Machine Learning Homework 6

Student Name: 吳政緯

Student ID: 0510507

Introduction

I implemented two different unsupervised clustering methods to analysis the images. First one is Kernel k-means algorithm and the second one is spectral clustering algorithm including ratio cut and normalize cut. Comparing with different hyper-parameters and do some experiments with different initial center points.

The approach

I. Compute each data distance in k-means(Kernel k-means Clustering)
Use the kernel function to compute each data distance to different cluster center points. The first term is putting each data points and cluster members into the kernel function. The second term is putting the cluster members and cluster members into kernel functions. Finally, we could get the matrix of each data distance to different cluster center. The following function has the weight, but I use the unweighted method. Setting all weight function as constant 1 value.

$$\frac{2\sum_{\mathbf{b}\in\pi_j} w(\mathbf{b})\phi(\mathbf{a})\cdot\phi(\mathbf{b})}{\sum_{\mathbf{b}\in\pi_j} w(\mathbf{b})} + \frac{\sum_{\mathbf{b},\mathbf{c}\in\pi_j} w(\mathbf{b})w(\mathbf{c})\phi(\mathbf{b})\cdot\phi(\mathbf{c})}{(\sum_{\mathbf{b}\in\pi_j} w(\mathbf{b}))^2}$$

II. Kernel k-means algorithm:

WEIGHTED_KERNEL_KMEANS(K, k, w, $C_1, ..., C_k$)
Input: K: kernel matrix, k: number of clusters, w: weights for each point
Output: $C_1, ..., C_k$: partitioning of the points
1. Initialize the k clusters: $C_1^{(0)}, ..., C_k^{(0)}$.
2. Set $t = 0$.
3. For each point $\mathbf{a}$, find its new cluster index as
$$j^*(\mathbf{a}) = \text{argmin}_j \|\phi(\mathbf{a}) - \mathbf{m}_j\|^2, \text{ using (2).}$$
4. Compute the updated clusters as
$$C_j^{t+1} = \{\mathbf{a} \ : \ j^*(\mathbf{a}) = j\}.$$
5. If not converged, set $t = t + 1$ and go to Step 3; Otherwise, stop.

In every run of the k-means, we update the cluster members according to each point distance to cluster center.

III. Create Similarity Matrix(Spectral Clustering)
We could create a fully connected graph by Gaussian Kernels. The S(x) is the spatial

information and C(x) is the color information of data x.

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

IV. Ratio cut and Normalize cut(Spectral Clustering)

Because the Simple cut method has the problem of choosing small and isolated clusters, we have to put the factor of cluster size into the minimization function.

A. Ratio cut :

Following equation is the function we have to minimize, but actually the problem will become an NP hard problem. We only can approximate the mincut and it has been derived that Ratio cut is approximately same as the unnormalized spectral clustering.

$$\mathrm{RatioCut}(A_1, \ldots, A_k) = \sum_{i=1}^{k} \frac{\mathrm{cut}(A_i, \overline{A}_i)}{|A_i|}$$

B. Normalized cut:

Use the following equation to find the eigenvalues and eigenvectors of z and use the normalized spectral clustering algorithm.

$$\text{Find } z \text{ in } \quad D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}z = \lambda z \quad \text{where} \quad z = D^{\frac{1}{2}}y$$

$$\mathrm{Ncut}(A_1, \ldots, A_k) = \sum_{i=1}^{k} \frac{\mathrm{cut}(A_i, \overline{A}_i)}{\mathrm{vol}(A_i)}$$

V. Normalized spectral clustering (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}.
- **Compute the first k eigenvectors $v_1, \ldots, v_k$ of L_{sym}.**
- Let $V \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $v_1, \ldots, v_k$ as columns.
- **Form the matrix $U \in \mathbb{R}^{n \times k}$ from V by normalizing the row sums to have norm 1,** that is $u_{ij} = v_{ij} / (\sum_k v_{ik}^2)^{1/2}$.
- For $i = 1, \ldots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of U.
- 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 $A_i = \{j | y_j \in C_i\}$.

VI. Unnormalized spectral clustering (Unnormalized Cut)

Unnormalized spectral clustering

Input: Similarity matrix $S \in \mathbb{R}^{n \times n}$, number k of clusters to construct

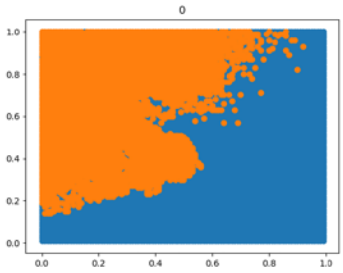
- Construct a similarity graph by one of the ways described in Section 2. Let W be its weighted adjacency matrix.
- Compute the unnormalized Laplacian L.
- **Compute the first k eigenvectors $v_1, \ldots, v_k$ of L.**
- Let $V \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $v_1, \ldots, v_k$ as columns.
- For $i = 1, \ldots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of V.
- 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 $A_i = \{j | y_j \in C_i\}$.

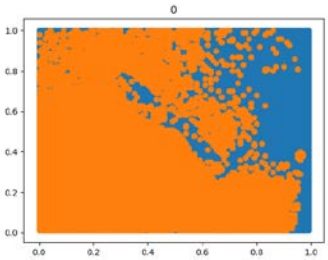
Discussion/Experiments

I. Visualize image 1 kernel k-means and spectral clustering with GIF (k = 2)
We can see that the Kernel k-means will generate more general solution, the part of sea in the picture doesn't show up and it takes more time in each iteration of k-means because of kernel k-means algorithm. The ratio cut spectral will cluster more extremely. Finally, the normalize cut perform well in clustering this image considering the time and iterations.

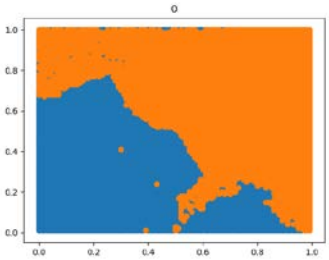
A. Kernel k-means (11 iterations) (2_kernel_1.gif)



B. Ratio Cut Spectral clustering (18 iterations) (2_ratio_1.gif)

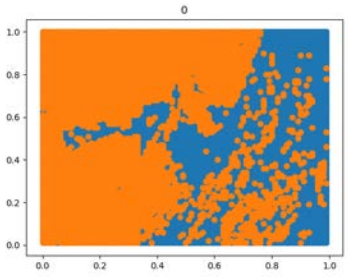


C. Normalize cut (1 iteration)

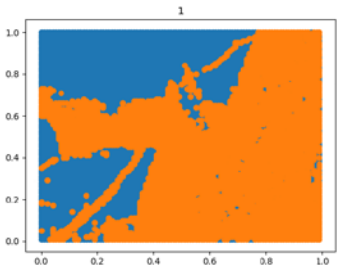


II. Visualize image 2 kernel k-means and spectral clustering with GIF (k = 2)
The result of ration cut cluster the image more clearly comparing with normalized cut. I think the ratio cut method will choose the small and isolated cluster.

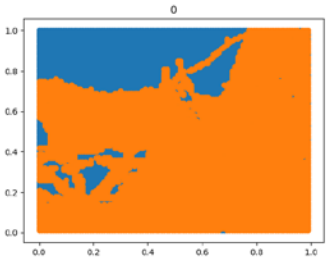
A. Kernel k-means (15 iterations) (2_kernel_2.gif)



B. Ratio Cut (42 iterations) (2_ratio_2.gif)



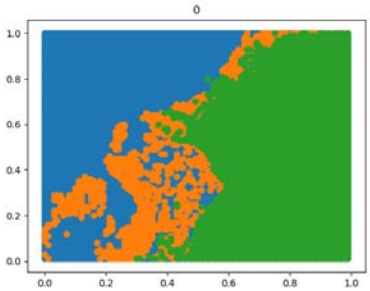
C. Normalize Cut(3 iterations) (2_normal_2.gif)



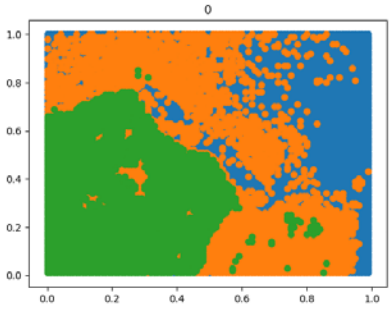
III. Try more clusters

A. 3 clusters (image 1)

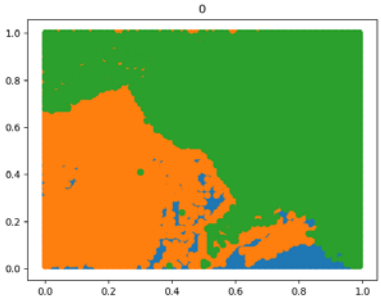
1. Kernel K-means (25 iterations) (3_kernel_1.gif)



2. Ratio cut(30 iterations) (3_ratio_1.gif)

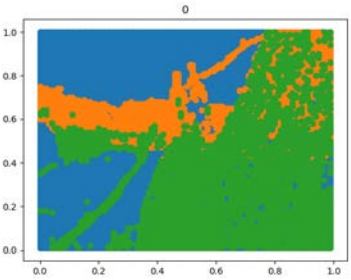


3. Normalize cut(20 iterations) (3_normal_1.gif)

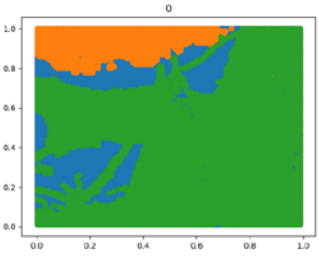


B. 3 cluster (image 2)

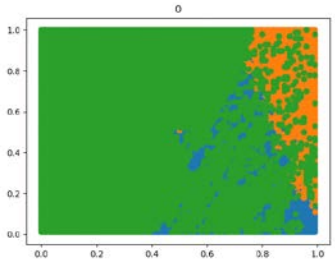
1. Kernel K-means(28 iterations) (3_kernel_2.gif)



2. Ratio cut(29 iterations) (3_ratio_2.gif)



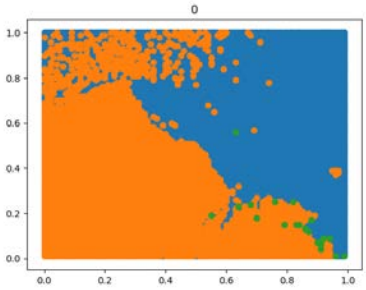
3. Normalize cut(14 iterations) (3_normal_2.gif)



IV. K-means++

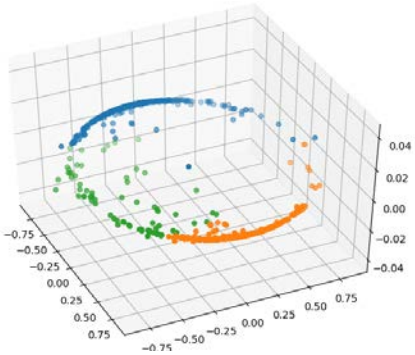
normalize cut (3_normal_1_plus.gif)

Originally, there are some variance of clustering result in different initial data center points. If we use the k-means++, the result would be more similar than k-means algorithm.



V. Coordinates in eigenspace

The following image is the data, normalize cut with k-means++, plotting in the eigenspace. We can observe that eigenvector separate the data clearly, so it just need few iterations.



Code explanation

I. RBF Kernel

The custom_kernel function combine the spatial information and color information.

```
def Rbf_kernel(data1, data2, sigma):
    result = np.exp( -sigma * scipy.spatial.distance.cdist(data1, data2, 'sqeuclidean') )
    return result

def custom_kernel(s1, s2, c1, c2, sigma1, sigma2):
    term1 = Rbf_kernel(s1, s2, sigma1)
    term2 = Rbf_kernel(c1, c2, sigma2)
    return  term1*term2
```

II. Kernel k-means

I would compute the data to each cluster to each cluster center by reference formula and every data point will be assigned to smallest distance cluster.

```
while(1):
    distance_matrix = np.zeros((k,10000))
    for i in range(k):
        cluster_pixel_axis = pixel_axis[data_cluster[i]]
        cluster_img        = img[data_cluster[i]]
        second_term_value  = custom_kernel(pixel_axis,cluster_pixel_axis , img , cluster_img, 0.5,0.5).sum(axis=1) / cluster_pixel_axis.shape[0]
        third_term_value   = custom_kernel(cluster_pixel_axis,cluster_pixel_axis,cluster_img,cluster_img,0.5,0.5).sum(axis=1) / cluster_pixel_axis
        third_term_value   = third_term_value.sum(axis=0)
        distance_matrix[i] = -2*second_term_value + third_term_value
    #assign
    belong_cluster = np.argmin(distance_matrix, axis=0)
```

III. Comparing two set:

I would compare original cluster and new cluster. If there is no new data point be updated, we would jump out of the loop.

```
for i in range(k):
    new_c = set(data_cluster[i])
    org_c = set(old_data_cluster[i])
    if new_c != org_c:
        converge = 1
```

IV. Build similarity matrix(Spectral Clustering)

Combine the spatial information and color information.

```
def buildSimilarityGraph(img):
    pixel_axis   = []
    for i in range(100):
        for j in range(100):
            temp = [j/100, 1-i/100]
            pixel_axis.append(temp)
    pixel_axis = np.array(pixel_axis).reshape(-1,2)
    s_term = Rbf_kernel(img, img, 0.5)
    t_term = Rbf_kernel(pixel_axis, pixel_axis, 0.5)
    new_kernel = s_term * t_term
    return new_kernel
```

V. Normalize cut:

After computing the Laplacian matrix, we should transform it to eigenvalues and

eigenvectors and the formula is in approach part.

```
SimilarityGraph  = buildSimilarityGraph(img)
degMatrix   = buildDegreeMatrix(SimilarityGraph)
lapMatrix   = unnormalizedLaplacian(SimilarityGraph, degMatrix)

#normalize cut
D1_2 = np.diag(np.diag(degMatrix )**-1./2)
Lsym = D1_2.dot(lapMatrix).dot(D1_2)
eigval, eigvec = np.linalg.eigh(Lsym)
eigen = sorted([(eigval[i], eigvec[:,i]) for i in range(len(eigval))], key=lambda t:t[0])

U = np.ndarray((10000, k))
for i in range(1,k+1):
    U[:,i - 1] = eigen[i][1]
    T = np.divide(U, np.repeat(np.linalg.norm(U, axis=1).reshape(10000, 1), k, axis=1))
```

VI. Spectral Clustering k-mean

Using the eigenvectors to do the k-means clustering, each iteration we will update the center point.

```
while(1):
    distance_matrix = np.zeros((k,10000))
    for i in range(k):
        temp = scipy.spatial.distance.cdist(transform_data, [centeriod[i]], 'sqeuclidean')
        distance_matrix[i] = np.transpose(temp)
    belong_cluster = np.argmin(distance_matrix, axis=0)
    old_data_cluster = data_cluster
```

VII. Ratio cut Clustering:

It is the unnormalize spectral clustering, this algorithm also uses the eigenvector to cluster the data.

```
SimilarityGraph  = buildSimilarityGraph(img)
degMatrix   = buildDegreeMatrix(SimilarityGraph)
lapMatrix   = unnormalizedLaplacian(SimilarityGraph, degMatrix)

#normalize cut
L = lapMatrix
eigval, eigvec = np.linalg.eigh(L)
eigen = sorted([(eigval[i], eigvec[:,i]) for i in range(len(eigval))], key=lambda t:t[0])

U = np.ndarray((10000, k))
for i in range(1,k+1):
    U[:,i - 1] = eigen[i][1]
```

VIII. K-means++ implementation

For the first cluster center, I will random choose from the data points. Now, we need k-1 cluster centers, and we need to compute the distance from nearest cluster center for each data. After computing all the distance, we will random choose one cluster center according to

the distance.

```
def kmeans_plus(transform_data):
    global k
    centeriod_list = []
    inital_center = random.randint(0, 10000)
    centeriod_list.append(inital_center)
    centeriod_array = np.asarray(centeriod_list)
    for i in range(k-1):
        data_to_center = scipy.spatial.distance.cdist(transform_data, transform_data[centeriod_array], 'sqeuclidean')
        probability = np.amin(data_to_center, axis=1) / np.sum(data_to_center)
        option = random.choices(population= [i for i in range(10000)],weights=probability)
        centeriod_array = np.append(centeriod_array,option)
    return centeriod_array
```