Applied Machine Learning

# Homework 10

Anand Kamble
Department of Computer Science
Florida State University

---

## Spectral Clustering on Image Data

In this report, we perform spectral clustering on the image `scene2.jpg` to segment it into clusters based on pixel intensity and color similarity. The image has dimensions $82 \times 128 \times 3$, representing the height, width, and RGB channels, respectively.

### Approach and Implementation

#### Data Loading and Normalization

We use the `skimage.io` library to load the image and normalize pixel values by dividing by 255:

```
from skimage import io
image = io.imread('scene2.jpg') / 255.0
height, width, channels = image.shape
n_pixels = height * width
```

#### Constructing the Affinity Matrix

The affinity matrix $A$ is constructed as a sparse matrix to save memory and computation time. For each pixel, we consider its immediate neighbors and compute the affinity:

```
import numpy as np
import scipy.sparse as sp

sigma = 0.1
data = []
rows = []
cols = []

for i in range(height):
    for j in range(width):
        idx = i * width + j
        I_i = image[i, j, :]

        # Neighbor positions
        neighbors = []
        if i > 0:
            neighbors.append((i - 1, j))
        if i < height - 1:
            neighbors.append((i + 1, j))
        if j > 0:
            neighbors.append((i, j - 1))
        if j < width - 1:
            neighbors.append((i, j + 1))

        for ni, nj in neighbors:
            idx_neighbor = ni * width + nj
            I_j = image[ni, nj, :]
            weight = np.exp(-np.sum((I_i - I_j) ** 2) / sigma ** 2)
            data.append(weight)
            rows.append(idx)
            cols.append(idx_neighbor)
```

We then build the sparse affinity matrix $A$:

```
A = sp.coo_matrix((data, (rows, cols)), shape=(n_pixels, n_pixels))
A = (A + A.transpose()) / 2  # Ensure symmetry
```

### Spectral Clustering Using Sparse SVD

To perform spectral clustering, we compute the normalized Laplacian matrix $L_{\text{sym}}$ and then obtain the first $k$ singular vectors using sparse SVD:

```
from scipy.sparse.linalg import svds

# Compute degree matrix D
degrees = np.array(A.sum(axis=1)).flatten()
D_inv_sqrt = sp.diags(1.0 / np.sqrt(degrees))

# Compute normalized Laplacian L_sym
L_sym = sp.eye(n_pixels) - D_inv_sqrt @ A @ D_inv_sqrt

# Compute first k+1 singular vectors
u, s, vt = svds(L_sym, k=k+1, which='SM')
eigenvectors = u[:, :k]
```

### Clustering and Visualization

We use the $k$-means algorithm on the eigenvectors obtained from the SVD to cluster the pixels:

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=k, n_init=10)
labels = kmeans.fit_predict(eigenvectors)
label_image = labels.reshape((height, width))
```

The label image is displayed using a colormap for visualization.

### Reconstructing the Image with Mean Colors

For each cluster, we compute the mean RGB values and reconstruct the image:

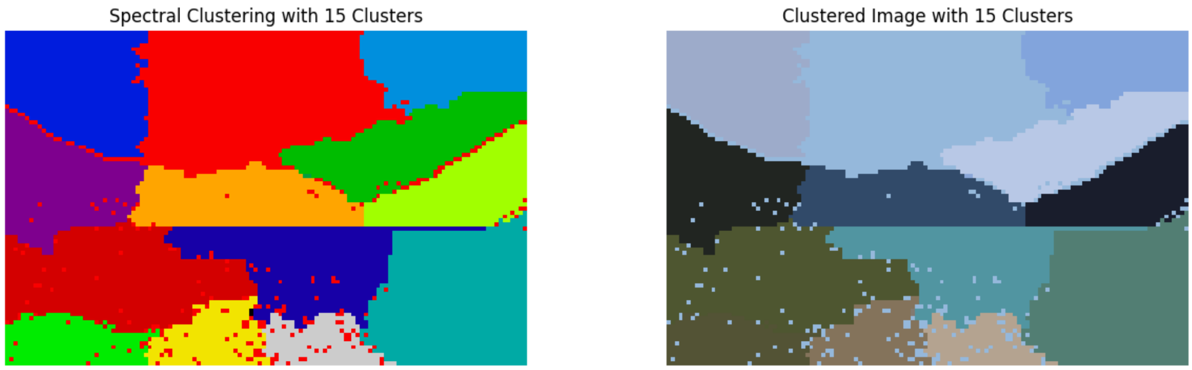
```
flat_image = image.reshape(-1, 3)
clustered_image = np.zeros_like(flat_image)

for cluster in range(k):
    mask = (labels == cluster)
    cluster_pixels = flat_image[mask]
    mean_color = cluster_pixels.mean(axis=0)
    clustered_image[mask] = mean_color

clustered_image = clustered_image.reshape((height, width, 3))
```

## Results

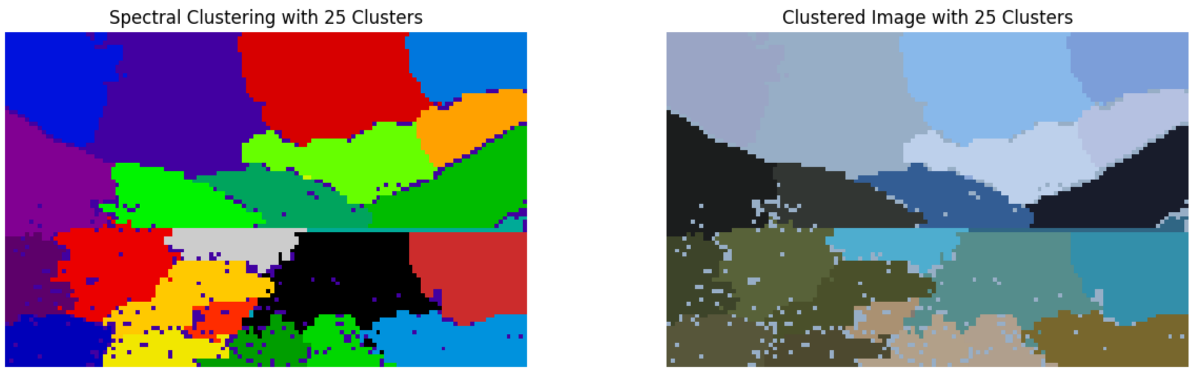
### Clustering with 15 Clusters




(a) Label Image with 15 Clusters (b) Reconstructed Image with Mean Colors

Figure 1: Spectral Clustering Results for $k = 15$

### Clustering with 25 Clusters




(a) Label Image with 25 Clusters (b) Reconstructed Image with Mean Colors

Figure 2: Spectral Clustering Results for $k = 25$

## Appendix: Code

```
import numpy as np
import scipy.sparse as sp
from scipy.sparse.linalg import eigsh
import matplotlib.pyplot as plt
from skimage import io
from sklearn.cluster import KMeans


def spectral_clustering(image_path, k, sigma=0.1):
    # Load image and normalize
    image: np.ndarray = io.imread(image_path) / 255.0
    height: int = image.shape[0]
    width: int = image.shape[1]
    channels: int = image.shape[2]
    n_pixels: int = height * width

    # Construct the affinity matrix A
    data: list = []
    rows: list = []
    cols: list = []

    for i in range(height):
        for j in range(width):
            idx: int = i * width + j
            I_i: np.ndarray = image[i, j, :]

            # Neighbor positions (left, right, up, down)
            neighbors: list = []
            if j > 0:
                neighbors.append((i, j - 1))
            if j < width - 1:
                neighbors.append((i, j + 1))
            if i > 0:
                neighbors.append((i - 1, j))
            if i < height - 1:
                neighbors.append((i + 1, j))

            for ni, nj in neighbors:
                idx_neighbor: int = ni * width + nj
                I_j: np.ndarray = image[ni, nj, :]
                weight: float = np.exp(-np.sum((I_i - I_j) ** 2) / sigma ** 2)
                data.append(weight)
                rows.append(idx)
                cols.append(idx_neighbor)

    # Build the sparse affinity matrix A
    A = sp.coo_matrix((data, (rows, cols)), shape=(n_pixels, n_pixels))
    A = (A + A.transpose()) / 2  # Ensure symmetry

    # Compute the normalized Laplacian
    degrees: np.ndarray = np.array(A.sum(axis=1)).flatten()
    degrees_sqrt_inv = 1.0 / np.sqrt(degrees)
    # Handle divisions by zero
    degrees_sqrt_inv[np.isinf(degrees_sqrt_inv)] = 0
    D_sqrt_inv = sp.diags(degrees_sqrt_inv)
    L_sym = sp.eye(n_pixels) - D_sqrt_inv.dot(A).dot(D_sqrt_inv)

    # Compute the first k+1 eigenvectors
    eigenvalues, eigenvectors = eigsh(L_sym, k=k+1, which='SM')
    eigenvectors = eigenvectors[:, 1:k+1]  # Skip the first eigenvector
```

```python
    # Perform k-means clustering
    kmeans: KMeans = KMeans(n_clusters=k, n_init=10)
    labels: np.ndarray = kmeans.fit_predict(eigenvectors)
    label_image: np.ndarray = labels.reshape((height, width))

    # Part a) Display the label image
    plt.imshow(label_image, cmap='nipy_spectral')
    plt.title(f'Spectral Clustering with {k} Clusters')
    plt.axis('off')
    plt.show()

    # Part b) Construct the clustered image
    flat_image: np.ndarray = image.reshape(-1, 3)
    clustered_image: np.ndarray = np.zeros_like(flat_image)

    for cluster in range(k):
        mask = (labels == cluster)
        cluster_pixels = flat_image[mask]
        mean_color = cluster_pixels.mean(axis=0)
        clustered_image[mask] = mean_color

    clustered_image = clustered_image.reshape((height, width, 3))

    # Display the clustered image
    plt.imshow(clustered_image)
    plt.title(f'Clustered Image with {k} Clusters')
    plt.axis('off')
    plt.show()


# Run spectral clustering with 15 clusters
spectral_clustering('scene2.jpg', k=15)

# Run spectral clustering with 25 clusters
spectral_clustering('scene2.jpg', k=25)
```