Link : https://hackmd.io/fJ2ZfZ2bQheWYZm0FjazdA?view

hw6

Code

Part1 Clustering algorithm and visualization

Part1 - Kernal K-means

Overview

```
class kmeasn:
    def train(self):
        img_records = []
        converge_iter = 0
        converge = 5

        # 1. Initialization
        n, d = self.data.shape
        # (1) initializa centers
        centers = self.init_centers()

        # (2) initialize dist
        dist = cdist(self.data, centers, 'sqeuclidean')

        # (3) initialize clusters
        clusters = np.zeros((n, self.k))
        clusters[np.arange(n), np.random.randint(self.k, size=n)] = 1
        img_records.append(clusters)
        # assign data to the closest cluster
        for i in range(n):
            clusters[i, np.argmin(dist[i])] = 1
        img_records.append(clusters)

        # 2. Lloyd's algorithm
        for iter in range(100):
            # E-step : classify all samples according to closet μk
            if self.kernal_mode:
                dist = self.__kernal_dist(self.data, clusters)
            else:
                dist = cdist(self.data, centers, 'sqeuclidean')

            # M-step : re-compute the centers μk in cluster Ck
            new_clusters = np.zeros_like(dist)
            for i in range(n):
                new_clusters[i, np.argmin(dist[i])] = 1
            new_centers = (new_clusters.T @ self.data) / (np.sum(new_clusters, axis=
            img_records.append(new_clusters)

            # Retrun upon convergence
            delta = np.linalg.norm(new_centers - centers)
            if delta < 1e-10:
                converge -= 1
            if converge == 0:
                converge_iter = iter + 1

            centers = new_centers
            clusters = new_clusters

        print(f'k-means : (k = {self.k}, {self.method})   ',
              bashc.GREEN +'is converged at iter [{converge_iter}' + bashc.ENDC)

        return img_records, converge_iter
```

Steps :

1. Read images and transform it to the gram matrix using the new rbf kernal
   - $k(x,x')=e^{-\gamma_s||S(x)-S(x')||}*e^{-\gamma_c||C(x)-C(x')||}$

```
def new_kernal(x, gamma_s=1e-3, gamma_c=1e-3):
    # dist_s = ||S(x) - S(x')||^2
    grid = np.indices((100, 100)).reshape((2, 10000, 1))
    sx = np.hstack((grid[0], grid[1]))
    ds = cdist(sx, sx, 'sqeuclidean')

    # dist_c = ||C(x) - C(x')||^2
    dc = cdist(x, x, 'sqeuclidean')

    # kenral = e^(-γs ds) * e^(-γs dc)
    kernel = np.exp(-gamma_s * ds) * np.exp(-gamma_c * dc)
    return kernel
```

2. Initialize centers and clusters.
   - Check Part3 for the details of init_centers()

3. Apply Lloyd's algorithm to perform k-means clustering, which recursively do EM steps.
   - E-step : calssify data to the closest μ_k
     - Calculate the distance from each data to the centers, saved as $dist_{nxk}$
     - For each entry s_{jk} of $dist_{nxk}$:

$$s_{jk}=||\phi(x_j-\mu_{\phi k})||=k(x_j,x_j)-2|C_k|\sum_n \alpha_{kn}k(x_j,x_n)+1|C_k|^2\sum_p\sum_q \alpha_{kp}\alpha_{kq}k(x_p,x_q)$$

```
def __kernal_dist(self, gram, ck):
    n, cnt_k = ck.shape
    dist = np.zeros_like(ck)

    # Ck = |Ck|
    Ck = np.sum(ck, axis=0)
    # Sjk = -2/|Ck| * ∑K(xj, xn) with xn∈ck
    Sjk = -2 * (gram @ ck) / Ck
    # Spq = 1/|Ck|^2 * ∑∑K(xp,xq) with xp, xq∈ck
    Spq =  np.ones((n, n)) @ Sjk / (Ck ** 2 + 1e-9)

    # distance_jk = K(xj, xj) - 2∑K(xj, xn)/|Ck| + ∑∑K(xp,xq)/|Ck|^2
    for j in range(n):
        for k in range(cnt_k):
            dist[j, k] = gram[j, j] + Sjk[j, k] + Spq[j, k]
    return dist
```

   - M-step : Re-compute the center μ_k of each cluster Ck

```
new_clusters = np.zeros_like(dist)
for i in range(n):
    new_clusters[i, np.argmin(dist[i])] = 1
new_centers = (new_clusters.T @ self.data) / (np.sum(new_clusters, axis=0, k
img_records.append(new_clusters)
```

Part1 - spectral clustering

- Overview

```
class spectral_clustering
    def train(self):
        # 1. Calculate Laplacian L
        self.__cal_laplacian()

        # 2. Do eigendecomposition to get eig_vals and eig_vecs
        k_lambs, U = self.__get_U(self.L, self.k)

        # 3. For i=1 to n, cluster yi ∈ Rk with k-means algorithm
        km_model = kmeans(U, k=self.k, method=self.kmeans_method, kernal_mode=False)
        img_records, iter = km_model.train()
        return  img_records, k_lambs, U
```

- Gram matrix K serves as the similarity graph W.
- To partition a more balanced graph, we use ratio / normalized cut as objective functions.
- Approximating the cut minimization problems, we have the solutions:
  - Note : assume U be the first kth normalized eigenvectors
  - Ratio cut :
    - $\min_{H \in R^{nk}} Tr(H^{'}LH)$ subject to $H^{'}H=I$ with Unnormalized Laplacian L = D - W
  - Normalized cut :
    - $\min_{T \in R^{nk}} Tr(T^{'}LT)$ subject to $T^{'}T=I$ with Normalized Laplacian L = Lsym = $D^{-0.5}LD^{-0.5T}$
  -

```
def __cal_laplacian(self):
    # L = D - W
    self.D = np.diag(np.sum(self.W, axis=1))
    self.L = self.D - self.W
```

  -

```
def __eig(self, L):
    # Define file pathes
    ...
    # Calculate eig_vals and eig_vecs
    if(self.train_mode):
        # RatioCut
        L = self.L
        # NormalizedCut
        if self.normalize:
            # L = Dsym = D^(-1/2)LD^(-1/2)
            _D = np.zeros_like(self.D)
            for i in range(len(self.D)):
                _D[i, i] = self.D[i, i] ** -0.5
            L = _D @ self.L @ _D

        eig_vals, eig_vecs = np.linalg.eigh(L)
        np.save(val_path, eig_vals)
        np.save(vec_path, eig_vecs)
        return eig_vals, eig_vecs
    # Load eig_vals and eig_vecs from files
    else:
        ...
```

- Important details :
  - We should pick the first kth eigenvectors with positive eigenvalue
  - Either ratio or normalized method should do normalization on eigenvectors

```
def __get_U(self, L, k):
    # 1. Get the eig_vals and eig_vecs
    eig_vals, eig_vecs = self.__eig(L)
    cut = 'normalized' if self.normalize else 'ratio'

    # 2. Get the first k non-zero eigenvalues λ
    sort_idx = np.argsort(eig_vals)
    start = 0
    while eig_vals[sort_idx[start]] <= 1e-9:
        start += 1
    k_idx = sort_idx[start : start + self.k]

    # U ∈ R_nk containing k eigenvectors
    U = eig_vecs[:, k_idx]
    lambs = eig_vals[k_idx]

    # Normalized U (since H'H / T'T = I)
    U /= np.linalg.norm(U, axis=1).reshape(-1,1)

    # (additional experiment : visualize the eigenvalues λ)
    # plot_dot(np.arange(len(eig_vals)), eig_vals, f'{cut}_with_k={self.k}')

    return lambs, U
```

- According to tho following algorithm, we should cluster $y_i \in T_{Rk}$ using k-means algorithm in the last step.

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 $u_1, \ldots, u_k$ 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 $A_i = \{j | y_j \in C_i\}$.

```
class spectral_clustering():
    def train(self):
    # 1. Calculate Laplacian L
    self.__cal_laplacian()

    # 2. Do eigendecomposition to get eig_vals and eig_vecs
    k_lambs, U = self.__get_U(self.L, self.k)

    # 3. For i=1 to n, cluster yi ∈ Rk with k-means algorithm
    km_model = kmeans(U, k=self.k, method=self.kmeans_method, kernal_mode=False)
    img_records, iter = km_model.train()
    return  img_records, k_lambs, U
```

Part1 - visualization

- Visualize clusters in each iteration, and colorize each cluster with different colors

```
k_color = colors.to_rgba_array(['skyblue','teal','cornflowerblue','yellowgreen','ora

# colorize each cluster with different colors
def  visualize_clusters(img_records, save_path, figsize=(100, 100, 4)):
    gif = []
    # for each iteration
    for i in range(len(img_records)):
        # Get the cluster of each data point in the current img
        c_id = np.argmax(img_records[i], axis=1)
        n = c_id.shape[0]
        # Colorize each cluster with different colors
        img = np.zeros(figsize, dtype=np.uint8)
        for q in range(100):
            for r in range(100):
                cluster = c_id[q * 100 + r]
                img[q, r] = k_color[cluster] * 255
        gif.append(img)
    imageio.mimsave(save_path, gif)
    return
```

Part2

Part2 - Try more clusters

- Initialize kmeans and spectral_clustering classes with k from 2 to 10.
  - kmeans

```
# main() of k-means
for i, kernal in enumerate(kernals):
    for j in range(9):
        # k = 2 to 10
        k = 2 + j

        # kmeans
        model = kmeans(kernal, k, method='default')
        record, _ = model.train()
        visualize_clusters(record, f'output_kmeans/img{i+1}_kmeans(k={k}).gif')

        # kmeans++
        model_kpp = kmeans(kernal, k, method='kmeans++')
        record_kpp, _ = model_kpp.train()
        visualize_clusters(record_kpp, f'output_kmeans/img{i+1}_kmeans++(k={k}).

        print('----------------------------------------------------------------
```

  - spectral clustering

```
# main() of spectral_clustering
for i, similarity in enumerate(kernals):
    for k in range(2, 11):
        params = {'root': f'img{i+1}',
                  'normalized': False,
                  'train_mode': train_mode,
                  'kmeans_method': kmeans_method}

        for cut_method in ['normalized', 'ratio'] :
            print(f'Spectral_clustering {cut_method}cut with k = {k}')

            # set the parameters
            params['normalized'] = True if cut_method == 'normalized' else False
            filename = f'output_spectral/img{i+1}_{cut_method}({kmeans_method}={

            # train the model
            model = spectral_clustering(similarity, k, params)
            records, eig_vals, eig_vecs = model.train()

            # visualization
            visualize_clusters(records, filename+'.gif')
            visualize_eigvecs(eig_vecs, records[-1], k, filename+'.png')
        print('----------------------------------------------------------------
```

Part3

- Try 2 methods to initialize the clusters

```
class kmeans():
    def __init__(self, data, k=2, method='default', kernal_mode=True):
        self.init_centers = self.__kmeanspp
        if method == 'kmeanspp'
        else self.__random
```

  - random
    - randomly pick k data as centers

```
def __random(self):
    idx = list(random.sample(range(0, self.data.shape[0]), self.k))
    return np.array(self.data[idx])
```

  - kmeans++
    - keep clustering data then picking the points with maximum distance from its centroid as a new center.

```
def __kmeanspp(self):
    n, d = self.data.shape
    centers = np.zeros((self.k, d))

    centers[0] = self.data[np.random.randint(n)]
    for i in range(1, self.k):
        # dist to each center
        dist = cdist(self.data, centers[:i], 'sqeuclidean')
        # classify to the closet center
        dist = np.min(dist, axis=1)
        # pick the farest point as center
        centers[i] = self.data[np.argmax(dist, axis=0)]
    return centers
```

Part4

- Visualize the eigenspace coordinatesas as points

```
# plot eigenvectors
def visualize_eigvecs(vecs, clusters, k, fig_path):
    fig = plt.figure()
    ax = fig.gca(projection="3d")

    n, d = vecs.shape
    # translate clusters from nxk to nx1
    clusters_id = np.argmax(clusters, axis=1)
    for ki in range(k):
        # k_idx : select data within cluster ki
        k_idx = clusters_id == ki
        # visualize using the first 3 axes
        x = vecs[k_idx, 0]
        y = vecs[k_idx, 1]
        if d < 3:
            ax.scatter(x, y, marker='.', c=[k_color[ki]])
        else:
            z = vecs[k_idx, 2]
            ax.scatter(x, y, z, marker='.', c=[k_color[ki]])

    plt.savefig(fig_path)
```

Experiments & Discussion

Part1

Show the clustering procedure

- Image1 : k=2

iter	0	10	last iter	gif
Kmeans iter=43				
Ratio iter=14				
Normalized iter=15				

- Image2

iter	0	10	last iter	gif
Kmeans iter=71				
Ratio iter=18				
Normalized iter=20				

Discussion

- Every models can classify 2 clusters easily.
- In comparison to spectral clustering, kmeans start from a more random state.
  - I guess this might because that kmeans randomly choose a 10000-dim point, while the kmeans step in spectral clustering choose a k-dim point.

Part2 & Part3

Experiments : Try more clusters and different initialization method

- Nameing rule
  - Ratio : Ratio + kmeans
  - Ratio++ : Ratio + kmeans++
- Color :

- image1 , γ_s=1e−3,γ_c=1e−3

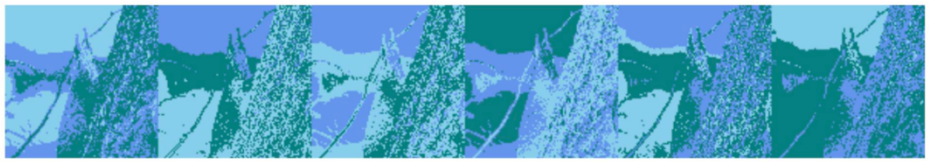
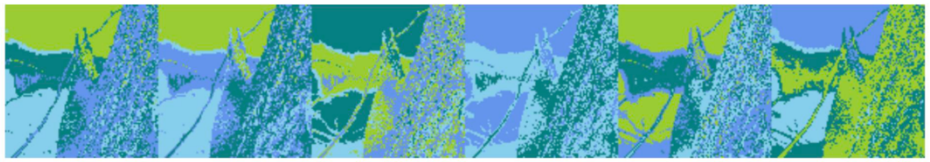
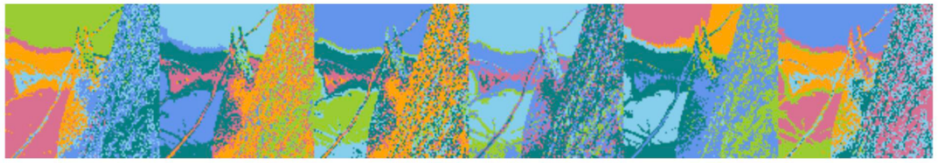
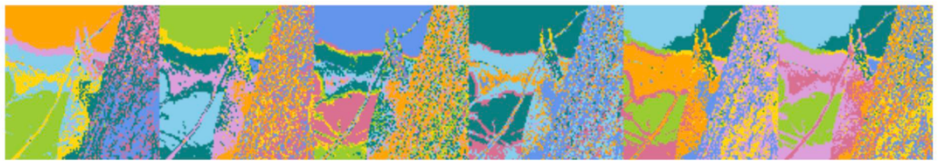
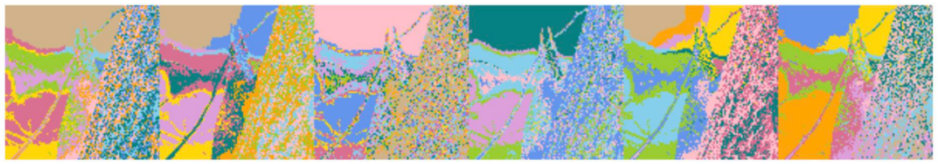
K	Kmeans	Kmean++	Ratio	Ratio++	Normalized	Normalized++
3						
4						
6						
8						
10						

	Observation
	1. RatioCut can only identify few clusters in this picture. 2. Except for RatioCut, all the otehr models make similar classificaitons. 3. Comparing Kmeans++ with Normalized++ at k=10, Kmeans++ tends to seperate clusters more apart from each other.

Part2	**Comparison betweens k**
-----	1. Regions can be clearly identified at k<=4. 2. As k increased, more regions are identified, such as plain, forest, beach, and island.

Part3	**Comparison betweens initialization methods**
-----	1. Kmeans initialize centers using random method, while kmeans++ initialize centers far from each other. We can see the difference especially at k=10. 2. There seems no difference between 2 initialization methods, since the iteration l set(100) is large enough for convergence.

- image2 , γ_s=1e−3,γ_c=1e−3

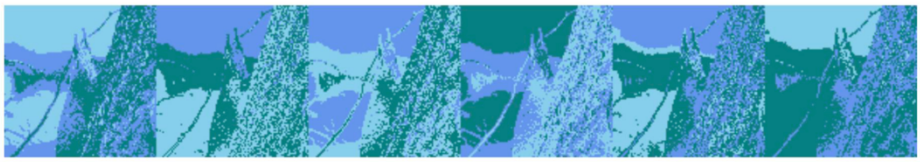
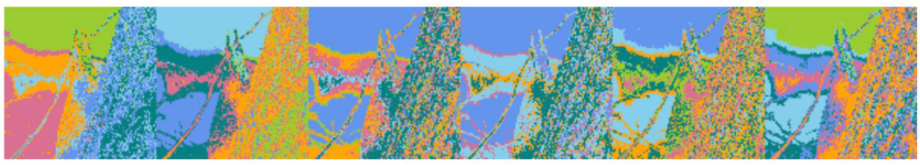
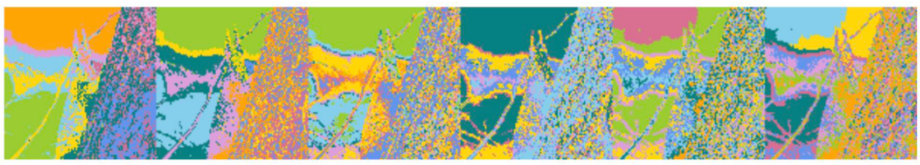
K	Kmeans	Kmean++	Ratio	Ratio++	Normalized	Normalized++
3						
4						
6						
8						
10						

	Observation
	1. Frankly speaking, I can't tell signifnicant difference betweens those models… 2. The most noisy model is RatioCut with random initialization. 3. k<=4 seperate objects and backgound clearly. 4. k=6 seperates background in reasonable way, but when it comes to k=8, some clusters become fragmented.

Part2	**Comparison betweens k**
-----	1. Regions can be clearly identified at k<=4. 2. As k increased, more regions are identified, while the model become sensitive. E.g., ambiguous parts such as the colorful(?) tree become noisy.

Part3	**Comparison betweens initialization methods**
-----	1. There seems no difference between 2 initialization methods, since the iteration is large enough for convergence. 2. However, changing initialization method from random to kmeans++ will improve both RatioCut and NormalizedCut models a little bit.

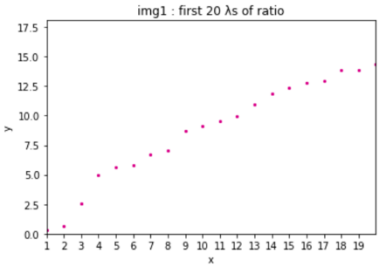
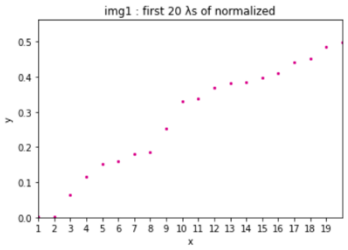
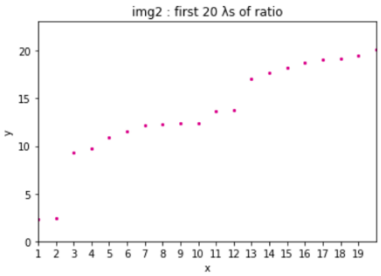
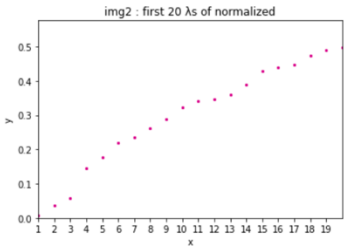
- Other experiment : image2 , γ_s=1e−4,γ_c=1e−3

K	Kmeans	Kmean++	Ratio	Ratio++	Normalized	Normalized++
3						
4						
6						
8						
10						

	Observation
	1. According to the tutorial, spectral clustering can be quite sensitive to changes in the similarity graph and to the choice of its parameters. Thus, in order to identify the rabbit, I decreased γs(emphasize on color) in kernal function, and finally, the rabbit is identified using Normalized models with k=10 2. Though the model can identify the rabbit, it's a disaster to detect the tree.

Discussion

- Part2 : What is the best cluster number?
  - According to the experiments, I will take k<=4 to identify backgound.
  - Yet, I will take k=8 when $\gamma_s=\gamma_c=1e-3$, and k=10 when $\gamma_s=1e-4,\gamma_c=1e-3$ to identify more objects.
  - According to the tutorial , eigengap may indicate the number of cuts. Thus, let's examine the eigenvalues(λs) :

	Ratio	**Normalized**
		
Discussion	There are 2 λs close to 0, so we conclude that the image can be classifed into 2 classes easily.	In addtion to the 2 λs, there is a gap at k=8
		
Discussion	There is a gap at k=2.	There is a gap at k=3.
Conclusion	k=2	k=3

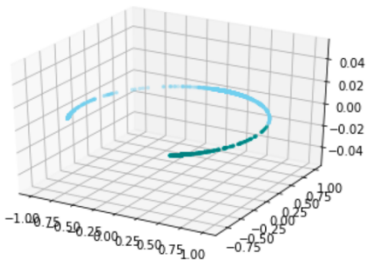
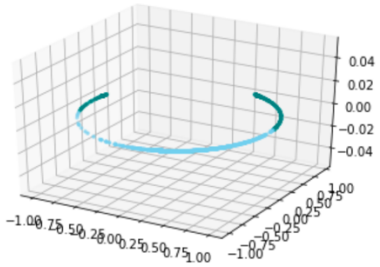
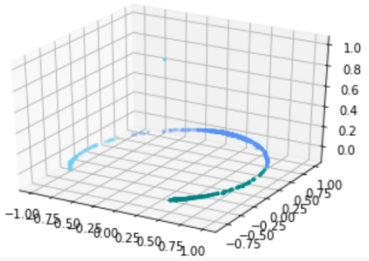
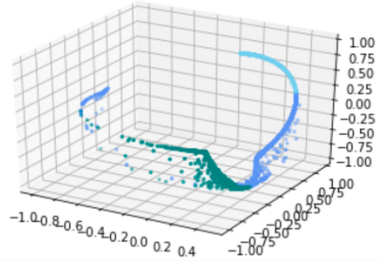
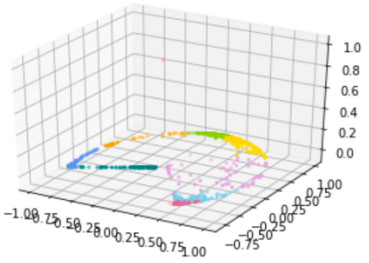
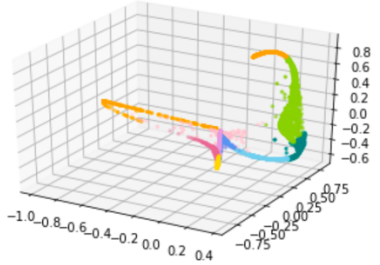
Part4

Observation

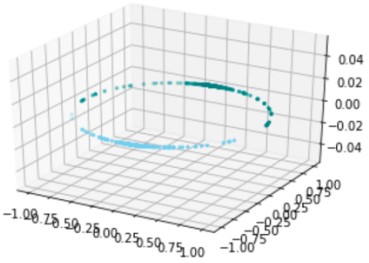
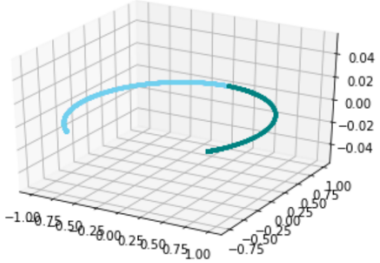
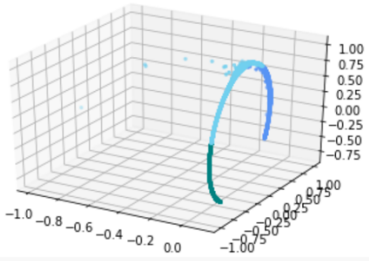
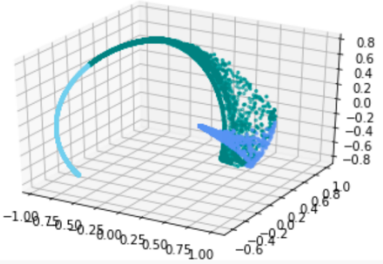
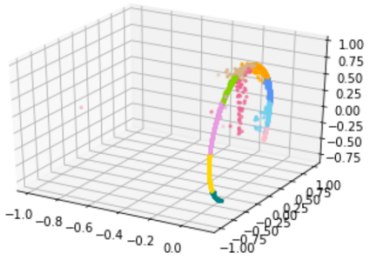
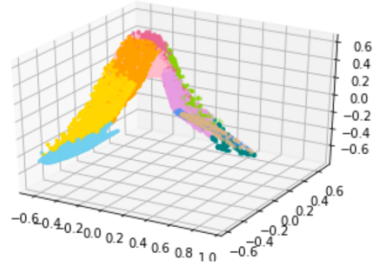
- The data points within same cluster have different but close coordinates in the eigenspace of graph Laplacian.

Experiment

- image1

K	Ratio	Normalized
2		
3		
10		

- image2

K	Ratio	Normalized
2		
3		
10		

Observations & Discussion

Performance

- Kmeans requires more iteration and becomes slower as K increases.
- Spectral clustering spends lots of time on eigendecomposition
  - However, if we save the eigenvalues/eigenvectors of the Laplacian matrix, it's not time-consuming.

Observation

- Based on the result of Part2&3
  - Only 2 dimensions of the eigenvectors in Ratiocut are useful in most cases.
  - In contrast, at least 3 dimension are useful in NormalizedCut.
  - This may explain why RatioCut can't identify pictures with large k.
- According to my experiments, NormalizedCut performs better than RatioCut on the 2 pictures.

Problem

- I encounter this error using scipy.linalg.eig, and solved by using eigh() instead.
- The first eigenvalue should be 0, but it turns out to be 1e-13~1e-15. I guess this is a floating point issue, so I take this number as 0 in my code.