INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Reporte Práctica 6 "Algoritmos de agrupación"

Integrantes: Cruz Barragan Ryan Nathanael - Rodríguez Vázquez Joshua Levi - Torres Abonce Luis Miguel

Correos: RNCB0963@gmail.com – joshualevirv@gmail.com – luiseishon9@gmail.com

I. Resumen

Dentro de la práctica se implementaron diversos algoritmos de agrupación (algoritmos de clustering), los cuales son métodos o técnicas de aprendizaje no supervisado que nos permiten dividir un conjunto de datos (clusters) en grupos. Los algoritmos desarrollados en MATLAB son: K-means, Mean Shift, Gaussian Mixture Models (GMM), Hierarchical Clustering y DBSCAN. El objetivo de estos algoritmos es buscar una optimización (maximización o minimización) entre objetos dentro o fuera de un cluster. Dentro de la práctica se muestra el funcionamiento de cada uno, así como sus características de manera gráfica con base a un conjunto aleatorio de datos.

II. Palabras clave

MATLAB, K-means, Mean Shift, Gaussian Mixture Models (GMM), Hierarchical Clustering, DBSCAN, cluster.

III. Introducción

Los algoritmos de agrupación (clustering) son métodos que trabajan mediante un aprendizaje no supervisado que trabajan en el ámbito del aprendizaje automático, se utiliza para agrupar un conjunto de objetos en clusters (subgrupos) con base a la similitud que tengan entre ellos. Su versatilidad permite identificar patrones y relaciones intrínsecas (no son evidentes a simple vista), facilita el análisis e interpretación de gran cantidad de datos no etiquetados. De esta manera, estos algoritmos pueden tener aplicaciones científicas (visión por computadora) y comerciales (segmentación de mercados).

El presente reporte pretende mostrar y analizar algunos de los algoritmos de agrupación implementados en el programa, los cuales en la actualidad son los más reconocidos y utilizados (K-means, Mean Shift, Gaussian Mixture Models (GMM), Hierarchical Clustering, DBSCAN). Cada uno tiene sus propias ventajas, desventajas, características y funcionamiento específico que permite tener un enfoque único y diferente para la agrupación de datos, que dependiendo de la situación, restricciones y especificaciones del problema se elige el más adecuado.

Algoritmo K-means. Tiene le propósito de agrupar conjuntos de datos en un número específico de clusters, se aplica en áreas como el análisis de imágenes, reconocimiento de patrones, marketing, etc.

Mean Shift. Es especialmente útil para identificar estructuras en conjuntos de datos complejos, sin contemplar el número de clusters a priori.

Gaussian Mixture Models (GMM). Se aplica para encontrar una representación probabilística de un conjunto de datos, tomando en cuenta que los conjuntos se generan mediante una combinación de varias distribuciones gaussianas.

Hierarchical Clusteringel. Se enfoca en que los objetos dentro del mismo cluster tengan más similitudes entre ellos, en comparación a otros que están en diferentes clusters. Se utiliza en áreas como la biología, marketing, minería de datos y reconocimiento de patrones.

DBSCAN. Destaca por su capacidad para identificar clusters de forma arbitraria y manejar el ruido en los datos, sin predefinir el número de clusters, mediante una dependencia de parámetros de densidad.

IV. Desarrollo

La implementación del programa se dividió en los cinco algoritmos antes mencionados, cada ejemplo mostrado en esta sección será definido por su pseudocódigo, así como su funcionamiento general.

A) K-means

El código desarrollado genera un conjunto de datos aleatorios compuesto por dos clusters mediante la adición de diferentes desviaciones y medias a dos conjuntos de puntos aleatorios. Luego, visualiza estos datos en un gráfico de dispersión. Posteriormente, aplica el algoritmo *K-means* para agrupar los

datos en dos clusters, obteniendo así los índices que indican a cuál cluster pertenece cada punto y las coordenadas de los centroides de los clusters. Finalmente, visualiza los clusters resultantes y los centroides en otro gráfico, diferenciando cada cluster con colores distintos y destacando los centroides con una marca especial.

Pseudocódigo

```
Inicio

1. GenerarDatosAleatorios()

    Fijar la semilla para reproducibilidad

    Generar datos aleatorios para dos clusters

2. VisualizarDatos(data)

    Crear una nueva figura

    Graficar los datos usando un diagrama de dispersión

    Establecer el título de la gráfica

3. DefinirNumeroDeClusters()

    Definir el número de clusters k como 2

4. AplicarKMeans(data, k)

    Aplicar el algoritmo K-means a los datos para obtener los índices de los
clusters y los centroides

5. VisualizarClusters(data, idx, centroids)

    Crear una nueva figura

    Graficar los datos coloreados por cluster

    Mantener la gráfica actual

    Graficar los centroides

    Establecer el título de la gráfica

    Añadir la leyenda

    Liberar la retención de la gráfica

Fin
```

B) Mean Shift

El código desarrollado genera un conjunto de datos aleatorios compuesto por dos clusters mediante la adición de diferentes desviaciones y medias a dos conjuntos de puntos aleatorios. Luego, visualiza estos datos en un gráfico de dispersión. Posteriormente, aplica el algoritmo Mean Shift, utilizando un ancho de banda definido, para agrupar los datos en clusters. Este proceso implica calcular las distancias euclidianas entre los puntos de datos y sus respectivos centros, iterativamente ajustando estos centros hasta que convergen. Finalmente, visualiza los clusters resultantes y los centroides en otro gráfico, diferenciando cada cluster con colores distintos y destacando los centroides con una marca especial.

Pseudocódigo

```
Inicio

1. GenerarDatosAleatorios()

    Fijar la semilla para reproducibilidad

    Generar datos aleatorios para dos clusters

2. VisualizarDatos(data)

    Crear una nueva figura

    Graficar los datos usando un diagrama de dispersión

    Establecer el título de la gráfica

3. DefinirParametroDeMeanShift()

    Definir el ancho de banda para Mean Shift

4. AplicarMeanShift(data, bandwidth)

    Inicializar variables para centros de clusters y asignaciones

    Para cada punto en los datos:

        Inicializar el centro actual como el punto de datos

        Mientras el centro actual cambie:

            Calcular distancias euclidianas entre el centro actual y todos los
puntos de datos

            Encontrar puntos dentro del ancho de banda

            Calcular el nuevo centro como la media de los puntos dentro del ancho
de banda

            Si el nuevo centro es igual al anterior, romper el bucle
```

```
            Actualizar el centro actual
        Si el centro actual no está en los centros de clusters:
            Añadir el centro actual a los centros de clusters
        Asignar el índice del centro de cluster al punto de datos
    Retornar centros de clusters y asignaciones
5. VisualizarClusters(data, assignments, cluster_centers)
    Crear una nueva figura
    Graficar los datos coloreados por cluster
    Mantener la gráfica actual
    Graficar los centroides
    Establecer el título de la gráfica
    Añadir la leyenda
    Liberar la retención de la gráfica
Fin
```

C) Gaussian Mixture Models (GMM)

El código desarrollado genera un conjunto de datos aleatorios compuesto por dos clusters mediante la adición de diferentes desviaciones y medias a dos conjuntos de puntos aleatorios. Luego, visualiza estos datos en un gráfico de dispersión. Posteriormente, ajusta un modelo de mezcla gaussiana (GMM) con dos componentes a los datos. Con el modelo ajustado, genera nuevas muestras basadas en la distribución del modelo. Finalmente, visualiza estas nuevas muestras en otro gráfico, diferenciando los datos generados y destacando los centroides del GMM con una marca especial.

Pseudocódigo

```
Inicio
1. GenerarDatosAleatorios()
    Fijar la semilla para reproducibilidad
    Generar datos aleatorios para dos clusters
    retornar data
2. VisualizarDatos(data)
```

```
    Crear una nueva figura
    Graficar los datos usando un diagrama de dispersión
    Establecer el título de la gráfica
3. AjustarModeloGMM(data, num_clusters)
    Ajustar un modelo de mezcla gaussiana (GMM) a los datos
    retornar gmm_model
4. GenerarMuestrasGMM(gmm_model, num_samples)
    Generar nuevas muestras basadas en el modelo ajustado
    retornar generated_data
5. VisualizarDatosGenerados(generated_data, gmm_model)
    Crear una nueva figura
    Graficar los datos generados usando un diagrama de dispersión
    Mantener la gráfica actual
    Graficar los centroides del modelo GMM
    Establecer el título de la gráfica
    Añadir la leyenda
    Liberar la retención de la gráfic
data = GenerarDatosAleatorios()
VisualizarDatos(data)
gmm_model = AjustarModeloGMM(data, 2)
generated_data = GenerarMuestrasGMM(gmm_model, 1000)
VisualizarDatosGenerados(generated_data, gmm_model)
Fin
```

D) Clustering Jerárquico

El código desarrollado genera un conjunto de datos aleatorios compuesto por dos clusters mediante la adición de diferentes desviaciones y medias a dos conjuntos de puntos aleatorios. Luego, calcula la matriz de enlace utilizando el método de enlace completo (complete linkage) para realizar un clustering jerárquico. A continuación, visualiza el dendrograma resultante del clustering jerárquico en una gráfica. Después, define el número de clusters y obtiene los índices de los clusters utilizando la matriz de enlace y el

número de clusters. Finalmente, visualiza los datos agrupados por los clusters obtenidos en otra gráfica, diferenciando cada cluster con colores distintos.

Pseudocódigo

```
Inicio

1. GenerarDatosAleatorios()

    Fijar la semilla para reproducibilidad

    Generar datos aleatorios para dos clusters

    retornar data

2. CalcularMatrizDeEnlace(data)

    Calcular la matriz de enlace usando el método de enlace completo

    retornar Z

3. VisualizarDendrograma(Z)

    Crear una nueva figura

    Graficar el dendrograma usando la matriz de enlace

    Establecer el título de la gráfica

4. DefinirNumeroDeClusters()

    Definir el número de clusters

    retornar k

5. ObtenerIndicesDeClusters(Z, k)

    Obtener los índices de los clusters usando la matriz de enlace y el número de
clusters

    retornar idx

6. VisualizarClusters(data, idx)

    Crear una nueva figura

    Graficar los datos agrupados por los clusters obtenidos

    Establecer el título de la gráfica

data = GenerarDatosAleatorios()

Z = CalcularMatrizDeEnlace(data)

VisualizarDendrograma(Z)

k = DefinirNumeroDeClusters()
```

```
idx = ObtenerIndicesDeClusters(Z, k)

VisualizarClusters(data, idx)

Fin
```

E) DBSCAN

El código desarrollado procesa una imagen realizando clustering K-means en sus píxeles para reducir la cantidad de colores. Primero, lee la imagen y la convierte a una matriz de doble precisión. Luego, convierte la imagen a una matriz de píxeles donde cada fila representa un píxel RGB. Después, define el número de clusters y aplica K-means a los píxeles de la imagen. Cada píxel se asigna al color del centroide más cercano. Finalmente, restaura la estructura original de la imagen y muestra tanto la imagen original como la procesada.

Pseudocódigo

```
Inicio

1. LeerImagen(ruta)
    Leer la imagen desde la ruta especificada
    retornar img
2. ConvertirADoblePrecision(img)
    Convertir la imagen a una matriz de doble precisión
    retornar img_double
3. ObtenerDimensiones(img_double)
    Obtener las dimensiones de la imagen (filas, columnas, canales)
    retornar filas, columnas
4. ConvertirAMatrizDePixeles(img_double, filas, columnas)
    Convertir la imagen a una matriz de píxeles (cada fila representa un píxel)
    retornar pixeles
5. DefinirNumeroDeClusters()
    Definir el número de clusters
    retornar num_clusters
6. AplicarKMeans(pixeles, num_clusters)
```

```
    Aplicar K-means a los píxeles de la imagen
    retornar indices_de_clusters, centroides
7. AsignarColores(pixeles, indices_de_clusters, centroides)
    Para cada píxel en la matriz de píxeles
        Asignar el color del centroide más cercano al píxel
    retornar pixeles_actualizador
8. RestaurarEstructuraDeImagen(pixeles_actualizados, filas, columnas)
    Restaurar la estructura de la imagen desde la matriz de píxeles actualizada
    retornar img_reconstruida
9. MostrarImagenes(img, img_reconstruida)
    Crear una nueva figura
    Mostrar la imagen original
    Mostrar la imagen procesada
img = LeerImagen('imagen.png')
img_double = ConvertirADoblePrecision(img)
filas, columnas = ObtenerDimensiones(img_double)
pixeles = ConvertirAMatrizDePixeles(img_double, filas, columnas)
num_clusters = DefinirNumeroDeClusters()
indices_de_clusters, centroides = AplicarKMeans(pixeles, num_clusters)
pixeles_actualizados = AsignarColores(pixeles, indices_de_clusters, centroides)
img_reconstruida = RestaurarEstructuraDeImagen(pixeles_actualizados, filas,
columnas)
MostrarImagenes(img, img_reconstruida)
Fin
```

V. Resultados

A) K-means

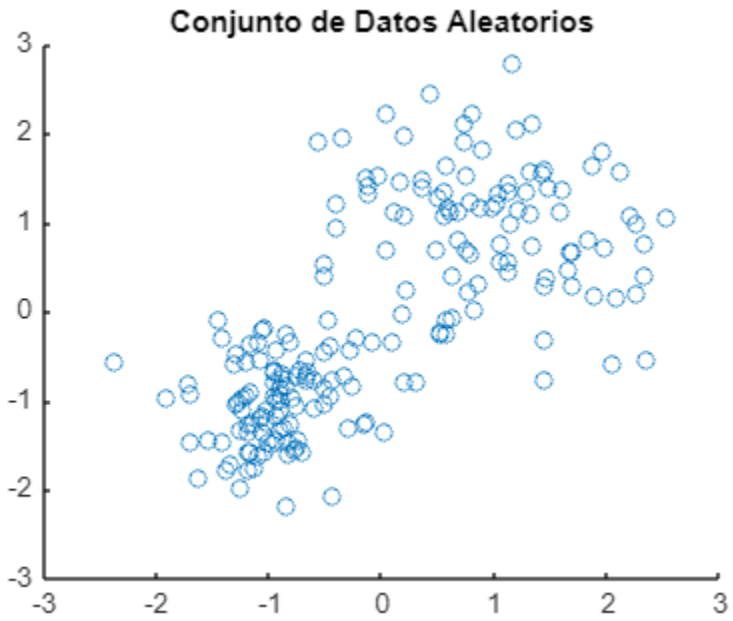

Figura 1. Conjunto de datos aleatorios del algoritmo K-means.

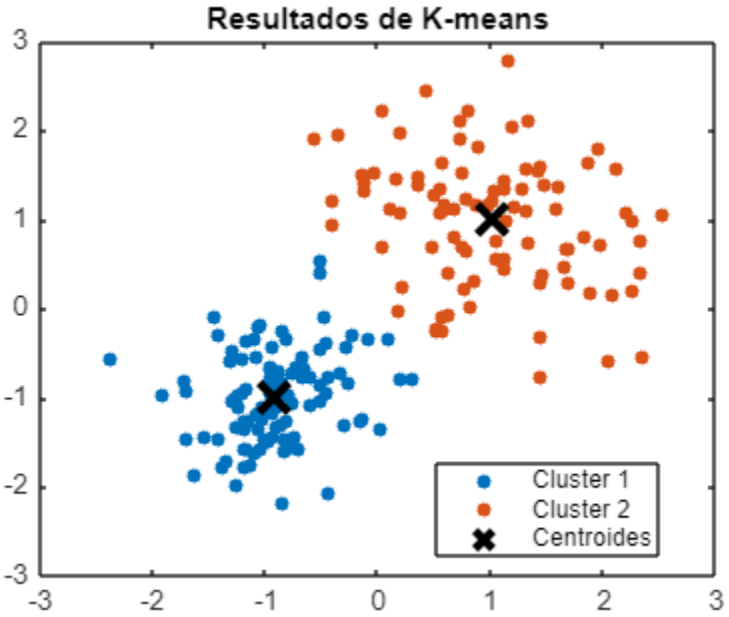

Figura 2. Resultados con el algoritmo K-means.

B) Mean Shift

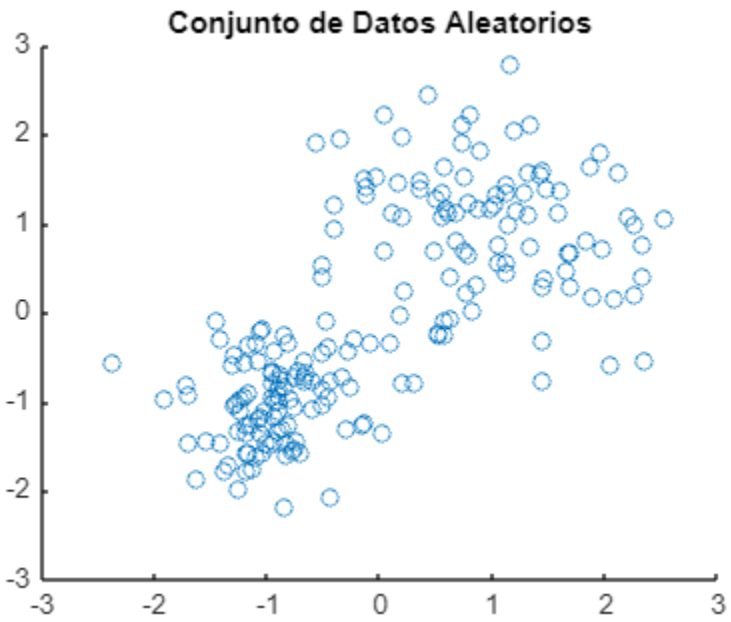

Figura 3. Conjunto de datos aleatorios del algoritmo Mean Shift.

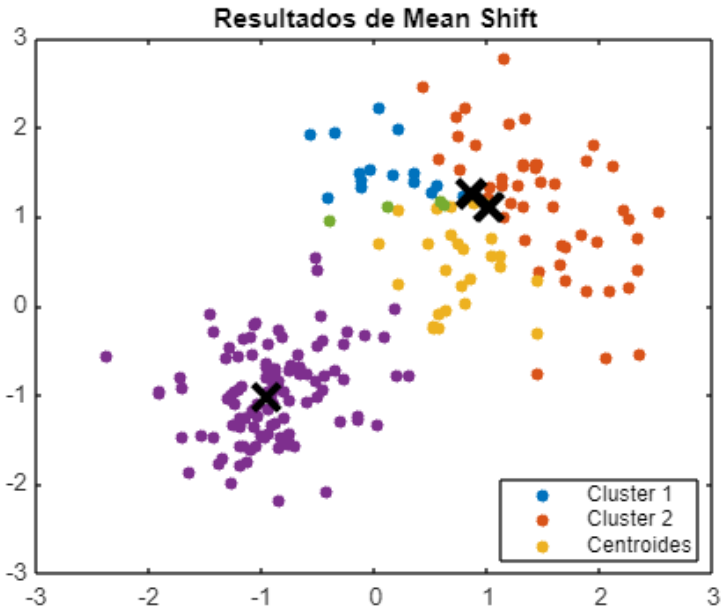

Figura 4. Resultados con el algoritmo Mean Shift.

C) Gaussian Mixture Models (GMM)

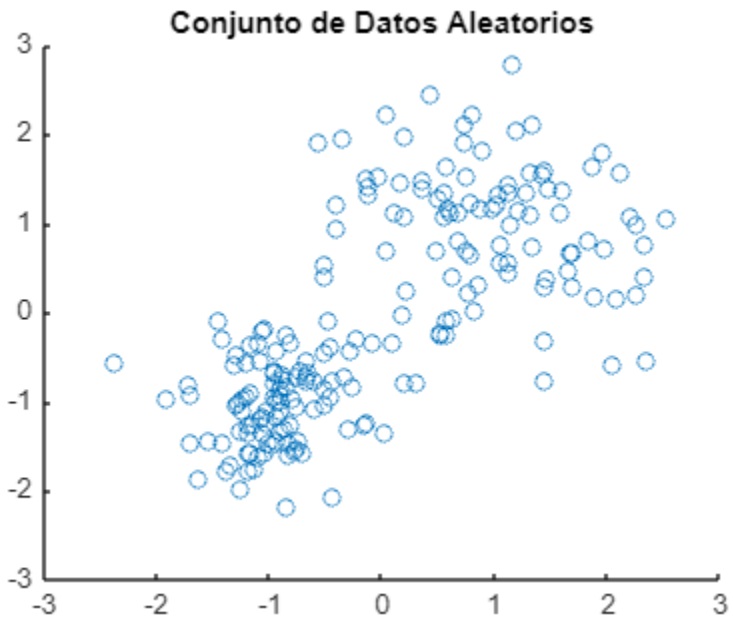

Figura 5. Conjunto de datos aleatorios del algoritmo GMM.

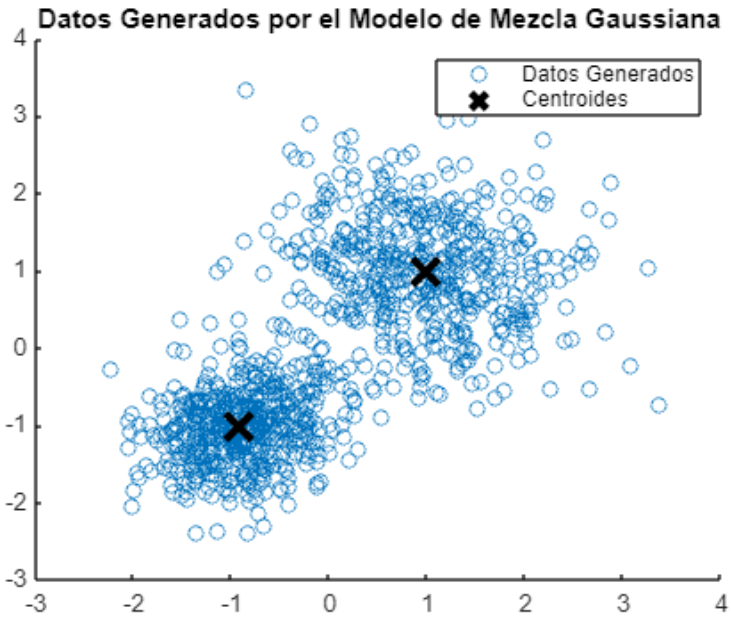

Figura 6. Datos generados por el algoritmo GMM.

D) Clustering Jerárquico

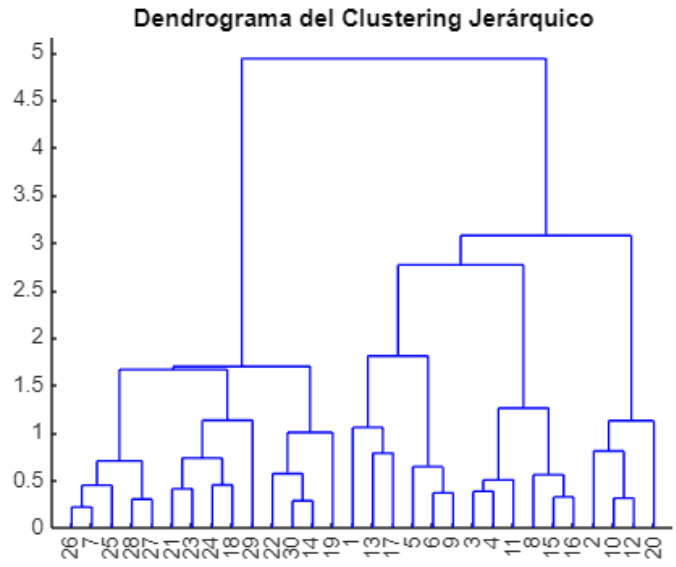

Figura 7. Dendrograma del algoritmo Clustering Jerárquico.

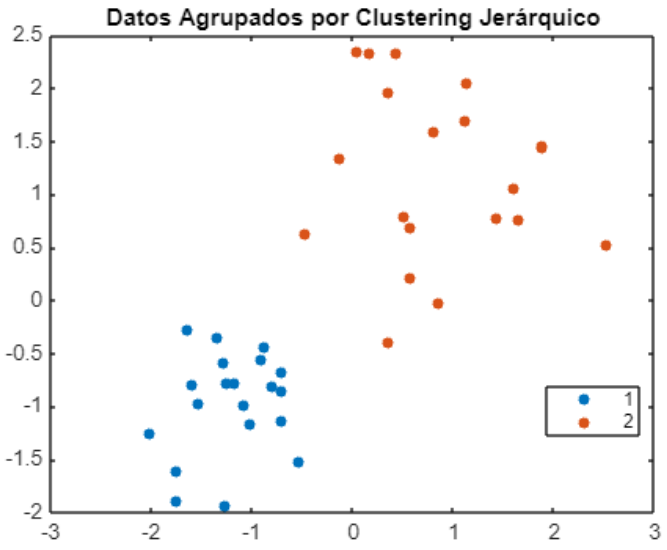

Figura 8. Datos agrupados por el algoritmo Clustering Jerárquico.

E) DBSCAN

Figura 9. Comparación de imágenes antes y después de ser procesada por el algoritmo K-means.

VI. Conclusión

Como se ha visto a lo largo del desarrollo y resultados de la práctica, los algoritmos de agrupación son fundamentales a la hora de trabajar con datos y como su nombre lo dice agruparlos de manera eficiente. Cada uno de ellos tiene sus ventajas, restricciones, funcionamiento y limitaciones. Por lo visto, ninguno es mejor que otro, sólo tienen que ser bien aplicados dependiendo en el esquema donde se este trabajando, así como las especificaciones dadas.

La diferencia que existe entre cada uno depende hacia donde lo queremos enfocar, algunos son más fáciles y eficiente de implementar, otros no requieren especificar el número de clusters, otros permiten un modelado de distribuciones complejas y algunos mas pueden darnos una visión completa de la estructura de los datos. Sin embargo, así como ventajas que pueden ofrecernos, todos tienen limitaciones a considerar: pueden no ser adecuados para datos no esféricos, pueden ser computacionalmente intensivos, requieren la especificación del número de componentes, son sensibles al ruido y a la escala de datos.

Entonces, como podemos ver esta práctica fue una ayuda muy valiosa para analizar y observar el comportamiento de cada uno de los algoritmos, que en un ejemplo más concreto pudimos aplicar una de las técnicas para procesar una imagen. De la misma manera, resalta la importancia de conocer y saber elegir el mejor algoritmo dependiendo la situación para obtener mejores resultados.

VII. Bibliografía

[1] Duda, R. O., Hart, P. E., & Stork, D. G. (2001). Pattern Classification. Wiley-Interscience.

[2] Geeksforgeeks. (2024, May 11th). K means Clustering – Introduction.

[Online]. Recuperado de: https://www.geeksforgeeks.org/k-means-clustering-

[3] Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: A review. ACM Computing Surveys (CSUR), 31(3), 264-323.

[4] MATLAB. (n.d.). K-means clustering. MathWorks.

https://www.mathworks.com/help/stats/k-means-clustering.html