclidean')))
    rbf2=squareform((gamma[1]*-pdist(X,'sqeuclidean')))
    return rbf1*rbf2
```

I follow the formula in homework spec and compute the kernel. Because the image is 100x100, S(x) is the coordinate (i/100,i%100). C(x) is the pixel value of image.

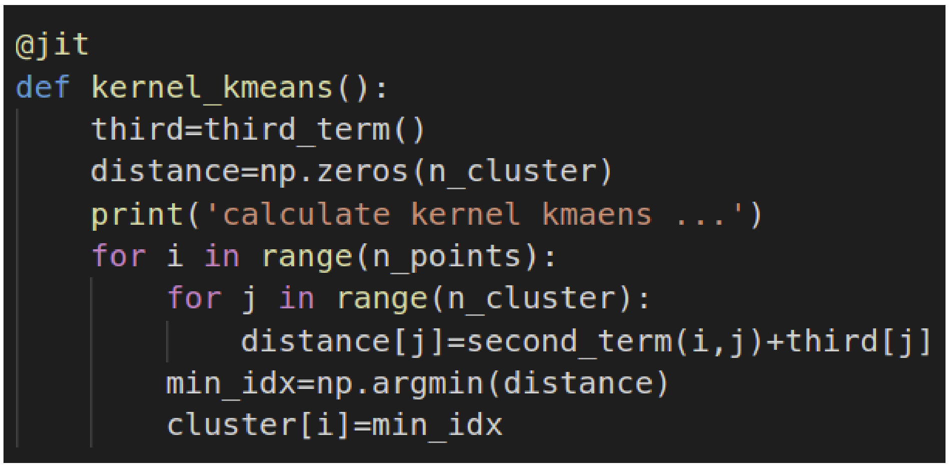


```python
warnings.simplefilter('ignore', category=NumbaWarning)
parser = argparse.ArgumentParser()
parser.add_argument('-k', required=True)
parser.add_argument('--name', required=True)
parser.add_argument('-m')
args = parser.parse_args()

img_name=args.name
method=args.m
img=cv2.imread(img_name+'.png')

n_cluster=int(args.k)
n_points=10000
colormap=[[255,0,0],[0,0,255],[0,255,0]]
img=img.reshape(n_points,3)
cluster=np.zeros(n_points,dtype=np.int)
K=kernel(img)
```

In the first, I decide the cluster number, the image that I want to load and initial method (random or k-means++)



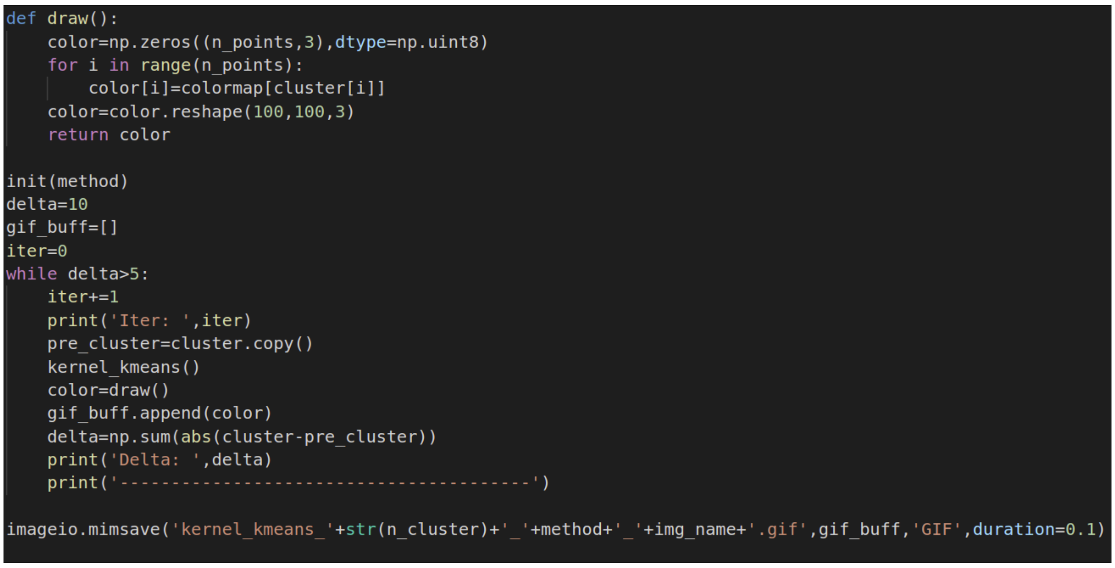
```python
def second_term(img_idx,cluster_idx):
    cluster_size=0
    result=0
    for i in range(n_points):
        if cluster[i]==cluster_idx:
            cluster_size+=1
            result+=K[img_idx,i]

    if cluster_size!=0:
        result=-2/cluster_size*result
    return result

@jit
def third_term():
    result=np.zeros(n_cluster)
    for k in range(n_cluster):
        for p in range(n_points):
            for q in range(n_points):
                if cluster[p]==k and cluster[q]==k:
                    # S=[p,q]
                    # C=[img[p],img[q]]
                    result[k]=result[k]+K[p,q]
    for k in range(n_cluster):
        cluster_size=np.count_nonzero(cluster==k)
        if cluster_size!=0:
            result[k]=result[k]/(cluster_size**2)
    return result
```



```python
@jit
def kernel_kmeans():
    third=third_term()
    distance=np.zeros(n_cluster)
    print('calculate kernel kmaens ...')
    for i in range(n_points):
        for j in range(n_cluster):
            distance[j]=second_term(i,j)+third[j]
        min_idx=np.argmin(distance)
        cluster[i]=min_idx
```

$$\left\|\phi(x_j) - \mu_k^{\phi}\right\| = \left\|\phi(x_j) - \frac{1}{|C_k|}\sum_{n=1}^{N} \alpha_{kn}\phi(x_n)\right\|$$
$$= \mathbf{k}(x_j, x_j) - \frac{2}{|C_k|}\sum_{n} \alpha_{kn}\mathbf{k}(x_j, x_n) + \frac{1}{|C_k|^2}\sum_{p}\sum_{q} \alpha_{kp}\alpha_{kq}\mathbf{k}(x_p, x_q)$$

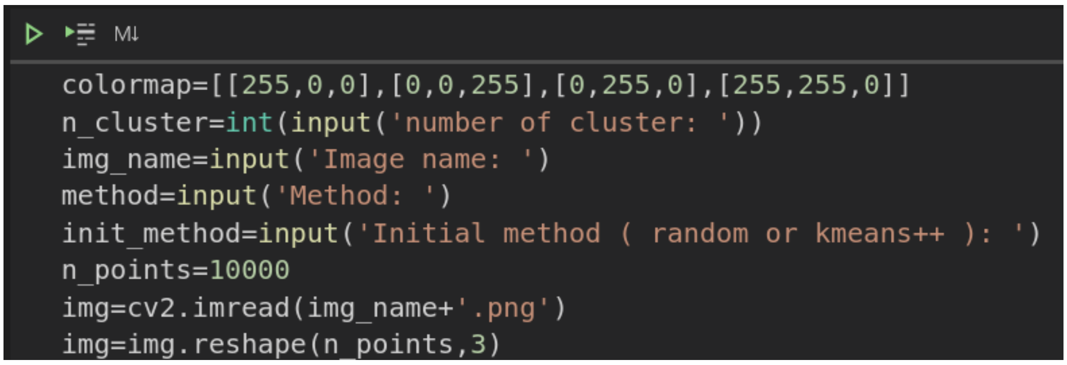
I follow this formula to calculate the kernel k-means. The first term is fixed so it doesn't need to compute it. Therefore the distance between every data point and mean will be second_term add third_term. After getting the distance, I can assign the cluster if its distance is shortest between the mean.
By the way, I use the "numba" package to speed up my python script.



```python
def draw():
    color=np.zeros((n_points,3),dtype=np.uint8)
    for i in range(n_points):
        color[i]=colormap[cluster[i]]
    color=color.reshape(100,100,3)
    return color

init(method)
delta=10
gif_buff=[]
iter=0
while delta>5:
    iter+=1
    print('Iter: ',iter)
    pre_cluster=cluster.copy()
    kernel_kmeans()
    color=draw()
    gif_buff.append(color)
    delta=np.sum(abs(cluster-pre_cluster))
    print('Delta: ',delta)
    print('----------------------------------------')

imageio.mimsave('kernel_kmeans_'+str(n_cluster)+'_'+method+'_'+img_name+'.gif',gif_buff,'GIF',duration=0.1)
```

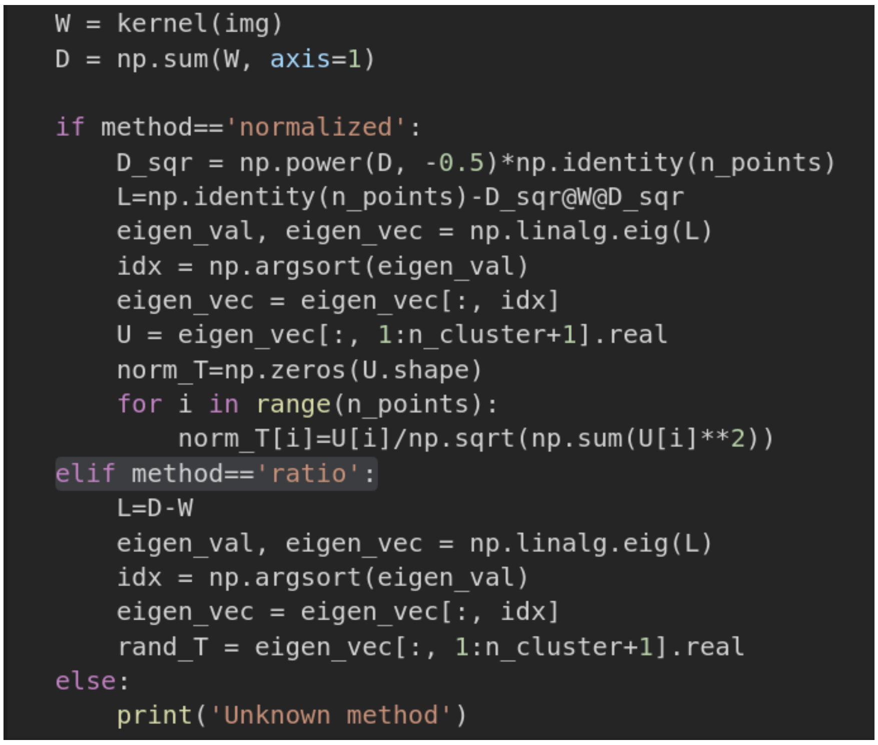
When finishing the kernel k-means, I draw the same color in same cluster. After finishing all iterations,I use the result to make the gif file by imageio.mimsave()

Spectral clustering

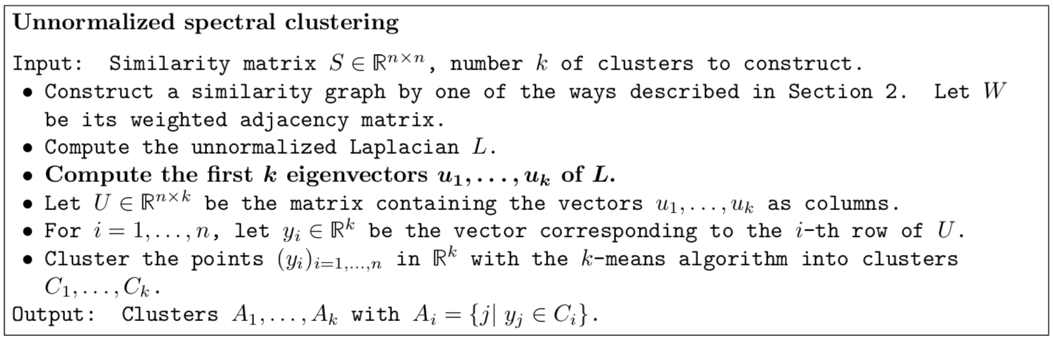


```
colormap=[[255,0,0],[0,0,255],[0,255,0],[255,255,0]]
n_cluster=int(input('number of cluster: '))
img_name=input('Image name: ')
method=input('Method: ')
init_method=input('Initial method ( random or kmeans++ ): ')
n_points=10000
img=cv2.imread(img_name+'.png')
img=img.reshape(n_points,3)
```

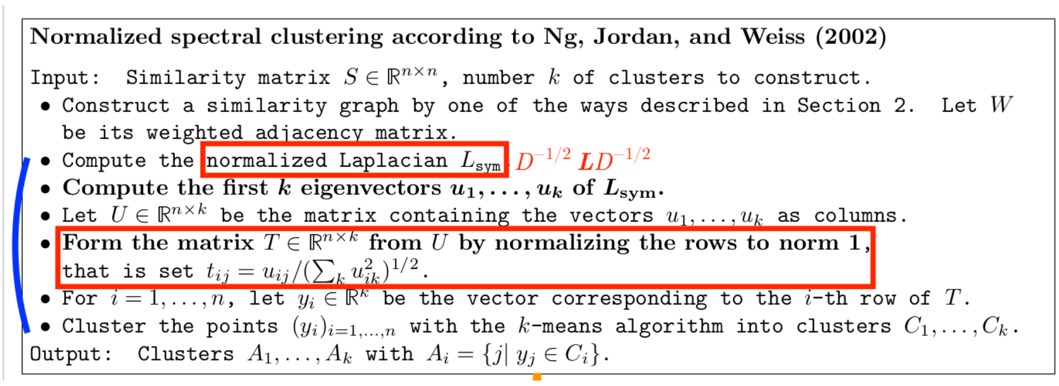
In the first, I decide cluster number, the image that I want to load,normalized cut or ratio cut and initial method (random or k-means++)



```
W = kernel(img)
D = np.sum(W, axis=1)

if method=='normalized':
    D_sqr = np.power(D, -0.5)*np.identity(n_points)
    L=np.identity(n_points)-D_sqr@W@D_sqr
    eigen_val, eigen_vec = np.linalg.eig(L)
    idx = np.argsort(eigen_val)
    eigen_vec = eigen_vec[:, idx]
    U = eigen_vec[:, 1:n_cluster+1].real
    norm_T=np.zeros(U.shape)
    for i in range(n_points):
        norm_T[i]=U[i]/np.sqrt(np.sum(U[i]**2))
elif method=='ratio':
    L=D-W
    eigen_val, eigen_vec = np.linalg.eig(L)
    idx = np.argsort(eigen_val)
    eigen_vec = eigen_vec[:, idx]
    rand_T = eigen_vec[:, 1:n_cluster+1].real
else:
    print('Unknown method')
```

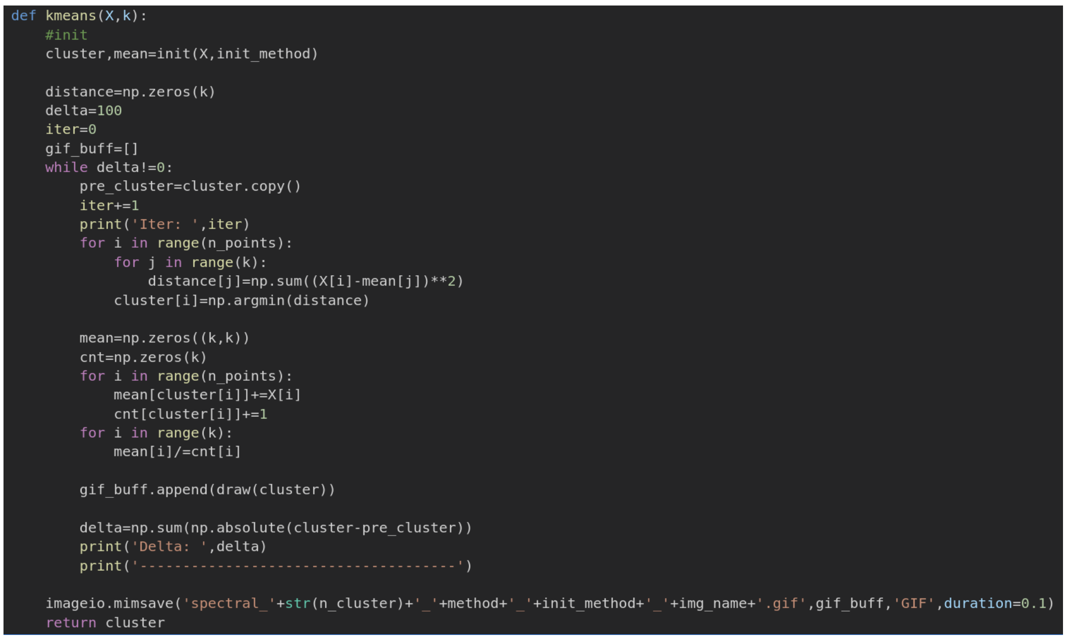


```
Unnormalized spectral clustering
Input: Similarity matrix $S \in \mathbb{R}^{n\times n}$, number $k$ of clusters to construct.
  • Construct a similarity graph by one of the ways described in Section 2. Let $W$
    be its weighted adjacency matrix.
  • Compute the unnormalized Laplacian $L$.
  • Compute the first $k$ eigenvectors $u_1, \dots, u_k$ of $L$.
  • Let $U \in \mathbb{R}^{n\times k}$ be the matrix containing the vectors $u_1, \dots, u_k$ as columns.
  • For $i = 1, \dots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the $i$-th row of $U$.
  • Cluster the points $(y_i)_{i=1,\dots,n}$ in $\mathbb{R}^k$ with the $k$-means algorithm into clusters
    $C_1, \dots, C_k$.
Output: Clusters $A_1, \dots, A_k$ with $A_i = \{j \mid y_j \in C_i\}$.
```

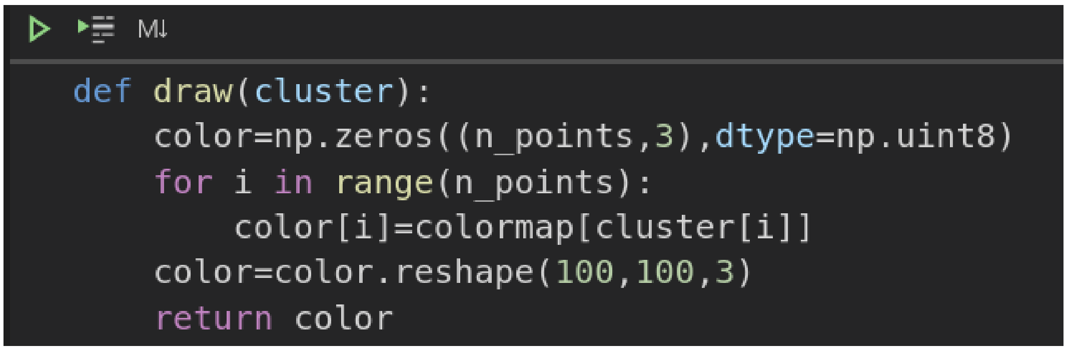


```
Normalized spectral clustering according to Ng, Jordan, and Weiss (2002)
Input: Similarity matrix $S \in \mathbb{R}^{n\times n}$, number $k$ of clusters to construct.
  • Construct a similarity graph by one of the ways described in Section 2. Let $W$
    be its weighted adjacency matrix.
  • Compute the normalized Laplacian $L_{\text{sym}}$ $D^{-1/2}LD^{-1/2}$
  • Compute the first $k$ eigenvectors $u_1, \dots, u_k$ of $L_{\text{sym}}$.
  • Let $U \in \mathbb{R}^{n\times k}$ be the matrix containing the vectors $u_1, \dots, u_k$ as columns.
  • Form the matrix $T \in \mathbb{R}^{n\times k}$ from $U$ by normalizing the rows to norm 1,
    that is set $t_{ij} = u_{ij}/(\sum_k u_{ik}^2)^{1/2}$.
  • For $i = 1, \dots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the $i$-th row of $T$.
  • Cluster the points $(y_i)_{i=1,\dots,n}$ with the $k$-means algorithm into clusters $C_1, \dots, C_k$.
Output: Clusters $A_1, \dots, A_k$ with $A_i = \{j \mid y_j \in C_i\}$.
```

I follow the spectral clustering algorithm pseudo code and implement the algorithm. In my code, W is similarity matrix and D is degree matrix. And I use the two matrix to calculate the graph Laplacian L. Based on normalized cut or ratio cut, it uses the different way to compute the L. After I get the L, I compute the first k(cluster number) eigenvectors of L.



```python
def kmeans(X,k):
    #init
    cluster,mean=init(X,init_method)

    distance=np.zeros(k)
    delta=100
    iter=0
    gif_buff=[]
    while delta!=0:
        pre_cluster=cluster.copy()
        iter+=1
        print('Iter: ',iter)
        for i in range(n_points):
            for j in range(k):
                distance[j]=np.sum((X[i]-mean[j])**2)
            cluster[i]=np.argmin(distance)

        mean=np.zeros((k,k))
        cnt=np.zeros(k)
        for i in range(n_points):
            mean[cluster[i]]+=X[i]
            cnt[cluster[i]]+=1
        for i in range(k):
            mean[i]/=cnt[i]

        gif_buff.append(draw(cluster))

        delta=np.sum(np.absolute(cluster-pre_cluster))
        print('Delta: ',delta)
        print('-----------------------------------')

    imageio.mimsave('spectral_'+str(n_cluster)+'_'+method+'_'+init_method+'_'+img_name+'.gif',gif_buff,'GIF',duration=0.1)
    return cluster
```

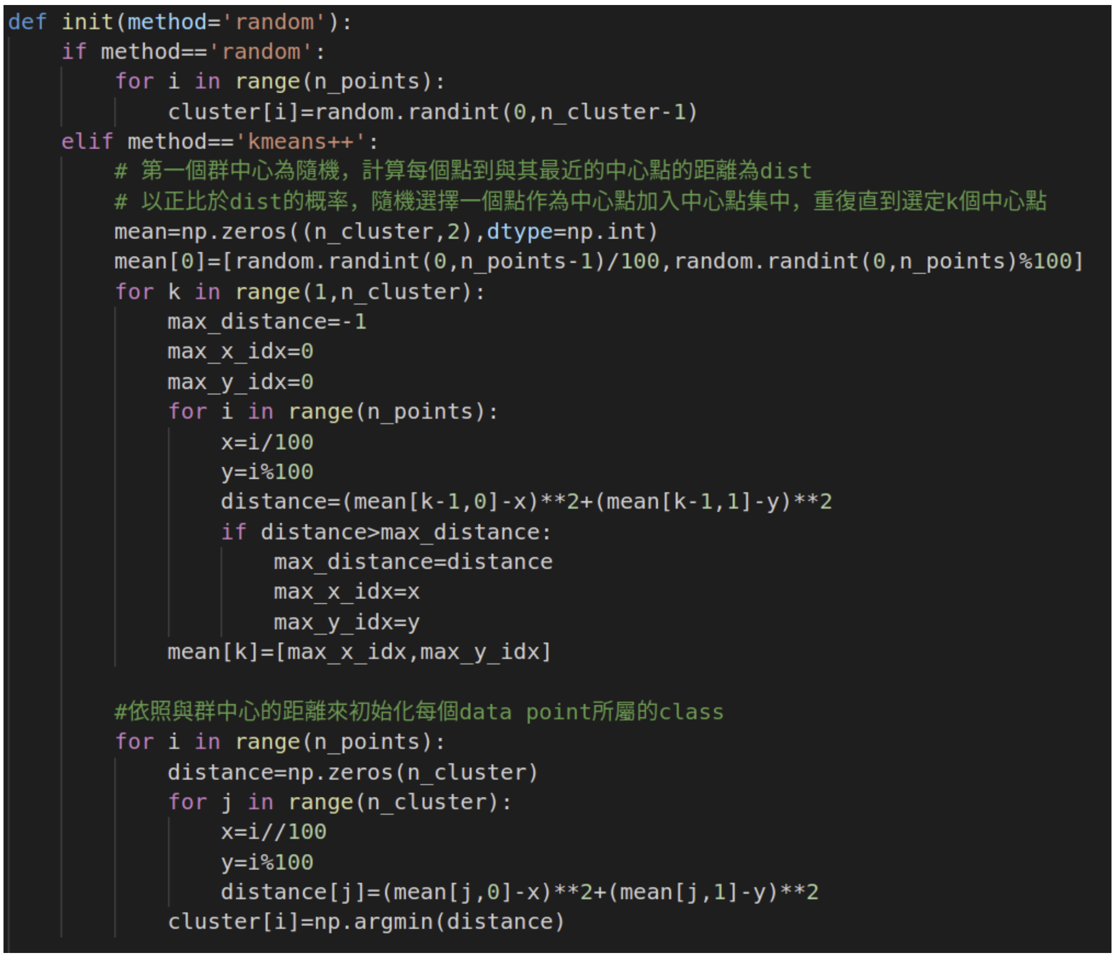


```python
def draw(cluster):
    color=np.zeros((n_points,3),dtype=np.uint8)
    for i in range(n_points):
        color[i]=colormap[cluster[i]]
    color=color.reshape(100,100,3)
    return color
```

I feed the first k eigenvectors into “k-means” function. Then, I can get the result of clustering. In “draw” function, I draw the same color in same cluster. And I use the result to make the gif file by imageio.mimsave().

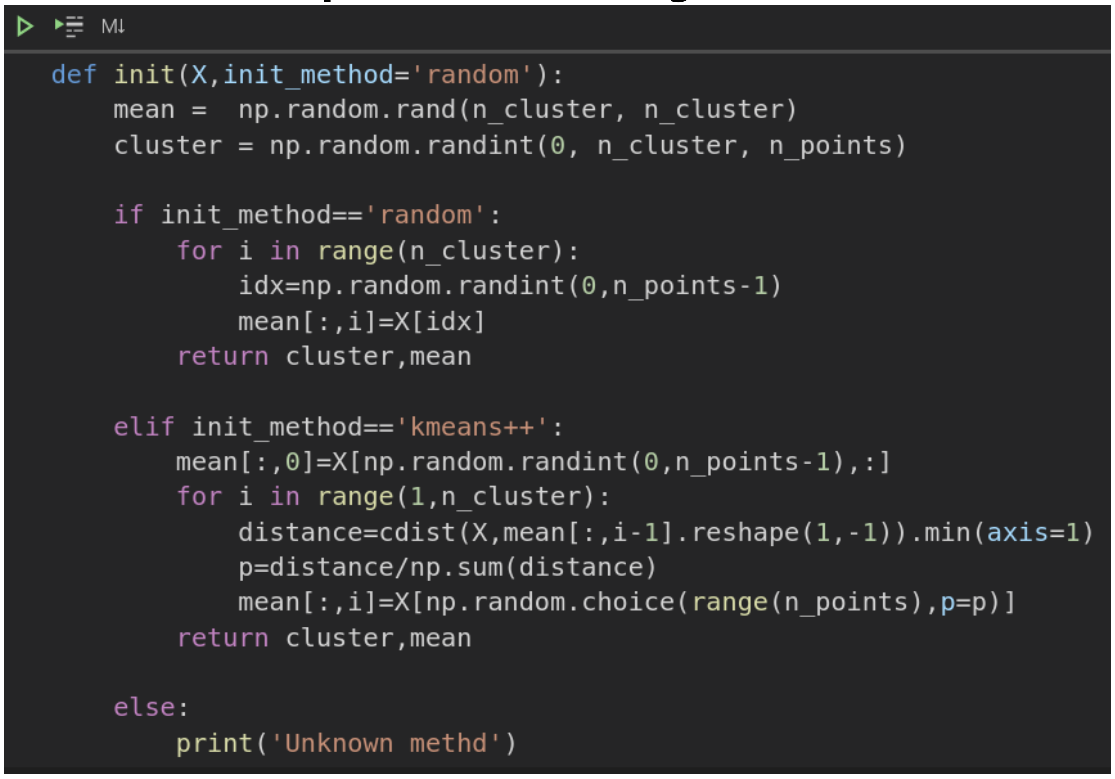
- **Part3**

kernel k-means



```python
def init(method='random'):
    if method=='random':
        for i in range(n_points):
            cluster[i]=random.randint(0,n_cluster-1)
    elif method=='kmeans++':
        # 第一個群中心為隨機，計算每個點到與其最近的中心點的距離為dist
        # 以正比於dist的概率，隨機選擇一個點作為中心點加入中心點集中，重復直到選定k個中心點
        mean=np.zeros((n_cluster,2),dtype=np.int)
        mean[0]=[random.randint(0,n_points-1)/100,random.randint(0,n_points)%100]
        for k in range(1,n_cluster):
            max_distance=-1
            max_x_idx=0
            max_y_idx=0
            for i in range(n_points):
                x=i/100
                y=i%100
                distance=(mean[k-1,0]-x)**2+(mean[k-1,1]-y)**2
                if distance>max_distance:
                    max_distance=distance
                    max_x_idx=x
                    max_y_idx=y
            mean[k]=[max_x_idx,max_y_idx]

        #依照與群中心的距離來初始化每個data point所屬的class
        for i in range(n_points):
            distance=np.zeros(n_cluster)
            for j in range(n_cluster):
                x=i//100
                y=i%100
                distance[j]=(mean[j,0]-x)**2+(mean[j,1]-y)**2
            cluster[i]=np.argmin(distance)
```

Spectral clustering



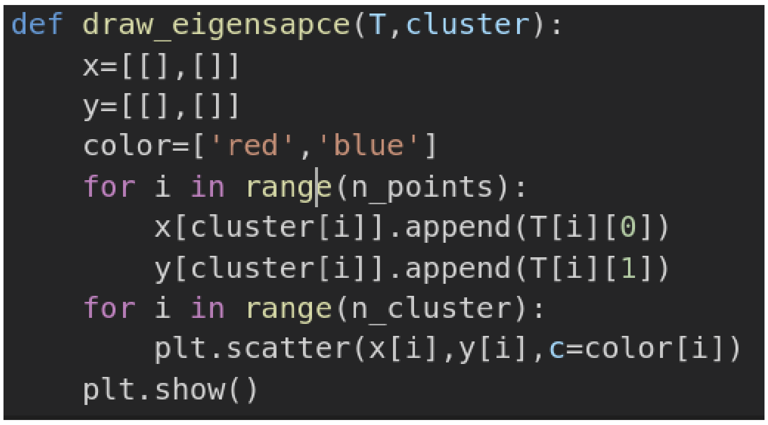
```
def init(X,init_method='random'):
    mean =  np.random.rand(n_cluster, n_cluster)
    cluster = np.random.randint(0, n_cluster, n_points)

    if init_method=='random':
        for i in range(n_cluster):
            idx=np.random.randint(0,n_points-1)
            mean[:,i]=X[idx]
        return cluster,mean

    elif init_method=='kmeans++':
        mean[:,0]=X[np.random.randint(0,n_points-1),:]
        for i in range(1,n_cluster):
            distance=cdist(X,mean[:,i-1].reshape(1,-1)).min(axis=1)
            p=distance/np.sum(distance)
            mean[:,i]=X[np.random.choice(range(n_points),p=p)]
        return cluster,mean

    else:
        print('Unknown methd')
```

In the part1 and part2, I initial the k-means with random method. I random assign the cluster center by choose the data points randomly.
In the part3, I use the k-means++ to initialized the cluster centers. The first cluster center is choose randomly.

In kernel k-means, the next center should far away the last center. It means the cluster will separate better. After getting the cluster center, I can compute the cluster of every data point.

In spectral clustering, the other cluster centers are used the the probability that is proportional in distance between every data point and the nearest cluster center to pick the data points to cluster centers. Repeat the step until choose the all k centers.

- **Part4**



```python
def draw_eigensapce(T,cluster):
    x=[[],[]]
    y=[[],[]]
    color=['red','blue']
    for i in range(n_points):
        x[cluster[i]].append(T[i][0])
        y[cluster[i]].append(T[i][1])
    for i in range(n_cluster):
        plt.scatter(x[i],y[i],c=color[i])
    plt.show()
```

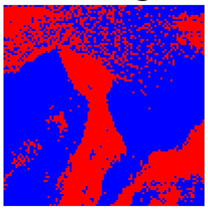
Eigenvectors are the coordinate in eigenspace. In order to visualized easily, I draw the eigenspace when cluster number is two. I put the eigenvectors of same cluster together. The first dimension of eigenvector takes as x and the second dimension as y.

b. experiments settings and results

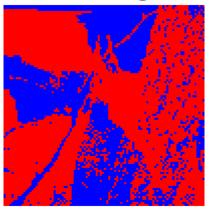
- **Part1**

  Kernel k-means

  - image1



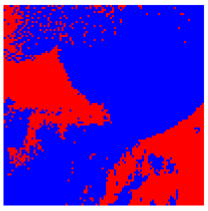
  - image2



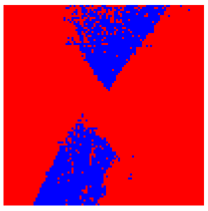
Spectral clustering

- image1

ratio

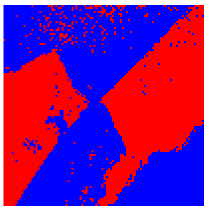


normalized

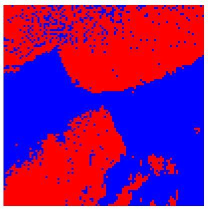


- image2

ratio



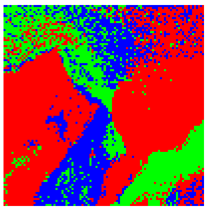
normalized



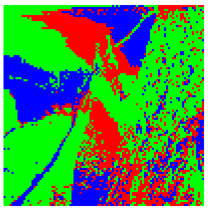
- **Part2**

Kernel k-means k=3

- image1



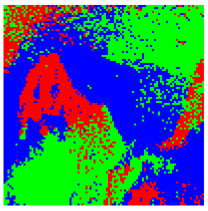
- image2



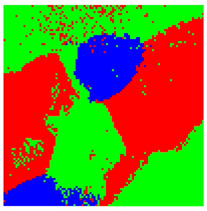
Spectral clustering k=3

- image1

ratio

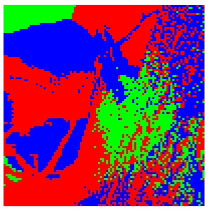


normalized

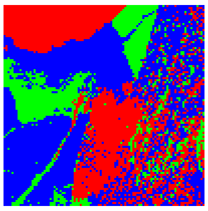


- image2

ratio



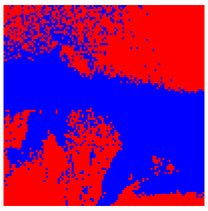
normalized



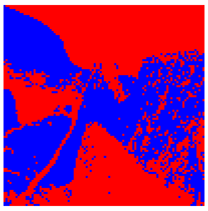
- **Part3**

Kernel k-means with k-means++

- image1



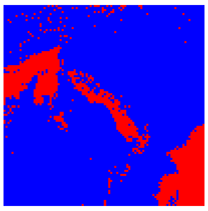
- image2



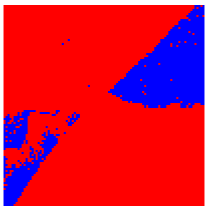
Spectral clustering with k-means++

- image1

ratio

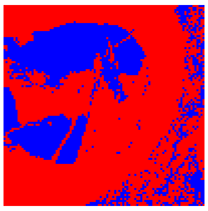


normalized

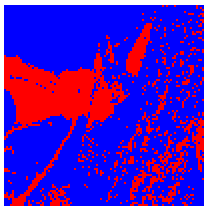


- image2

ratio



normalized

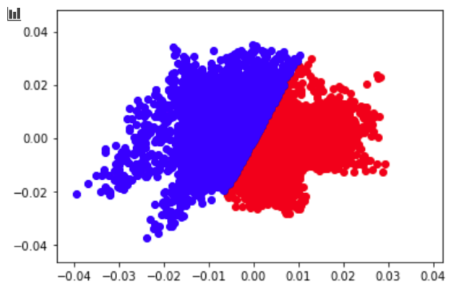


- **Part4**

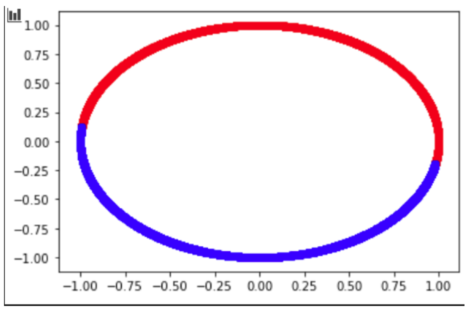
  we can find the data points within the same cluster have the similar coordinates in the eigenspace in this experiment.

  - image1

  ratio

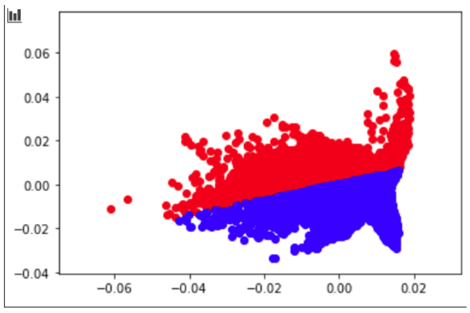


  normalized

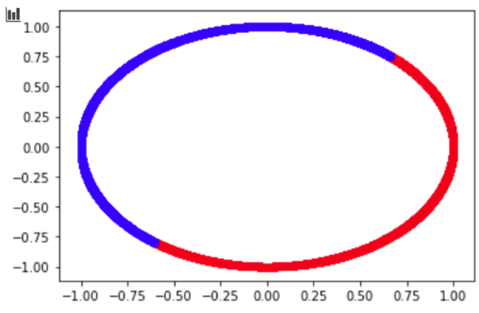


- image2

  ratio



  normalized



c. observations and discussion

I take the image1, and the cluster number is 2 for the execution time experiment.

- Normalized cut with random initialization takes 12 iteration
- ratio cut with random initialization takes 24 iteration

In my observation, I find the ratio cut will take more time to finish the clustering.