Machine Learning Homework 6 Report

Yuan-Ming Chang
Student ID: A132525

1 Code & Detailed Explanations

1.1 Part1: Clustering & Visualization

The defined kernel function is:

$$\mathbf{k}(x, x') = e^{-\gamma_s \|S(x)-S(x')\|^2} \times e^{-\gamma_c \|C(x)-C(x')\|^2} \tag{1}$$

Hyperparameters:

- $\mathcal{S}(x)$: The spatial information of the data point x.
- $\mathcal{C}(x)$: The color information of the data point x.
- γs**:** Controls the sensitivity of the kernel to differences in spatial-related features.
- γc**:** Controls the sensitivity to differences in color-related features.

Code:

```python
def compute_kernel(feature, gamma_s, gamma_c):
    # Extract coordinates and colors
    coords = feature[:, :2]
    colors = feature[:, 2:]
    # Compute pairwise squared differences in spatial coordinates
    diff_s = cdist(coords, coords, metric='sqeuclidean')
    # Compute pairwise squared differences in color space
    diff_c = cdist(colors, colors, metric='sqeuclidean')
    # Compute the kernel matrix
    kernel_matrix = np.exp(-gamma_s * diff_s) * np.exp(-gamma_c * diff_c)
    return kernel_matrix
```

I use **scipy.spatial.distance** to compute the pairwise squared differences in both spatial and color coordinates. I set the parameter of `compute_kernel` function as $\gamma_s = \gamma_c = 10^{-4}$ to compute the kernel function in each pixels.

Kernel K-means

At the beginning of the algorithm, each pixel is randomly assigned to one of the two clusters. This random initialization provides the starting point for the iterative kernel k-means optimization. For each pixel, I need to compute three terms based on the kernel function which I created previously, follow the below equation:

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

- The first term measures the self-similarity of the pixel.
- The second term captures the similarity between the pixel and the current cluster.
- The third term reflects the internal coherence (compactness) of the cluster.

The algorithm compares the distances of each pixel to all clusters and assigns the pixel to the cluster with the minimum distance. This assignment step is repeated for multiple epochs until the label assignments converge, that is, no pixel changes its cluster. If convergence is not reached, the process terminates after a maximum of 30 epochs.

Code:

```
def kernel_kmeans(epoch, k, kernel, image_size, dir_name, kmeanspp):
        for i in range(100):
            for j in range(100):
                idx = i*100 + j
                #print("(", i, ", ", j, ")")
                #print(idx)
                dist = []
                # first term
                term1 = kernel[idx, idx]
                #print(first)
                for c in range(k):
                    # idx_c: cluster c 的 datapoint index
                    idx_c = label_idx[c]
                    term2 = term2_idx[c][idx]
                    term3 = term3_c[c]

                    if len(idx_c) > 0:
                        final_term = term1 - 2*(1/len(idx_c))*term2 + (1/(len(idx_c))**2)*term3
                    else:
                        final_term = float('inf')  # 避免空群錯誤
                    dist.append(final_term)
                new_labels.append(np.argmin(dist))
                total_loss += dist[np.argmin(dist)]
                #print(dist)
                #print(np.argmin(dist))
        #print(f'Original Labels: {labels}')
        #print(f'New Labels: {new_labels}')
        losses.append(total_loss)
        print(f'Total kernel K-means loss: {total_loss:.2f}')
        save_epoch_visual(np.array(new_labels), image_size, e, dir_name)
        if np.array_equal(new_labels, labels):
            print(f"Converged at epoch {e}")
            break
        labels = np.array(new_labels)
    return labels, losses
```

Spectral Clustering

Both **Ratio Cut** and **Normalized Cut (Ncut)** are graph-based spectral clustering methods. They follow a similar pipeline but differ in the construction of the Laplacian matrix and the post-processing of eigenvectors.

1. **Construct the similarity matrix W:**

   I load the kernel function, encoding pairwise similarity between data points, typically based on spatial and appearance features (e.g., Gaussian kernel).

2. **Compute the degree matrix D:**

   The degree matrix is a diagonal matrix defined as:

   $$D_{ii} = \sum_j W_{ij}$$

3. **Compute the Laplacian matrix:**

   - For **Ratio Cut**, use the **unnormalized Laplacian**:

     $$L = D - W$$

   - For **Normalized Cut**, use the **symmetric normalized Laplacian**:

     $$L_{\text{sym}} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$$

     (implemented using a stabilized inverse with a small constant to avoid division by zero)

4. **Eigen decomposition:**

   Solve the eigenvalue problem using a symmetric eigensolver:

   $$L\mathbf{v} = \lambda\mathbf{v} \quad \text{or} \quad L_{\text{sym}}\mathbf{v} = \lambda\mathbf{v}$$

   Select the eigenvectors corresponding to the second smallest eigenvalues of matrix $H \in R^{N\times k}$.

5. **(Only for Normalized Cut) Row-normalize the eigenvectors:**

   For Ncut, normalize each row of H to have unit ℓ_2-norm:

   $$H_i \leftarrow \frac{H_i}{\|H_i\|_2}$$

   This step helps improve clustering quality by mapping each point to the unit sphere in the spectral space.

6. **Apply k-means:**

   Perform k-means clustering on the rows of H to obtain the final cluster assignments. Each row corresponds to a data point (e.g., pixel), represented in a new space defined by the selected eigenvectors. The clustering process is run for a maximum of 30 epochs or until convergence is reached.

Code:

Normalize cut

```python
def spectral_normalized_cut(W, epoch, k, image_size, dir_name, kmeanspp):
    D = np.diag(W.sum(axis=1))
    D_inv_sqrt = np.diag(1.0 / np.sqrt(W.sum(axis=1) + 1e-8))  # avoid division by zero
    L = D - W
    L_sym = D_inv_sqrt @ L @ D_inv_sqrt  # symmetric normalized Laplacian
    # Step 2: Eigen decomposition
    eigvals, eigvecs = np.linalg.eigh(L_sym)
    H = eigvecs[:, 1:k+1]  # skip trivial all-1 eigenvector
    # Step 3: Row-normalize H (optional but standard for Ncut)
    H = H / (np.linalg.norm(H, axis=1, keepdims=True) + 1e-8)
    # Step 4: Spectral K-means (same as before)
    if kmeanspp == 0:
        labels = np.random.randint(0, k, size=N)
    else:
        labels = kmeans_plusplus_init_spectral(H, k)
    losses = []

    for e in range(epoch):
        print(f"Epoch {e+1}:")
        for c in range(k):
            print(f"Cluster {c}: {np.sum(labels == c)} points")
        centroids = []
        for c in range(k):
            if np.any(labels == c):
                centroids.append(H[labels == c].mean(axis=0))
            else:
                centroids.append(np.zeros(H.shape[1]))
        centroids = np.array(centroids)

        new_labels = np.argmin(((H[:, np.newaxis, :] - centroids[np.newaxis, :, :]) ** 2).sum(axis=2), axis=1)
```

Ratio cut

```python
def spectral_ratio_cut(W, epoch, k, image_size, dir_name, kmeanspp):
    # Step 1: Degree and Laplacian
    D = np.diag(W.sum(axis=1))
    L = D - W

    # Step 2: Eigen decomposition
    eigvals, eigvecs = np.linalg.eigh(L)  # Since L is symmetric (use eigh)
    print(eigvals[:10])
    #print(eigvecs)
    H = eigvecs[:, 1:k+1]  # take 2nd to Nth eigenvectors
    #print(H)
    #print(H.shape)

    # Step 3: Initialize cluster labels
    if kmeanspp == 0:
        labels = np.random.randint(0, k, size=N)
    else:
        labels = kmeans_plusplus_init_spectral(H, k)
    losses = []
    #print(H[labels == 0])
    for e in range(epoch):
        print(f"Epoch {e+1}:")
        for c in range(k):
            print(f"Cluster {c}: {np.sum(labels == c)} points")
        centroids = []
        for c in range(k):
            if np.any(labels == c):
                centroids.append(H[labels == c].mean(axis=0))
            else:
                centroids.append(np.zeros(H.shape[1]))
        centroids = np.array(centroids)

        new_labels = np.argmin(((H[:, np.newaxis, :] - centroids[np.newaxis, :, :]) ** 2).sum(axis=2), axis=1)
```

1.2 Part2: Try more clusters

In addition to setting the number of clusters $k = 2$, I also experimented with clustering the image into 3 and 4 parts. The key idea is to select the eigenvectors corresponding to the second to $(k+1)$-th smallest eigenvalues, forming the spectral embedding matrix $H \in R^{N \times k}$, where each row represents a pixel in the eigenspace. For initialization, I randomly assign each pixel to one of the k clusters as the starting labels for the iterative k-means process.

1.3 Part3: Try different initializations

In this part, I implemented and evaluated the performance of the k-means++ initialization strategy for both kernel k-means and spectral clustering. KMeans++ is an improved initialization method for the KMeans algorithm. Instead of randomly selecting all initial centers, it selects the first center randomly, and each subsequent center is chosen with probability proportional to the squared distance from the nearest existing center. This strategy helps to spread out the initial centers, leading to faster convergence and better clustering performance compared to standard KMeans.

Kernel K-means

In `kmeans_plusplus_init_kernel` function:

I implements the k-means++ initialization using the precomputed kernel (Gram) matrix. It selects the first center randomly and then chooses subsequent centers with probability proportional to their squared kernel distance from the closest existing center.

Spectral Clustering

For spectral clustering, the input is the matrix $H \in R^{N \times k}$, where each row is a data point embedded in the eigenspace.

The `kmeans_plusplus_init_spectral` function:

- Runs multiple attempts (default: 10) to find the best initialization.
- Check whether the cluster sizes are reasonable. If any cluster contains only one point, the initialization is considered invalid.
- For each attempt, centers are selected with probabilities proportional to the squared Euclidean distances in the eigenspace. For each data point, the distance is defined as the squared distance to its nearest existing center.
- After assigning each point to its nearest center, the total intra-cluster loss is computed.
- The best initialization (lowest loss) is kept, and the result is visualized.

Code:

Kernel K-means

```python
def kmeans_plusplus_init_kernel(kernel, k):
    N = kernel.shape[0]
    centers = []
    # Random select the center
    first = np.random.randint(0, N)
    centers.append(first)

    for _ in range(1, k):
        dists = []
        for i in range(N):
            min_dist = float('inf')
            for c in centers:
                # kernel distance^2 = k(x,x) - 2k(x,c) + k(c,c)
                dist_sq = kernel[i, i] - 2 * kernel[i, c] + kernel[c, c]
                min_dist = min(min_dist, dist_sq)
            dists.append(min_dist)
        dists = np.array(dists)
        probs = dists / np.sum(dists)
        new_center = np.random.choice(N, p=probs)
        centers.append(new_center)

    # 依據距離最小 assign labels
    dists_to_centers = np.zeros((N, k))
    for idx, c in enumerate(centers):
        dists_to_centers[:, idx] = kernel.diagonal() - 2 * kernel[:, c] + kernel[c, c]
    labels = np.argmin(dists_to_centers, axis=1)
    return labels
```

Spectral Clustering

```python
def kmeans_plusplus_init_spectral(features, k, retries=10):
    for attempt in range(retries):
        centers = []

        # Random select the center
        first_center_idx = np.random.randint(0, N)
        centers.append(features[first_center_idx])

        # 根據距離平方機率分布選新中心，距離dist越遠被選成center越高
        for _ in range(1, k):
            dist = []
            # 對每個 datapoint 算離他最近距離的中心的距離
            for x in features:
                min_dist_sq = float('inf')
                for c in centers:
                    d = np.linalg.norm(x - c)**2
                    if d < min_dist_sq:
                        min_dist_sq = d
                dist.append(min_dist_sq)

            dist = np.array(dist)
            probs = dist / np.sum(dist)
            new_center_idx = np.random.choice(N, p=probs)
            centers.append(features[new_center_idx])
```

```python
# Assign labels
centers = np.array(centers)
dists_to_centers = np.linalg.norm(features[:, np.newaxis, :] - centers[np.newaxis, :, :], axis=2)
labels = np.argmin(dists_to_centers, axis=1)

# 檢查群大小是否合理，若其中一群只有一點，無效
cluster_sizes = np.bincount(labels, minlength=k)
if np.min(cluster_sizes) <= 1:
    print(f"[Attempt {attempt+1}] Discarded: one cluster only has 1 point.")
    continue

# 計算 loss：每點到其中心的距離平方總和
loss = np.sum(np.linalg.norm(features - centers[labels], axis=1) ** 2)
print(f"[Attempt {attempt+1}] OK, loss = {loss:.2f}, min cluster size = {np.min(cluster_sizes)}")

if loss < best_loss:
    best_loss = loss
    best_labels = labels
```

1.4 Part4: Experiments on the coordinates in the eigenspace

In spectral clustering, we construct a graph Laplacian from the similarity matrix and compute its eigenvectors. The eigenvectors corresponding to the smallest (non-zero) eigenvalues capture the global structure of the data and are used as features for clustering.

To analyze the clustering behavior in the eigenspace, we implemented two visualization methods:

- **Eigenvector value plot:**

  For each eigenvector (corresponding to the 2nd to $(k+1)$-th smallest eigenvalues), we plotted its values across all data points (indexed from 0 to $N-1$), with each point colored according to its cluster label. This visualization helps determine whether a particular eigenvector provides discriminative power for clustering. A well-structured pattern indicates that the eigenvector contributes meaningfully to separating clusters by colors, while flat or noisy patterns suggest weak or no clustering signal. The function `plot_single_eigenvector(H, labels, eig_idx, dir_name)` was used to generate these plots.

- **2D projection of eigenspace features:**

  Based on our observations, the first and second non-trivial eigenvectors (i.e., corresponding to the 2nd and 3rd smallest eigenvalues) exhibited the strongest ability to distinguish different classes in the data. As a result, we focused our projection and analysis on these two dimensions, as they captured the most meaningful variation for clustering. Higher-order eigenvectors, in contrast, often appeared flat or noisy, contributing little to class separation. The function `plot_eigen_projection(H, labels, dir_name)` was used for this purpose.

These visualizations were repeated for different values of k (number of clusters), using the first k non-trivial eigenvectors for analysis. We observed that while the first and second eigenvectors often provide strong clustering signals, higher-order eigenvectors tend to carry less meaningful structure, especially when the eigenvalue spectrum exhibits a clear eigen gap. This behavior confirms that spectral clustering works best when the number of clusters aligns with the intrinsic dimensionality of the informative eigenspace.

Code:

plot_single_eigenvector

```
def plot_single_eigenvector(H, labels, eig_idx, dir_name):
    plt.figure(figsize=(8, 2))
    plt.scatter(range(len(H)), H[:, eig_idx], c=labels, cmap='tab10', s=10)
    plt.title(f"Eigenvector {eig_idx+1} (sorted by index)")
    plt.xlabel("Data Index")
    plt.ylabel(f"Value in Eigenvector {eig_idx+1}")
    plt.grid(True)
    plt.tight_layout()
    plt.savefig(f"{dir_name}/eigenspace{eig_idx+1}.png")
    plt.show()
```

plot_eigen_projection

```
def plot_eigen_projection(H, labels, dir_name):
    plt.figure(figsize=(6, 5))
    plt.scatter(H[:, 0], H[:, 1], c=labels, cmap='tab10', s=5)
    #plt.title("title")
    plt.xlabel("Eigenvector 1")
    plt.ylabel("Eigenvector 2")
    plt.grid(True)
    plt.savefig(f"{dir_name}/eigenspace_feature.png")
    plt.show()
```

Declaration

```
plot_eigen_projection(H, labels, dir_name)
for i in range(k):
    plot_single_eigenvector(H, labels, i, dir_name)
```

2 Experiments settings and results

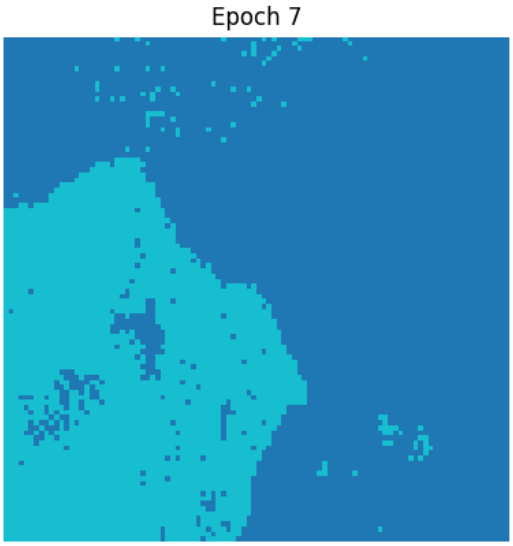
2.1 Part1

For the maximum number of epochs, I set both kernel k-means and spectral clustering to 30. The title of each plot indicates the number of epochs at which the algorithm converged. I set the parameter of `compute_kernel` function as $\gamma_s = \gamma_c = 10^{-4}$.

image1.png

Kernel K-means

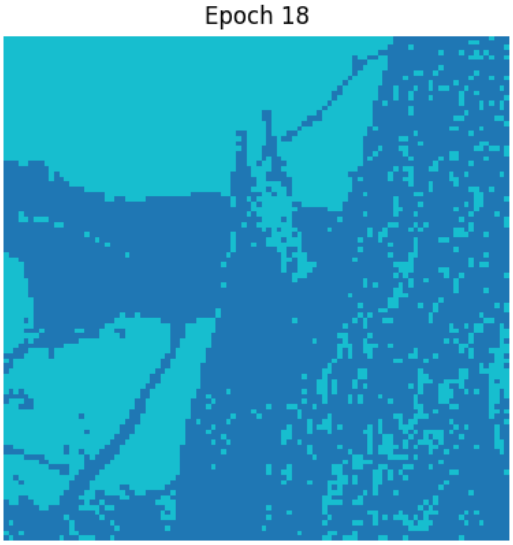
Ratio cut



Normalize cut

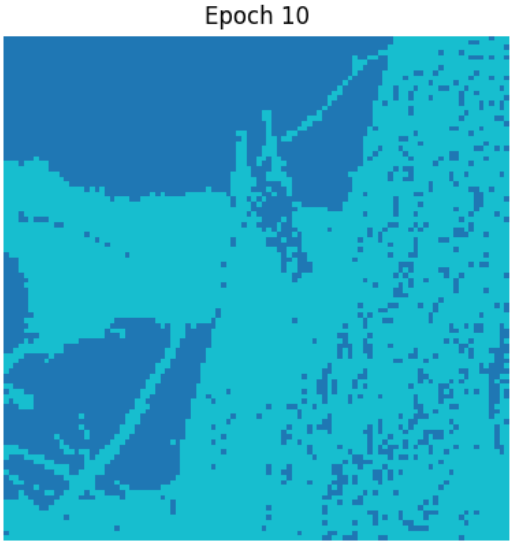
image2.png

Kernel K-means



Ratio cut

Normalize cut

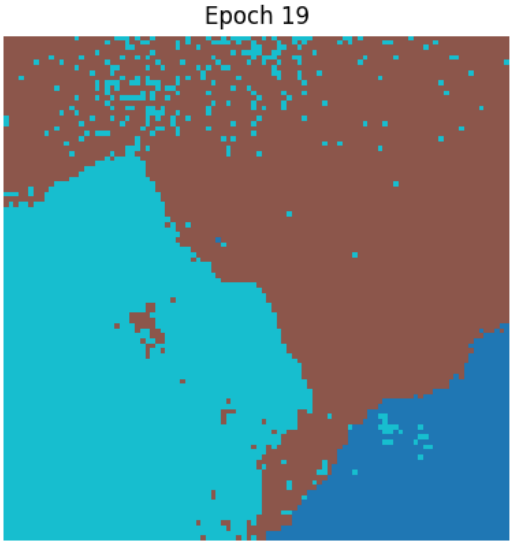


2.2 Part2: Try more clusters

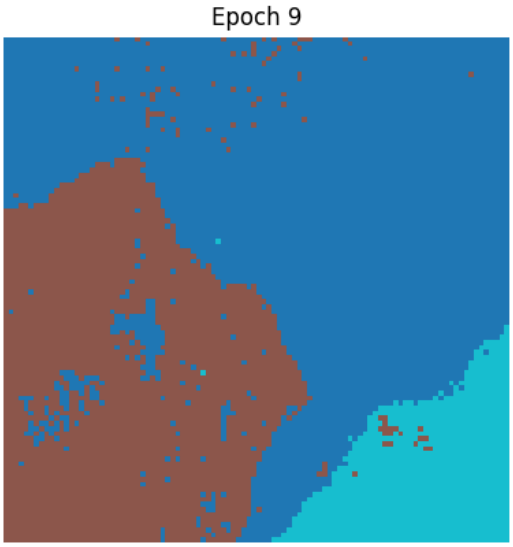
In addition to clustering with k=2, I also experimented with clustering the images into 3 and 4 classes to observe how the algorithms perform with more clusters.

image1.png (k = 3)

Kernel K-means



Ratio cut



Normalize cut

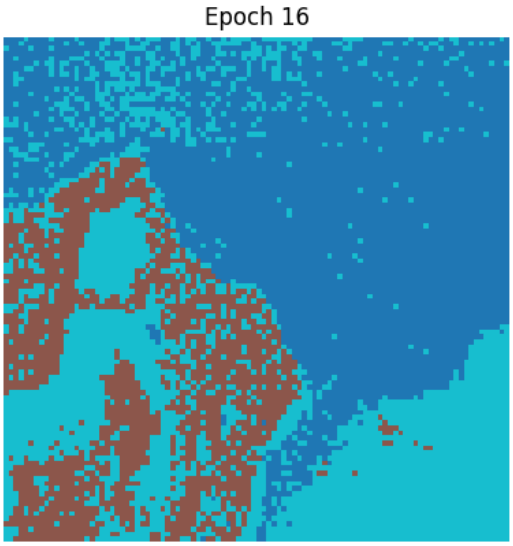
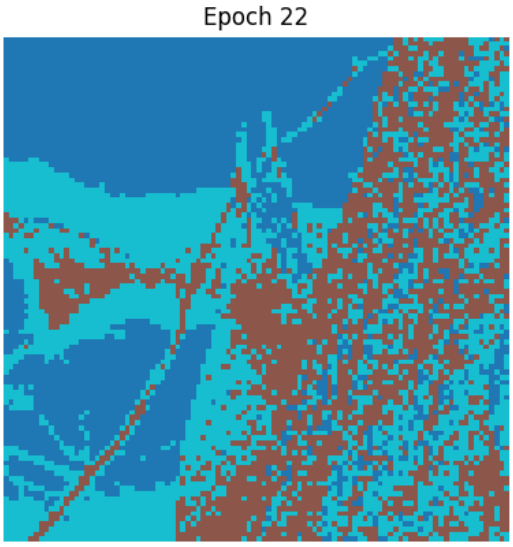
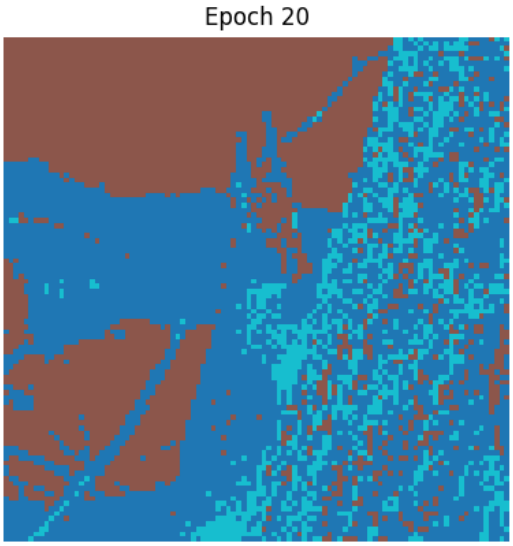


image2.png (k = 3)

Kernel K-means



Ratio cut



Normalize cut

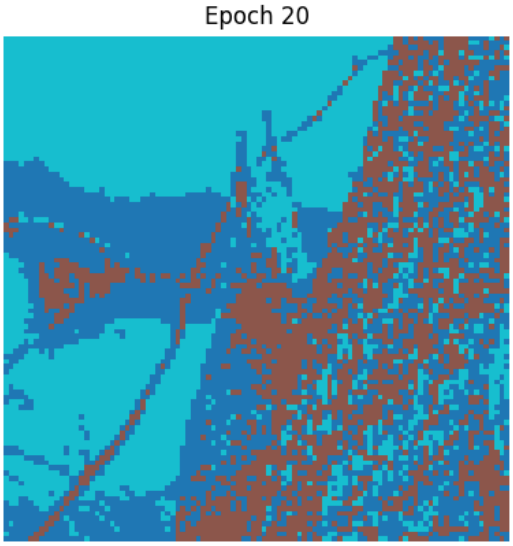
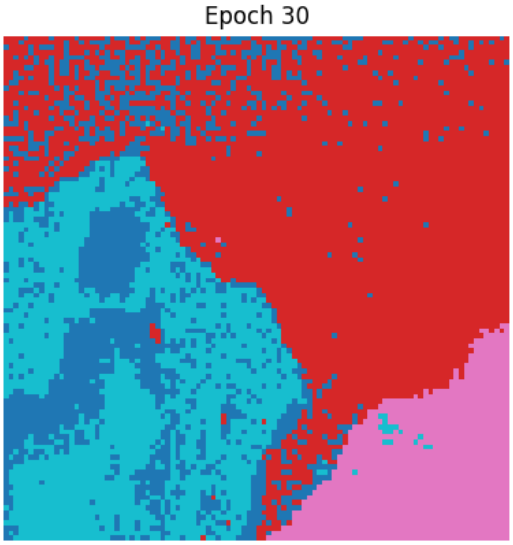
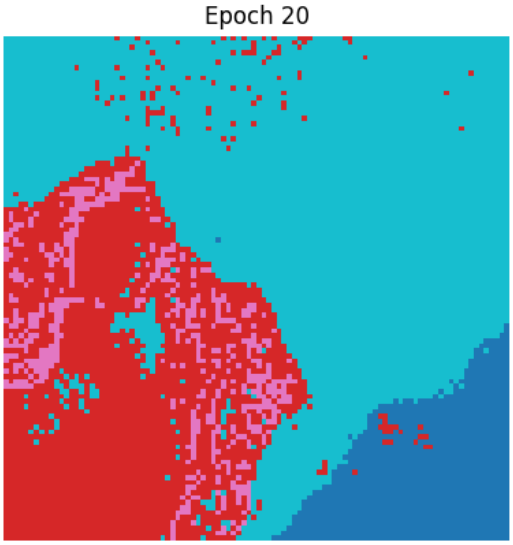


image1.png (k = 4)

Kernel K-means



Ratio cut



Normalize cut

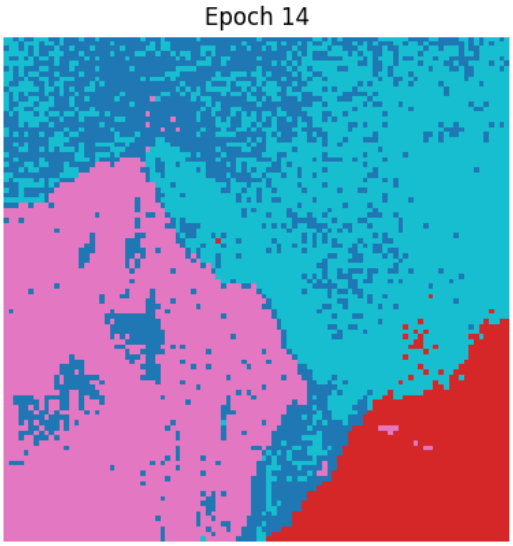
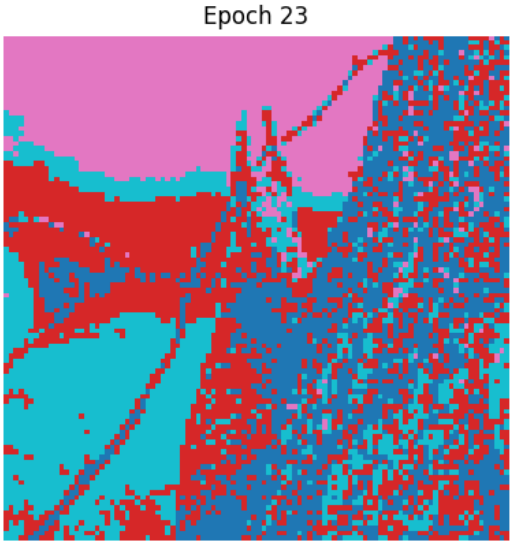
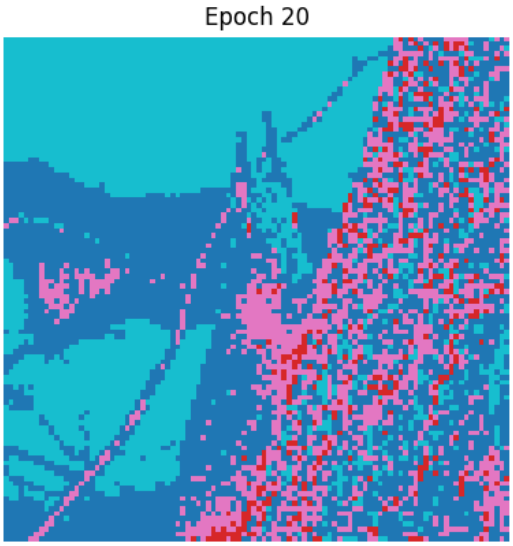


image2.png (k = 4)

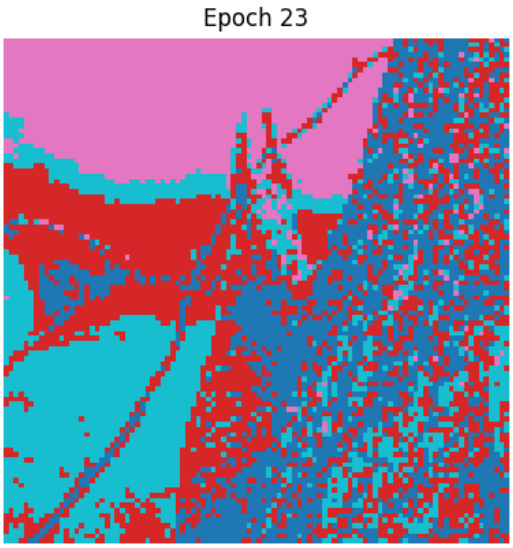
Kernel K-means



Ratio cut



Normalize cut



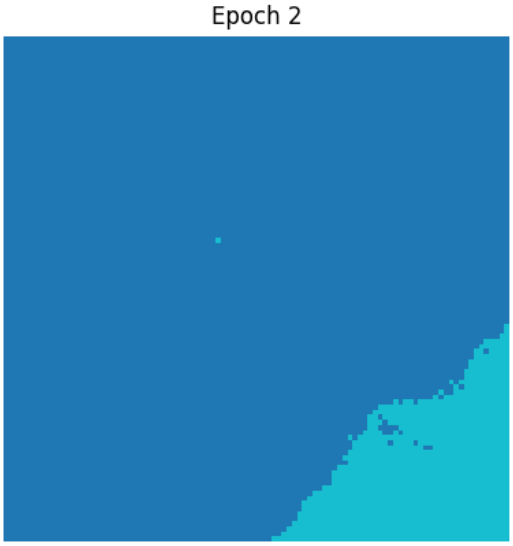
2.3 Part3: Try different initializations

For each condition above, I additionally try kmeans++ for initialization of the centers.

image1.png (k = 2)

Kernel K-means

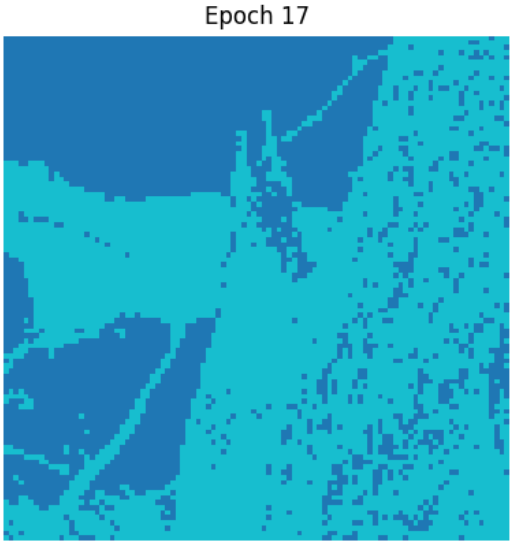
Ratio cut



Normalize cut

image2.png (k = 2)

Kernel K-means



Ratio cut

Normalize cut

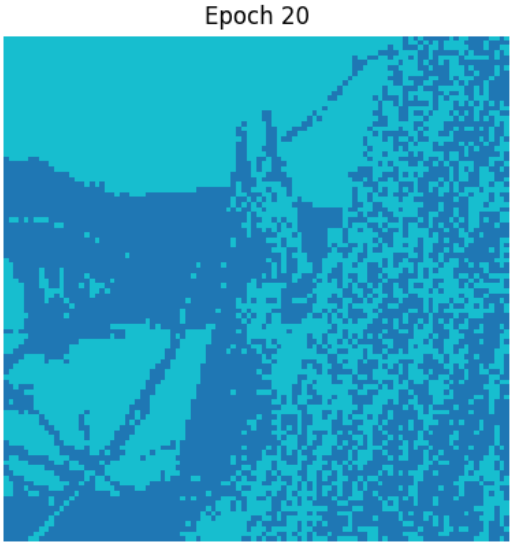
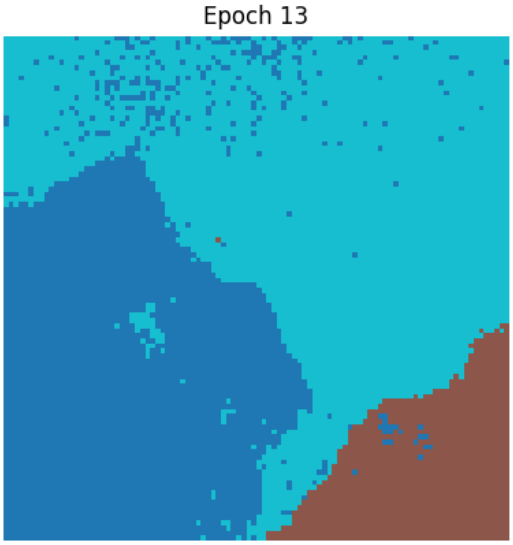
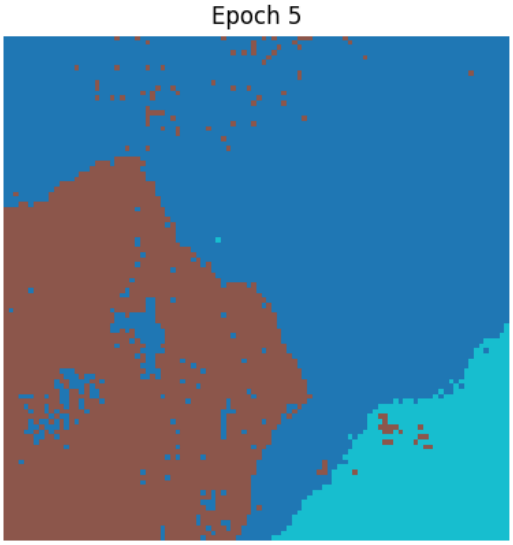


image1.png (k = 3)

Kernel K-means



Ratio cut



Normalize cut

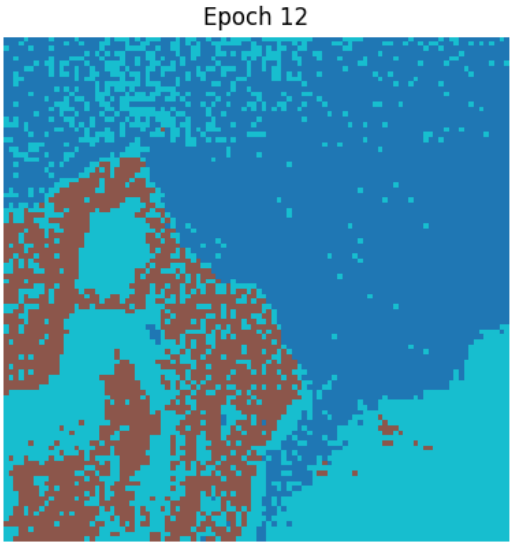
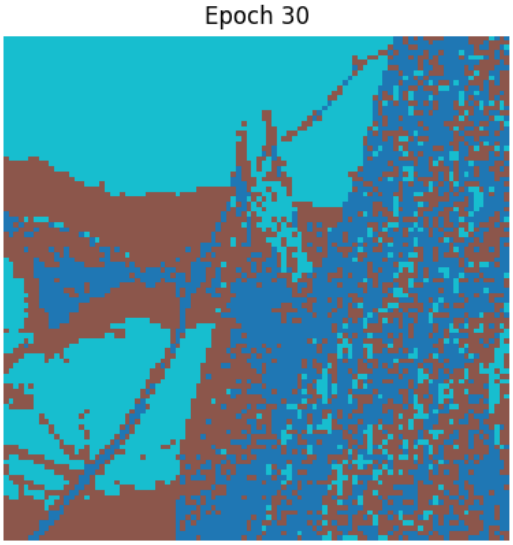
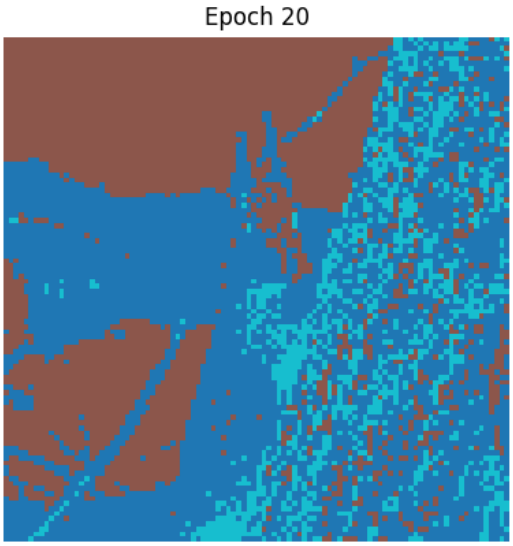


image2.png (k = 3)

Kernel K-means



Ratio cut



Normalize cut

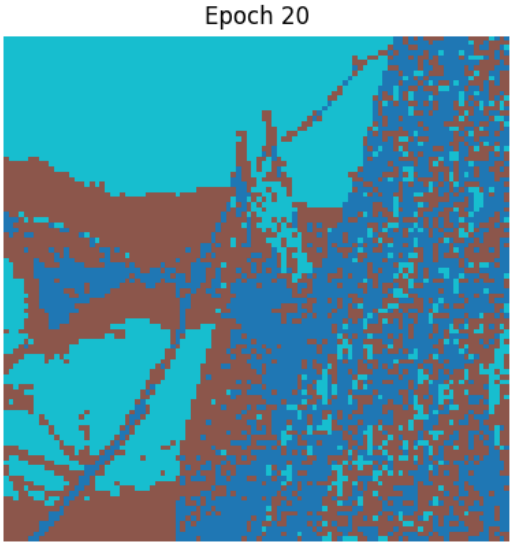
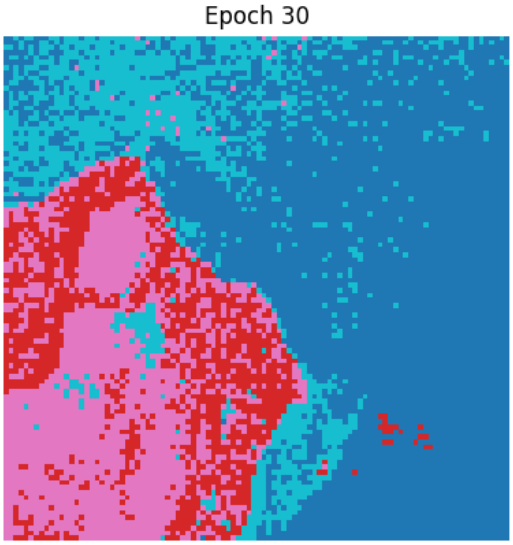


image1.png (k = 4)

Kernel K-means



Ratio cut

Normalize cut

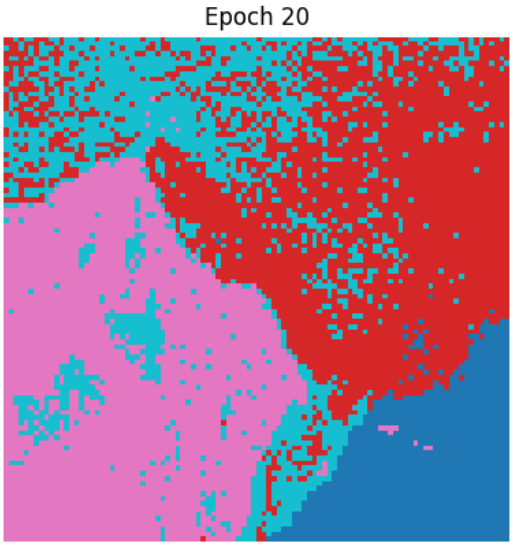
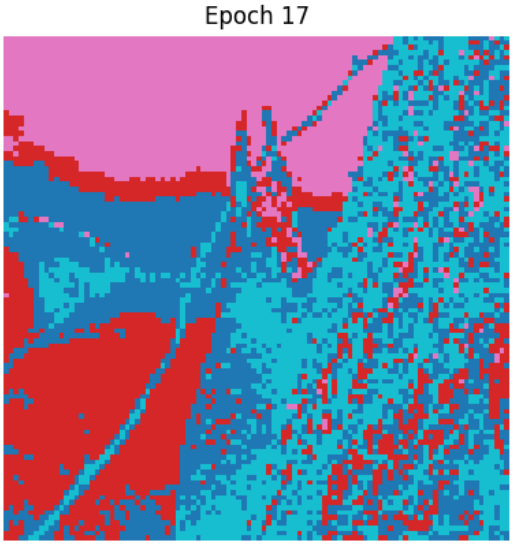
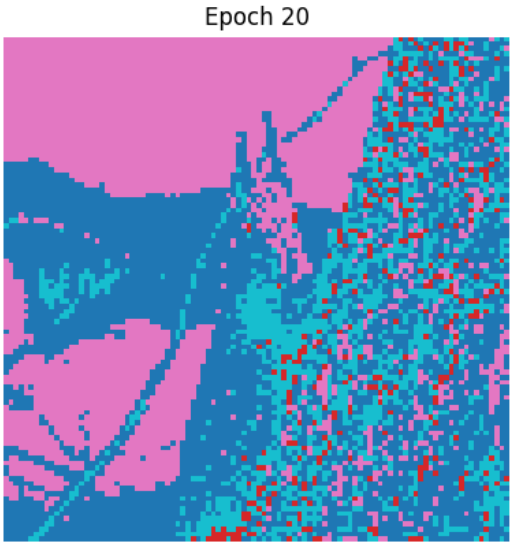


image2.png (k = 4)

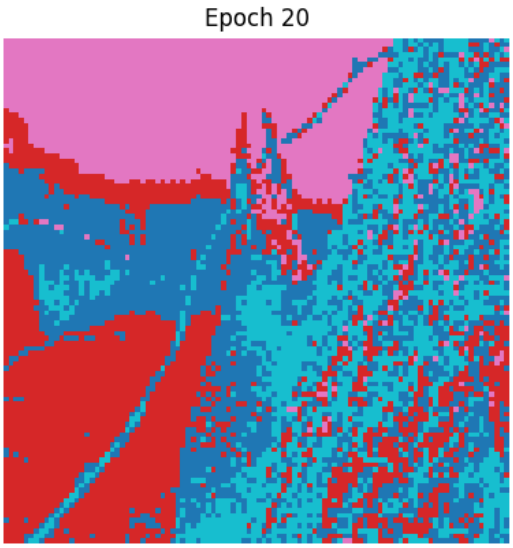
Kernel K-means



Ratio cut



Normalize cut

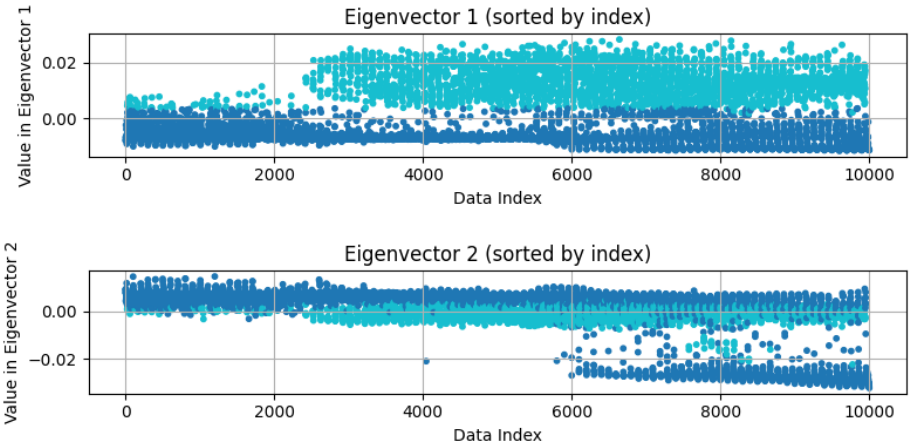


2.4 Part4: Experiments on the coordinates in the eigenspace

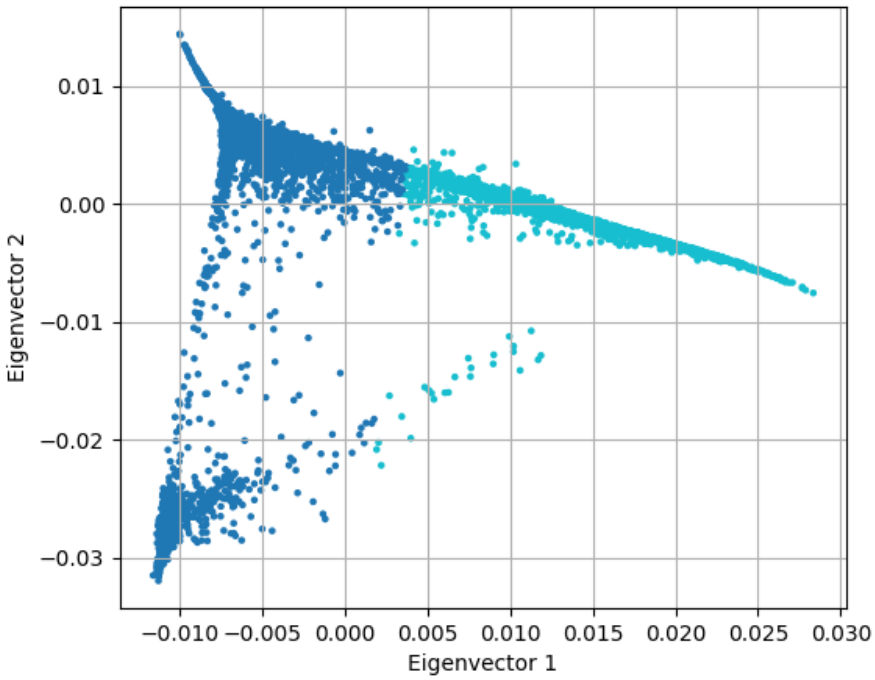
Eigenvector value plot:

image1.png (k = 2)

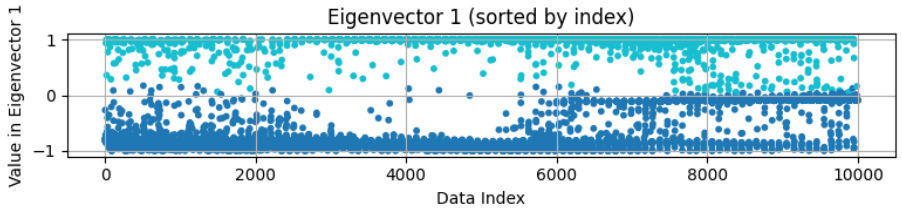
Ratio cut

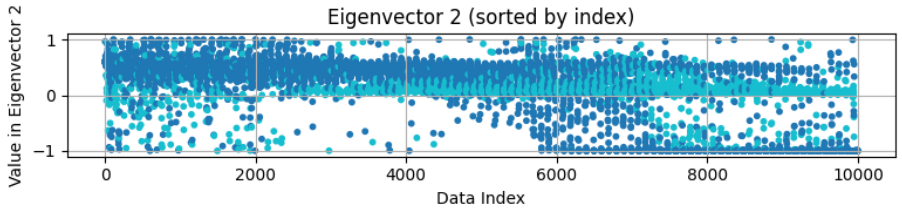


2D projection of eigenspace features:



Normalize cut





2D projection of eigenspace features:

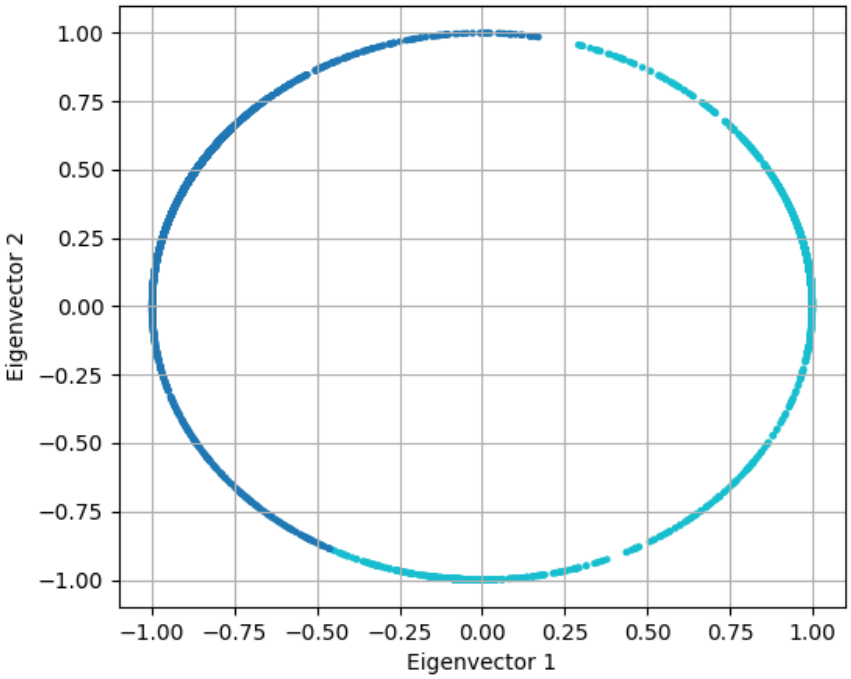
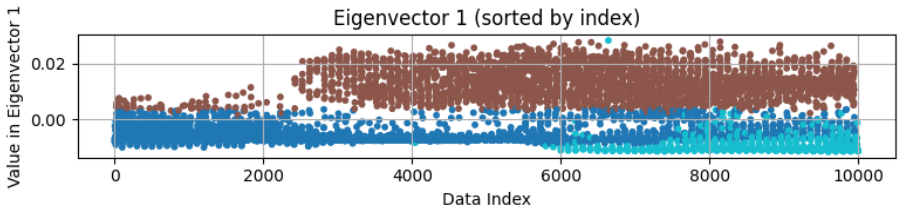
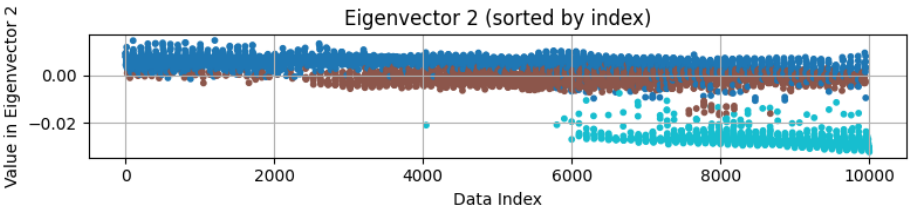
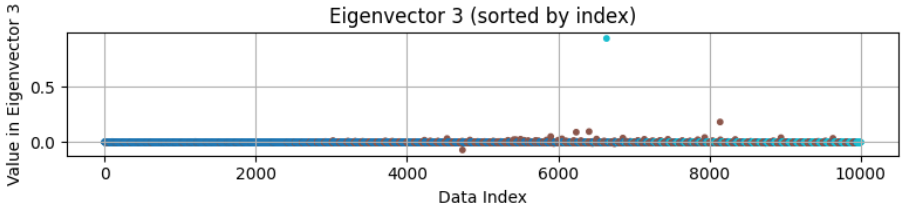


image1.png (k = 3)

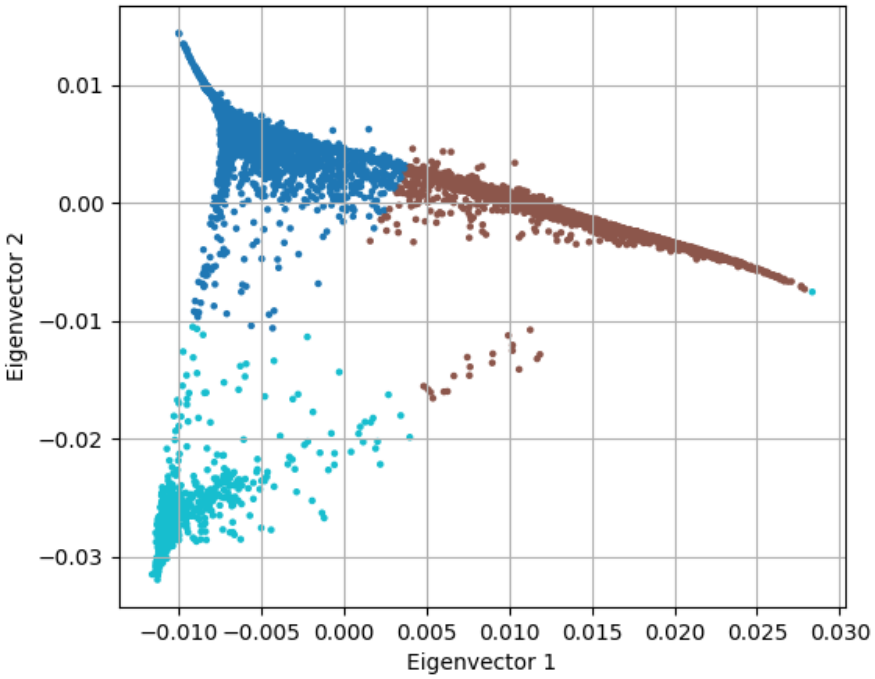
Ratio cut



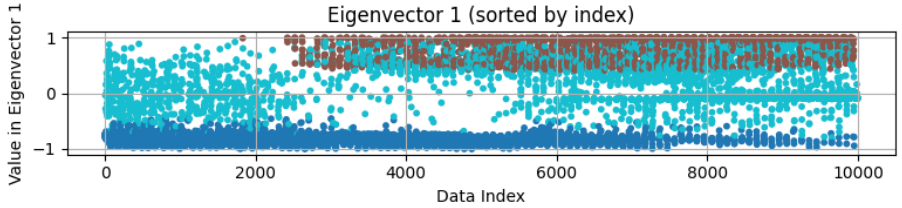


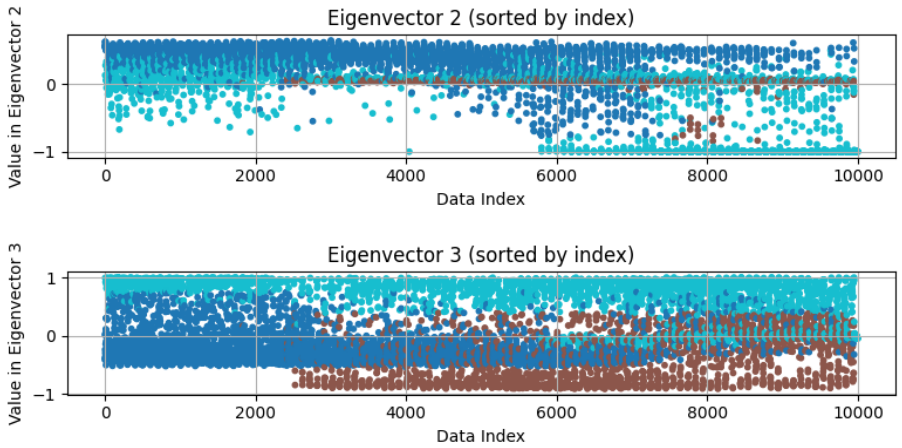


2D projection of eigenspace features:



Normalize cut





2D projection of eigenspace features:

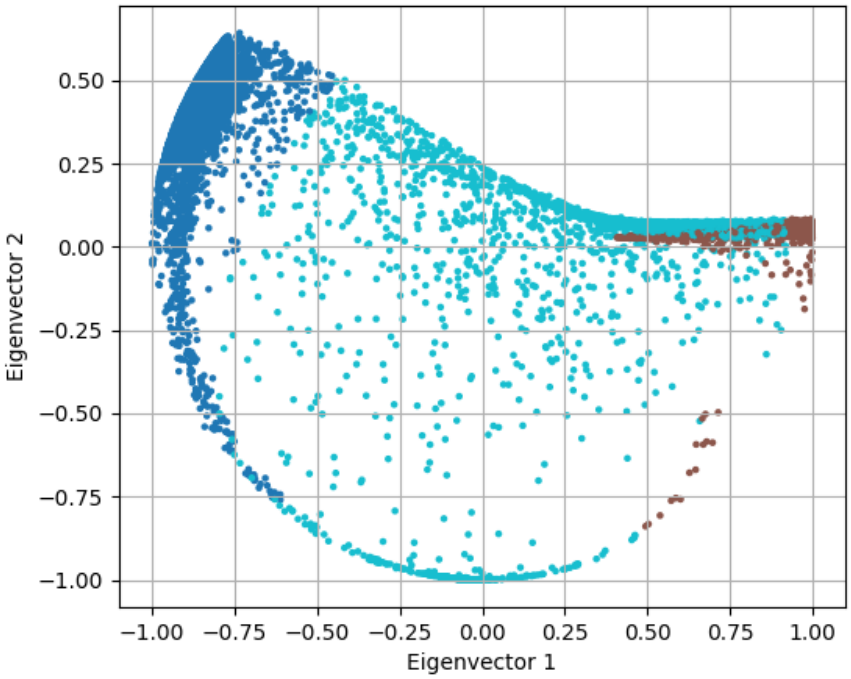
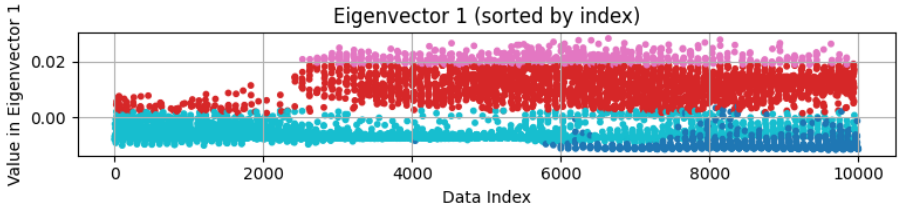
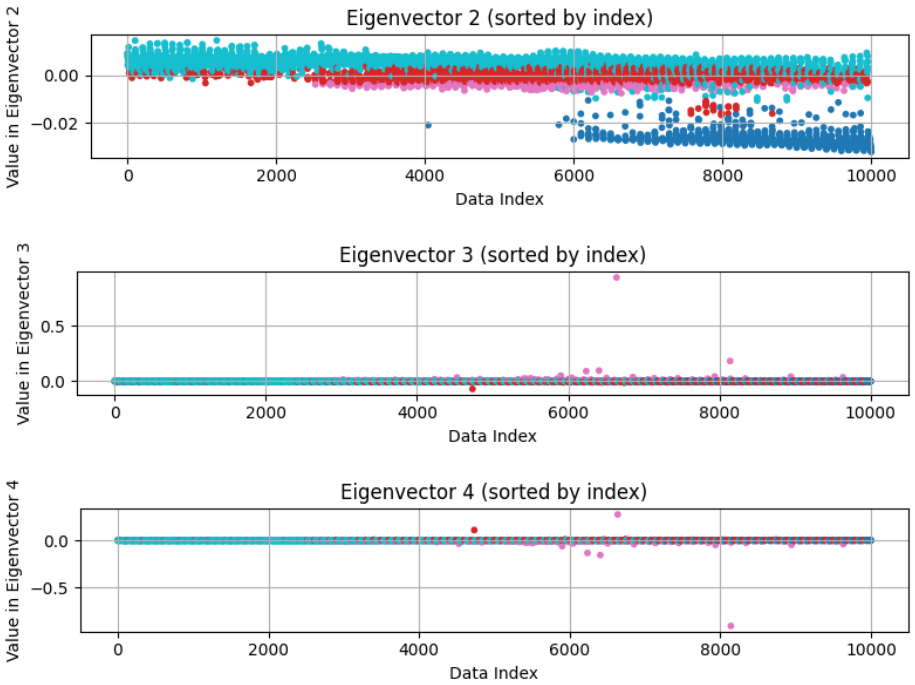


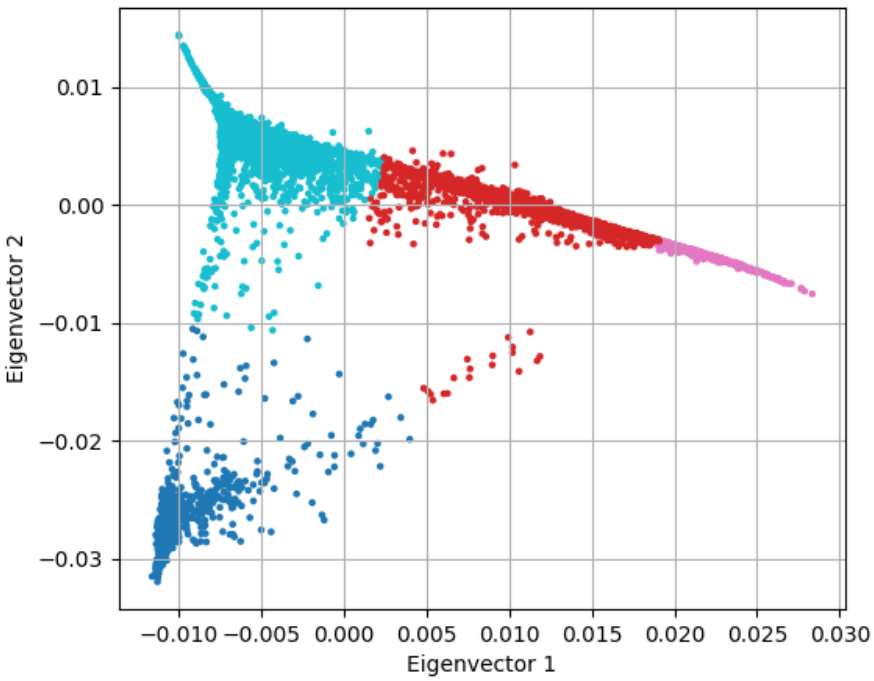
image1.png (k = 4)

Ratio cut

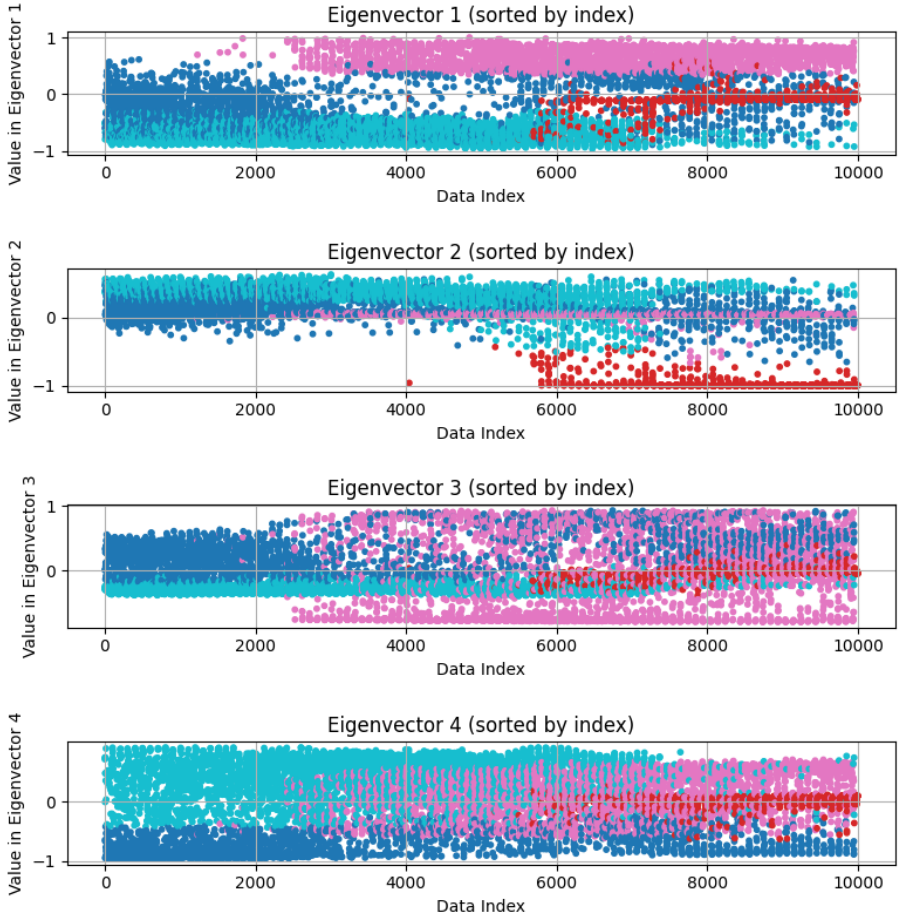




2D projection of eigenspace features:



Normalize cut



2D projection of eigenspace features:

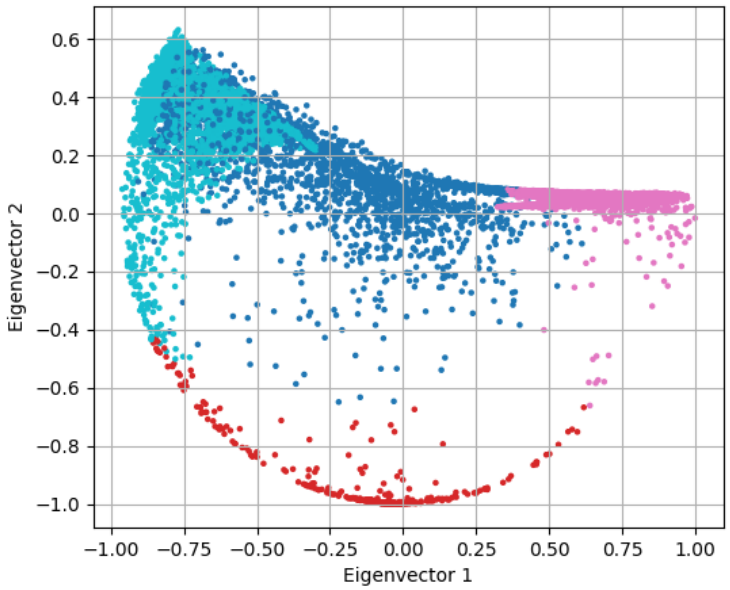
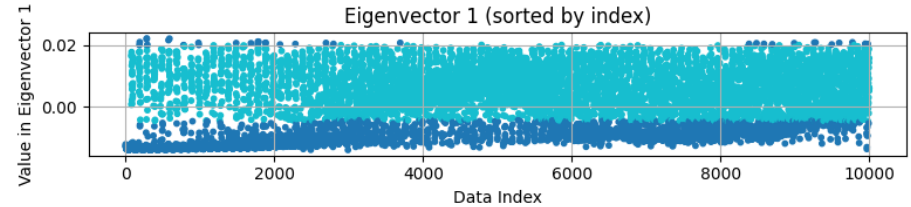
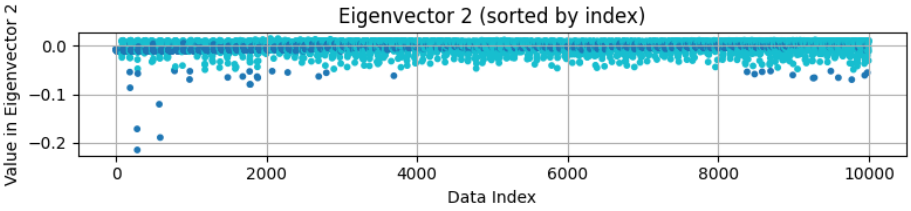


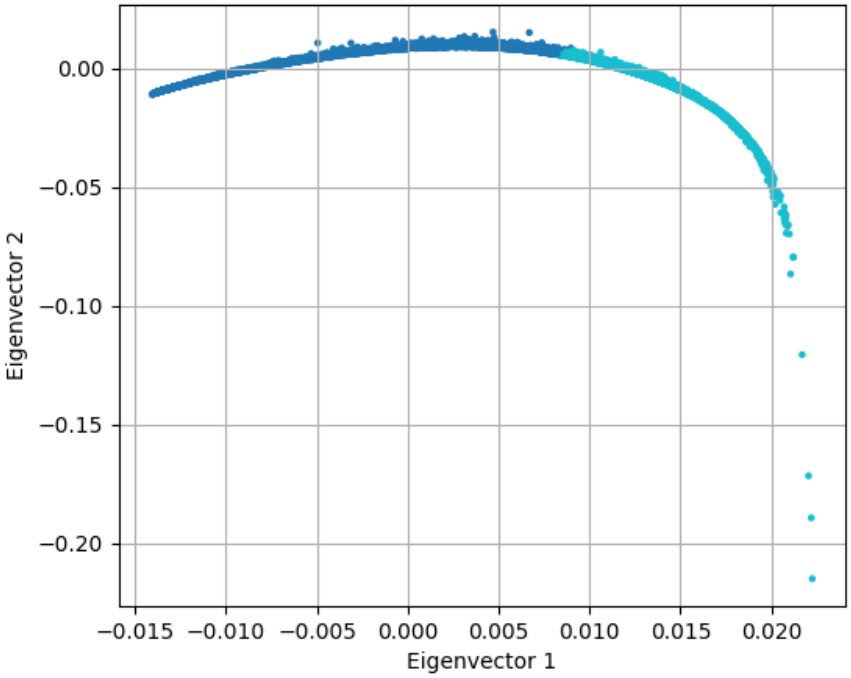
image2.png (k = 2)

Ratio cut

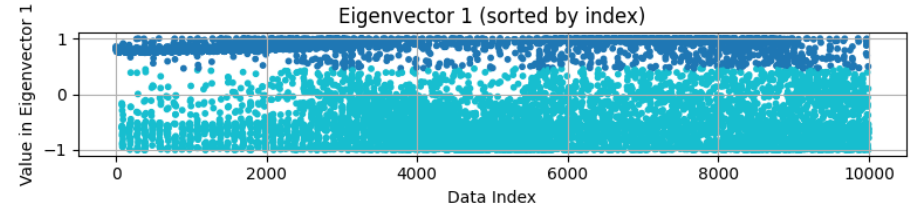


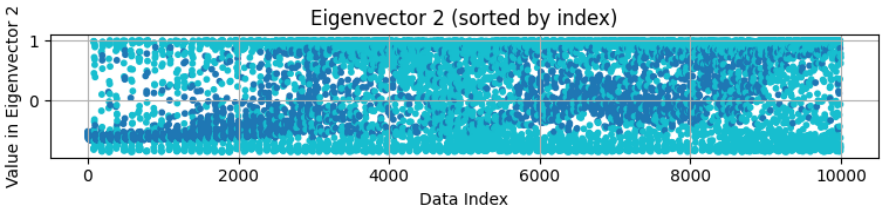


2D projection of eigenspace features:



Normalize cut





2D projection of eigenspace features:

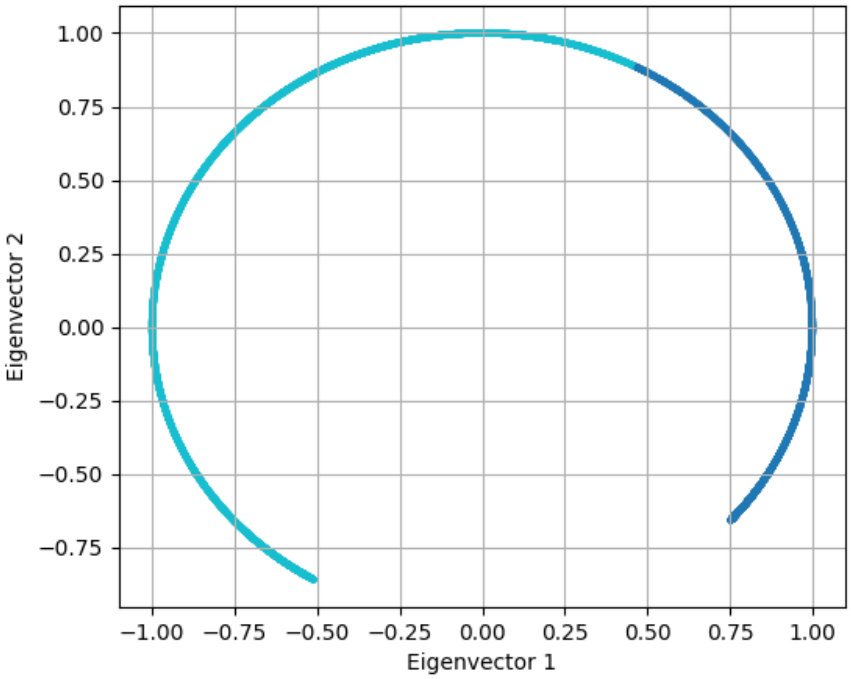
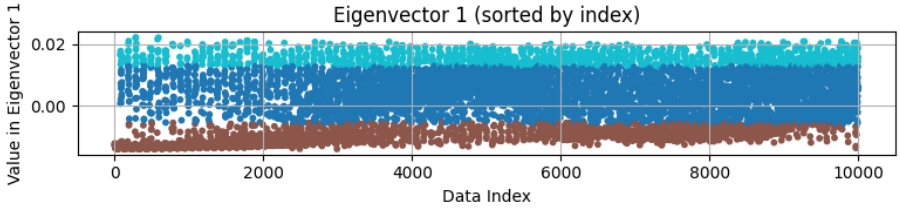
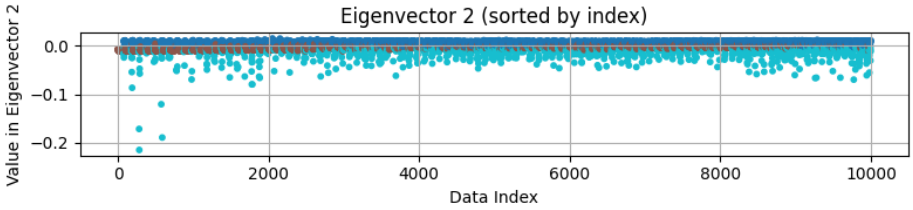
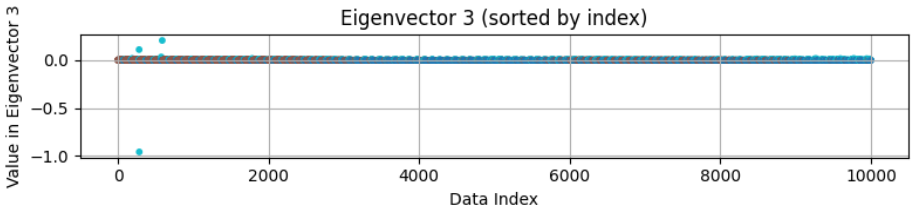


image2.png (k = 3)

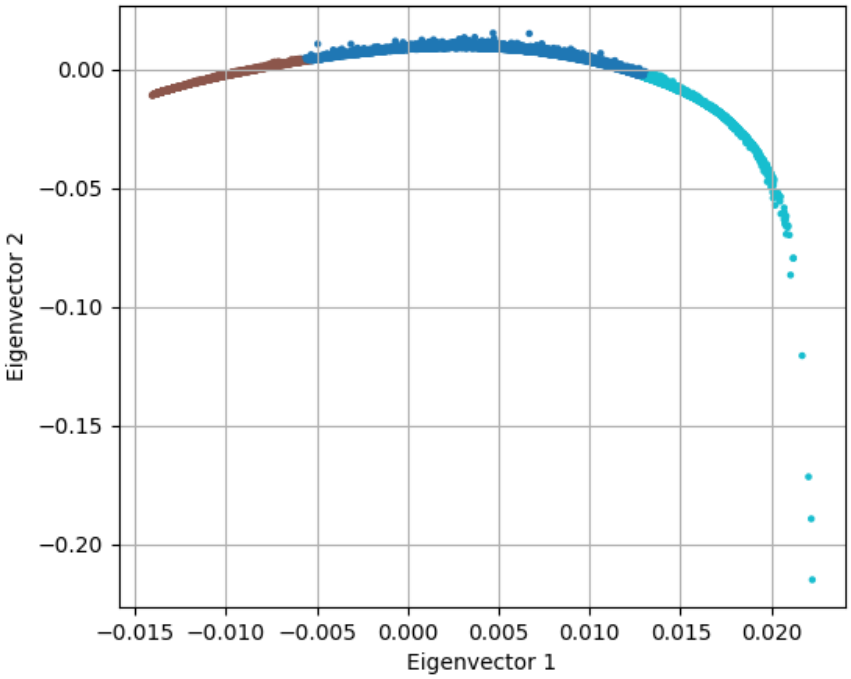
Ratio cut



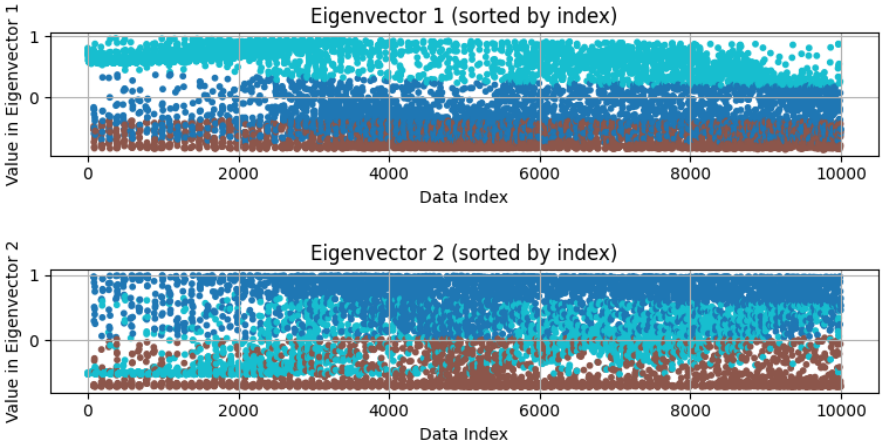


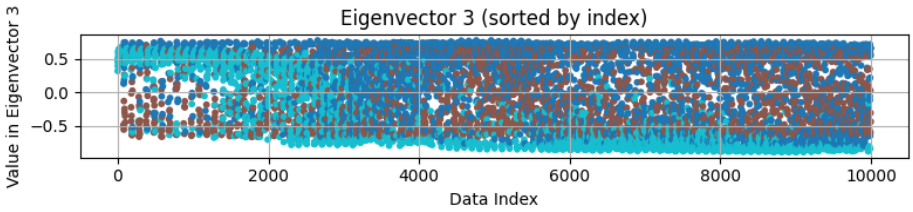


2D projection of eigenspace features:



Normalize cut





2D projection of eigenspace features:

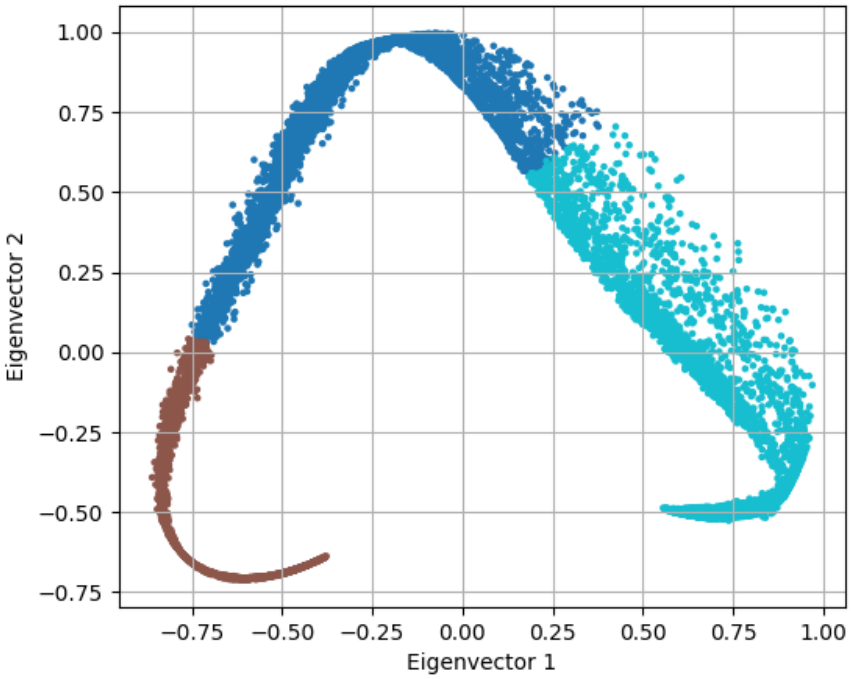
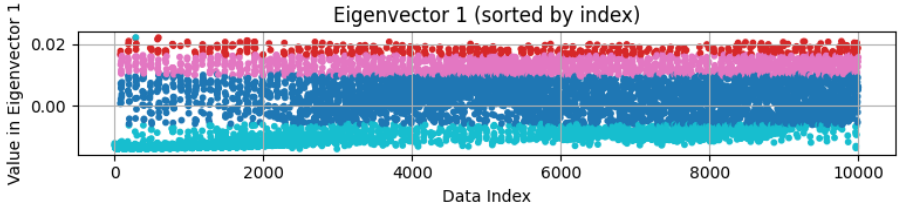
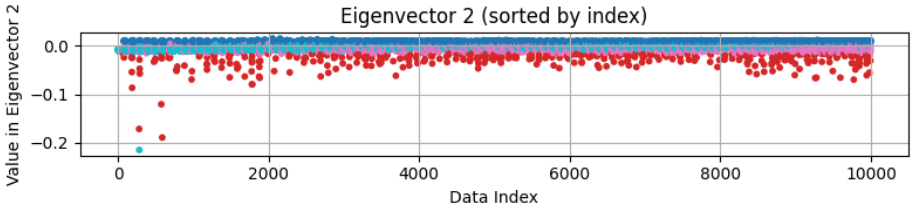
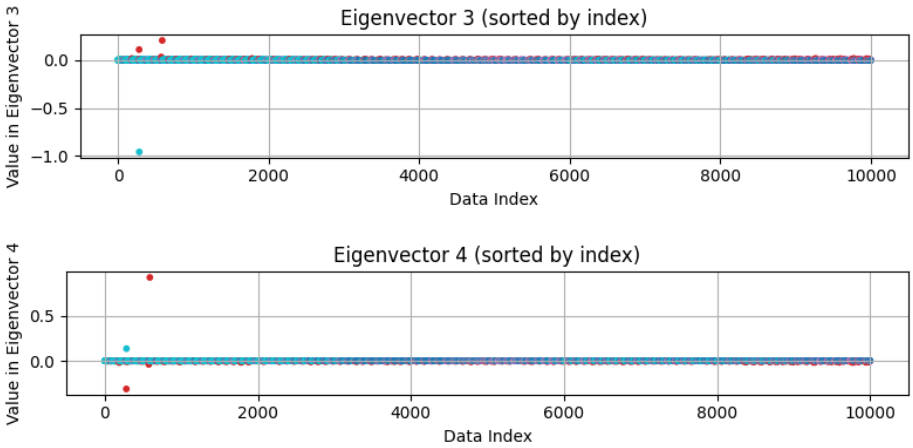


image.png (k = 4)

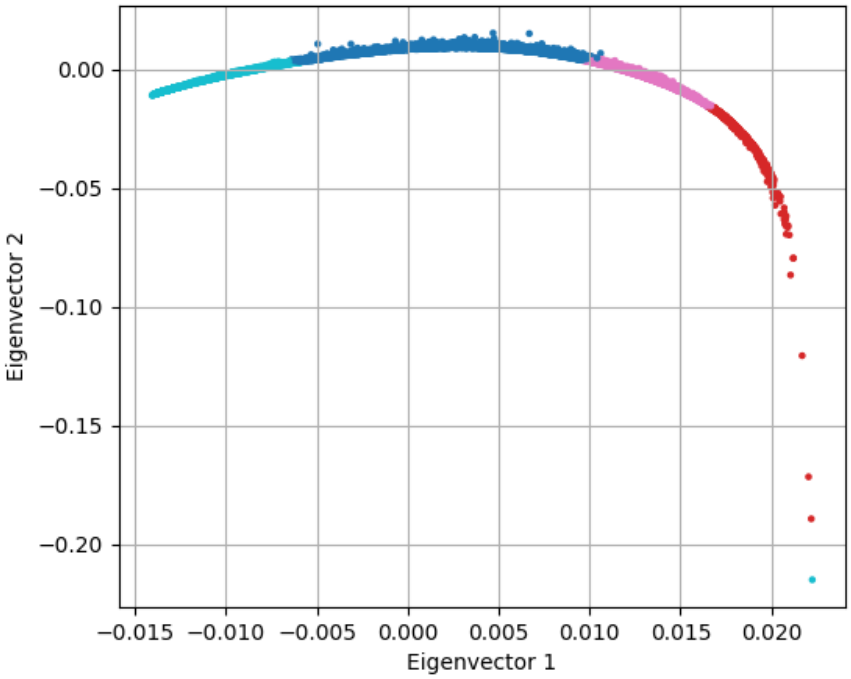
Ratio cut



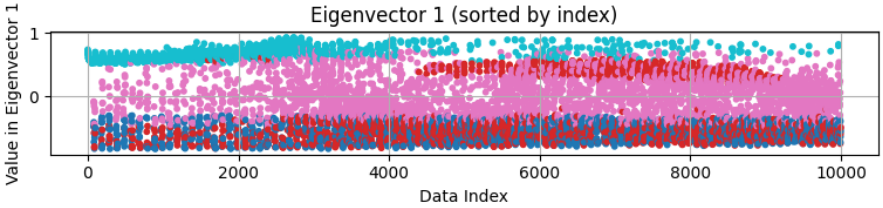


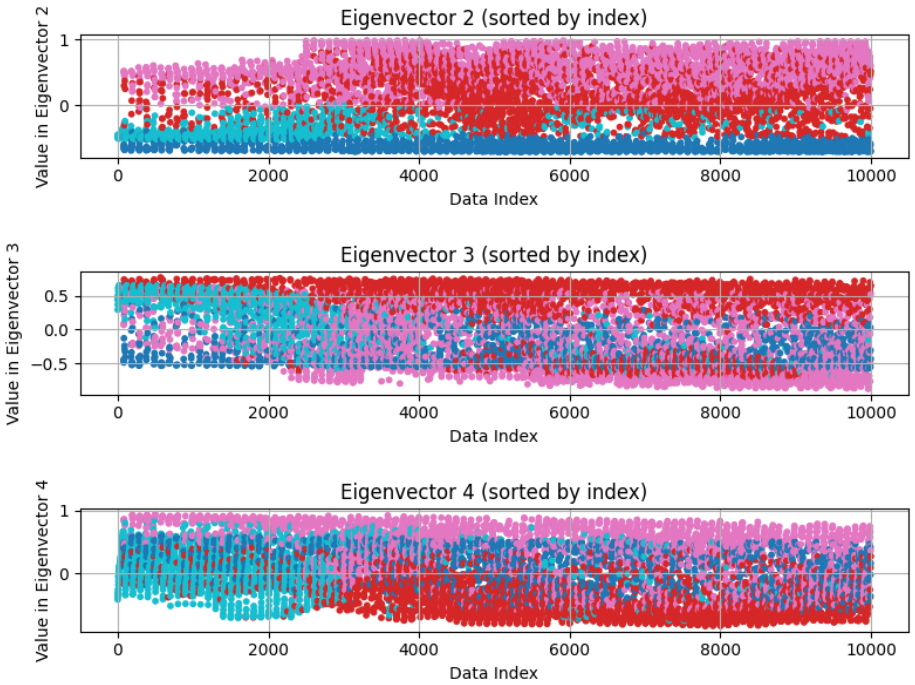


2D projection of eigenspace features:

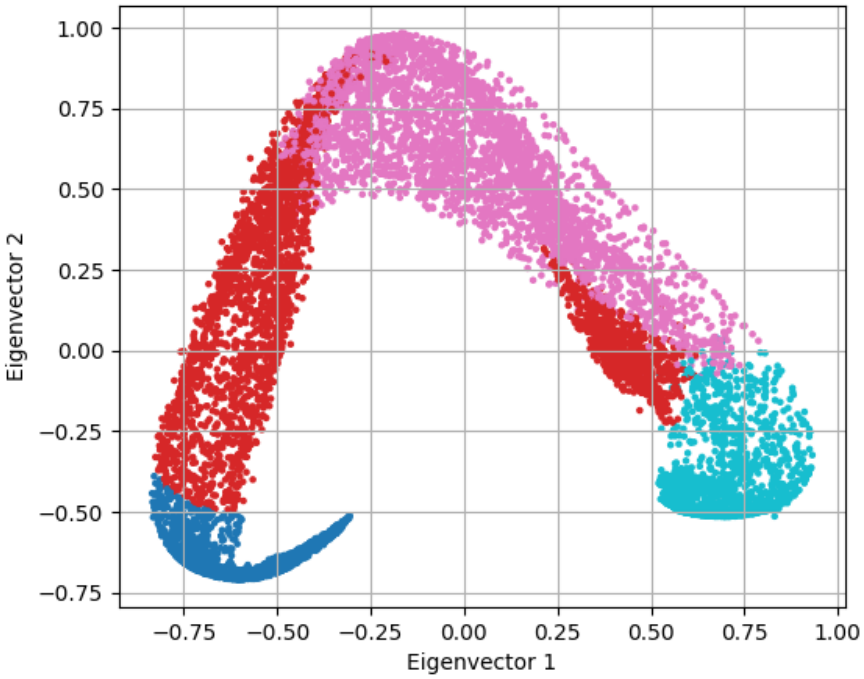


Normalize cut





2D projection of eigenspace features:



3 Observations and discussion

3.1 Comparison between different clustering methods

Part 1

1. **Convergence behavior**

   Kernel k-means converges at epoch 8 (image1), epoch 18 (image2); spectral clustering converges even faster (e.g., epoch 4 or 10), which significantly fewer than the maximum epoch limit (30), which indicates stable convergence behavior.

2. **Kernel K-means produces smooth boundaries**

   For both images, kernel k-means results in large, smooth cluster regions with relatively clean edges. This is because kernel k-means optimizes cluster membership in the transformed kernel space, which encourages compact, homogeneous clusters.

3. **Ratio Cut tends to produce irregular segments**

   - Ratio cut produces less regular or noisier boundaries, particularly in image2.png.
   - This method tries to minimize the cut value relative to the cluster size, which may result in clusters that are unbalanced or spatially scattered.)

4. **Normalized Cut shows better spatial coherence than Ratio Cut**

   Compared to Ratio Cut, Normalized Cut usually results in more spatially coherent clusters because it accounts for the total connection strength within each cluster.In both images, the results of normalized cut look more structured and cleaner than ratio cut, especially in image1.

5. **Image content affects clustering difficulty**

   Clustering on image2.png is clearly harder than on image1.png across all three methods. It may cause by three reasons:

   - The boundaries are more complex.
   - More noise and texture in the image leads to noisier segmentation maps.
   - Image2 requires more epochs (e.g., kernel k-means reaches 18 epochs for image2 vs. 8 for image1).

In Part 1, we observe that all clustering methods converge well before reaching the maximum epoch limit. Kernel k-means tends to produce smoother and more coherent regions, especially in simpler images like `image1.png`. Ratio cut, on the other hand, often results in noisier boundaries due to its sensitivity to cluster size. Normalized cut generally performs better than ratio cut in terms of spatial consistency. We also observe that `image2.png` presents a greater challenge for all algorithms, as its structural complexity leads to more fragmented clusters and slower convergence.

Part 2

I experimented with larger numbers of clusters ($k = 3$, $k = 4$), and made the following observations:

1. **Kernel K-means maintains coherent boundaries but starts to over-segment at $k = 4$**

   - At $k = 3$, kernel k-means still produces clean and spatially coherent clusters, especially in `image1.png`, where it separates the large foreground and background, and further isolates a third region.
   - However, at $k = 4$, the result starts to look over-segmented, with some smaller clusters appearing noisy or scattered. This indicates that the intrinsic structure of the data may not support four strong, separable regions.

2. **For spectral clustering, the results become less stable as k increases**

   As k increases to 3 and especially 4, the segmentation results tend to become more fragmented and irregular, particularly in fine-grained regions such as texture boundaries or areas with high visual complexity (e.g., in `image2.png`). While the overall cluster shapes remain roughly consistent, local inconsistencies become more frequent. This suggests that spectral methods—both ratio cut and normalized cut—are sensitive to cluster balance and may struggle when the number of clusters exceeds the intrinsic structure of the data.

3. **Signs of eigen gap limitation appear at $k = 4$**

   In all three methods, the results at $k = 4$ do not clearly improve or provide meaningful subdivisions compared to $k = 3$. This supports the idea (from Part 4 eigenspace analysis) that the data may not support more than 3 meaningful clusters, and the 4th cluster is likely artificial. I will validate this in the result interpretation in Part 4.

Part 3

In this part, I try the K-means++ method to improve the initial task. Based on the observation of the result:

1. **Faster or more stable convergence**

   Across many settings, K-means++ initialization led to faster convergence compared to random initialization. Even when epochs are similar (e.g., kernel k-means on image2.png), the convergence is smoother and more stable.

2. **More consistent cluster boundaries**

   K-means++ helps avoid poor initial seeds that lead to suboptimal clustering boundaries. The kernel K-means result appears more symmetrical and balanced across regions. Spectral methods also show reduced noise and more coherent boundaries (e.g., normalized cut in image2.png (k=3)).

3. **Reduced over-segmentation at higher k**

   When k=4, results with k-eans++ appear less fragmented compared to random init.

   - In image1.png, kernel k-means has smoother transitions between the four clusters.
   - In image2.png, although visual noise still exists, the major structures are more clearly separated.

   Overall, these results suggest that K-means++ provides a more robust initialization strategy, especially as the number of clusters increases or the data becomes more complex.

Part 4

To understand the clustering behavior in spectral clustering, we analyzed the eigenvectors of the Laplacian matrices and their corresponding eigenvalue spectra. Our observations are presented from two perspectives:

1. Comparison between Ratio Cut and Normalized Cut

- **Eigenvector structure:** In both methods, the first non-trivial eigenvectors (i.e., those corresponding to the smallest non-zero eigenvalues) provided the most discriminative power for clustering. However, as k increases, we observed growing instability. Ratio cut often produced cleaner and more structured eigenvector value distributions, possibly due to its use of the unnormalized Laplacian, which better preserves affinity information without row normalization.

- **2D projection quality:** When projecting data into the eigenspace spanned by the first two eigenvectors, ratio cut frequently yielded more visually interpretable and stable cluster boundaries, especially on `image1.png`. While normalized cut is designed to improve cluster balance, the row normalization sometimes introduced distortion, reducing separability.

- **Sensitivity to k:** Both methods showed degraded clustering quality as k increased beyond the support of the spectral structure. Normalized cut degraded more gradually, while ratio cut occasionally produced irregular or fragmented segments.

2. Comparison between `image1.png` and `image2.png`

- **Eigenspace clarity:** `image1.png` exhibited stronger spectral structure, with clearer eigenvector patterns and well-separated clusters in the 2D projections. In contrast, `image2.png` had noisier eigenvectors and more overlap in projections, indicating weaker separability.

- **Label overlap onset:** For `image1.png`, label overlap began at the third non-trivial eigenvector (Eigenvector 4), whereas in `image2.png`, overlap started earlier—at the second non-trivial eigenvector (Eigenvector 3)—revealing a more limited useful eigenspace.

- **Eigenvalue spectrum analysis:** To validate my assumption, I also do he eigenvalue spectrum to find the eigengap.
  - `image1.png`:

    $$[2.5 \times 10^{-12},\ 285.27,\ 477.99,\ 808.15, \dots] \Rightarrow \text{Gap: } 808.15 - 477.99 = 330.16$$

    Indicates support for up to $k = 3$ meaningful clusters.
  - `image2.png`:

    $$[-2.76 \times 10^{-12},\ 357.72,\ 946.77, \dots] \Rightarrow \text{Gap: } 946.77 - 357.72 = 589.05$$

    Suggests only the first non-trivial eigenvector contributes significantly, supporting $k = 2$.

Conclusion

- Spectral clustering effectiveness strongly depends on the structure of the Laplacian eigenspace. The first few non-trivial eigenvectors carry the most useful information; beyond that, clustering becomes less reliable.
- The number of clusters should be guided by the observed eigengap. For `image1.png`, the eigengap suggests $k \leq 3$ is appropriate, while for `image2.png`, $k = 2$ is the safest choice.
- Ratio cut and normalized cut differ in behavior. Ratio cut tends to retain more raw spectral structure, which sometimes leads to better visual separation, especially in simpler images like `image1.png`.

3.2 Compare the execution time of different settings

To evaluate the execution time under different conditions, I incorporated the `time` library to measure the runtime of each clustering method. Note that this timing experiment was conducted separately from the visualization experiments described earlier, so the number of epochs until convergence may differ slightly between runs.

The measured execution times and epochs for different images, methods, values of k, and initialization strategies are summarized in Table 1 and Table 2.

I have some observations of the tables:

- While normalized cut provides strong theoretical guarantees for balanced partitions, it is computationally heavier. Ratio cut is efficient but potentially less robust. Kernel K-means strikes a middle ground—highly sensitive to initialization, but able to perform efficiently with K-means++ and well-tuned parameters.
- In spectral clustering, a significant portion of the runtime is spent on computing the eigenvalues of the Laplacian matrix. Additionally, when using K-means++ for initialization, I perform 10 attempts to find the best initial centers, which can become a computational bottleneck.

Table 1: Execution time and number of epochs on `image1.png` under different settings

Method	k	**Image**	**Initialization**	**Execution Time (s)**	**Epochs**
Kernel K-means	2	image1	Random	16.86	11
Kernel K-means	2	image1	KMeans++	14.00	9
Kernel K-means	3	image1	Random	21.49	7
Kernel K-means	3	image1	KMeans++	24.58	15
Kernel K-means	4	image1	Random	79.07	30
Kernel K-means	4	image1	KMeans++	104.19	30
Ratio Cut	2	image1	Random	140.19	4
Ratio Cut	2	image1	KMeans++	173.72	2
Ratio Cut	3	image1	Random	90.56	10
Ratio Cut	3	image1	KMeans++	222.96	4
Ratio Cut	4	image1	Random	94.46	10
Ratio Cut	4	image1	KMeans++	99.22	8
Normalized Cut	2	image1	Random	247.05	4
Normalized Cut	2	image1	KMeans++	254.95	3
Normalized Cut	3	image1	Random	257.07	15
Normalized Cut	3	image1	KMeans++	256.62	7
Normalized Cut	4	image1	Random	254.63	19
Normalized Cut	4	image1	KMeans++	261.09	19

Table 2: Execution time and number of epochs on `image2.png` under different settings

Method	k	**Image**	**Initialization**	**Execution Time (s)**	**Epochs**
Kernel K-means	2	image2	Random	36.19	25
Kernel K-means	2	image2	KMeans++	35.84	21
Kernel K-means	3	image2	Random	86.20	27
Kernel K-means	3	image2	KMeans++	14.00	8
Kernel K-means	4	image2	Random	29.02	19
Kernel K-means	4	image2	KMeans++	166.23	30
Ratio Cut	2	image2	Random	102.09	12
Ratio Cut	2	image2	KMeans++	190.34	12
Ratio Cut	3	image2	Random	94.32	28
Ratio Cut	3	image2	KMeans++	202.43	24
Ratio Cut	4	image2	Random	201.41	30
Ratio Cut	4	image2	KMeans++	110.34	12
Normalized Cut	2	image2	Random	249.36	15
Normalized Cut	2	image2	KMeans++	242.51	7
Normalized Cut	3	image2	Random	236.08	12
Normalized Cut	3	image2	KMeans++	308.07	14
Normalized Cut	4	image2	Random	225.8	16
Normalized Cut	4	image2	KMeans++	272.09	28

3.3 Anything want to discuss

Nonlinear Clustering in Spectral Embedding Space

In this project, I applied k-means clustering in the spectral embedding space formed by the first few non-trivial eigenvectors of the graph Laplacian. While k-means is simple and efficient, it assumes clusters to be convex and isotropic in Euclidean space. However, as observed in some projections—particularly in `image2.png`—the embedded data may form curved or elongated manifolds. In such cases, applying nonlinear clustering methods such as DBSCAN or spectral density-based clustering could potentially yield better results, as they do not require specifying the number of clusters and can adapt to irregular shapes. Exploring alternative clustering strategies in the eigenspace could be a promising direction for improving spectral clustering performance on complex data.

Effect of Convergence Criteria and Maximum Epochs on Execution Time

In this project, the convergence condition for all clustering methods was defined as the point when no data point changes its cluster assignment during an epoch. This "no-change" condition ensures that the algorithm halts as soon as a local optimum is reached, potentially saving time compared to always running the maximum number of epochs.

However, in cases where convergence is slow or not achieved (e.g., due to poor initialization or ambiguous cluster boundaries), the algorithm continues until the preset maximum number of epochs is reached. This design means that execution time is influenced both by the convergence behavior and the maximum epoch limit. As a result, some experiments—particularly those that hit the maximum epoch limit—may not reflect the actual cost of reaching convergence, but rather the cost of being capped by the upper bound.

Therefore, when comparing execution time across methods or initialization strategies, it is important to consider whether the algorithm terminated by convergence or by reaching the maximum allowed iterations.

Choosing the Number of Clusters Based on Image Structure

Our experiments show that the appropriate value of k can vary significantly depending on the complexity and content of the image. In `image1.png`, both the eigenspace projections and the eigenvalue spectrum suggest clear structural separation up to $k = 3$, with a noticeable eigengap after the third smallest eigenvalue. This aligns with the visual observation of three coherent regions in the image. In contrast, `image2.png` exhibits a strong eigengap after the second eigenvalue, and clustering quality noticeably degrades when $k > 2$. This indicates that `image2.png` may intrinsically support only two strong clusters.

These findings suggest that the intrinsic structure of the image data should guide the choice of k, and that spectral properties—especially the eigengap—can serve as useful indicators. However, eigengap alone is not always sufficient. In some cases, even when the eigengap is ambiguous, visual quality of segmentation may still remain acceptable. Therefore, a combined strategy using both spectral analysis and empirical testing is recommended when selecting k for real-world applications.

4 How to run my code

Execute the instruction: **python main.py**

Input:

- Choose different methods (0: Kernel k-means, 1: Spectral ratio cut, 2: Spectral normalized cut)
- Number of cluster (k)
- Initialization method (0: random, 1: k-means++)

```
PS C:\Users\alany\OneDrive\桌面\nthu\113-2\Machine Learning\Labs\Lab6> python .\main.py
Mode (0: Kernel k-means, 1: Spectral ratio cut, 2: Spectral normalized cut): 1
k: 4
Initialize Label (0: random, 1: k-means++): 1
```

GIF files:

The GIF files generated during the experiments are stored in the directories **kernel_kmeans** and **spectral**, corresponding to kernel K-means and spectral clustering methods, respectively. The directory **imageid++** contains the results obtained using K-means++ initialization.