Instituto Tecnológico y de Estudios Superiores de Monterrey

ESCUELA DE INGENIERÍA Y CIENCIAS

Inteligencia Artificial Avanzada para la Ciencia de Datos I

Manual de Usuario - Implementación Manual

ML: K-Means para Compresión de Colores sobre Imágenes.

Presenta:

Miguel Ángel Pérez Ávila

Profesor:

Dr. Víctor Manuel de la Cueva Hernández

Sante Fe, Ciudad de México a 15 de Septiembre del 2025

Índice

Introducción

El presente documento presenta la implementación y metodología de ejecución de un programa escrito en Python que realiza procesos afines al algoritmo de machine learning no supervisado K Means para realizar clustering de datos. Las funciones implementadas incluyen:

- La inicialización aleatoria de Centroides.
- El cálculo y asociación de las muestras a un cluster según su distancia.
- El ajuste de ubicación para los k Centroides utilizados
- La ejecución del algoritmo completo para un número definido de iteraciones.

Implementación

Es importante mencionar que para le ejecución de este programa es necesario contar con las siguientes librerías en el entorno de ejecución y una versión de python igual o cercana a la utilizada para esta implementación siendo esta la 3.11.13:

- **numpy** (Manipulación de vectores y operaciones matemáticas en general).
- **matplotlib** (Graficación y visualización de datos).
- **cv2** (Lectura Vectorial de Imágenes).

En caso de no contar con alguna de las anteriores se pueden instalar y/o actualizar en bash mediante:

```
// En caso de utilizar conda
user@pc: conda activate <dev_env>

user@pc: pip install numpy matplotlib cv2
```

En el archivo de nombre KMeans.py se implementan las siguientes funciones utilizadas para este proyecto:

```
# Función para inicializar aleatoriamente K Centroides indicados.
#
# - Input:
#       - X : Vector de datos
#       - k : Número de clusters deseado
# - Return :
#       - centroids : list() -> lista de k centroides inicializados
def kMeansInitCentroids(X, k)

# Función para encontrar los centroides mas cercanos para cada punto.
#
# - Input:
#       - X : Vector de datos
#       - inital_centroids : Vector de centroides y su ubicación
```

```
# - Return :
#       - dataCentroidsIdx : list() -> lista de clusters asociadas a cada i ésima muestra
def findClosestCentroids(X, inital_centroids)

# Función para modificar la ubicación de los centroides utilizando la media de los puntos.
#
# - Input:
#       - X : Vector de datos
#       - idx : vector de relación entre los datos y el cluster al que se encuentra
asignado.
#       - K : número de k clusters
# - Return :
#       - newCentroids : list() -> Vector actualizado de centroides y su ubicación
def computeCentroids(X, idx, K)

# Función para ejecutar algoritmo kmeans.
#
# - Input:
#       - X : Vector de datos
#       - initial_centroids : Vector inicial de centroides y su ubicación
#       - max_iters : número iteraciones a ejecutar la actualización de centroides
#       - drawCentroids : bool para gráficar los puntos y la trayectoria de los centroides
# - Return :
#       - historyCentroidsCoords : list() -> Vector que contiene la trayectoria de la
ubicación de los centroides
#       - dataCentroidsIdx : list() -> vector de relación entre los datos y el cluster al
que se encuentra asignado.
def runkMeans(X, initial_centroids, max_iters, drawCentroids = True)
```

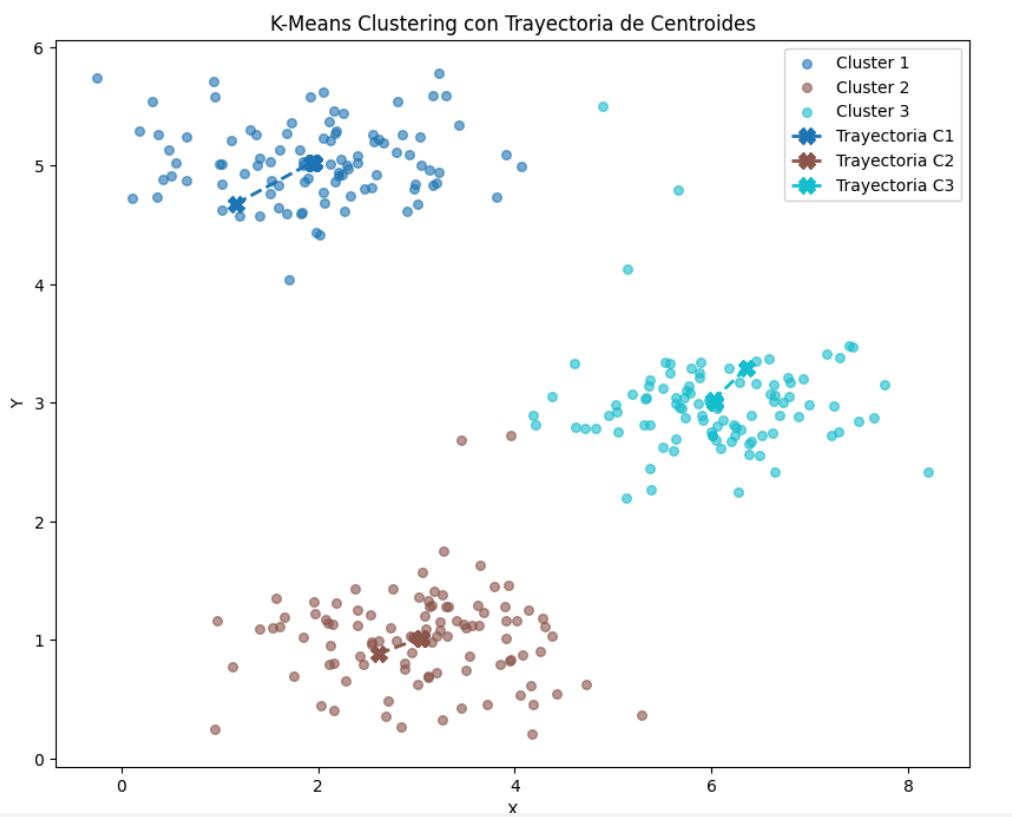
Ejemplos de Ejecución

A continuación se muestran dos ejemplos de ejecución para describir la manera en la que se deben utilizar las funciones anteriormente mencionadas utilizando los datos de prueba:

1. **Clustering de datos ex7data2.txt:** Para aplicar KMeans al conjunto de datos brindado "ex7data2.txt" se ejecutan las funciones anteriormente descritas de la siguiente manera:

```
# Importación de librerías
import numpy as np
from KMeans import *

# Lectura de archivo
file = open("../data/ex7data2.txt", "r")

# Vector para almacenar datos
pointsData = []

# Almacenar datos leidos del archivo
for line in file.readlines():
    aux = line[:-1].split(" ")
    del aux[0]
    for i in range(len(aux)):
        aux[i] = float(aux[i])
    pointsData.append(aux)

# Cast de list a np.array omitiendo los últimos dos valores del archivo de texto que parecen
ser
# saltos de línea
pointsData = np.array(pointsData[:-2])

# Definir K cluster deseados
k = 3

# Inicializar centroides de forma aleatoria
centroids = kMeansInitCentroids(pointsData, k)

# Ejecutar Kmeans y obtener el historial con la trayectoria de los centroides y las
asignaciones
# de clusters para cada dato. Se ingresan los centroides inicializados anteriormente
dataCentroidsIdx, historyCentroidsCoords = runkMeans(pointsData, centroids, 20,
drawCentroids=True)
```

- **Resultado:**



2. **Clustering aplicado a la compresión de colores en imágenes:** A continuación se presenta el código de muestra para aplicar KMeans sobre la imagen "brid_small.png" para realizar su compresión de calidad a los 16 colores más representativos:

```
# Importación de librerías
import cv2
import numpy as np
from KMeans import *

# Lectura de archivo con opencv
path = "../data/bird_small.png"
img_array = cv2.imread(path)

# Convertir matriz de BGR a RGB, ya que la lectura se hace BGR automáticamente
img_array = cv2.cvtColor(img_array, cv2.COLOR_BGR2RGB)

# Aplanar la matriz para obtener un vector de (128x128)x3
pointsData = matrix2Flat(img_array)

# Definir k clusters deseados
k = 16

# Inicializar Centroides aleatoriamente
```

```
centroids = kMeansInitCentroids(pointsData, k)

# Ejecutar Kmeans y obtener el historial con la trayectoria de los centroides y las
asignaciones
# de clusters para cada dato. Se ingresan los centroides inicializados anteriormente
dataCentroidsIdx, historyCentroidsCoords = runkMeans(pointsData, centroids, 60,
drawCentroids=False)

# Centroides obtenidos
color_centroids = np.array(historyCentroidsCoords[-1])
print("Centroides Obtenidos: ", color_centroids.shape)

compressedImage = []
# Remplazar puntos por centroide que generaliza su codificación RGB
for i in range(len(dataCentroidsIdx)):
    compressedImage.append(color_centroids[dataCentroidsIdx[i]])

# Nueva imagen
compressedImage = np.array(compressedImage)

# Redimensionar al tamaño original
compressedImage = np.reshape( compressedImage, shape=img_array.shape)

# Convertir a uint8 para rango válido 0-255
compressedImage = np.clip(compressedImage, 0, 255).astype(np.uint8)

# ----- Mostrar imagen analizada

# Crear un subplot con 1 fila y 2 columnas
fig, axes = plt.subplots(1, 2, figsize=(10, 5))

# Mostrar primera imagen
axes[0].imshow(img_array)
axes[0].set_title("Imagen Original")
axes[0].axis("off")

# Mostrar segunda imagen
axes[1].imshow(compressedImage)
axes[1].set_title("Imagen Compresa a K: "+ str(k) + " Colores")
axes[1].axis("off")

# Ajustar espacios y mostrar
plt.tight_layout()
plt.show()
```

- **Resultados:**

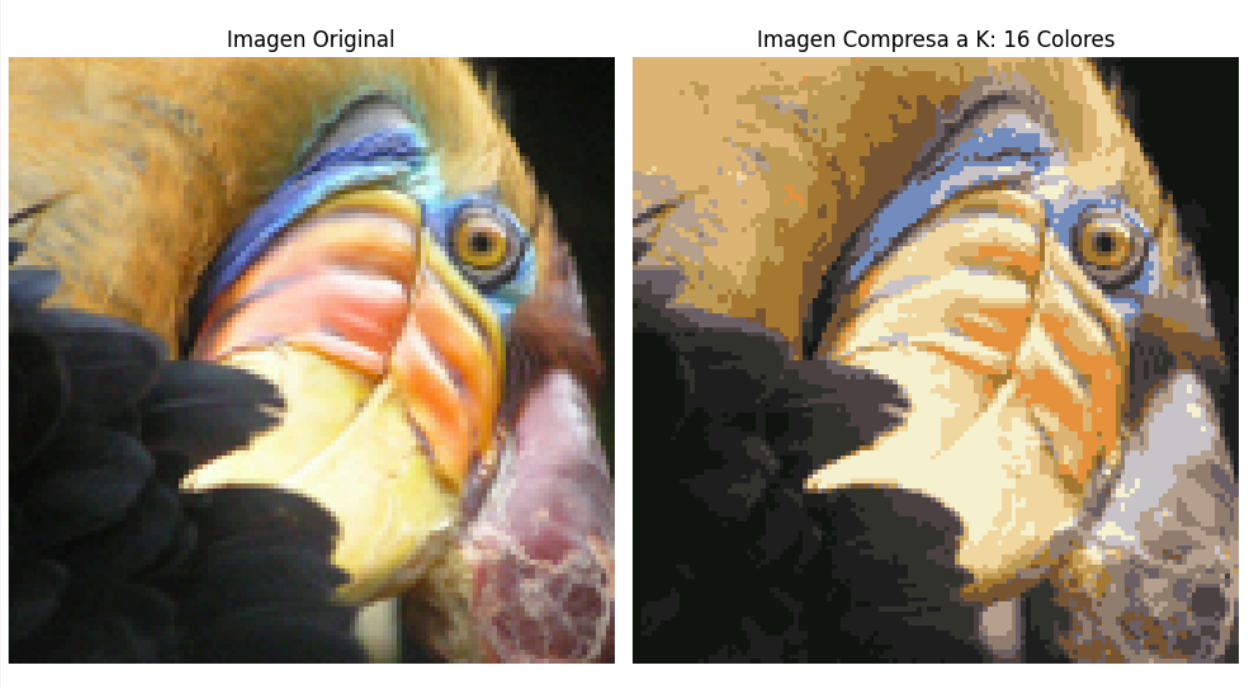


Imagen Original
Imagen Compresa a K: 16 Colores

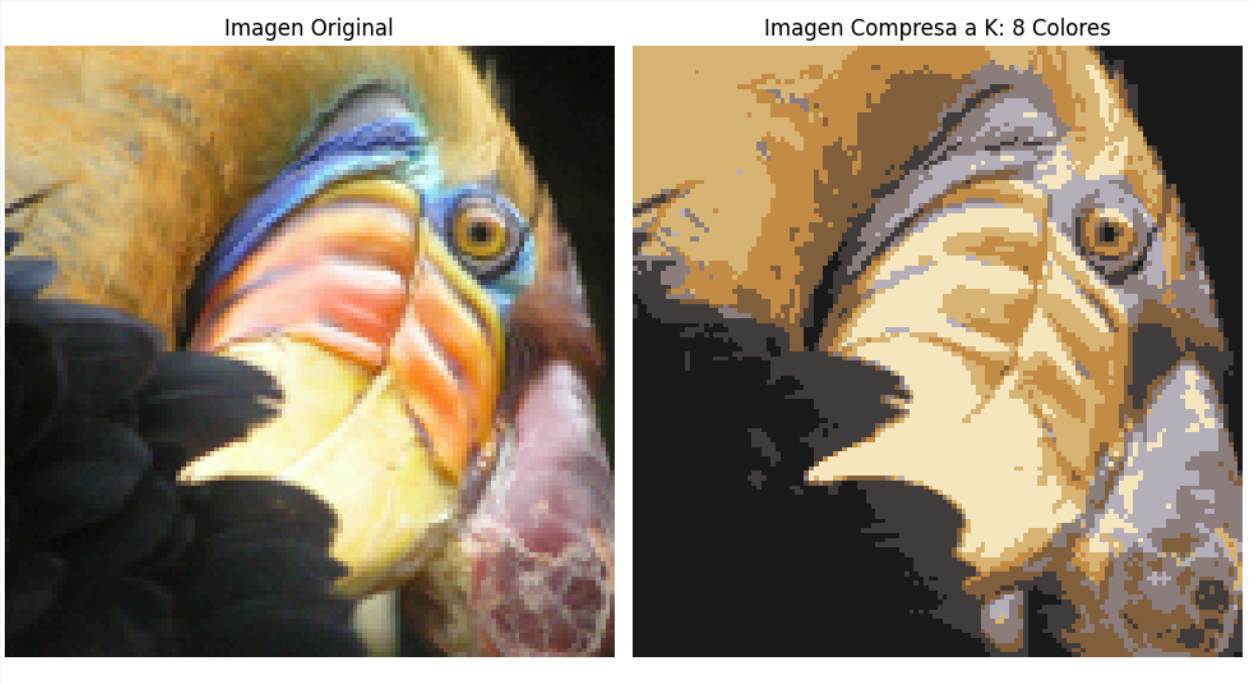
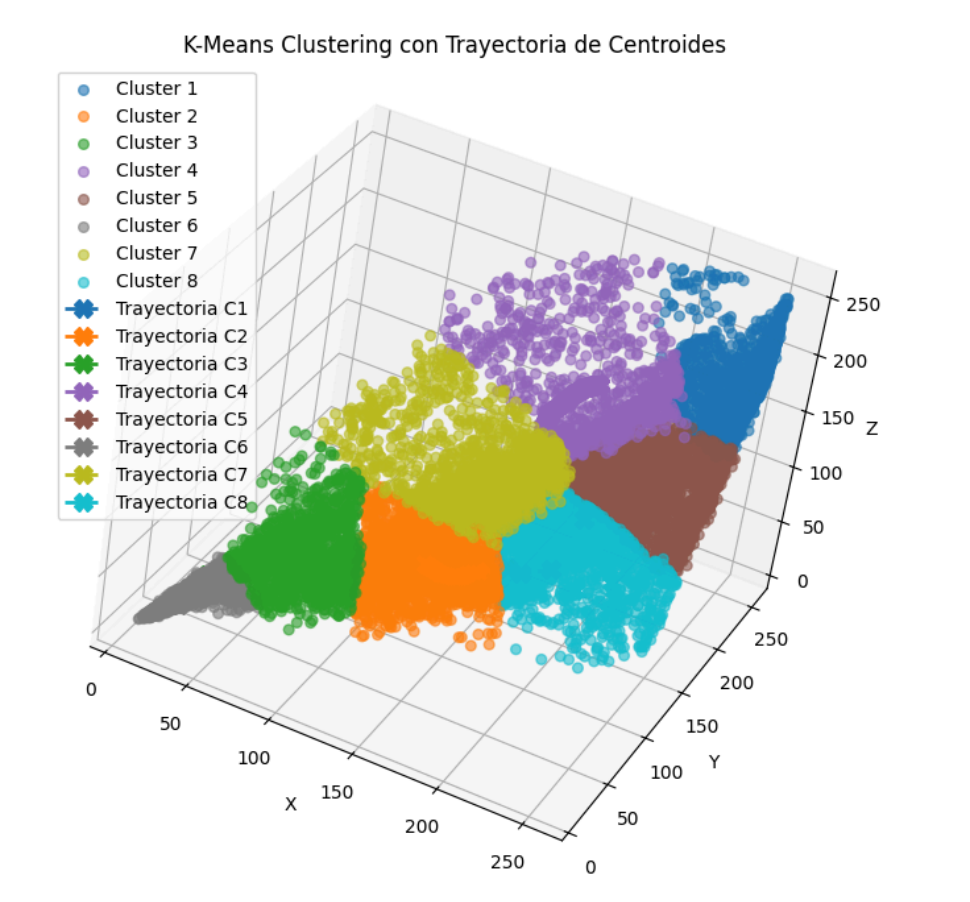


Imagen Original
Imagen Compresa a K: 8 Colores



Conclusión

El presente proyecto permitió implementar y comprender a profundidad el algoritmo de K-Means, un método no supervisado de clustering. La implementación desarrollada permitió manejar la inicialización de centroides, la asignación de puntos a los clusters más cercanos y el ajuste iterativo de los centroides mediante la media de los datos asignados, siguiendo la metodología tradicional del algoritmo.

Al aplicar K-Means a los datos del archivo ex7data2.txt, se observó que los puntos se agrupan efectivamente según la cercanía a los centroides, mostrando visualmente cómo los clusters se consolidan a lo largo de las iteraciones. Asimismo, la compresión de la imagen bird_small.png evidenció que K-Means puede reducir significativamente la cantidad de colores de una imagen manteniendo la esencia visual, generando una representación más compacta sin pérdida crítica de información perceptual.