ML Homework 6

Bo-Cheng Huang, 309652022

1 Kernel K-means and Spectral Clustering

1.1 Code with detailed explanations

```python
img1=cv2.imread('image1.png')
img2=cv2.imread('image2.png')
X=np.zeros((10000,2)) #spatial information
for i in range(10000):
    X[i][0]=i//100+1
    X[i][1]=i%100+1
Y1=img1.reshape((10000,3)) #color information
Y2=img2.reshape((10000,3))
```

Listing 1: Data preprocessing of 2 images

For both kernel k-means and spectral clustering, I use the kernel defined in spec to compute the Gram matrix K.

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

where $S(x)$ is the spatial information of data x, and $C(x)$ is the color information of data x. I use $\gamma_s = \gamma_c = 0.001$ as initial parameters. This function is the kernel of kernel k-means and the similarity function of spectral clustering.

```python
def kernel(S,C,gamma1,gamma2):
    '''
    S: (n,2)-array, spatial information of data
    C: (n,3)-array, color information of data
    gamma1, gamma2: parameters
    '''
    RBFS=dis.squareform(np.exp(-gamma1*dis.pdist(S,'sqeuclidean')))
    RBFC=dis.squareform(np.exp(-gamma2*dis.pdist(C,'sqeuclidean')))
    return RBFS*RBFC+np.identity(np.size(S,0))
```

Listing 2: Kernel

In kernel k-means method, it uses kernel trick to compute the Euclidean norm of a feature space and the formula in slides is

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

where x_j is a data point, C_k is a cluster set and α_{kn} is an indicator whether data point x_n in cluster C_k or not. Then computing distance between data points and center, choose the closest cluster and update the label of each data point.

```
def clustering(list1,K,k):
    '''
    list1: initial centers of every cluster
    K: kernel function
    k: number of clusters
    return (n,)-array, the index of data
    '''
    cluster=np.zeros((100,100))
    alpha=np.zeros((k,10000))
    for i in range(k):
        alpha[i][list1[i]]=1
    for t in range(10):
        dist=np.ones((10000,k))
        mean=np.zeros(k)
        for i in range(k):
            mean[i]=1/np.sum(alpha[i])**2*np.sum(np.outer(alpha[i],alpha[i])*K)
        for i in range(10000):
            for j in range(k):
                dist[i,j]-=2/np.sum(alpha[j])*np.sum(alpha[j]*K[i])
                dist[i,j]+=mean[j]
        for i in range(10000):
            ind=np.argmin(dist[i])
            cluster[i//100][i%100]=ind
            for j in range(k):
                alpha[j][i]=0
            alpha[ind][i]=1
        plt.imsave('new_'+str(t)+'.png',cluster)
        return cluster.reshape(1,-1)
```

Listing 3: Clustering

To do kernel k-means for images, I use k-means++ to find initial center of every cluster. (init_list2 is in part 3) After doing 10 times of k-means, there are

10 images containing the label of data points.

```
K1=kernel(X,Y1,0.001,0.001)
K2=kernel(X,Y2,0.001,0.001)
k=2
kmeans=init_list2(np.log(1/K1),k)
clustering(kmeans,K1,k)
kmeans=init_list2(np.log(1/K2),k)
clustering(kmeans,K2,k)
```

Listing 4: kernel k-means of image 1/image 2

In spectral clustering method, for ratio cut, following the algorithm of unnormalized spectral clustering in slides, choose H containing first 2^{nd} to $(k+1)^{th}$ eigenvectors of L then do k-means to H.

```
def ratio_cut(K):
    '''
    K: kernel of data
    return (n)-array, eigenvalue and (n,n)-array, eigenvector
    '''
    D=np.diag(np.sum(K,axis=1))
    L=D-K
    eigenvalue,eigenvector=np.linalg.eigh(L)
    return eigenvalue,eigenvector
eigenvaluer1,eigenvectorr1=ratio_cut(K1)
sort_indexr1=np.argsort(eigenvaluer1)
k=2
H=eigenvectorr1[:,sort_indexr1[1:k+1]]
```

Listing 5: Find eigenvectors of L

To do k-means to H, I use the linear kernel $k(x,x') = x^T x'$ and k-means++ to cluster rows of H.

```
Hdist=dis.squareform(dis.pdist(H,'sqeuclidean'))
ratio=init_list2(Hdist,k)
clustering(ratio,H@H.T,k)
```

Listing 6: Ratio cut of image 1

For normalized cut, following the algorithm of normalized spectral clustering according to Ng, Jordan, and Weiss in slides, choose T containing first 2^{nd} to $(k+1)^{th}$ eigenvectors of $L_{sym} = D^{-\frac{1}{2}}LD^{-\frac{1}{2}}$.

```
def normal_cut(K):
    '''
    K: kernel of data
    return (n)-array, eigenvalue and (n,n)-array, eigenvector
    '''
    D=np.diag(np.sum(K,axis=1))
    L=D-K
    D_inverse_square_root=np.diag(1/np.diag(np.sqrt(D)))
    L_sym=D_inverse_square_root@L@D_inverse_square_root
    eigenvalue,eigenvector=np.linalg.eigh(L_sym)
    return eigenvalue,eigenvector
eigenvaluen1,eigenvectorn1=normal_cut(K1)
sort_indexn1=np.argsort(eigenvaluen1)
k=2
T=eigenvectorn1[:,sort_indexn1[1:k+1]]
```

Listing 7: Find eigenvectors of $D^{-\frac{1}{2}}LD^{-\frac{1}{2}}$

Normalizing the rows of T to norm 1, we have a new matrix H. To do k-means to H, I use the linear kernel $k(x, x') = x^T x'$ and k-means++ to cluster rows of H same as ratio cut.

```
sums=np.sqrt(np.sum(np.square(T),axis=1)).reshape(-1,1)
H=T/sums
Hdist=dis.squareform(dis.pdist(H,'sqeuclidean'))
normal=init_list2(Hdist,k)
clustering(normal,H@H.T,k)
```

Listing 8: Normalized cut of image 1

In part 1, use the images obtained to make a GIF by package imageio.

```
imagelist=[]
for i in range(10):
    imagelist.append('new_'+str(i)+'.png')
with imageio.get_writer('newgif.gif', mode='I') as writer:
    for filename in imagelist:
        image = imageio.imread(filename)
        writer.append_data(image)
```

Listing 9: Making GIF

In part 2, I only need to change the parameter k to 3 or 4 as in part 1. (The codes of part 2 are same as Listing 4,6,8)

In part 3, for the initialization of k-means clustering, I use randomized choices and k-means++ to get the center of each cluster.

```
def init_list1(k):
    '''
    k: number of clusters
    return the list containing index of k centers
    '''
    list1=[]
    for i in range(k):
        x=rd.randint(0,9999)
        list1.append(x)
    return list1
```

Listing 10: Randomized initialization

For k-means++, first I choose a data point x randomly and add to list, then compute the weight W related to euclidean distance between x and other points x'. Next, I use roulette wheel selection to pick next point $\tilde{x}$ add to list and compute the weight of other points, use the minimum weight between list and other points as the new weight. Repeat the steps until list has k points. (see Listing 11)

For kernel k-means, the weight is defined as

$$W(x,x') = \gamma_s \|S(x) - S(x')\|^2 + \gamma_c \|C(x) - C(x')\|^2.$$

For k-means in spectral clustering, the weight is defined as

$$W(x,x') = \|x - x'\|^2.$$

In part 4, I plot all data points from rows of an ndarray H which contains k eigenvectors of L (or L_{sym}) by spectral clustering. We can plot two eigenvectors of L (or L_{sym}), x-axis is first and y-axis is second. (see Listing 12)

For instance, after doing Listing 6, I have an array H and cluster index A, then plot the first column of H as x-axis and the second column of H as y-axis. (see Listing 13)

```python
def init_list2(Mat,k):
    '''
    Mat: (n,n)-array, weight of all pairs of data
    k: number of clusters
    return the list containing index of k centers
    '''
    list1=[]
    weight=np.zeros(10000)
    x=rd.randint(0,9999)
    list1.append(x)
    weight+=Mat[x]
    for i in range(k-1):
        prob=np.cumsum(weight/np.sum(weight))
        x=rd.random()
        for j in range(10000):
            if x<prob[j]:
                list1.append(j)
                break
        dis=np.zeros((2,10000))
        dis[0]=weight
        dis[1]=Mat[j]
        weight=np.min(dis,axis=0)
    return list1
```

Listing 11: kmeans++ initialization

```python
def plot_eigenspace(x_axis,y_axis,class_cluster):
    plt.figure()
    colors=['purple','orange']
    for colors,i in zip(colors,np.arange(2)):
        plt.scatter(x_axis[class_cluster==i],y_axis[class_cluster==i],c=colors,s=3)
    plt.xlabel('$1^{st}$ Eigenvector')
    plt.ylabel('$2^{nd}$ Eigenvector')
    plt.show()
```

Listing 12: Plot eigenspace

```python
A=clustering(ratio,H@H.T,k)
x_a=H[:,0]
y_a=H[:,1]
plot_eigenspace(x_a.reshape(1,-1),y_a.reshape(1,-1),A.astype(int))
```

Listing 13: Example of using plot_eigenspace

1.2 Experiments settings and results

In part 1, with $k = 2$ and kmeans++, these are results of image 1 and image 2 by 3 methods. (In Figure 1 to 10, click the picture (GIF) it will move) For image 1, it separates island and sea. For image 2, it separates sky from tree and rabbit. I observe that using kernel k-means takes more iteration to converge while using spectral clustering doesn't.

Figure 1: kernel k-means and spectral clustering (ratio cut, normalized cut) with $k = 2$ for image 1

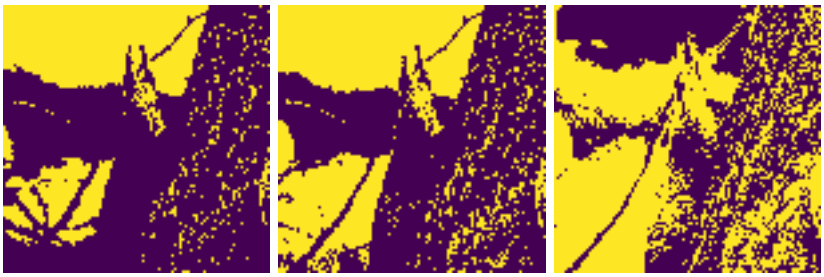

Figure 2: kernel k-means and spectral clustering (ratio cut, normalized cut) with $k = 2$ for image 2

In part 2, with $k = 2, 3, 4$ and kmeans++, these are results of image 1 and image 2 by 3 methods. It is more likely to classify different regions of an image by more clusters. For image 1, there are roughly 4 regions (island, sky, sea, etc.). But for image 2, since it is not clear to classify different parts in image (rabbit, tree, sky, etc.) by the RGB value of pixels, the performance of image 1 is greater than image 2.

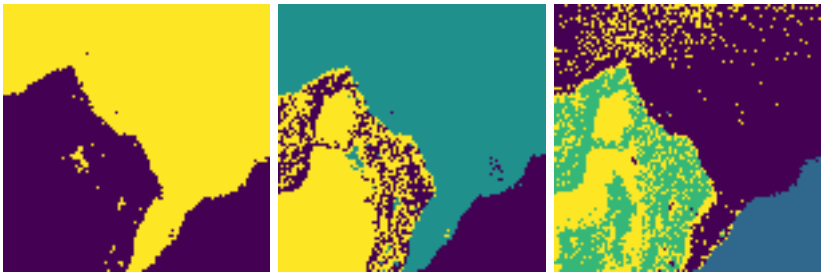

Figure 3: kernel k-means from $k = 2$ to $k = 4$ for image 1

Figure 4: Spectral clustering (ratio cut) from $k = 2$ to $k = 4$ for image 1

Figure 5: Spectral clustering (normalized cut) from $k = 2$ to $k = 4$ for image 1

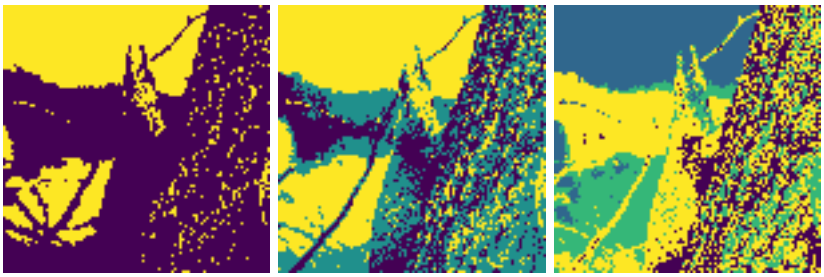

Figure 6: kernel k-means from $k = 2$ to $k = 4$ for image 2

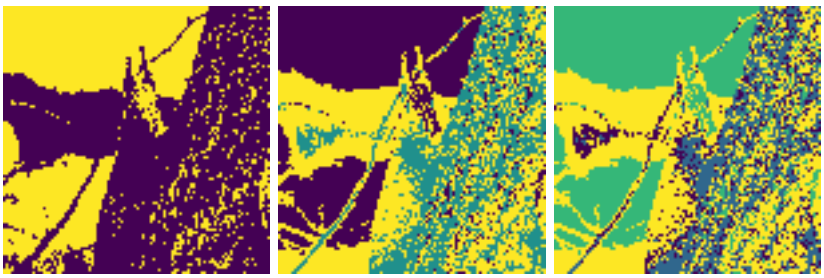

Figure 7: Spectral clustering (ratio cut) from $k = 2$ to $k = 4$ for image 2

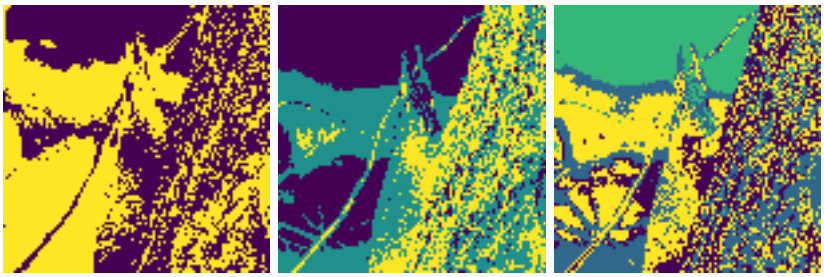

Figure 8: Spectral clustering (normalized cut) from $k = 2$ to $k = 4$ for image 2

In part 3, if I just randomly choose some point as center of clusters, then it has low probability to do the classification well. For k-means++, since the algorithm ensures it has higher probability to select centers with longer distance, the performance of doing k-means would be better than random choice. But if there are some extreme points inside image, then the probability that a extreme point is picked by k-means++ is very high. (see Figure 11) To prevent such condition, consider the mean μ of dataset and if one point is too far from μ, then this point shouldn't be a center of cluster.

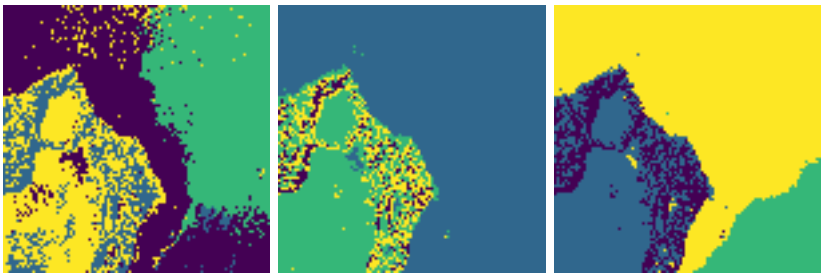

Figure 9: Random choice using kernel k-means and spectral clustering (ratio cut, normalized cut) with $k = 4$ for image 1

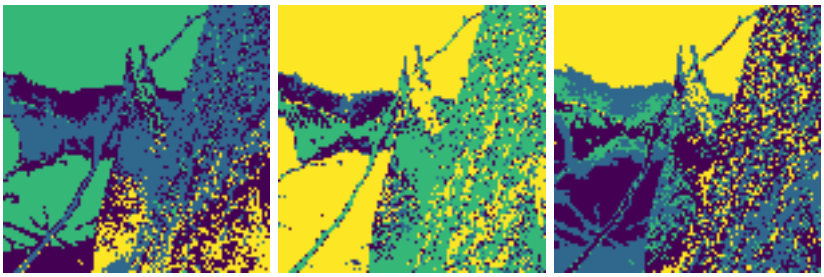

Figure 10: Random choice using kernel k-means and spectral clustering (ratio cut, normalized cut) with $k = 4$ for image 2

Figure 11: Bad exterme point by k-means++ and ratio cut of image 2

In part 4, first consider the Laplacian matrix L of unnormalized cut (ratio cut). In Figure 12, H contains eigenvectors of L of Listing 6, the range of x-axis is $[-0.03, 0.01]$ and the range of y-axis is $[-0.015, 0.01]$. It is mainly cut at $x = 0.004$. In figure 13, H contains eigenvectors of L of Listing 6, except the extreme point $(-1, -0.11)$, the range of x-axis is $[-0.1, 0]$ and the range of y-axis is $[-0.02, 0.02]$. It is mainly cut at $y = 0$. Note that if $(-1, -0.11)$ is chosen to be center of a cluster, then there is always a point in that cluster.

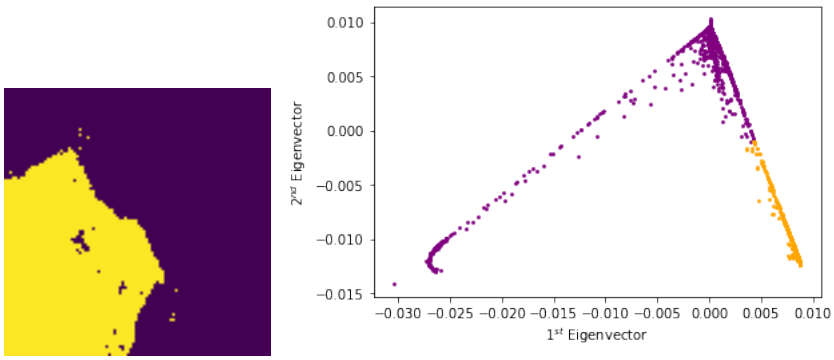

Figure 12: Ratio cut and scatter of H for image 1

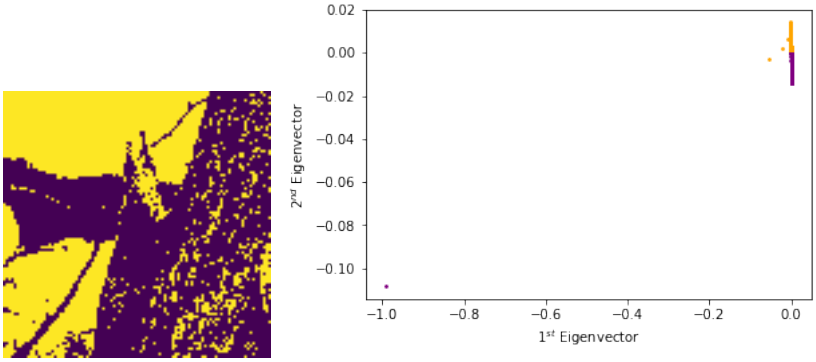

Figure 13: Ratio cut and scatter of H for image 2

Next, consider the symmetric normalized Laplacian matrix L_{sym} of normalized cut (normalized cut). In figure 14, there are 2 graphs for image 1, T contains eigenvectors of L_{sym} on bottom and H is norm 1 version of T on top of Listing 8. After changing the rows into norm 1, the graph is like a ring of eigenspace coordinates. Before takeing norm 1, the range of x-axis is $[-0.033, 0.01]$ and the range of y-axis is $[-0.013, 0.022]$. It is mainly cut at $y = 0$. After takeing norm 1, it is mainly cut along $y = 0.25x - 0.15$.

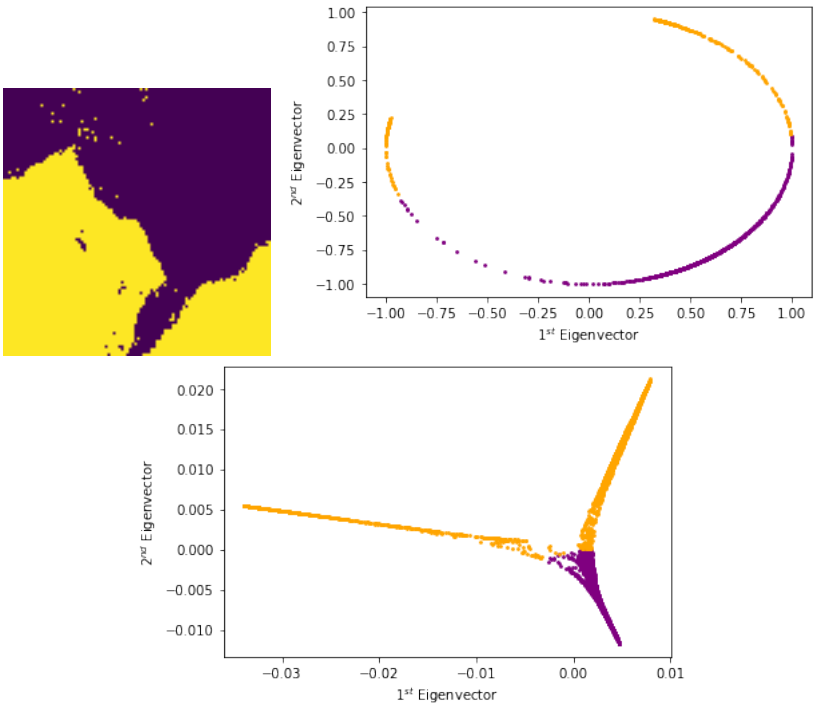

Figure 14: Normalized cut and scatter of H (right-top), T (bottom) for image 1

In Figure 15, there are 2 igraphs for image 2 T contains eigenvectors of L_{sym} on bottom and H is norm 1 version of T on top as Figure 14. After changing the rows into norm 1, the graph is like a ring of eigenspace coordinates. Before takeing norm 1, the range of x-axis is $[-0.018, 0.018]$ and the range of y-axis is $[-0.019, 0.018]$. It is mainly cut at $y = 0$. After takeing norm 1, it is mainly cut along $y = 0.17x + 0.07$.

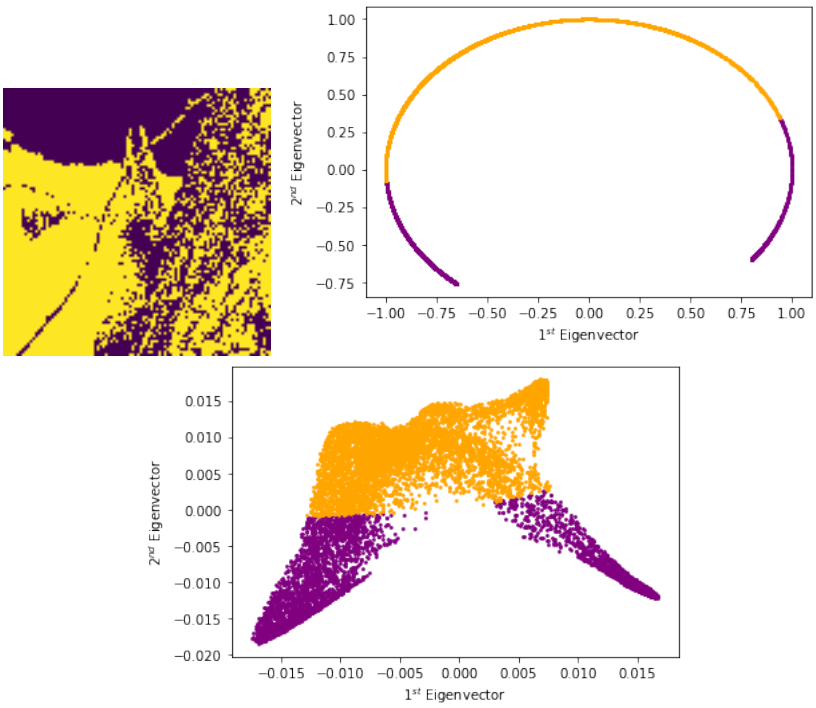

Figure 15: Normalized cut and scatter of H (right-top), T (bottom) for image 2

1.3 Observations and discussion

Along the procedure of HW6, to select the parameters γ_s and γ_c, if they are too large (e.g. $\gamma_s = \gamma_c = 1$), then the Gram matrix would be contains lots of zero and other nonzero terms are very small. For images like image 1, using color information is enough to classify different regions. For images like image 2, it needs to pay more attention about the spatial information related to each pixels.

Consider the initialization of k-means algorithm, since either random choice or k-means++ is a probalisitic way, it needs to take times to get a better result of clustering. To construct a similarity graph, I use Gram matrix (fully connected graph) in HW6 and there are some other constructions such as restricting the number of neighbors of each data points. Comparing the original k-means algorithm with kernal k-means, it's more efficient and flexible to use kernel trick instead of the original version.

In spectral clustering method, it needs more times (6 minites) to compute the eigenvectors of a matrix, but fortunately, the matrix L and L_{sym} are symmetric so I can use a faster function eigh of numpy instead of eig.

To plot data points about eigenspaces with more clusters, first separating them pairwisely then make $\lceil \frac{k}{2} \rceil$ graphs. By the way, plot data using 3-dimensional space is also a way to see the relationship between eigenspaces and clusters.