

ML_HW6

312657008_江祐姍

In this homework, I used Python and imported the following packages:

- import numpy as np
- from PIL import Image
- from scipy.spatial.distance import cdist
- import matplotlib.pyplot as plt
- from matplotlib.animation import FuncAnimation
- import os

1. Code with detailed explanations

i. Part 1.

kernel k-means

First, I employ the following kernel function:

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

where $S(x)$ is the spatial information (i.e. the coordinate of the pixel) of data x , and $C(x)$ is the color information of data x . γ_s and γ_c are hyper-parameters

```
def Gram_matrix(data_s, data_c, gamma_s=0.001, gamma_c=0.001):
    kernel_s = np.exp(-gamma_s * cdist(data_s, data_s, 'sqeuclidean'))
    kernel_c = np.exp(-gamma_c * cdist(data_c, data_c, 'sqeuclidean'))
    kernel = kernel_s * kernel_c
    return kernel

def initial(kernel, k):
    n = kernel.shape[0]
    center = np.random.choice(n, size=k, replace=False)
    mean = kernel[center, :]
    return mean

# K-means
def k_means(kernel, k, max_iterations=100):
    mean = initial(kernel, k)
    diff = 10
    history = []
```

```

iteration = 0

while diff > 1e-6 and iteration < max_iterations:
    # E-step
    c = np.zeros(kernel.shape[0], dtype=int)
    for i in range(kernel.shape[0]):
        distances = [np.linalg.norm(kernel[i] - mean[j]) for j in range(k)]
        c[i] = np.argmin(distances)

    history.append(c.copy())
    # M-step
    previous_mean = mean.copy()
    mean = np.zeros((k, kernel.shape[1]), dtype=kernel.dtype)
    counters = np.zeros(k)

    for i in range(kernel.shape[0]):
        mean[c[i]] += kernel[i]
        counters[c[i]] += 1

    for i in range(k):
        if counters[i] > 0:
            mean[i] /= counters[i]
        else:
            mean[i] = kernel[np.random.randint(kernel.shape[0])]
    diff = np.linalg.norm(mean - previous_mean)
    iteration += 1
return history

```

Function: initial

The initial function initializes the means (cluster centers) for the K-means algorithm.

- Input:

-kernel: A 2D array representing the data points.

-k: The number of clusters.

- Process:

-n is the number of data points, determined by the first dimension of kernel.

-Randomly selects k distinct indices from the data points to serve as the initial cluster centers.

-Extracts these points from kernel to form the initial cluster means.

- Output:

-mean: A 2D array containing the initial cluster centers.

Function: k_means

-The k_means function implements the K-means clustering algorithm.

- Input:

-kernel: A 2D array where each row represents a data point.

k: The number of clusters.

-max_iterations: The maximum number of iterations allowed (default: 100).

- Initialization:

-Calls initial to set the initial cluster means.

-Sets diff to 10 (a large initial value to ensure the loop starts).

-Initializes history to record the clustering assignments at each iteration.

-Sets iteration counter to 0.

- Iterative Process:

-The algorithm iterates until either the change in cluster means (diff) is smaller than 10^{-6} , or the maximum number of iterations is reached.

- E-step:

-For each data point, calculates its distance to all cluster centers using vectorized operations for efficiency.

-Assigns the data point to the nearest cluster.

- M-step:

-Updates the cluster means by computing the average of all points assigned to each cluster.

-Handles empty clusters (clusters with no assigned points) by randomly selecting a point from the dataset as a new cluster center.

-Tracks the movement of cluster centers by computing the Euclidean norm of the difference between the current and previous cluster means (diff).

- Output:

- history: A list where each element represents the cluster assignments for all data points at a specific iteration. This is also useful for visualization purposes.

spectral clustering[normalize cut] & spectral clustering[ratio cut]

```

def Laplacian(kernel, cut):
    W = kernel
    D = np.diag(np.sum(W, axis=1))
    L = D - W # ratio cut
    if cut == 0:
        D_sqrt_inv = np.diag(1/np.diag(np.sqrt(D)))
        L = D_sqrt_inv @ L @ D_sqrt_inv
    return L

def eigen(L, k, cut):
    eigval, eigvec = np.linalg.eig(L)

    sorted_index = np.argsort(eigval)
    U = eigvec[:, sorted_index[1: k+1]]
    if cut == 0:
        U /= np.sqrt(np.sum(np.power(U, 2), axis=1)).reshape(-1,1)
    return U

```

Function: Laplacian

-The Laplacian function computes the graph Laplacian matrix used in spectral clustering.

- Input:

-kernel: The adjacency matrix W , which represents the graph's edge weights between data points.

-cut: A flag indicating the type of cut (0 for normalized cut, otherwise for ratio cut).

- Process

-Constructs the degree matrix D , where each diagonal entry is the sum of weights connected to the corresponding node.

-Computes the Laplacian matrix $L = D - W$ for the ratio cut.

-If cut == 0, the normalized Laplacian matrix L_{sym} is calculated as: $L_{sym} = D^{-\frac{1}{2}} L D^{\frac{1}{2}}$
 Here, $D^{-\frac{1}{2}}$ is the diagonal matrix with elements $1/\sqrt{d_i}$ where d_i is the degree of node i .

- Output:

-Returns L , the Laplacian matrix (normalized or unnormalized, depending on cut).

Function: eigen

The eigen function computes the eigenvectors of the Laplacian matrix, which are then used for spectral clustering.

- Input:

-L: The Laplacian matrix (output from Laplacian).

-k: The number of clusters.

-cut: A flag for whether to normalize the eigenvectors (0 for normalized cut).

- Process:

-Computes all eigenvalues and eigenvectors of L

-Sorts the eigenvalues in ascending order and selects the eigenvectors corresponding to the smallest k non-zero eigenvalues.

-If cut == 0, applies normalization to the eigenvectors. Each row of the resulting matrix U is normalized as:

$$T_{ij} = \frac{u_{ij}}{\sqrt{\sum_k u_{ik}^2}}$$

This ensures the rows have unit length, which is critical for normalized cut.

- Output:

U : A matrix containing k eigenvectors as columns, optionally normalized. The rows may be normalized or unnormalized depending on the cut type.

Then, apply K-means (above mentioned in kernel k-means) clustering on the rows of U to form k clusters.

ii. Part 2

The code remains the same, except for changing the value of k to 3 or 4.

iii. Part 3

k-means++

The purpose of the k-means++ initialization method is to select the initial cluster centers in a way that improves the convergence rate and the final clustering results of the k-means algorithm. Unlike the standard k-means, which selects initial centers randomly, k-means++ selects the first center randomly but then chooses subsequent centers based on their distance from already chosen centers, ensuring that they are spread out.

```
def initial(kernel, k):
    n = kernel.shape[0]
    centers = []
    first_center = np.random.randint(n)
    centers.append(first_center)
```

```

    for _ in range(1, k):
        distances = np.min([np.linalg.norm(kernel - kernel[center], axis=1) ** 2 for center in centers], axis=0)
        probabilities = distances / np.sum(distances)
        next_center = np.random.choice(n, p=probabilities)
        centers.append(next_center)
    mean = kernel[centers, :]
    return mean

```

Improved cluster initialization: The centers are more spread out compared to the random selection, which helps the k-means algorithm converge faster and reduce the likelihood of poor local minima.

Selection based on distance: New centers are chosen based on their distance from already selected centers, ensuring that the selected points are not too close to each other.

Randomness with a bias: Although the selection of centers is still random, the probability distribution biases the selection towards points that are farther away from the already chosen centers.

iv. Part 4

```

def visualize_eigenspace(U, c, k, cc, image_number):

    U = np.real(U)

    c = np.real(c)
    if U.shape[1] > 2:
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        ax.scatter(U[:, 0], U[:, 1], U[:, 2], c=c, cmap='viridis')
        ax.set_title(f'Coordinates in the Eigenspace_spectral clustering (k={k})')
        ax.set_xlabel('dim 1')
        ax.set_ylabel('dim 2')
        ax.set_zlabel('dim 3')
        save_path = fr"C:\Users\yoush\Desktop\機器學習\HW6\png\part_d\image_{image_number}\spectral_clustering({cc})_k={k}_final_image.png"
        os.makedirs(os.path.dirname(save_path), exist_ok=True)
        plt.savefig(save_path)
        plt.show()
    else:
        plt.figure()
        colors = ['r', 'g', 'b', 'y', 'c', 'm', 'b']
        for i in range(k):
            cluster_points = U[c == i]
            plt.scatter(cluster_points[:, 0], cluster_points[:, 1], color=colors[i % len(colors)], label=f'Cluster {i}')

        plt.title(f'Coordinates in the Eigenspace_spectral cluster

```

```

ing (k={k})')
    plt.xlabel('dim 1')
    plt.ylabel('dim 2')
    plt.legend()
    save_path = fr"C:\Users\yoush\Desktop\機器學習\HW6\png\part
_d\image_{image_number}\spectral_clustering({cc})_k={k}_final_image.png"
    os.makedirs(os.path.dirname(save_path), exist_ok=True)
    plt.savefig(save_path)
    plt.show()

```

This code visualizes the coordinates in the eigenspace resulting from spectral clustering, with a distinction between 3D and 2D visualization based on the dimensionality of the eigenvectors.

- **3D Visualization:** If the eigenvectors' dimensionality exceeds 2, the code creates a 3D scatter plot using the first three eigenvectors. The points are color-coded based on their cluster assignments, with the color map 'viridis' used to represent different clusters. The plot includes a title and axis labels, and the visualization is displayed and saved to a specified path on the local machine.
- **2D Visualization:** If the eigenvectors are in 2D, the code generates a 2D scatter plot. The points are again color-coded based on their cluster assignments (with a color list ensuring a variety of colors for different clusters). A title and axis labels are added to the plot, and the visualization is both displayed and saved.

For both visualizations, the results are saved as PNG images in a structured directory, named according to the clustering method, cluster size, and experiment number.

2. Experiments settings and results & discussion

Since GIFs do not animate in PDFs, the last frame of each GIF will be used as a static image in the document. The remaining GIFs are stored in corresponding folders: part_a contains part_1, part_b contains part_2, and part_c contains part_3.

Set $\gamma_s = 0.001, \gamma_c = 0.001$ in the all codes.

Kernel k-means: Effective for separating clusters when spatial and color information are significantly distinct.

Spectral Clustering:

- **Normalize Cut:** Achieved better performance in evenly sized cluster scenarios.
- **Ratio Cut:** Favored cases where clusters varied in size but maintained internal consistency.

image_1



[k_means]

Kernel k-means demonstrated robustness when clusters were linearly inseparable in the original feature space. However, its performance highly depends on kernel matrix construction and the choice of hyperparameters. Misaligned settings resulted in overlaps between clusters.

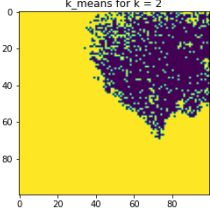
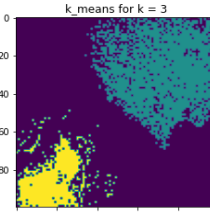
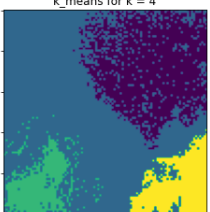
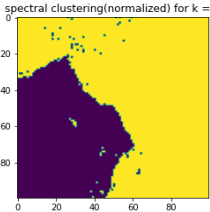
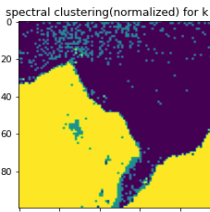
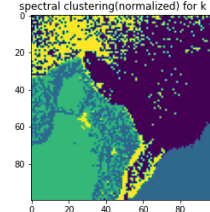
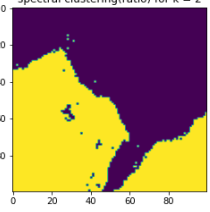
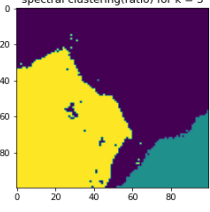
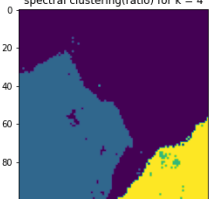
When the k value increased, the clustering boundaries became clearer, reflecting improved separation of data points into more distinct groups.

Spectral clustering's results varied based on the cut type:

- **Normalize Cut:** Improved when clusters were well-separated and comparable in size.

- **Ratio Cut:** Performed better in imbalanced scenarios, as it minimized inter-cluster weights relative to intra-cluster weights.

For **ratio cut**, the results showed minimal differences between $k=$ and $k=4$, indicating its stability and effectiveness in handling varying cluster counts.

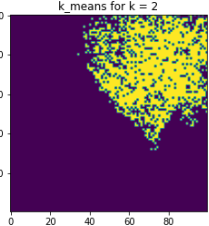
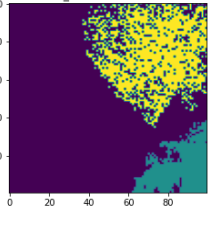
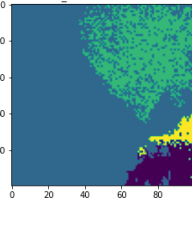
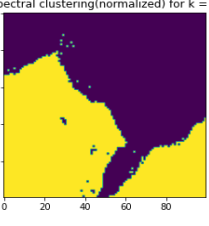
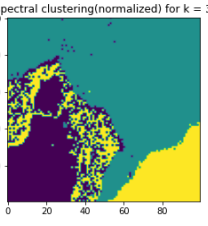
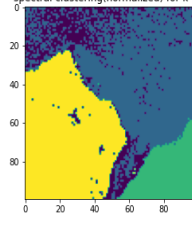
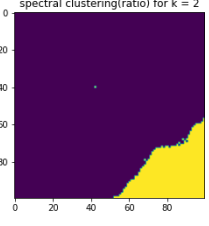
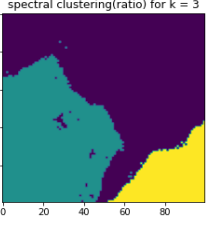
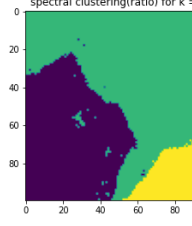
	k= 2	k = 3	k= 4
kernel k-means			
spectral clustering (normalize cut)			
spectral clustering (ratio cut)			

[k_means++]

The k-means++ initialization enhanced kernel k-means by mitigating sensitivity to initial cluster assignments. This reduced the chances of poor local minima and improved final cluster quality, particularly evident in $k=3$.

When k values increased, the initialization further helped by distributing initial centers across more diverse regions, leading to clearer and more distinct cluster boundaries.

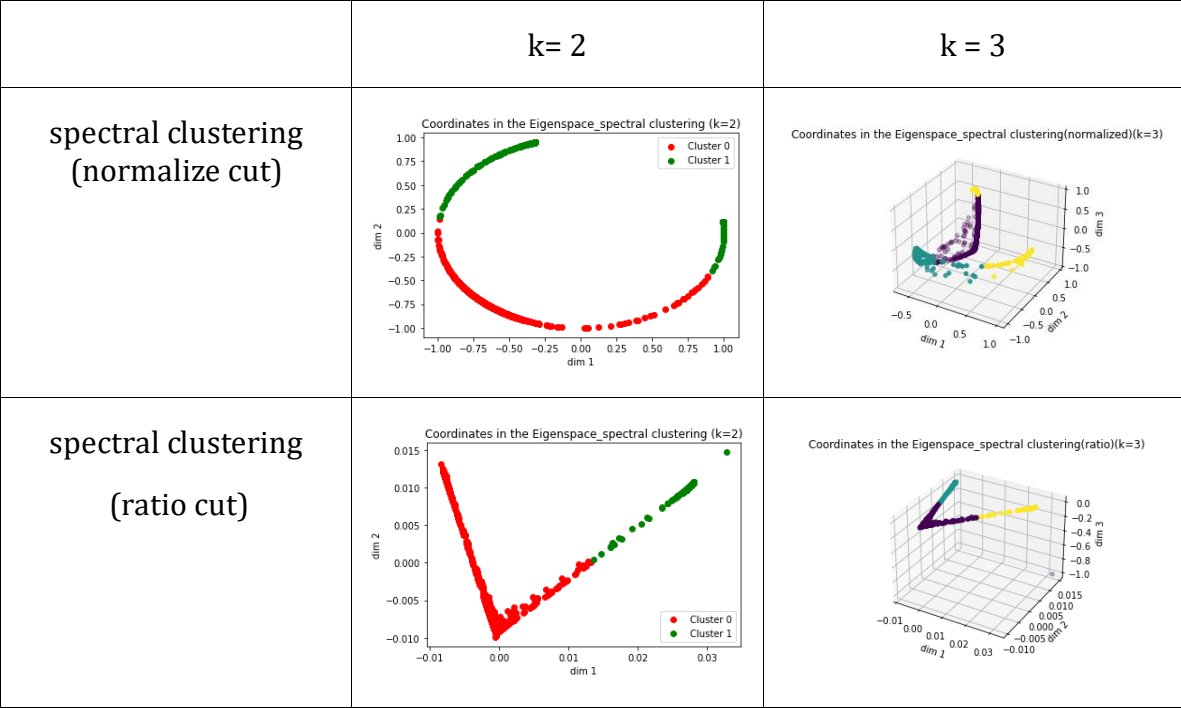
However, **k-means++** underperformed in kernel k-means compared to standard k-means. For instance, at $k=4$, the classification of the lower-left cluster was suboptimal, suggesting that kernel k-means required more precise initialization or adjustments to the kernel parameters to achieve comparable accuracy.

	k= 2	k= 3	k= 4
kernel k-means			
spectral clustering (normalize cut)			
spectral clustering (ratio cut)			

coordinates in the eigenspace [k_means++]

The eigenspace visualizations for spectral clustering provide insight into how data points are distributed across key dimensions:

- In **2D visualizations**, the clusters are well-separated and aligned with their respective groupings. This confirms the ability of the clustering algorithms to isolate major features within the dataset.
- In **3D visualizations**, the clusters appear to be projected along two dominant axes, suggesting that the third dimension contributes less variance. This aligns with observations that some data may be inherently low-dimensional despite embedding into higher-dimensional spaces.



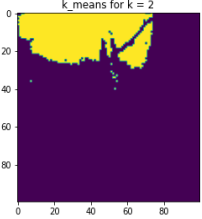
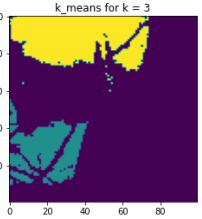
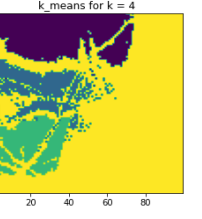
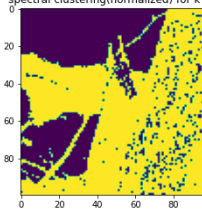
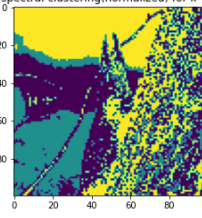
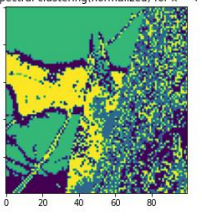
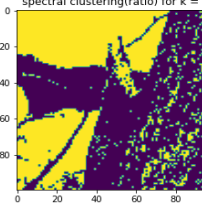
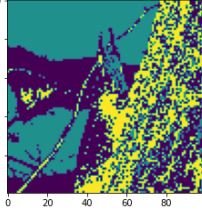
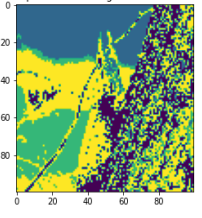
In **Image 1**, both methods are effective regardless of whether k=2 or k=3.

image_2



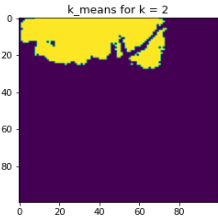
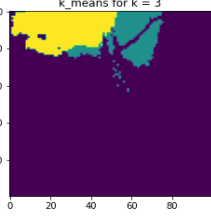
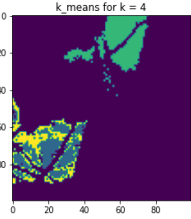
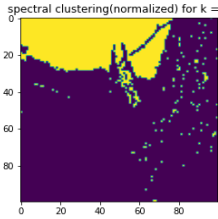
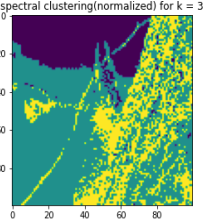
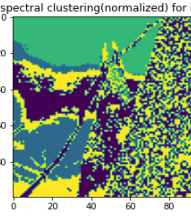
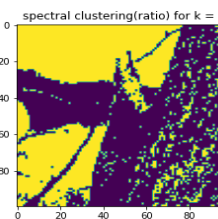
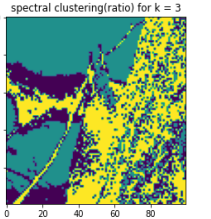
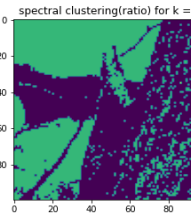
[k_means]

As the value of k increases, the cluster boundaries become clearer. In the case of spectral clustering (ratio cut), there is a noticeable difference between k=3 and k=4.

	k = 2	k = 3	k = 4
kernel k-means			
spectral clustering (normalize cut)			
spectral clustering (ratio cut)			

[k_means++]

As the value of k increases, the cluster boundaries become clearer. In the case of spectral clustering (ratio cut) using k-means++, there is no significant visual difference between k = 3 and k = 4.

	k= 2	k= 3	k= 4
kernel k-means			
spectral clustering (normalize cut)			
spectral clustering (ratio cut)			

coordinates in the eigenspace [k_means++]

The eigenspace visualizations for spectral clustering provide insight into how data points are distributed across key dimensions:

- In **2D visualizations**, the clusters are well-separated and aligned with their respective groupings. This confirms the ability of the clustering algorithms to isolate major features within the dataset.
- In **3D visualizations**, the clusters appear to be projected along two dominant axes, suggesting that the third dimension contributes less variance. This aligns with observations that some data may be inherently low-dimensional despite embedding into higher-dimensional spaces.

	k= 2	k= 3
spectral clustering (normalize cut)	<p>Coordinates in the Eigenspace_spectral clustering(normalized)(k=2)</p>	<p>Coordinates in the Eigenspace_spectral clustering(normalized)(k=3)</p>
spectral clustering (ratio cut)	<p>Coordinates in the Eigenspace_spectral clustering(ratio)(k=2)</p>	<p>Coordinates in the Eigenspace_spectral clustering(ratio)(k=3)</p>

In **spectral clustering (ratio cut)**, it can be observed that for $k=2$, one dimension has a value of 0, and for $k=3$, two dimensions have values of 0. This phenomenon is not observed in **spectral clustering (normalize cut)**.

The difference may arise due to the way the Laplacian matrix is normalized in the normalized cut approach. By normalizing the Laplacian, the eigenvalues and eigenvectors are adjusted, preventing zero values in the resulting dimensions. In contrast, the ratio cut uses the unnormalized Laplacian, which can lead to zero-valued dimensions depending on the data structure and connectivity.

3. Observations and discussion

i. Compare the performance between different clustering methods

✓ **Comparison for image1:**

In the k-means method, it is observed that when $k=2$, the classification results of spectral clustering (both normalize and ratio cuts) outperform the kernel k-means method, with spectral clustering (ratio cut) providing even better results. As k increases, each method can distinguish between more clusters. In the case of $k=4$, spectral clustering (normalize cut) offers a more refined division into smaller regions.

In the k-means++ method, when $k=2$, the best classification result shifts to spectral clustering (normalize cut). Even at $k=4$, spectral clustering (normalize cut) still provides the most accurate division.

✓ **Comparison for image2:**

In the case of k-means, spectral clustering (both normalize and ratio cuts) can distinguish detailed contours, with spectral clustering (ratio cut) providing even better results. On the other hand, kernel k-means only captures rough contours. As k increases, although kernel k-means can distinguish finer details, it still does not perform as well as the other two methods. Under these conditions, regardless of k , I believe spectral clustering (ratio cut) gives the best results.

In the case of k-means++, for $k=2$ and $k=3$, spectral clustering (ratio cut) still performs best, but at $k=4$, both spectral clustering (normalize and ratio cuts) provide good divisions.

ii. Compare the execution time of different settings

The execution time difference between k-means and k-means++ is notable. K-means++ typically requires slightly more time during the initialization phase due to its probabilistic selection of initial cluster centers. However, this results in better cluster initialization, leading to faster convergence and potentially fewer iterations compared to standard k-means. It can also be observed that kernel k-means executes faster than spectral clustering (both normalized cut and ratio cut) regardless of whether k-means or k-means++ is used. This is because spectral clustering requires additional computations for eigenvalues and eigenvectors, which

significantly affect its execution speed. Moreover, the runtime differences between the two methods of spectral clustering are minimal.

iii. Anything you want to discuss

I made changes by exploring other initialization methods and focusing on kernel k-means.

```
#1
def stratified_initialization(kernel, k):
    n = kernel.shape[0]
    indices = np.arange(n)
    np.random.shuffle(indices)
    partitions = np.array_split(indices, k)
    return kernel[np.array([np.random.choice(partition) for partition in
n partitions])]

#2
def density_initialization(kernel, k):
    densities = np.sum(kernel, axis=1)
    indices = np.argsort(densities)[-k:]
    return kernel[indices]

#3
def max_min_distance_initialization(kernel, k):
    n = kernel.shape[0]
    centers = [np.random.randint(n)]
    for _ in range(k - 1):
        distances = np.min(cdist(kernel[centers], kernel), axis=0)
        next_center = np.argmax(distances)
        centers.append(next_center)
    return kernel[centers]

def initial(kernel, k, init_method):
    if init_method == "stratified":
        return stratified_initialization(kernel, k)
    elif init_method == "density":
        return density_initialization(kernel, k)
    elif init_method == "max_min":
        return max_min_distance_initialization(kernel, k)
```

1. Stratified Initialization:

The stratified_initialization method shuffles the indices of the kernel and splits them into k partitions. One random point is selected from each partition to serve as the initial centroids.

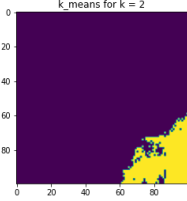
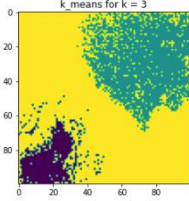
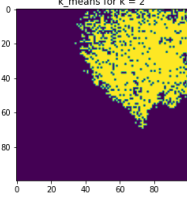
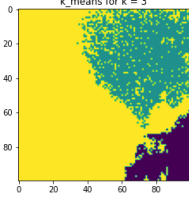
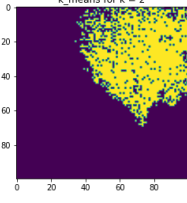
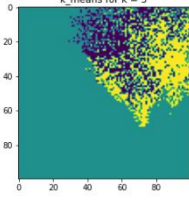
2. Density Initialization:

The `density_initialization` method calculates the density of each point by summing its values along the rows (axis 1). It then selects the k points with the highest densities as the initial centroids.

3. Max-Min Distance Initialization:

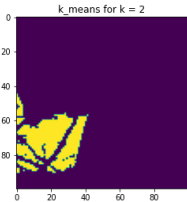
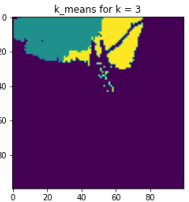
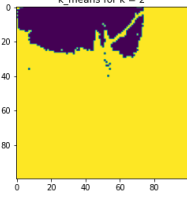
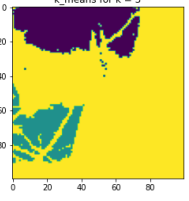
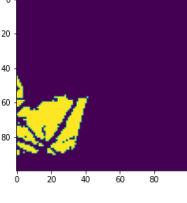
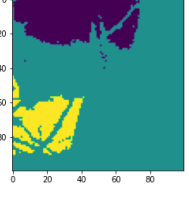
The `max_min_distance_initialization` method starts by randomly selecting one point. Then, for each subsequent centroid, it selects the point that is farthest from all already selected centroids, based on the minimum distance between points.

In the initial function, you can choose between these initialization methods by specifying the `init_method` argument, which defaults too "spectral".

Image1	$k = 2$	$k = 3$
Stratified Initialization:		
Density Initialization		
Max-Min Distance Initialization		

- In **Image 1**, when clustering into two groups ($k=2$), the classification results show minimal differences across the methods. Most methods group the data points in the upper-right region, while the **stratified** method uniquely places them in the lower-right region. This difference might arise from the stratified method's emphasis on balancing or prioritizing specific features during the clustering process.

When $k=3$, the **max-min** method exhibits a different clustering approach than others. Instead of following the broader cluster boundaries observed in other methods, it subdivides the already distinct regions into finer subgroups. This results in a more granular separation within the existing clusters, highlighting subtle differences that other methods may not capture.

Image2	$k = 2$	$k = 3$
Stratified Initialization:	 A 90x90 pixel plot showing a dark purple background with a yellow, branching shape in the lower-left corner. The plot is titled 'k_means for k = 2'.	 A 90x90 pixel plot showing a dark purple background with a yellow, branching shape in the lower-left corner. The plot is titled 'k_means for k = 3'.
Density Initialization	 A 90x90 pixel plot showing a dark purple background with a yellow, branching shape in the lower-left corner. The plot is titled 'k_means for k = 2'.	 A 90x90 pixel plot showing a dark purple background with a yellow, branching shape in the lower-left corner. The plot is titled 'k_means for k = 3'.
Max-Min Distance Initialization	 A 90x90 pixel plot showing a dark purple background with a yellow, branching shape in the lower-left corner. The plot is titled 'k_means for k = 2'.	 A 90x90 pixel plot showing a dark purple background with a yellow, branching shape in the lower-left corner. The plot is titled 'k_means for k = 3'.

- In **Image 2**, when clustering into two groups ($k=2$), the **density** method successfully distinguishes the animal's ears as a separate cluster. Meanwhile, the **stratified** and **max-min distance** methods focus on the branches in the lower-left corner as distinct clusters.

When $k=3$, both the **density** and **max-min distance** methods separate the animal's ears and the lower-left branches into distinct clusters. However, the **stratified** method takes a different approach, segmenting the background sky with varying colors as one cluster while distinguishing the animal's ears. This highlights stratified's tendency to prioritize differences in color or texture for segmentation.