ML_HW6 Report

1. Kernel K-means

- Read data

```
image1 = imageio.imread('image1.png')
image2 = imageio.imread('image2.png')
```

- Define a function can get kernel

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

S(x) is the spatial information of data, the value in each pixel is the (x,y) location in image. Shape of S(x) is (10000,2)

C(x) is the color information of data, the value in each pixel is its RGB values. Shape of S(x) is (10000,3)

```
def kernel(data, gamma_s, gamma_c):


    def spatial_information(data):
        S = np.empty((data.shape[0]*data.shape[1],2))
        for i in range(data.shape[0]):
            for j in range(data.shape[1]):
                S[i*data.shape[0] + j] = [i,j]
        return S

    def color_information(data):
        return data.reshape(data.shape[0]*data.shape[1] , 3)

    S = spatial_information(data)
    C = color_information(data)

    S_dist = squareform(pdist(S,'sqeuclidean'))
    S_K = np.exp(-gamma_s * S_dist)

    C_dist = squareform(pdist(C,'sqeuclidean'))
    C_K = np.exp(-gamma_c * C_dist)

    K = S_K*C_K

    return K
```

- Define the function can do kernel k-means and then draw pictures.

```
def kernel_kmeans(data, K, group, plus, title):

    if plus:
        a = find_init_a(data, group)
    else:
        a = random_a(data, group)

    ims = []
    fig = plt.figure()
    iter_num = 0
    while True:
        d = distance(K, a , group)
        group_indices = np.argmin(d, axis=1)
        s = group_indices.reshape(100,100)

        plt.imshow(s,animated=True)
        plt.title(title + str(group) + 'group iter:{}'.format(iter_num+1))
        plt.savefig(title + "/" + title + str(group) + "group_" +str(iter_num) + ".png")
        plt.show()

        new_a = np.zeros((K.shape[0] , group))
        for g in range(group):
            new_a[:, g] = (group_indices == g)


        if (a == new_a).all():
            break
        a = new_a
        iter_num+=1

    return iter_num
```

kernel_kmeans has a parameter called “plus”, if “plus” == True, then the the group initialized method will follow k-means++.
When “plus” == True, at the beginning of *kernel_kmeans* will call the *find_init_a* function to set the initial group.

```
def find_int_a(data,group):

    X = get_spatial_merge_color(data)
    center = find_init_center(X,group)
    group_indices = clustering(X, center,group)

    a = np.zeros((data.shape[0] * data.shape[1] , group))
    for g in range(group):
        a[:, g] = (group_indices == g)

    return a
```

In the *find_init_a* function, first, I try to get the spatial and color information at the same time, so I define a function called *get_spatial_merge_color*, it will return a matrix that the value of each pixel is [S,C](S is spatial information, C is color information), the shape of the matrix is (10000,5)

Then it will go into the function called *find_init_center*, it will base on the method that k-mean++ find the initial group, the principle of this method is let every initial group center as far as possible. The function will return a array include every group center.

```
def find_init_center(data,group):

    key = np.random.randint(0,data.shape[0])
    step =0
    center_list = []

    while step<10:
        if step == 0:
            seed = data[key]
        else:
            distance_list = np.sqrt(np.power(data - seed, 2).sum(axis=1)).tolist()
            dx = sum(distance_list)
            random = np.random.random()*dx
            for index in range(len(distance_list)):
                random -= distance_list[index]
                if random <=0:
                    seed  = distance_list[index]
                    break
            x_index  = index
            seed = data[x_index]
            center_list.append(seed)

            if len(center_list) == group:
                break

        step +=1

    center_list = np.array(center_list)

    return center_list
```

Calculate the euclidean distance between each point and those centers, then merge them to the closet center. *find_init_a* will return the initial alpha array “a”. “a_{kn}” == 1if the data point x_n is assigned to the k-th cluster. Otherwise, “a_{kn}” == 0.

The kernel distance between each points base on the formula below:

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

and I defined a function called *distance* to compute this.

```
def distance(K,a,group):
    d = np.zeros((K.shape[0] , group))
    for g in range(group):
        a_k = a[:, g]
        ck_size = a_k.sum()
        d[:,g] = np.diagonal(K)
        d[:,g] -= (2 / ck_size) * (a_k@K)
        d[:,g] += (1 / (ck_size*ck_size)) * ((a_k[..., np.newaxis] @ a_k[np.newaxis, ...]).T * K).sum()

    return d
```

Assign each point to the closet center, update “a” in each iterative until it converge.

- Run the *kenel_means* in different group number, and then make every pictures into gif.

The parameters in kernel function I set, image1 is gamma_s = 0.001, and gamma_c = 0.001, image2 is gamma_s = 0.0005, and gamma_c = 0.001.In the beginning, I set them all equal to 1, then I found that neither the kernel k-means nor spectral clustering can work, finally I realized that when the L2 norm between two point is big, the RBF value will be really small, moreover, the kernel function we defined was multiplying two RBF kernels in order to consider spatial similarity and color similarity at the same time. After multiplying two small values, t’s will make the kernel really really small , even 0. That why the kernel k-means and spectral clustering can’t work. After I set gamma_s and gamma_c below to 1, the situation get better.
In image1,Gamma_s = 0.001, and gamma_c = 0.001 can get the best clustering I have tried. In image2, there have depth of field, so I set Gamma_s = 0.0005, and gamma_c = 0.001, it means spatial is more important than color information.

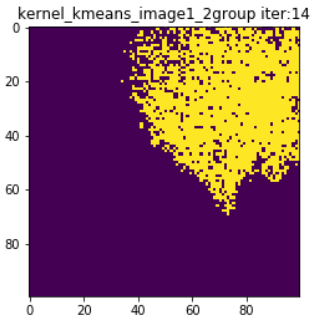
```
def make_gif(iter_num, group, title):
    images = []
    for i in range(iter_num+1):
        im = title + "/" + title+ "{}group_{}.png".format(group, i)
        images.append(imageio.imread(im))
    imageio.mimsave( "gif/" + title + str(group) + "group" +".gif", images, duration = 0.3)
```

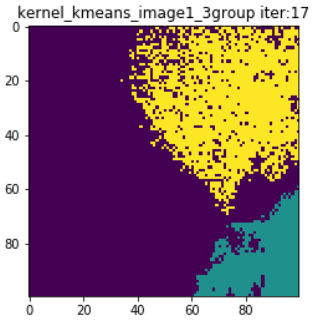
```
def kernel_kmeans_draw(image, dataname,K, plus):
    if plus:
        title = "kernel_kmeans+_{}_".format(dataname)
    else:
        title = "kernel_kmeans_{}_".format(dataname)
    for group in [2,3,4]:
        iter_num = kernel_kmeans(image, K , group, plus, title)
        make_gif(iter_num, group, title)
```

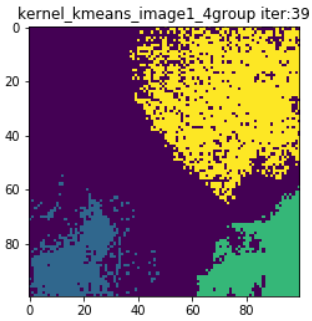
Below are the result (I just show the final cluster result here. The cluster assignments of data points in each iteration will be show in gif.), every image have 3 pictures, represents 2,3,4 group respectively.

1. Image1

```
K = kernel(image1, 0.001, 0.001)
kernel_kmeans_draw(image1,"image1", K, plus = False)
```

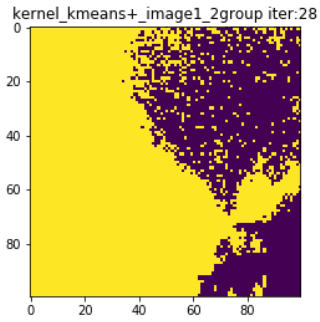


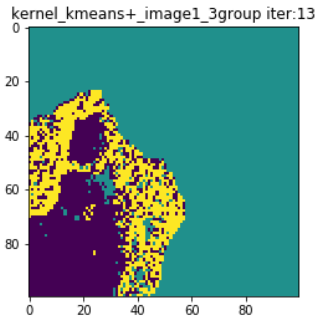


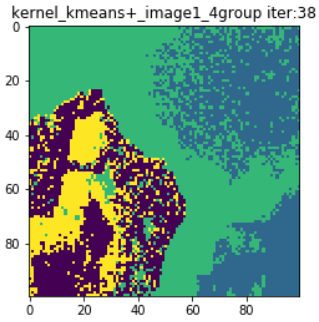


2. Image1 (k-means++)

```
K = kernel(image1, 0.001, 0.001)
kernel_kmeans_draw(image1,"image1", K,plus = True)
```

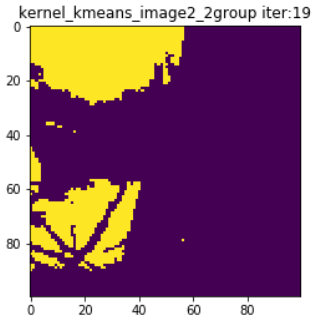


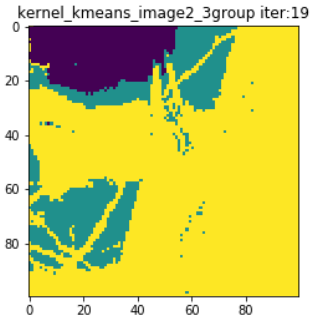


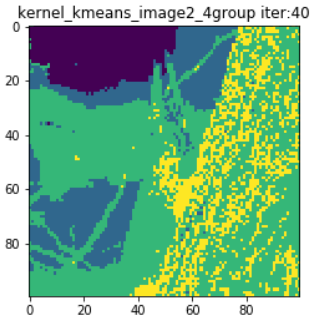


3. Image2

```
K = kernel(image2, 0.0005, 0.001)
kernel_kmeans_draw(image2,"image2", K, plus = False)
```

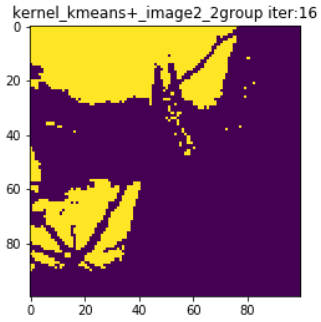


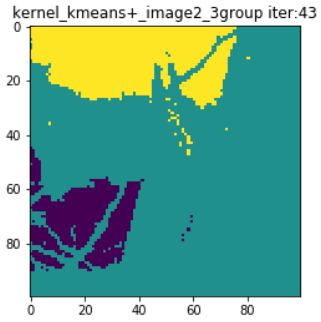


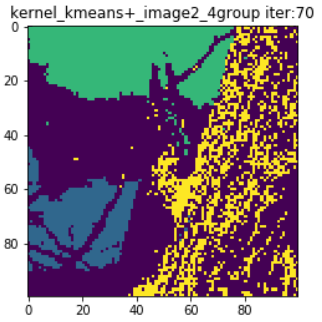


4. Image2 (k-means++)

```
K = kernel(image2, 0.0005, 0.001)
kernel_kmeans_draw(image2,"image2", K, plus = True)
```







- Discussion

In my opinion, choose k-means++ to be the initialization method indeed has better clustering result than normal k-means. In image1, kernel k-means++ can separate the land from the sea when group number is 3,4. Also, in image2, kernel k-means++ can separate the rabbit from others more clearly. I think this is because the distances between each initial group center are farer, it can make the clustering more reliable.

2. Spectral Clustering

- Both ratio cut and normalized cut is based on the algorithm in lecture slides.

ratio cut

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 $u_1, \ldots, u_k$ of L.**
- Let $U \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $u_1, \ldots, u_k$ as columns.
- For $i = 1, \ldots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of U.
- Cluster the points $(y_i)_{i=1,\ldots,n}$ in $\mathbb{R}^k$ with the k-means algorithm into clusters $C_1, \ldots, C_k$.

Output: Clusters $A_1, \ldots, A_k$ with $A_i = \{j | y_j \in C_i\}$.

normalized cut

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_{sym} $D^{-1/2} L D^{-1/2}$
- **Compute the first k eigenvectors $u_1, \ldots, u_k$ of L_{sym}.**
- Let $U \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $u_1, \ldots, u_k$ 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 $i = 1, \ldots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of T.
- Cluster the points $(y_i)_{i=1,\ldots,n}$ with the k-means algorithm into clusters $C_1, \ldots, C_k$.

Output: Clusters $A_1, \ldots, A_k$ with $A_i = \{j | y_j \in C_i\}$.

- Define the function to return array “U” that include the first K eigenvectors of L. (K : number of group)

```
def get_eig_radio(data,group):

    if data == "image1":
        data = image1
    else:
        data == image2

    W = kernel(data,0.001,0.001)
    D = np.diag(W.sum(axis=1))
    L = D-W
    eigenvalues, eigenvectors = np.linalg.eig(L)

    U = eigenvectors[:, np.argsort((eigenvalues))[:group]]

    return U
```

```
def get_eig_normalized(data,group):

    if data == "image1":
        data = image1
    else:
        data == image2

    W = kernel(data,0.0005,0.001)
    D = np.diag(W.sum(axis=1))
    D_sqrt = np.diag((W.sum(axis=1))**-0.5)
    L = D-W
    L_sym = D_sqrt@L@D_sqrt

    eigenvalues, eigenvectors = np.linalg.eig(L_sym)
    U = eigenvectors[:, np.argsort((eigenvalues))[:group]]

    return U
```

- Define a function called *kmeans, do k-means algorithm.*

  Same as kernel k-means, there is a parameter called “plus”, if “plus” == True, the clustering process will be K-means++ algorithm.

```
def random_centor_init(data,group):
    #random init centors
    center_list = []
    for i in range(group):
        key = np.random.randint(0,data.shape[0])
        center_list.append(data[key])

    center_list = np.array(center_list)

    return center_list
```

```
def kmeans(data, group , plus, title):

    if plus:
        center = find_init_center(data, group)
    else:
        center = random_centor_init(data, group)

    iter_num = 0
    plt.figure()
    while True:
        group_indices = clustering(data, center, group)

        plt.imshow(group_indices.reshape(100,100))
        plt.title(title + str(group) + 'group iter:{}'.format(iter_num+1))
        plt.savefig(title + "/" + title + str(group) + "group_" +str(iter_num) + ".png")
        plt.show()

        new_center = []
        for g in range(group):
            index = np.where(group_indices == g)[0]
            new_center.append(data[index].mean(axis=0))
        new_center = np.array(new_center)

        if (new_center==center).all():
            break
        center = new_center

        iter_num+=1

    return iter_num
```

- Run the *spectral_clustering_ratio_draw or spectral_clustering_normalized_draw* in different group number, and make every pictures into gif.

  Follow the algorithm in the lecture slides, in ratio cut let “U” be the data points, cluster them with the K-means algorithm. In normalized cut, first form the matrix T, and let “U” be the data points, cluster them with the K-means algorithm.

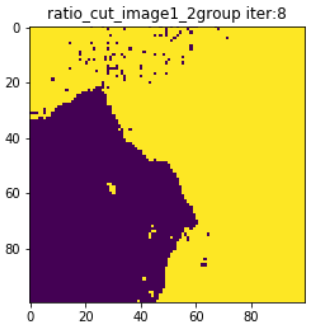
```
def spectral_clustering_ratio_draw(image, plus):
    if plus:
        title = "ratio_cut+_{}_".format(image)
    else:
        title = "ratio_cut_{}_".format(image)

    for group in [2,3,4]:
        U = get_eig_radio(image, group)
        iter_num = kmeans(U, group, plus , title)
        make_gif(iter_num, group, title)
```

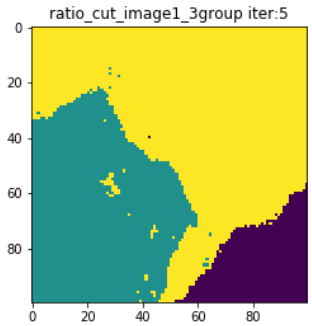
```
def spectral_clustering_normalized_draw(image, plus):

    if plus:
        title = "normalized_cut+_{}_".format(image)
    else:
        title = "normalized_cut_{}_".format(image)

    for group in [2,3,4]:
        U = get_eig_normalized(image, group)
        T = np.zeros_like(U)
        for i in range(10000):
            T[i, :] = U[i, :] / np.sqrt((U[i, :] * U[i, :]).sum())
        iter_num = kmeans(T, group, plus, title)
        make_gif(iter_num, group, title)
```

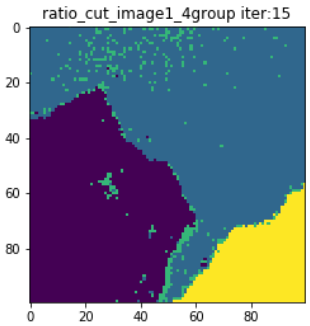
  Below are the result (I just show the final cluster result here. The cluster assignments of data points in each iteration will be show in gif.), every image have 3 pictures, represents 2,3,4 group respectively.

  1. Image1 ratio cut

```
spectral_clustering_ratio_draw("image1", plus =False)
```

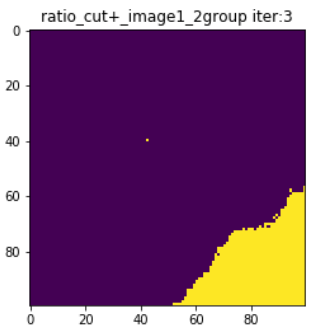


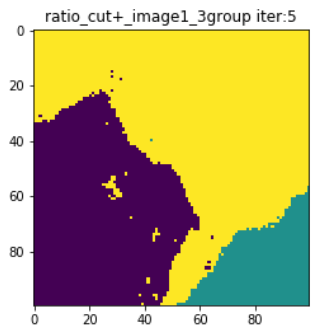


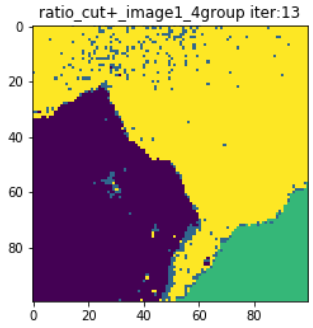


2. Image1 ratio cut (k-means++)

```
spectral_clustering_ratio_draw("image1", plus =True)
```

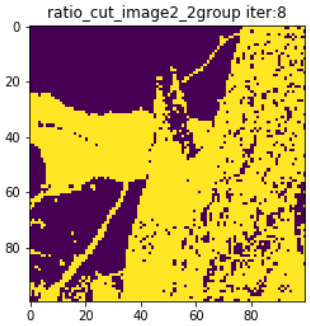


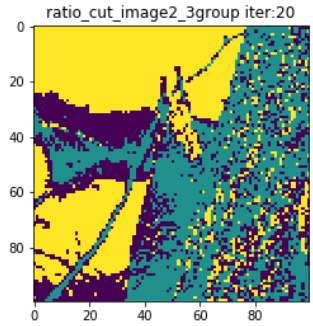


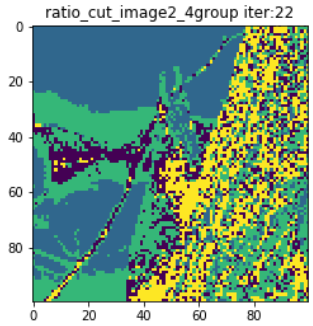


3. Image2 ratio cut

```
spectral_clustering_ratio_draw("image2", plus =False)
```

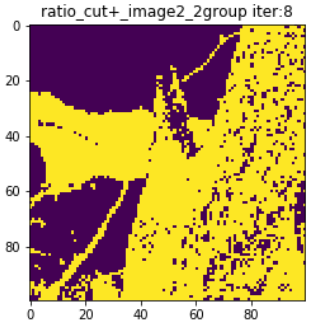


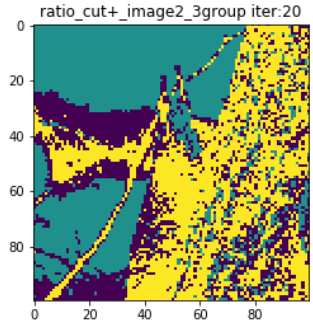


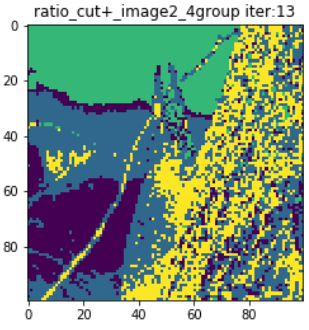


4. Image2 ratio cut (k-means++)

```
spectral_clustering_ratio_draw("image2", plus =True)
```

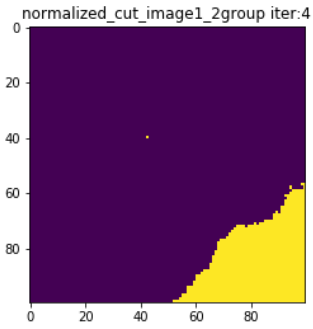


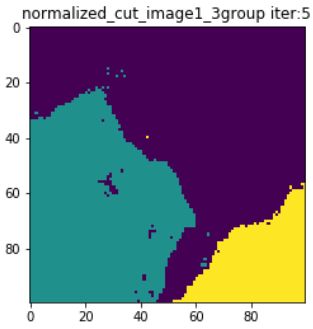


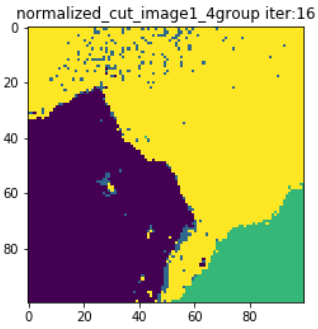


5. Image1 normalized cut

```
spectral_clustering_normalized_draw("image1", plus = False)
```

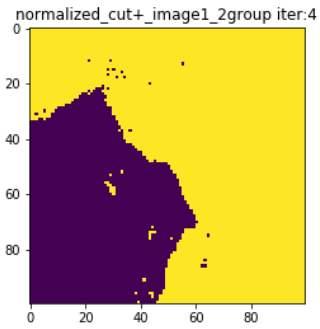


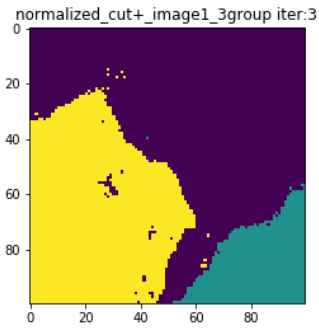


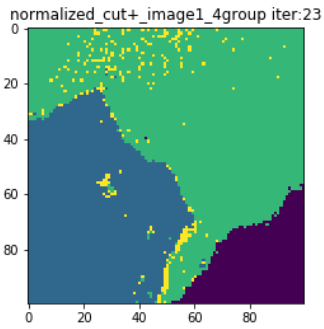


6. Image1 normalized cut (k-means++)

```
spectral_clustering_normalized_draw("image1", plus =True)
```

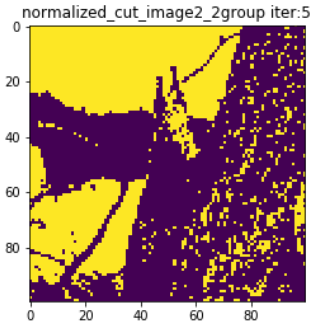


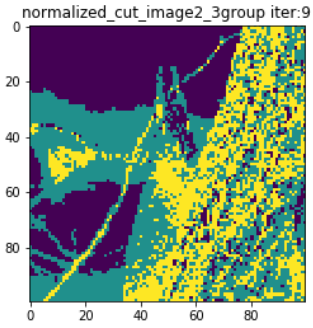


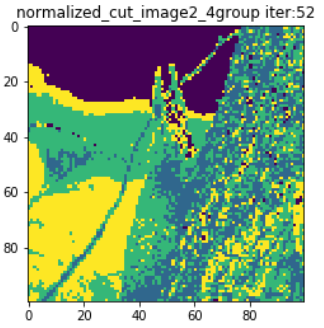


7. Image2 normalized cut

```
spectral_clustering_normalized_draw("image2", plus = False)
```

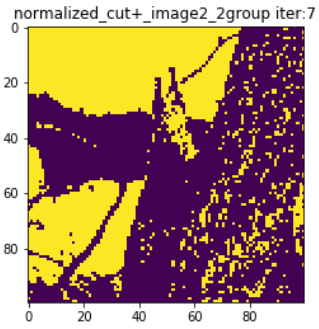


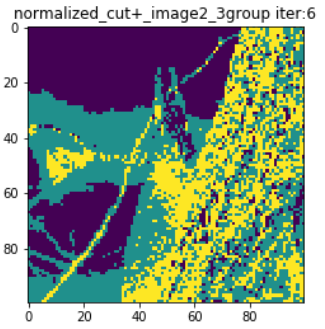


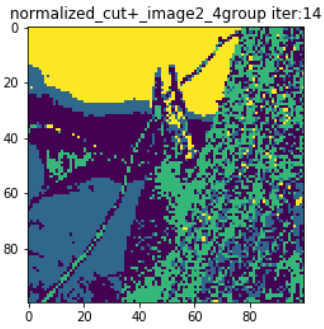


8. Image2 normalized cut (k-means++)

```
spectral_clustering_normalized_draw("image2", plus = True)
```







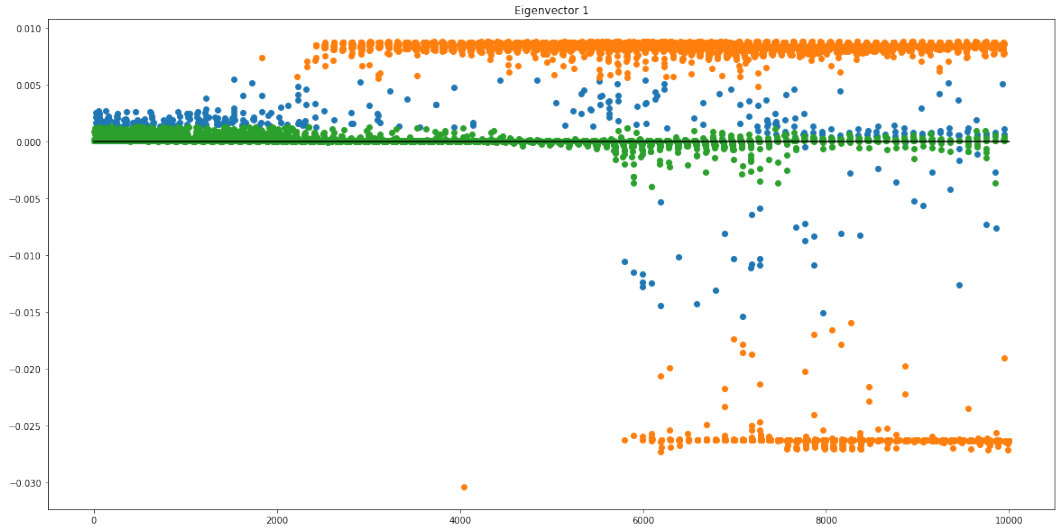
- Discussion

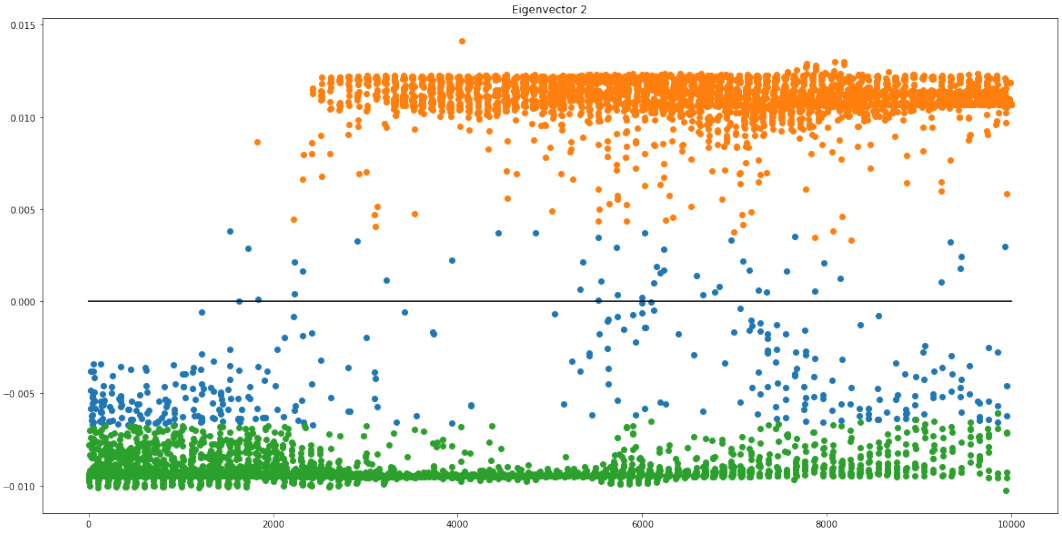
  Compare to kernel k-means, I think spectral clustering has better performance. However, I can't really tell ratio cut or normalized cut and k-means or k-means++ which ones are better just depends on my eyes. Also, the result depends on the group initialization.
  The reason why I set the spatial information more important than color information is that I wish to separate the rabbit and the tree from the background, not just cluster them by color. Although I think it indeed has better clustering result compare to gamma_s = gamma_c, but it still has a lots of improvement space.

3. Check data points in eigenspace

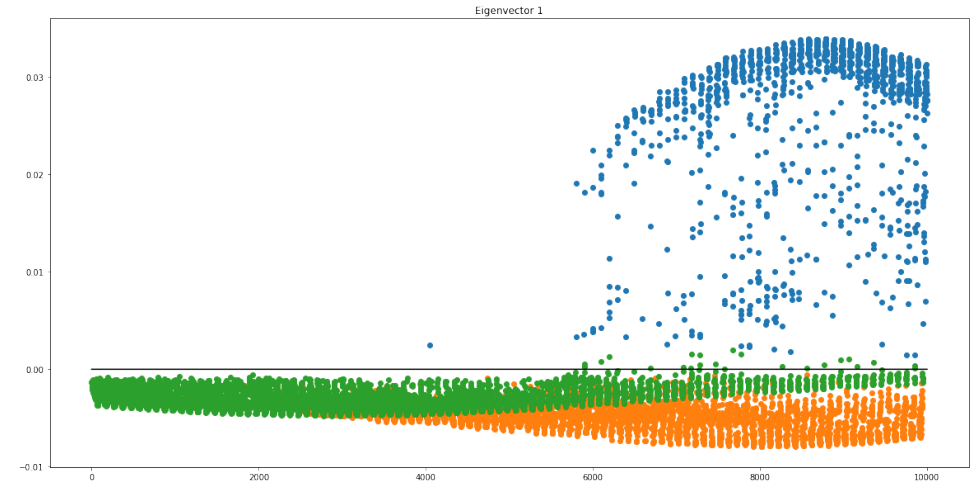
Use group =3, and image1 to do this experiment. I draw out all points and different group used different color. X-axis is the points index, y-axis is the values in Eigenvector. The black line is located on 0.

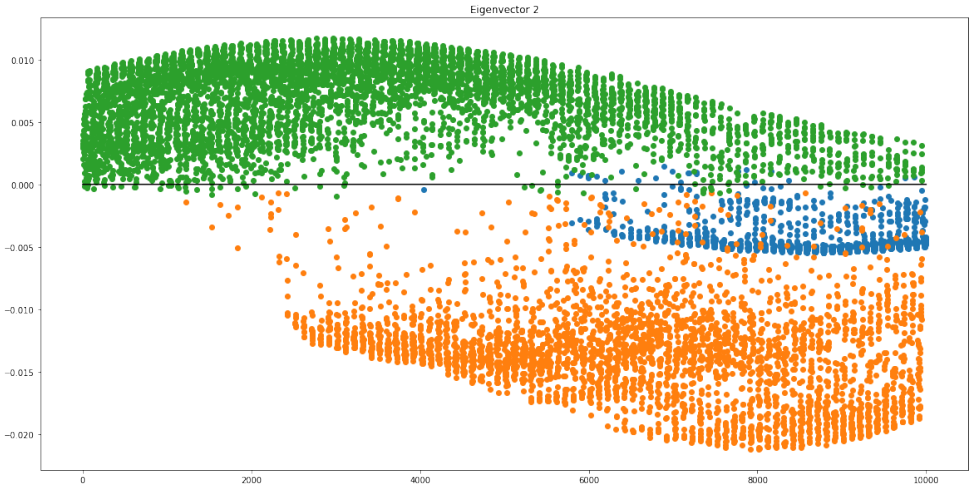
Ratio cut





Normalized cut





- Discussion

  In both ratio cut and normalized cut method, two eigenvector both can separate the whole data set into two group roughly. Most of the points in the same eigenspace assign to the same group. Notably, there have some points in the same group but locate in the different eigenspace, and this situation is more severe in the ratio cut. Although I can't tell the clustering result of ratio cut and normalized cut which is better depends on my eyes, in this experiment, the result of normalized cut seems better. And I think it's just the reason why we do "Normalize", normalization ease the problem cause by the different of the size of the group.