ML hw6 Report

註：pdf檔的GIF不會動

Please ref https://hackmd.io/DjBqaeTDTr-yNy7xOZev_A

Code with detailed explanations

Part 1

Initialize

```
CLUSTER_NUM = k
GIF_path = './GIF'
colormap= np.random.choice(range(256),size=(100,3))
```

k值即number of cluster，以下均用k表示

read image

```
def openImage(path):
    image = cv2.imread(path)
    H, W, C = image.shape
    image_flat = np.zeros((W * H, C))
    for h in range(H):
        image_flat[h * W:(h + 1) * W] = image[h]
```

讓image flat方便計算

precomputed kernel

```
def precomputed_kernel(X, gamma_s, gamma_c):
    n=len(X)
    # S(x) spacial information
    S=np.zeros((n,2))
    for i in range(n):
        S[i]=[i//100,i%100]
    print(pdist(S,'sqeuclidean').shape)
    K=squareform(np.exp(-gamma_s*pdist(S,'sqeuclidean')))*squareform(np.exp(-gamma_c*pdist(X,'sqeuclidean')))
    print(K.shape)
    return K
```

按規定的kernel建立(10000, 10000)的K

kmeans++ init

```
# kmeans++ init
Cluster = np.zeros((CLUSTER_NUM, Gram.shape[1]))
Cluster[0]=Gram[np.random.randint(low=0,high=Gram.shape[0],size=1),:]
for c in range(1,CLUSTER_NUM):
        Dist=np.zeros((len(Gram),c))
        for i in range(len(Gram)):
            for j in range(c):
                Dist[i,j]=np.sqrt(np.sum((Gram[i]-Cluster[j])**2))
        Dist_min=np.min(Dist,axis=1)
        sum=np.sum(Dist_min)*np.random.rand()
        for i in range(len(Gram)):
            sum-=Dist_min[i]
            if sum<=0:
                Cluster[c]=Gram[i]
                break
```

Kmeans++ init步驟

K-means++算法
Step 1: 从数据集中随机选取一个样本作为初始聚类中心c_1；
Step 2: 首先计算每个样本与当前已有聚类中心之间的最短距离(即与最近的一个聚类中心的距离)，用$D(x)$表示；接着计算每个样本被选为下一个聚类中心的概率 $\frac{D(x)^2}{\sum_{x\in X} D(x)^2}$。最后，按照轮盘法选择出下一个聚类中心；
Step 3: 重复第 2 步直到选择出共K个聚类中心；

註：D(x)即為Dist_min[x]

kmeans

```
diff = 1e9
eps = 1e-9
count = 1
# Classes of each Xi
C=np.zeros(len(Gram),dtype=np.uint8)
segments=[]
while diff > eps:
    # E-step
    for i in range(len(Gram)):
        dist=[]
        for j in range(CLUSTER_NUM):
            dist.append(np.sqrt(np.sum((Gram[i]-Cluster[j])**2)))
        C[i]=np.argmin(dist)
```

E-step : 根據最短距離算出每個點的Cluster belonging

```
#M-step
New_Mean=np.zeros(Cluster.shape)
for i in range(CLUSTER_NUM):
    belong=np.argwhere(C==i).reshape(-1)
    for j in belong:
        New_Mean[i]=New_Mean[i]+Gram[j]
    if len(belong)>0:
        New_Mean[i]=New_Mean[i]/len(belong)

diff = np.sum((New_Mean - Cluster)**2)
Cluster=New_Mean
# visualize
segment = visualize(C, CLUSTER_NUM, H, W)
segments.append(segment)
print('iteration {}'.format(count))
for i in range(CLUSTER_NUM):
    print('k={}: {}'.format(i + 1, np.count_nonzero(C == i)))
print('diff {}'.format(diff))
print('--------------------')
cv2.imshow('', segment)
cv2.waitKey(1)
```

M-step : 根據E-step assign好的cluster belonging計算新的mean

```
def visualize(C,k,H,W):
    '''
    @param C: (10000) belonging classes ndarray
    @param k: #clusters
    @param H: image_H
    @param W: image_W
    @return : (H,W,3) ndarray
    '''
    colors= colormap[:k,:]
    res=np.zeros((H,W,3))
    for h in range(H):
        for w in range(W):
            np.array(res)[h,w,:] = np.array(colors)[C[h*W+w]]

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

為此刻的每個分群填上不同顏色

Spectral clustering

Compute L

```
D=np.diag(np.sum(W,axis=1))
L=D-W
```

計算L

Compute eigenvalue, eigenvector

```
if cut == 'ratio':
    eigenvalue,eigenvector=np.linalg.eig(L)

np.save('{}_eigenvalue_{:.3f}_{:.3f}_unnormalized'.format(path.split('.')
[0],gamma_s,gamma_c),eigenvalue)

np.save('{}_eigenvector_{:.3f}_{:.3f}_unnormalized'.format(path.split('.')
[0],gamma_s,gamma_c),eigenvector)
```

取L的eigenvalue, eigenvector並存起來（避免重複計算）

```
if cut == 'normalized':
    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.eig(L_sym)
    np.save('{}_eigenvalue_{:.3f}_{:.3f}_normalized'.format(path.split('.')
[0],gamma_s,gamma_c),eigenvalue)

np.save('{}_eigenvector_{:.3f}_{:.3f}_normalized'.format(path.split('.')
[0],gamma_s,gamma_c),eigenvector)
```

若為normalized cut，則取L_sym的eigenvalue, eigenvector

```
if cut == 'ratio':

eigenvalue=np.load('{}_eigenvalue_{:.3f}_{:.3f}_unnormalized.npy'.format(pa
th.split('.')[0],gamma_s,gamma_c))

eigenvector=np.load('{}_eigenvector_{:.3f}_{:.3f}_unnormalized.npy'.format(
path.split('.')[0],gamma_s,gamma_c))
if cut == 'normalized':
    D_inverse_square_root=np.diag(1/np.diag(np.sqrt(D)))
    L_sym=D_inverse_square_root@L@D_inverse_square_root

eigenvalue=np.load('{}_eigenvalue_{:.3f}_{:.3f}_normalized.npy'.format(path
.split('.')[0],gamma_s,gamma_c))

eigenvector=np.load('{}_eigenvector_{:.3f}_{:.3f}_normalized.npy'.format(pa
th.split('.')[0],gamma_s,gamma_c))
```

若已經有存檔的eigenvalue及eigenvector，可以直接將它們讀進來。

Compute U from eigenvector

```
sort_index = np.argsort(eigenvalue)
U = eigenvector[:, sort_index[1:1+k]]
```

```
if cut == 'normalized':
    sums=np.sqrt(np.sum(np.square(U),axis=1)).reshape(-1,1)
    U=U/sums
C, segments = kmeans(k, U, H, W)
```

按大小排序eigenvalue，取除第二後最小的k個建立U，若為normalized cut則必須normalize row to norm 1，最後將U做kmeans得最終的Cluster belonging

Part 2

只需修改k為3, 4即可

Part 3

除了Part 1提到的kmeans++外，我還用了random及gaussian來init

```
if init == 'random':

random_pick=np.random.randint(low=0,high=Gram.shape[0],size=CLUSTER_NUM)
    Cluster = Gram[random_pick,:]
if init == 'gaussian':
    X_mean=np.mean(Gram,axis=0)
    X_std=np.std(Gram,axis=0)
    for c in range(Gram.shape[1]):
        Cluster[:,c]=np.random.normal(X_mean[c],X_std[c],size=CLUSTER_NUM)
```

random即隨機選擇，gaussian即用所有data的mean及variance建構gaussian distribution並sample

Part 4

```
def plot_eigenvector(xs,ys,zs,C):
    fig=plt.figure()
    ax=fig.add_subplot(111,projection='3d')
    markers=['o','^','s']
    for marker,i in zip(markers,np.arange(3)):
        ax.scatter(xs[C==i],ys[C==i],zs[C==i],marker=marker)

    ax.set_xlabel('eigenvector 1st dim')
    ax.set_ylabel('eigenvector 2nd dim')
    ax.set_zlabel('eigenvector 3rd dim')
    plt.show()
```

在k=3的情況下做出3維的示意圖，分別代表3個不同的eigenvector，並將3個不同的cluster用不同的marker表示

Experiments settings and results (20%) & discussion

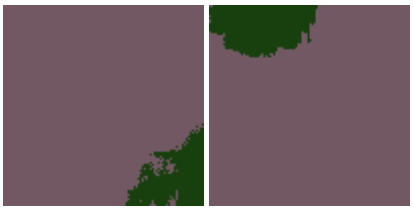
I use gamma_s = 0.001 and gamma_c = 0.001 below.

input image

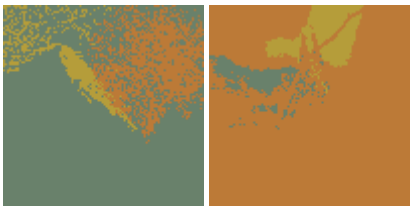
Part 1&2

kernel-kmeans++

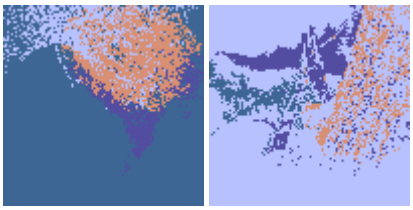
k=2



k=3

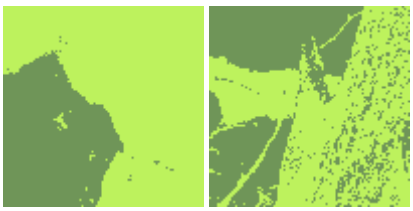


k=4

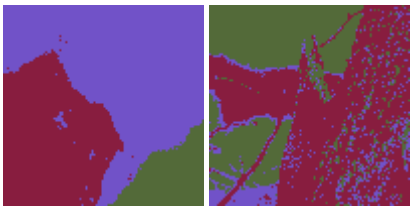


ratio cut

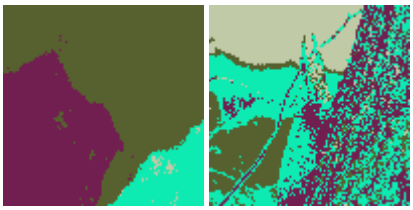
k=2



k=3



k=4

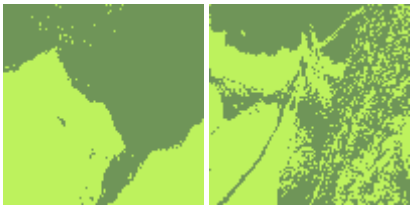


Discussion

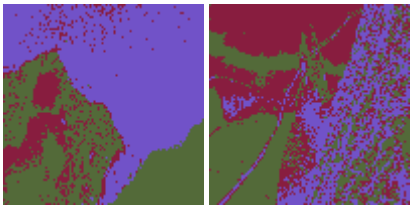
對image1而言，ratio cut的點都較集中，然而對image2而言，在樹上還是有一些交錯的點，推測是因為樹上白的和綠的顏色相差太多所致，但以分群結果來看，在現實世界中，同一物體都是連著的，ratio cut的效果看起來較kernel kmeans好。

normalized cut

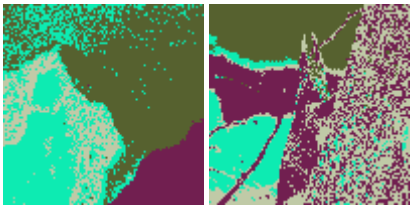
k=2



k=3



k=4



Discussion

除k=2外，結果大致和ratio cut相似，但在這k=2的情形下，normalized cut似乎較ratio cut有更好的表現。

Part 3 different initialization method

kernel kmeans

k=2 random

看起來跟kmeans++的差不多，可能是k只有2的關係

k=2 gaussian

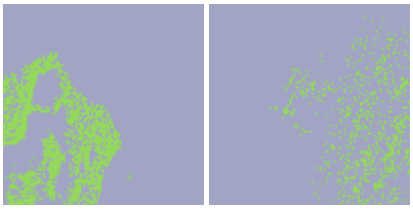
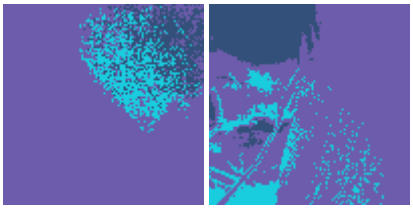
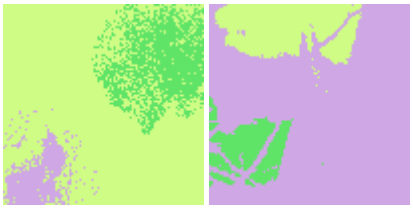


image1上看起來完全不同，由此可知，不同initialize方式確實會產生不同的結果。

k=3 random

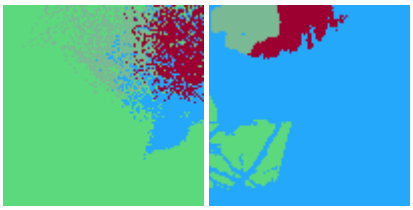


k=3 gaussian



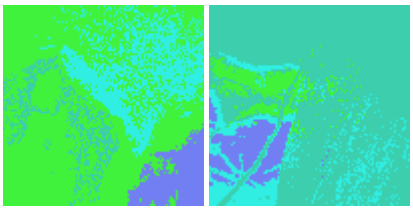
gaussian的結果看起來比較好，推測是init在較中間（接近mean）的位置，所以比較不會因為init太偏而收斂到local minimum

k=4 random



在image2的表現很明顯不如kmeans++

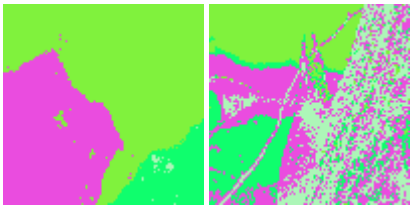
k=4 gaussian



gaussian還是較random好，在image1甚至較kmeans++好

ratio cut

k=4 random

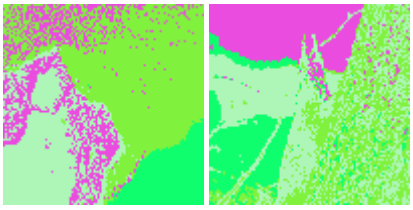


k=4 gaussian

這裡看起來都差不多（和kmeans++）

normalized cut

k=4 random



k=4 gaussian

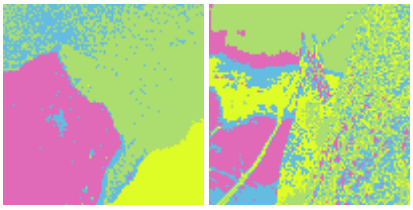


image1明顯gaussian分的比random好

Discussion

以這兩張圖的情形看起來，以kmeans++和gaussian init幾乎效果都較隨機init好。

Part 4 eigenspace visualization(k=3)

ratio cut

image 1

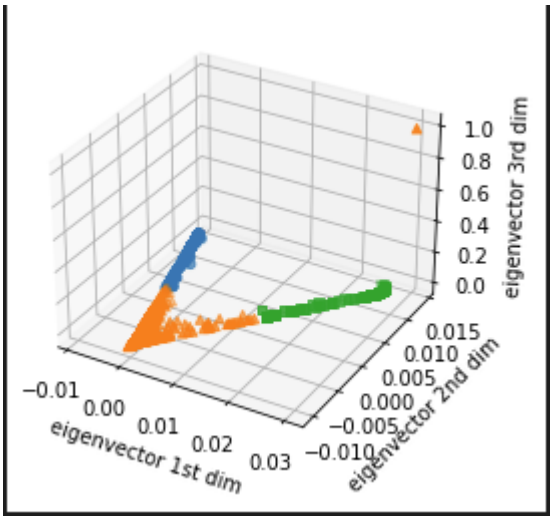
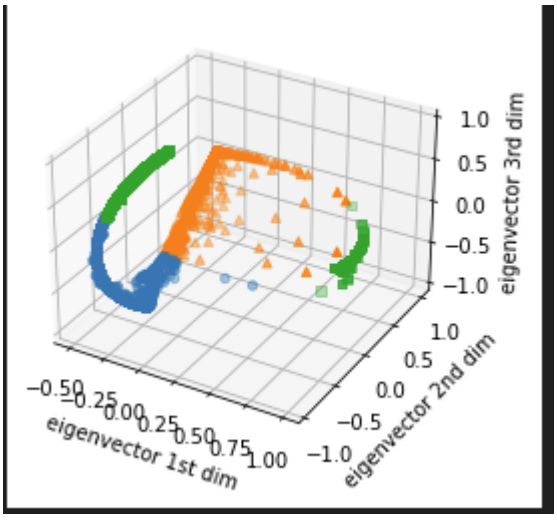


image 2



Discussion

確實不同cluster在graph Laplacian的eigenspace上的coordinates上是分開的

normalized cut

image 1

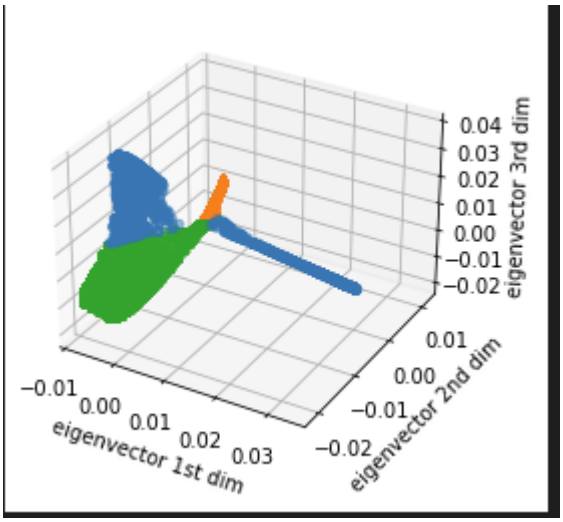
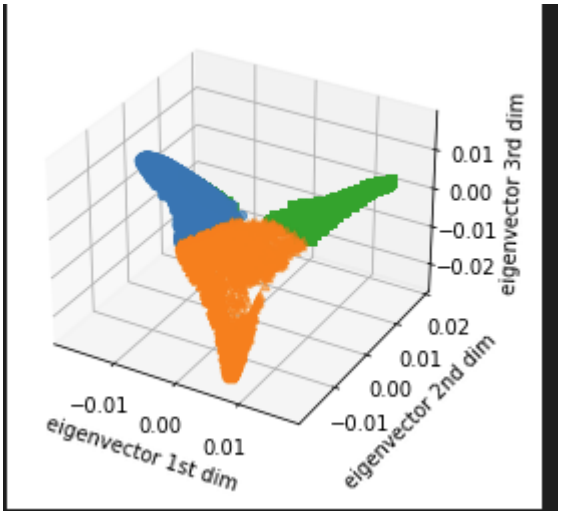


image 2



Discussion

不同cluster在normalized graph Laplacian的eigenspace上的coordinates上也是分開的 Note : All use Gaussian init

---

Observations and discussion

這次的作業跑了大概有好幾十次的clustering，各種clustering方法，各種initialization方法，各種k，如果要做個summary的話，我認為，如果要做圖片的clustering，且用的是同一種kernel，initialization選gaussian或kmeans++應該比較適合，clustering方法選ratio cut或normalized cut應該會較用普通kmeans好（點分佈的較集中，和我們肉眼的分群應該較接近），至於k的話，其實k是最重要的，像image1，k=3最適合，但像image2，k=4或5才是最佳的，至於要怎麼找k呢，並不在這次作業的範圍內，在data science的領域中，我們會以手肘法(elbow method)或輪廓係數法（Silhouette Coefficient）來挑選適合的k值。

Elbow method

其概念是基於 SSE（sum of the squared errors，誤差平方和）作為指標，去計算每一個群中的每一個點，到群中心的距離。算法如下：

$$SSE = \sum_{i=1}^{k} \sum_{p \in C_i} |p - mi|^2$$

根據 K 與 SSE 作圖，可以從中觀察到使 SSE 的下降幅度由「快速轉為平緩」的點，一般稱這個點為拐點（Inflection point），我們會將他挑選為 K。因為該點可以確保 K 值由小逐漸遞增時的一個集群效益，因此適合作為分群的標準。

Silhouette method

輪廓係數法的概念是「找出相同群凝聚度越小、不同群分離度越高」的值，也就是滿足 Cluster 一開始的目標。其算法如下：

$$S = \frac{b - a}{max(a, b))}$$

其中，凝聚度（a）是指與相同群內的其他點的平均距離；分離度（b）是指與不同群的其他點的平均距離。S 是指以一個點作為計算的值，輪廓係數法則是將所有的點都計算 S 後再總和。S 值越大，表示效果越好，適合作為 K。

Ref : https://blog.v123582.tw/2019/01/20/K-means-%E6%80%8E%E9%BA%BC%E9%81%B8-K/

因為逼近期末時間緊迫，Elbow method和Silhouette method的實作之後會補在github

https://github.com/franktseng0718/ML

Note

To see full report : https://hackmd.io/DjBqaeTDTr-yNy7xOZev_A

註：pdf檔的GIF不會動