# CSE6740 HM1

yfan393 Fan

13 September 2024

## 1 Foundations in Probability, Linear Algebra, and Matrix Calculus

### (a) Probability

$$\begin{aligned}
E_y\left[E_x[x \mid y]\right] &= \int_y \left(\int_x x\, p(x \mid y)\, dx\right) p(y)\, dy \\
&= \int_x \int_y x\, p(x \mid y)\, p(y)\, dy\, dx \\
&= \int_x \int_y x\, p(x, y)\, dy\, dx \\
&= \int_x x\, p(x)\, dx \\
&= E[X]
\end{aligned} \tag{1}$$

$$\begin{aligned}
\mathrm{Var}(X) &= E[X^2] - (E[X])^2 \\
&= E_y\left[E_x[X^2 \mid y]\right] - (E[X])^2 \\
&= E_y\left[\mathrm{Var}_x(X \mid y) + (E_x[X \mid y])^2\right] - (E[X])^2 \\
&= E_y[\mathrm{Var}_x(X \mid y)] + E_y[(E_x[X \mid y]^2] - (E[X])^2 \\
&= E_y[\mathrm{Var}_x(X \mid y)] + E_y[(E_x[X \mid y]^2] - (E_y[E_x(X \mid y)])^2 \\
&= E_y[\mathrm{Var}_x(X \mid y)] + \mathrm{Var}_y E_x(X \mid y)
\end{aligned} \tag{2}$$

## (b) Pairwise Independence

I'll use tossing three coins as a counter example. All the possible events are listed as below:

| $\text{Coin}_1$ | $\text{Coin}_2$ | $\text{Coin}_3$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
| 1 | 1 | 1 |

Table 1: Tossing Three coins

$$\Pr(A) = \Pr\{\text{Coin}_1 = \text{Coin}_2\} = \frac{1}{2}$$

$$\Pr(B) = \Pr\{\text{Coin}_2 = \text{Coin}_3\} = \frac{1}{2}$$

$$\Pr(C) = \Pr\{\text{Coin}_3 = \text{Coin}_1\} = \frac{1}{2}$$

$$\Pr(A \cap B) = \Pr\{\text{Coin}_1 = \text{Coin}_2 = \text{Coin}_3\} = \frac{1}{4} = \Pr(A)\Pr(B)$$

$$\Pr(A \cap C) = \Pr\{\text{Coin}_1 = \text{Coin}_2 = \text{Coin}_3\} = \frac{1}{4} = \Pr(A)\Pr(C)$$

$$\Pr(B \cap C) = \Pr\{\text{Coin}_1 = \text{Coin}_2 = \text{Coin}_3\} = \frac{1}{4} = \Pr(B)\Pr(C)$$

Thus, events $A$, $B$, and $C$ are pairwise independent. However,

$$\Pr(A \cap B \cap C) = \Pr\{\text{Coin}_1 = \text{Coin}_2 = \text{Coin}_3\} = \frac{1}{4} \neq \Pr(A)\Pr(B)\Pr(C)$$

So, events $A$, $B$, and $C$ are not mutually independent.

## (c) Linear Algebra

An operator $M$ is idempotent if $M^2 = M$

### For $A^+A$:

$$(A^+A)(A^+A) = A^+(AA^+)A = A^+A$$

Moore-Penrose condition $AA^+A = A$. Thus, $A^+A$ is idempotent.

### For $AA^+$:

$$(AA^+)(AA^+) = A(A^+A)A^+ = AA^+$$

Moore-Penrose condition $A^+AA^+ = A^+$. Thus, $AA^+$ is idempotent.

When $A \in \mathbb{R}^{m\times n}$ has linearly independent columns, the matrix $A^TA \in \mathbb{R}^{n\times n}$ is invertible. The Moore-Penrose inverse for matrices with full column rank is:

$$A^+ = (A^TA)^{-1}A^T$$

$$A^+A = (A^TA)^{-1}A^TA = (A^TA)^{-1}(A^TA) = I_n$$

Thus, $A^+$ is a left inverse to $A$.

A right inverse satisfies $AA^+ = I_m$. For $A \in \mathbb{R}^{m\times n}$ with $m > n$, a right inverse does not generally exist because $AA^+$ typically does not equal the identity matrix $I_m$.

Let the singular value decomposition (SVD) of $A$ be: s

$$A = U\Sigma V^T$$

where:

- $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are orthogonal,
- $\Sigma \in \mathbb{R}^{m\times n}$ is diagonal with singular values $\sigma_1, \sigma_2, \ldots, \sigma_r$.

$$A^+ = V\Sigma^+U^T$$

where:

$$\Sigma^+ = \text{diag}\left(\frac{1}{\sigma_1}, \frac{1}{\sigma_2}, \ldots, \frac{1}{\sigma_r}\right)$$

## (d) Matrix and Vector Calculus

$$A(x)A^{-1}(x) = I_n$$

$$\frac{d}{dx}\left(A(x)A^{-1}(x)\right) = \frac{d}{dx}I_n = 0$$

$$\frac{dA(x)}{dx}A^{-1}(x) + A(x)\frac{dA^{-1}(x)}{dx} = 0$$

$$\frac{dA^{-1}(x)}{dx} = -A^{-1}(x)\frac{dA(x)}{dx}A^{-1}(x)$$

$$\frac{d}{dx}\left(A^{-1}(x)\right) = -A^{-1}(x)\frac{dA(x)}{dx}A^{-1}(x)$$

The Jacobian matrix $J_f(x) \in \mathbb{R}^{n\times m}$ is defined as:

$$J_f(x) = \frac{df(x)}{dx} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_m} \end{bmatrix}$$

Thus, the derivative $\frac{df(x)}{dx}$ is the Jacobian matrix, which is an $n \times m$ matrix.

By the chain rule, the derivative of $g(f(x))$ with respect to $x$ is given by:

$$\frac{d}{dx}g(f(x)) = \frac{\partial g}{\partial f} \cdot \frac{\partial f}{\partial x}$$

Thus, the total derivative of $g(f(x))$ with respect to $x$ is:

$$\frac{dg(f(x))}{dx} = J_g(f(x))J_f(x)$$

## 2 Maximum Likelihood

### (a) Exponential distribution

$$f(x \mid \lambda) = \begin{cases} \lambda e^{-\lambda x} & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

$$\ell(\lambda) = \log \prod_{i=1}^{n} f(x \mid \lambda) = \sum_{i=1}^{n} \log f(x \mid \lambda)$$

$$\ell(\lambda) = \begin{cases} \sum_{i=1}^{n} \log(\lambda e^{-\lambda x_i}) = \sum_{i=1}^{n} (\log \lambda - \lambda x_i) & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

$$\ell(\lambda) = n \log \lambda - \lambda \sum_{i=1}^{n} x_i$$

$$\frac{d\ell(\lambda)}{d\lambda} = \frac{n}{\lambda} - \sum_{i=1}^{n} x_i = 0$$

$$\lambda = \frac{n}{\sum_{i=1}^{n} x_i}$$

$$\lambda = \frac{1}{\bar{x}}$$

### (b) Pareto distribution

$$f(x \mid x_0, \theta) = \begin{cases} \theta x_0^{\theta} x^{-\theta-1}, & \text{if } x \geq x_0, \theta > 0 \\ 0, & \text{otherwise} \end{cases}$$

$$\ell(\theta) = \log \prod_{i=1}^{n} f(x \mid x_0, \theta) = \sum_{i=1}^{n} \log f(x \mid x_0, \theta)$$

$$\ell(\theta) = n \log \theta + n\theta \log x_0 - (\theta + 1) \sum_{i=1}^{n} \log x_i$$

$$\frac{d\ell(\theta)}{d\theta} = \frac{n}{\theta} + n \log x_0 - \sum_{i=1}^{n} \log x_i = 0$$

$$\theta = \frac{n}{\sum_{i=1}^{n} \log x_i - n \log x_0}$$

### (c) Poisson distribution

$$P(x = k \mid \lambda) = \frac{\lambda^k e^{-\lambda}}{k!}, \quad \forall k \in \{0, 1, 2, \dots\}$$

$$\ell(\lambda) = \log \prod_{i=1}^{n} f(x = k \mid \lambda) = \sum_{i=1}^{n} \log f(x = k \mid \lambda)$$

$$\ell(\lambda) = \left( \sum_{i=1}^{n} x_i \right) \log \lambda - n\lambda - \sum_{i=1}^{n} \log x_i!$$

$$\frac{d\ell(\lambda)}{d\lambda} = \frac{\sum_{i=1}^{n} x_i}{\lambda} - n = 0$$

$$\lambda = \frac{1}{n} \sum_{i=1}^{n} x_i = \bar{x}$$

## 3 Eigenvalues and Eigenvectors

### (a)

$$x_1 = \arg\max_{\|x\|=1} x^T M x$$

$$x_i = \arg\max_{\|x\|=1} x^T M x, \quad \text{subject to} \quad x^T x_j = 0, \quad j < i$$

$$M x_1 = \lambda_1 x_1$$

$$M x_{k+1} = \lambda_{k+1} x_{k+1}$$

### (b)

**Input:** $M \in \mathbb{R}^{n \times n}$, $\epsilon, \text{max\_iter}$

**Output:** $\Lambda = [\lambda_1, \dots, \lambda_n], V = [v_1, \dots, v_n]$
**Steps:**s

1. Initialize: $V = [\,], \Lambda = [\,]$

2. For each $i = 1$ to $n$:

   sInitialize random vector $v_i \leftarrow \frac{v_i}{\|v_i\|}$ Orthogonalize: $v_i \leftarrow v_i - (v_i^T v_j)v_j \quad$ for $j < i$, normalize $v_i$ Power iteration: Update $v_i \leftarrow Mv_i$, normalize $v_i$ after each iteration, repeat until $\|v_i^{(k+1)} - v_i^{(k)}\| < \epsilon$ Compute eigenvalue: $\lambda_i = v_i^T M v_i$ Store: Append $v_i$ to $V$, $\lambda_i$ to $\Lambda$s

3. Return: $\Lambda, V$

## 4 Clustering

### (a)

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|x_n - \mu_k\|^2$$

$$\frac{dJ}{d\mu_k} = (-2) \sum_{n=1}^{N} r_{nk}(x_n - \mu_k) = 0$$

$$\mu_k = \frac{\sum_{n=1}^{N} r_{nk} x_n}{\sum_{n=1}^{N} r_{nk}}$$

### (b)

Since:

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|x_n - \mu_k\|^2 > 0$$

it's bounded below by 0.

For each iteration, we do two steps: (1) Assign each data point to its nearest cluster center to minimize J; (2) Update the cluster center for specified data sets to minimize J. These two steps guarantee that after each iteration J will decrease or at least remain the same. Consequently, after finite steps, J will converge to a local optimum.

## (c)

Average linkage is most likely to produce results similar to K-means. First, after assigning each data point to a cluster, K-means updates the cluster to the average position of the assigned data points, meaning it takes into account the overall distribution within each cluster. Similarly, average linkage merges groups by minimizing the average distance between clusters, which aligns well with K-means' approach.

In contrast, single linkage tends to merge elongated clusters, as it only considers the closest points between clusters. On the other hand, complete linkage produces very compact clusters by focusing on minimizing the maximum distance between clusters.

## (d)

No. the time complexity $O(n^3)$ and space complexity $O(n^2)$ of the hierarchical clustering algorithm are not optimal for the single linkage. The $O(n^3)$ time complexity comes from a quite exhaustive approach: a full pairwise distance matrix of $O(n^2)$ time complexity is calculated at the start and recalculated for n-1 merging steps before resulting in a single cluster that contains all data points. Consequently, the overall time complexity ends up with $O(n^3)$. Additionally, it requires significant memory allocated to store $n^2$ pairwise distance values, leading to a space complexity of $O(n^2)$.

Here's pseudocode for Single Linkage Hierarchical Clustering:

```
Dataset $D = \{x_1, x_2, \ldots, x_n\}$
Initialization:
1. Create a list of n clusters where each cluster contains a single point $x_i \in D$.
2. Compute the pairwise distance matrix $dist(i, j)$ for all points $x_i, x_j$.

While the number of clusters is greater than 1 do:
3. Find the two closest clusters $C_i$ and $C_j$ such that:

        For each pair of clusters (i, j):
        If this distance is less than mindist:
        Update mindist to this distance
        store (i, j) for merge
```

where $x_i \in C_i$ and $x_j \in C_j$.

4. Merge clusters $C_i$ and $C_j$ into a new cluster $C_{new}$:
Create a new cluster by combining clusters i and j Update the list of clusters by removing $C_i$ and $C_j$ and adding the new combined cluster $C_{new}$.
5. Update the distance matrix:
Create a new distance matrix with dimensions equal to the updated number of clusters.
For each pair of clusters (m, n): Compute the minimum distance between points in $C_m$ and $C_n$ Update the distance matrix with this minimum distance.
**End While**

**Return** the hierarchy of clusters.

By using a priority queue, we reduce the time complexity of hierarchical clustering to $O(n^2 \log n)$ while maintaining a space complexity of $O(n^2)$. This approach works for single, complete, and average linkage clustering.

Use a priority queue to store the pairwise distances between clusters efficiently, improving the time complexity:

**Time Complexity:**

- Initial distance computations: $O(n^2)$
- For each of the $n - 1$ merge steps:
  - Extracting the smallest distance from the priority queue: $O(\log n)$
  - Recalculating distances for the new cluster: $O(n)$

  Total time complexity: $O(n^2 \log n)$.

**Space Complexity:**

- Storing the distance matrix: $O(n^2)$
- Storing the priority queue: $O(n^2)$ Thus, the space complexity remains $O(n^2)$.

## 5 Programming: Image compression

### (1)

For K-medoids algorithm, I did several implementations:

- First K data points were chosen as initial medoids.
- Only Euclidean distance is tried due to the long run time.
- After assigning data points to medoids, the pairwise distances within each group are calculated, and the medoid with the lowest sum of distances to other points within the group is selected as the new medoid.
- When medoids selection doesn't change, the function will skip the iteration and print out the number of iterations.
- Finally, to avoid excessive computation, a maximum iteration limit of 200 is set.

### (2)

I attached a picture of a dog for test.

### (3)

Smaller values of K will produce more generalized clusters with less details. With larger values of K, the clusters may become more detailed and closer to the actual colors in the image, but too large a K might lead to over-fitting.

Smaller values of K generally converge faster since there are fewer medoids to update. Larger values of K may take more time due to the increased number of medoids and computations needed for distance calculations.

The number of iterations might increase with K, especially if the data is complex and requires more iterations to stabilize.

### (4)

Different initial medoids can lead to different final clusters because the algorithm might converge to different local minima. When I use sequential selection other than random selection, the result is not desirable.

The convergence time might vary depending on the initial medoid selection. Poor initializations could lead to more iterations or slower convergence.

## (5)

K-medoids tends to produce worse results compared to K means. Also, it takes longer compared to K-means because it calculates the pairwise distance in each loop, which is a nightmare for large datasets.

# HW1_programming

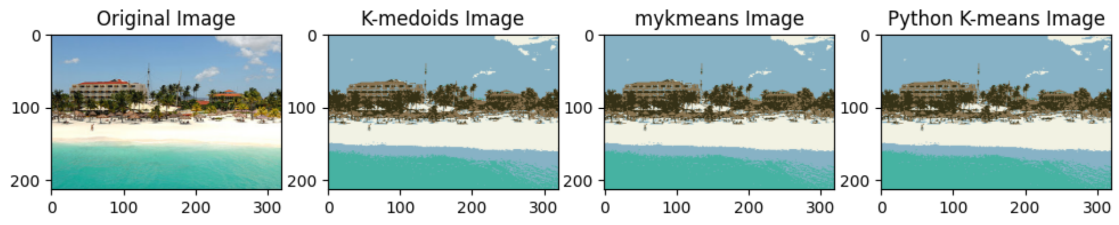
September 24, 2024

[1]:
```
!pip install imageio
!pip install scikit-learn
!pip install scipy
import numpy as np
import imageio.v2 as imageio
from matplotlib import pyplot as plt
from sklearn.cluster import KMeans
from scipy.sparse import csc_matrix
from scipy.spatial.distance import euclidean
from scipy.spatial.distance import cityblock
from scipy.spatial.distance import chebyshev
import scipy.sparse as sp
import sys
import os
import math


# Summarize the raw image data set features
# def summarize_array(array):
#     print("Image Data:")
#     print(array)
#     print("\nShape:", array.shape)
#     print("Size:", array.size)
#     print("Data type:", array.dtype)
#     print("Mean:", np.mean(array))
#     print("Min:", np.min(array))
#     print("Max:", np.max(array))
#     print("Standard deviation:", np.std(array))
#     print("Sum:", np.sum(array))
#     print("First element:", array.flat[0])

def mykmeans(pixels, K):
    # be careful, 3-D image data should be reshaped to 2-D pixels data before␣
 ↪input
    # Randomly initialize centroids within the data points;
    centers = pixels[np.random.choice(pixels.shape[0], K, replace=False),:]
    iterations = 200
    for iter in range(0, iterations):
        # norm squared of the centroids;
        squared_of_centers = np.sum(np.power(centers.T, 2), axis = 0, keepdims␣
 ↪= True)
        # in computing minimum Euclidean distance, we need to maximize the␣
 ↪cross term
        cross_term = (2 * np.dot(pixels,centers.T) - squared_of_centers)
        classes = np.argmax(cross_term, axis = 1)
        # update data assignment matrix; The assignment matrix is a sparse␣
 ↪matrix, with size n x K. only the entries specified will be 1, others zeros
        update_assignment_matrix = csc_matrix( (np.ones(pixels.shape[0]) ,(np.
 ↪arange(0,pixels.shape[0],1), classes)), shape=(pixels.shape[0], K) )
        # count the number of data points assigned to each cluster
        count = update_assignment_matrix.sum(axis=0)
        # assign new clusters based on average value
        centers = np.array((update_assignment_matrix.T.dot(pixels)).T / count).T
    return classes, centers


def mykmedoids(pixels, K):
    # be careful, 3-D image data should be reshaped to 2-D pixels data before␣
 ↪input
    medoids = pixels[np.random.choice(pixels.shape[0], K, replace=False),:]
    iterations = 200
    for iter in range(0, iterations):
        distances = np.array(compute_dist(pixels, medoids))
        classes = np.argmin(distances,axis=1)

        # after clustering data sets into K groups, update the medoids by␣
 ↪selecting one data point with lowest total distance with others within the␣
 ↪group
        new_medoids = np.zeros((K, pixels.shape[1]))
        for i in range(K):
            # Only select data points which belong to the same group
            cluster_points = pixels[np.where(classes == i)[0]]
            if len(cluster_points) > 0:
                # distances = np.array(compute_dist(cluster_points,␣
 ↪cluster_points)) # This way would compute too much out of memory             ␣
 ↪
                mean_point=np.mean(cluster_points,axis=0)
                distances = np.linalg.norm(cluster_points - mean_point, axis=1)
                new_medoids_index = np.argmin(distances)
                new_medoids[i]= cluster_points[new_medoids_index]

        # Check for convergence
        if np.all(medoids == new_medoids):
            print(f"Converged after {iter+1} iterations.")
            break

        medoids = new_medoids

    return classes, medoids


def compute_dist(X,medoids):
    N,D = X.shape
    K = medoids.shape[0]
    return np.sqrt(np.sum((np.reshape(X,[N,1,D])-np.
↪reshape(medoids,[1,K,D]))**2,axis=2))

def main():
    # Load the image files directly

    directory  = os.path.join(os.getcwd(), "content")  # Get the directory of␣
↪the script
    # Loop through all image files in the directory
    for filename in os.listdir(directory):
        if filename.endswith(('.png', '.jpg', '.jpeg', '.bmp', '.tiff', '.
↪gif')):
            image_file_name = os.path.join(directory, filename)
            K = 5  # You can adjust the number of clusters here

            im = np.asarray(imageio.imread(image_file_name))
            pixels = im.reshape(-1, 3) # reshaple the 3-D data to 2-D data set


            # Apply K-medoids
            classes, centers = mykmedoids(pixels, K)
            mykmedoids_im = np.asarray(centers[classes].reshape(im.shape), im.
↪dtype)
            imageio.imwrite(os.path.basename(os.path.
↪splitext(image_file_name)[0]) + '_converted_mykmedoids_' + str(K) + os.path.
↪splitext(image_file_name)[1], mykmedoids_im)

            # Apply K-means
            classes, centers = mykmeans(pixels, K)
            mykmeans_im = np.asarray(centers[classes].reshape(im.shape), im.
↪dtype)
            imageio.imwrite(os.path.basename(os.path.
↪splitext(image_file_name)[0]) + '_converted_mykmeans_' + str(K) + os.path.
↪splitext(image_file_name)[1], mykmeans_im)
```

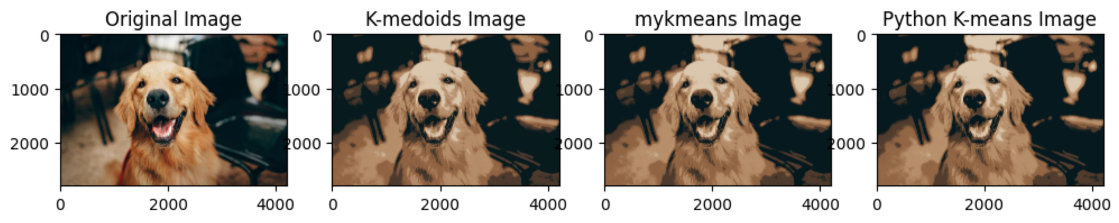
```
            # Directly call python function "kmeans"
            kmeans = KMeans(n_clusters=K, random_state=0)
            kmeans.fit(pixels)
            inertia = kmeans.inertia_ # sum of squared distances of samples to␣
↪their closest cluster center
            iterations = kmeans.n_iter_ # The number of iterations the␣
↪algorithm ran before convergence

            centers = kmeans.cluster_centers_ # Get the cluster centers and␣
↪labels
            classes = kmeans.labels_
            replace_pixels_to_clusters = np.asarray(centers[classes].reshape(im.
↪shape), im.dtype) # Map each pixel to the nearest cluster center, converts␣
↪the data type of the array to uint8 ensures that the pixel values are in the␣
↪range [0, 255]
            pyfunc_image = replace_pixels_to_clusters.reshape(im.shape)  #␣
↪Reshape back to the original image shape

            fig, ax = plt.subplots(1, 4, figsize=(12, 6)) # Display the␣
↪original and segmented images
            ax[0].imshow(im)
            ax[0].set_title("Original Image")
            ax[1].imshow(mykmedoids_im)
            ax[1].set_title("K-medoids Image")
            ax[2].imshow(mykmeans_im)
            ax[2].set_title("mykmeans Image")
            ax[3].imshow(pyfunc_image)
            ax[3].set_title("Python K-means Image")
            plt.show()



if __name__ == '__main__':
    main()
```

```
Requirement already satisfied: imageio in
c:\users\yuanting\appdata\local\programs\python\python312\lib\site-packages
(2.35.1)
Requirement already satisfied: numpy in
c:\users\yuanting\appdata\local\programs\python\python312\lib\site-packages
(from imageio) (2.1.0)
Requirement already satisfied: pillow>=8.3.2 in
c:\users\yuanting\appdata\local\programs\python\python312\lib\site-packages
(from imageio) (10.4.0)
Requirement already satisfied: scikit-learn in
c:\users\yuanting\appdata\local\programs\python\python312\lib\site-packages
(1.5.2)
```

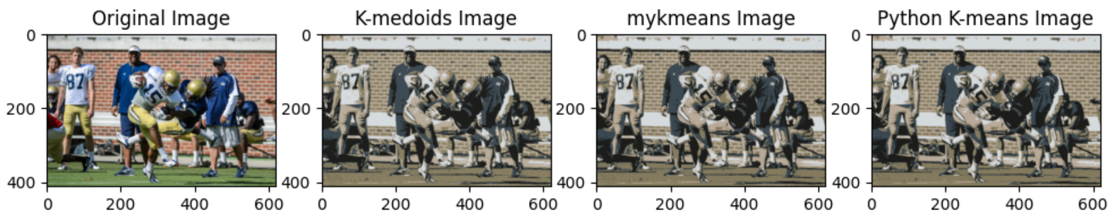
```
Requirement already satisfied: numpy>=1.19.5 in
c:\users\yuanting\appdata\local\programs\python\python312\lib\site-packages
(from scikit-learn) (2.1.0)
Requirement already satisfied: scipy>=1.6.0 in
c:\users\yuanting\appdata\local\programs\python\python312\lib\site-packages
(from scikit-learn) (1.14.1)
Requirement already satisfied: joblib>=1.2.0 in
c:\users\yuanting\appdata\local\programs\python\python312\lib\site-packages
(from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
c:\users\yuanting\appdata\local\programs\python\python312\lib\site-packages
(from scikit-learn) (3.5.0)
Requirement already satisfied: scipy in
c:\users\yuanting\appdata\local\programs\python\python312\lib\site-packages
(1.14.1)
Requirement already satisfied: numpy<2.3,>=1.23.5 in
c:\users\yuanting\appdata\local\programs\python\python312\lib\site-packages
(from scipy) (2.1.0)
Converged after 11 iterations.
```



```
Converged after 30 iterations.
```



```
Converged after 23 iterations.
```


Original Image
K-medoids Image
mykmeans Image
Python K-means Image
0
200
400
0
200
400
600