Assignment-6: K-Means Clustering

Rezwan-Ul-Alam (ID: 2011659042)
Md. Nur Alam Jowel (ID: 2012355042)
Raian Ruku (ID: 2013409642)

Email addresses:

rezwan.alam1@northsouth.edu
alam.jowel@northsouth.edu
raian.ruku@northsouth.edu

I. Introduction

K-Means clustering is a popular unsupervised machine learning algorithm used to partition a dataset into K clusters. Each cluster is represented by its centroid, and the objective is to minimize the variance within each cluster. In this assignment, we explores the application of K-Means clustering on a 2D dataset and its use in image compression. We will discuss the methodology, evaluation, results, and discussions derived from our experiments.

II. Method

The methodology for this assignment involves two main parts:

1. Clustering a 2D dataset.

2. Using K-Means for image compression.

III. Clustering a 2D Dataset

At first,We loaded a 2D dataset from a .npy file and applied the K-Means algorithm to cluster the data points. Here is below the steps are:

*A. **Data Loading***

a. First, we load the 2D data from a file. This data consists of points in a 2-dimensional space. Here is the code:

```
X = np.load("/content/drive/MyDrive/data/kmeans2d.npy")
print("First␣five␣elements␣of␣X␣are:\n", X[:5])
print('The␣shape␣of␣X␣is:', X.shape)
```

b. The first five elements and the shape of the dataset are printed to understand its structure and size.

```
First five elements of X are:
 [[1.84207953 4.6075716 ]
 [5.65858312 4.79996405]
 [6.35257892 3.2908545 ]
 [2.90401653 4.61220411]
 [3.23197916 4.93989405]]
The shape of X is: (300, 2)
```

B. Initialization

a. Randomly select initial centroids from the data points. This step is crucial as the choice of initial centroids can influence the final clusters. b. The function kmeans_initial_centroids randomly permute the dataset indices and selects the first K points

as the initial centroids. The code of the function is given below:

```
def kmeans_initial_centroids(X, K):
                randidx = np.random.permutation(X.shape[0])
               centroids = X[randidx[:K]]
               return centroids
```

IV. Iteration

1. Execute the K-Means algorithm for a fixed number of iterations.

Here is the code:

```
def find_closest_centroids(X, centroids):
    m = centroids.shape[0]
    n = X.shape[0]
    idx = np.zeros(X.shape[0], dtype=int)
    for i in range(n):
        distance = []
        for j in range(m):
            norm = np.linalg.norm(X[i] - centroids[j])
            distance.append(norm)
        idx[i] = np.argmin(distance)
    return idx

def compute_centroids(X, idx, K):
    m, n = X.shape
    centroids = np.zeros((K, n))
    for k in range(K):
        points = X[idx == k]
        centroids[k] = np.mean(points, axis=0)
    return centroids

def kmeans(X, initial_centroids, max_itr):
    m, n = X.shape
    K = initial_centroids.shape[0]
    centroids = initial_centroids
    previous_centroids = centroids
    idx = np.zeros(m)
    for i in range(max_itr):
        idx = find_closest_centroids(X, centroids)
        centroids = compute_centroids(X, idx, K)
    return centroids, idx
```

2. find_closest_centroids: For each data point, this function calculates the distance to each centroid and assigns the point to the nearest centroid. Here below the code:

```
def find_closest_centroids(X, centroids):

    m = centroids.shape[0]
    n = X.shape[0]
    idx = np.zeros(X.shape[0], dtype=int)
    for i in range(n):
          distance = []
          for j in range(m):
              norm = np.linalg.norm(X[i] - centroids[j])
              distance.append(norm)

          idx[i] = np.argmin(distance)
    return idx
```

3. compute_centroids: This function updates the centroids by calculating the mean position of all points assigned to each centroid. Here below the code:

```
compute_centroids(X, idx, K):
    m, n = X.shape
    centroids = np.zeros((K, n))

    for k in range(K):
          points = X[idx == k]
          centroids[k] = np.mean(points, axis = 0)
    return centroids
```

4. kmeans: This function repeatedly assigns points to the nearest centroid and recalculates centroids for a set number of iterations. Here below the code:

```
def kmeans(X, initial_centroids, max_itr):

    m, n = X.shape
    K = initial_centroids.shape[0]
    centroids = initial_centroids
    previous_centroids = centroids
    idx = np.zeros(m)
    for i in range(max_itr):
        idx = find_closest_centroids(X, centroids)
        centroids = compute_centroids(X, idx, K)
    return centroids, idx
```

V. Visualization

a. Plot the clustered data with centroids for different values of K.
b. We use matplotlib to create scatter plots of the clustered data points for different numbers of clusters (K=3 to K=6). Each plot shows the data points colored by their assigned cluster and the centroids marked with red 'x' symbols.

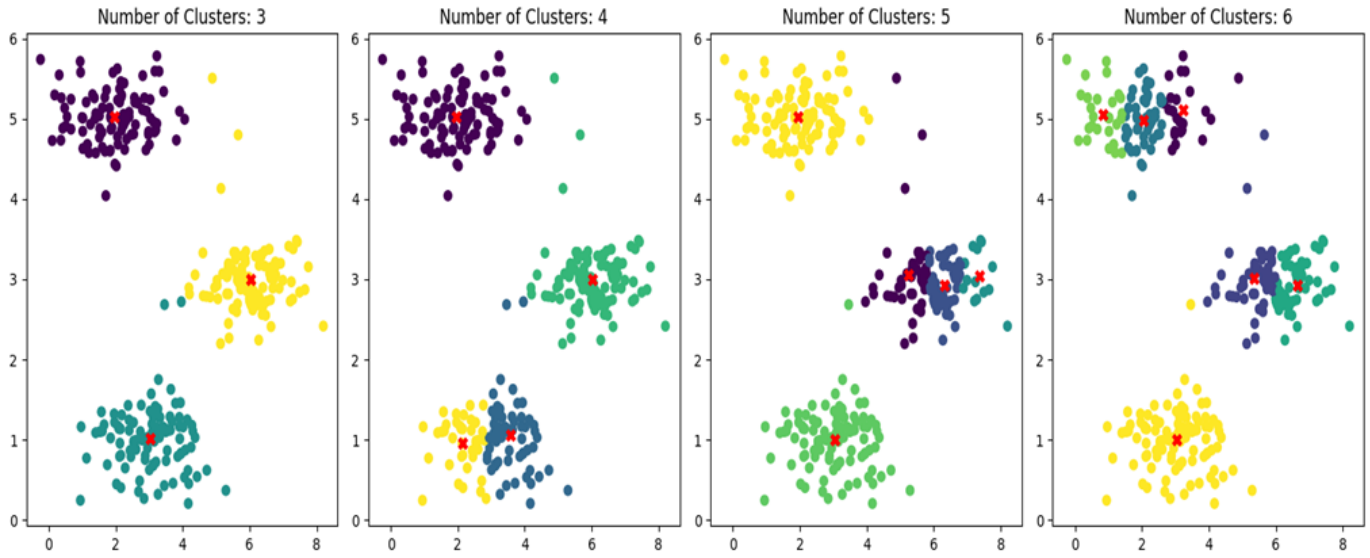
Here is the code:

```
maximum_clusters = 6
minimum_clusters = 3

fig, axs = plt.subplots(1, maximum_clusters - 2, figsize=(15, 5))
max_iteration = 10
#different numbers of clusters
for idx, clusters in enumerate(range(minimum_clusters, maximum_clusters + 1)):
initial_centroid = X[np.random.choice(range(len(X)), size=clusters)]
centroids, labels = kmeans(X, initial_centroid,max_iteration)
axs[idx].scatter(X[:, 0], X[:, 1], c=labels)
axs[idx].scatter(centroids[:, 0], centroids[:, 1], marker='x', c='r', linewidths=3)
axs[idx].set_title(f"Number of Clusters: {clusters}")

plt.tight_layout()
plt.show()
```

Here is the visualization output:



VI. Image Compression

Image compression using K-Means involves treating each pixel as a data point in a 3D space (RGB color space). Here below the steps:

A. Data Preparation

a. Load and preprocess the image by normalizing pixel values to the range [0, 1].
b. The image is loaded and displayed to understand its structure. We normalize the pixel values and reshape the image into a matrix where each row represents a pixel's RGB values.

Here is the code below:

```
# Divide by 255 so that all values are in the range 0 - 1
original_img = original_img / 255

# Reshape the image into an m x 3 matrix where m = number of pixels
# (in this case m = 128 x 128 = 16384)
# Each row will contain the Red, Green and Blue pixel values
# This gives us our dataset matrix X_img that we will use K-Means on.

X_img = np.reshape(original_img, (original_img.shape[0] * original_img.shape[1], 3))
X_img.shape
```

*B. **Initialization***

Randomly select initial centroids from the pixel values.

Code:

```
K = 8
max_iteration = 10

initial_centroids = kmeans_initial_centroids(X_img, K)

centroids, label = kmeans(X_img, initial_centroids, max_iteration)
```

*C. **Reconstruction***

Replace each pixel's color with its corresponding centroid's color to compress the image.

Code:

```
X_recovered = centroids[label, :]
X_recovered = np.reshape(X_recovered, original_img.shape)
```

*D. **Visualization***

a. Compare the original and compressed images.

b. The original and compressed images are displayed side by side for visual comparison. The compressed image uses a reduced color palette, effectively demonstrating the compression.

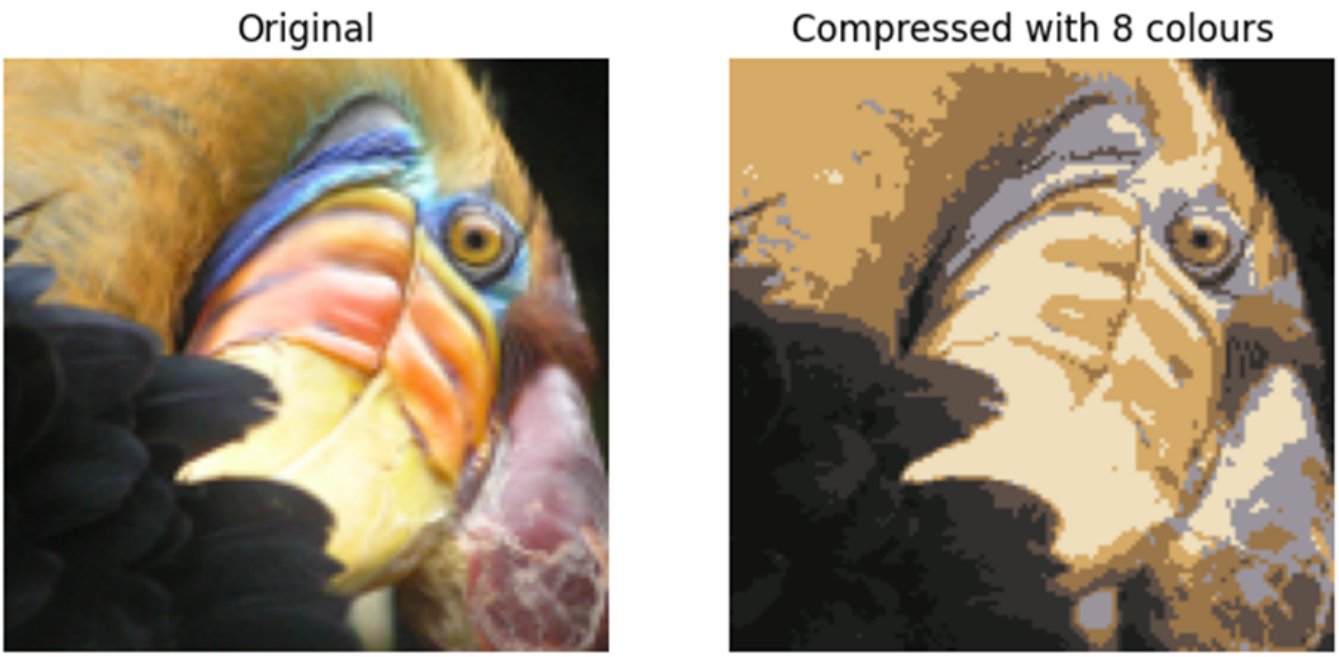
Here is the code below:

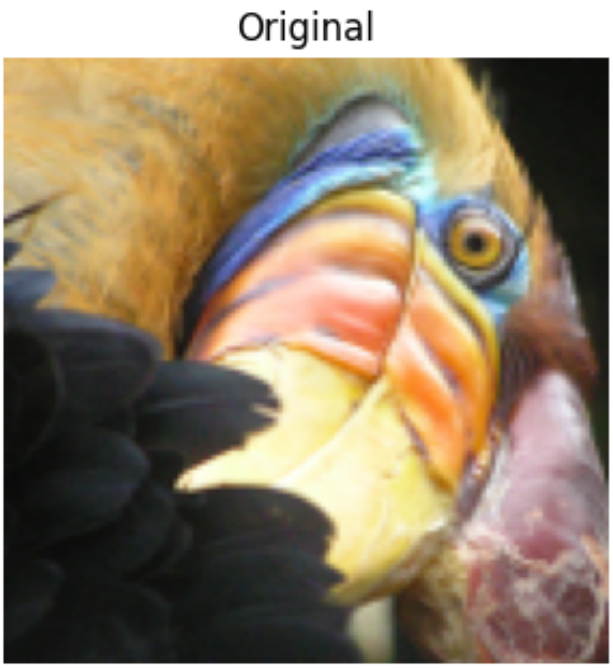
```
# Display original image
fig, ax = plt.subplots(1,2, figsize=(8,8))
plt.axis('off')

ax[0].imshow(original_img*255)
ax[0].set_title('Original')
ax[0].set_axis_off()

# Display compressed image
ax[1].imshow(X_recovered*255)
ax[1].set_title('Compressed with %d colours'%K)
ax[1].set_axis_off()
```

Here is the visualization output:





VII. Evaluation

A. *Clustering*

To evaluate the clustering results, we plotted the clustered data points and centroids for different values of K (from 3 to 6). This visual inspection helps us understand how well the data is partitioned.

B. *Image Compression*

For image compression, we evaluated the results by visually comparing the original and compressed images. The quality of compression is judged by how closely the compressed image resembles the original while reducing the number of colors.

VIII. Results

A. *2D Dataset Clustering*

Here below are the results of clustering the 2D dataset for different values of K:

1. K=3:

a. Three distinct clusters with clear separation.

b. Centroids are well positioned at the center of each cluster.

2. K=4:

a. Four clusters with good separation.

b. Centroids effectively represent each cluster.

3. K=5:

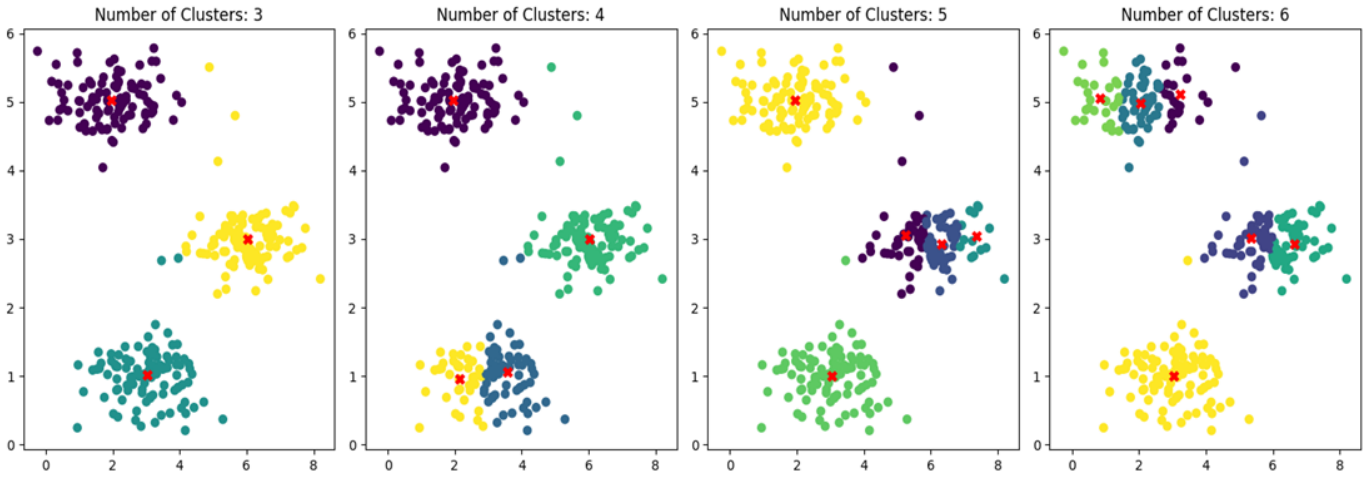
a. Five clusters with increasing granularity.

b. Centroids continue to accurately represent the clusters.

4. K=6:

a. Six clusters with more detailed partitioning.

b. Centroids maintain good cluster representation.



*B. **Image Compression***

The image compression results for K=8 is shown below:

*1) **Original Image**:*

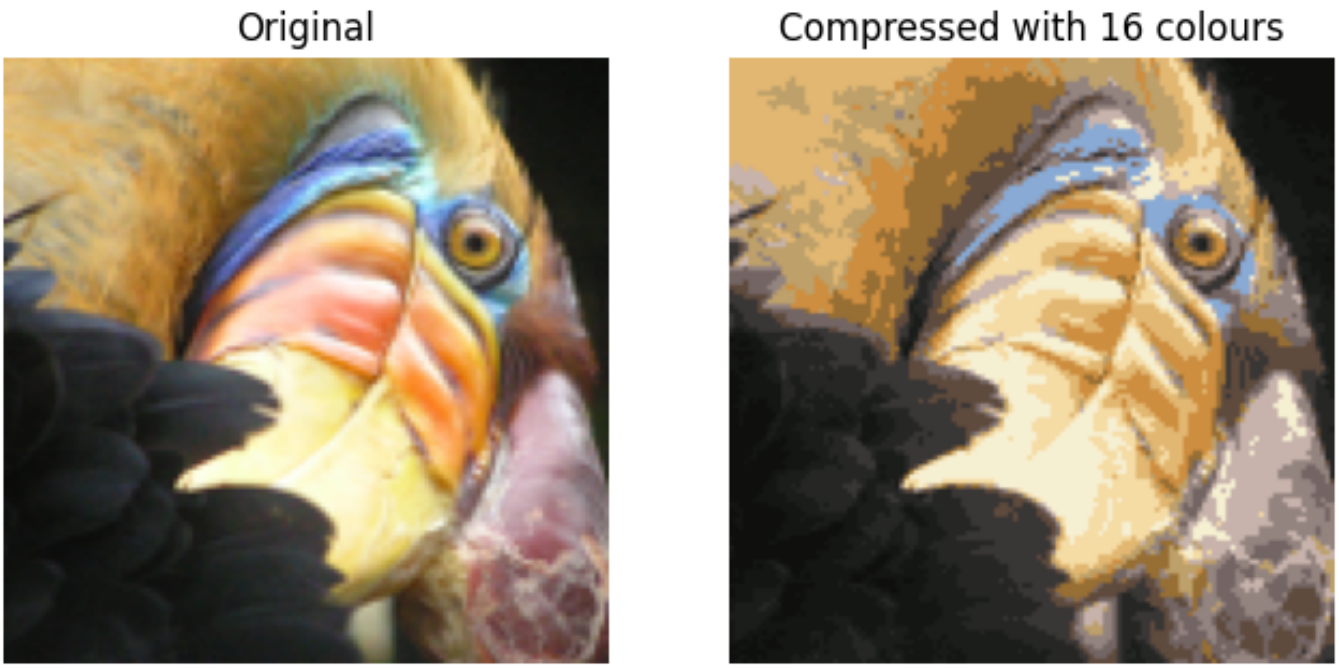
a. Full color range with 128x128 pixels.

b. Shape: (128, 128, 3).

*2) **Compressed Image:**:*

a. Reduced color palette with K=16.

b. Visual quality is high, closely resembling the original.



IX. Discussion:

1. The K-Means algorithm effectively clusters 2D data. For the 2D dataset, increasing the number of clusters provides finer granularity and better representation of data points.

2. In image compression, K-Means significantly reduces the number of colors while maintaining visual quality, demonstrating its utility in practical applications.