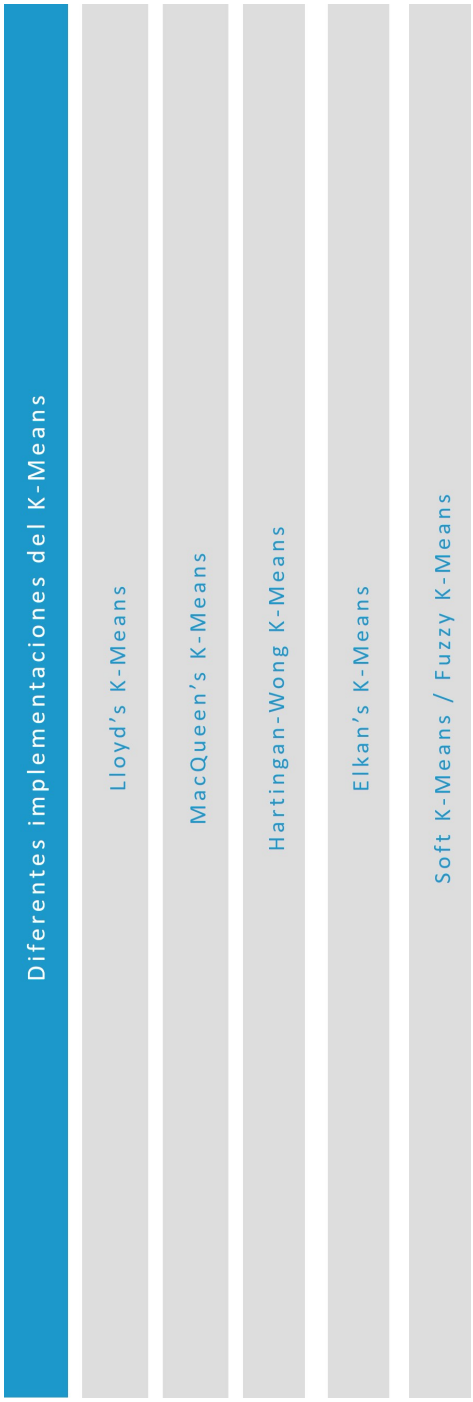
Aprendizaje Automático No Supervisado

Tema 3. Diferentes implementaciones de K-Means

Índice

Esquema



3.1. Introducción y objetivos

Entre las diversas técnicas disponibles, los algoritmos de K-Means se destacan por su simplicidad y eficacia en la creación de clústeres a partir de grandes volúmenes de datos. En este tema, profundizaremos en los diferentes algoritmos de K-Means y sus variantes, como Lloyd's, MacQueen's, Hartigan-Wong, Elkan's, y Soft K-Means.

Cada uno de estos algoritmos presenta particularidades que los hacen adecuados para distintos escenarios y tipos de datos. Por ejemplo, mientras que Lloyd's es el más simple y fácil de implementar, algoritmos como Elkan's ofrecen mejoras significativas en eficiencia mediante la reducción de cálculos de distancia, haciendo uso de propiedades geométricas avanzadas.

Asimismo, exploraremos el algoritmo de Soft K-Means, que introduce la noción de pertenencia difusa, permitiendo que los puntos de datos pertenezcan parcialmente a múltiples clústeres, una técnica invaluable en contextos donde los límites entre grupos no están claramente definidos.

No solo nos limitaremos a entender la teoría detrás de estos algoritmos, sino que también pondremos en práctica estos conocimientos mediante la implementación y análisis de casos de estudio reales. Utilizando herramientas y librerías de Python, como Scikit-learn y fcmeans, los estudiantes podrán desarrollar conocer técnicas y analizar cuáles son las ventajas y desventajas de cada uno de los algoritmos vistos.

Los **objetivos** que nos trazamos en este tema son:

- Comprender la teoría detrás de los algoritmos de K-Means y sus variantes. Estudiar los fundamentos matemáticos y lógicos que sustentan los algoritmos de Lloyd's, MacQueen's, Hartigan-Wong, Elkan's y Soft K-Means, y entender cómo cada uno maneja la partición de datos.

- Desarrollar habilidades prácticas en la implementación de algoritmos de K-Means.

Ideas clave

Utilizar Python y librerías como Scikit-learn y fcmeans; los estudiantes implementarán y compararán diferentes algoritmos de K-Means, aplicándolos a diversos conjuntos de datos.

- Analizar y evaluar la eficiencia y efectividad de diferentes algoritmos de K-Means. Análisis comparativos para evaluar la convergencia, la eficiencia computacional y la precisión de los algoritmos en diferentes escenarios.

- Aplicar algoritmos de K-Means a problemas reales. Aplicar los algoritmos estudiados en casos de estudio reales, como la compresión de imágenes, segmentación de clientes y diagnóstico médico, desarrollando soluciones prácticas y efectivas a problemas complejos.

3.2. Lloyd's k-Means

El **algoritmo de Lloyd,** comúnmente conocido como **K-means,** es un algoritmo de agrupamiento que particiona un conjunto de datos en K clústeres. Ya vimos en detalle cómo funcionaba este algoritmo junto con su implementación. Sin embargo, a modo de referencia, se indican los pasos del algoritmo, así como una posible implementación genérica en Python.

Algoritmo K-Means:

- Inicializa los centroides.
- En cada iteración:
  - Asigna cada punto al clúster más cercano.
  - Calcula nuevos centroides.
  - Verifica si los centroides han cambiado significativamente (convergencia).
- Finaliza cuando los centroides ya no cambian significativamente o se alcanza el número máximo de iteraciones.

```
def kmeans(X, K, max_iters=100, tolerance=1e-4):
    # Inicializar los centroides
    centroids = initialize_centroids(X, K)

    for i in range(max_iters):
        # Asignar puntos a los clústeres más cercanos
        cluster_labels = assign_clusters(X, centroids)

        # Calcular nuevos centroides
        new_centroids = update_centroids(X, cluster_labels, K)

        # Verificar la convergencia
        if np.all(np.abs(new_centroids - centroids) < tolerance):
            break
```

```
        centroids = new_centroids

    return centroids, cluster_labels
```

3.3. MacQueen's K-Means

Cuando Lloyd's actualiza las asignaciones de puntos de datos, no actualiza los centroides. Esto es un problema porque con cada nueva asignación, el centroide cambia la posición. Puede que un punto esté asignado erróneamente a un centroide porque no fue actualizado.

MacQueen intenta solucionar el anterior problema actualizando los centroides con cada nueva asignación. Esto conlleva más tiempo de cálculo (Wohlenberg, 2021). La inicialización es la misma que Lloyd's, en este caso se itera sobre los puntos de datos, al reasignarlos se vuelven a calcular los centroides asignados. El algoritmo se repite hasta que ningún punto cambie de grupo asignado, igual que con el algoritmo Lloyd.

- **Inicialización de centroides:** el ciclo comienza seleccionando aleatoriamente K puntos como centroides iniciales. Los centroides son puntos que representan el centro de cada uno de los K grupos que se quieren identificar.
- **Asignación de puntos a centroides de manera secuencial:** para cada punto del conjunto de datos se calcula la distancia (generalmente distancia Euclidiana) entre ese punto y cada uno de los centroides. Luego se asigna el punto al centroide más cercano.
- Se **recalculan los centroides de los clústeres** después de cada asignación como la media de todos los puntos de datos asignados a cada clúster.
- **Repetición:** los pasos de asignación de centroides y de actualización de centroides se repiten iterativamente hasta que los centroides dejen de cambiar de manera significativa o hasta que se alcance un número máximo de iteraciones.
- **Convergencia:** cuando los centroides permanecen constantes entre iteraciones y los puntos están asignados a los grupos de manera estable, podemos hablar de

convergencia.

Diferencias clave entre Lloyd's y MacQueen's K-Means

- **Lloyd's** actualiza los centroides en pasos discretos, recalculando después de asignar todos los puntos de datos. En términos de eficiencia, es más eficiente, ya que se generan actualizaciones en un solo momento. Requiere más iteraciones para converger debido a su naturaleza de grandes cambios de centroides en cada iteración.

- **MacQueen's** actualiza los centroides de manera continua, recalcula el centroide cada vez que asigna un punto de datos, garantizando que cada punto este asignado al clúster correcto. Es menos eficiente que Lloyd's si estamos frente a un gran conjunto de datos. Si es un conjunto de datos que llega como flujo continuo su comportamiento es mejor que el algoritmo Lloyd's. En términos de convergencia, converge más rápido, ya que el número de iteraciones se ajustan continuamente a los centroides.

3.4. Hartingan-Wong K-Means

Tanto Lloyd's como MacQueen no eligen centroides iniciales eficientes y puede que el algoritmo no converja a causa de una mala inicialización.

El **algoritmo Hartigan-Wong** asigna los puntos de datos a los centroides de forma aleatoria. Luego se calcula cada centroide como la media de los puntos asignados. Esto resulta en que todos los centroides estén inicialmente en posiciones similares, lo que reduce la probabilidad de converger a un óptimo local (Slonim, Aharoni y Crammer, 2013). En consecuencia, un punto de datos puede ser asignado a un centroide que no es el más cercano; si disminuye la función objetivo, mejora la calidad total de los grupos.

- Inicialización de centroides: el ciclo comienza seleccionando aleatoriamente K puntos como centroides iniciales.
- Al azar se asignan los puntos a un centroide.
- Se calcula el centroide como la media de los puntos asignados.
- Se repite hasta que exista convergencia o hasta el número máximo de iteraciones.

Diferencias clave entre Hartigan-Wong y Lloyd's K-Means

- Lloyd's actualiza los centroides después de una asignación completa de puntos a los clústeres. Hartigan-Wong realiza actualizaciones más frecuentes y locales al considerar la reubicación de puntos individuales entre clústeres durante las interacciones.
- Hartigan-Wong puede requerir menos iteraciones para converger, ya que optimiza de manera continua las ubicaciones de los puntos de datos.
- Lloyd's es más simple de implementar y menos complejo computacionalmente. Hartigan-Wong puede ser más complejo de implementar.

Ideas clave

3.5. Elkan's K-Means

El **algoritmo de Elkan** es una variante del algoritmo K-Means; un método de *clustering* usado en el aprendizaje automático para particionar un conjunto de datos en k grupos o clústeres. El algoritmo de Elkan se diseñó para acelerar la convergencia del K-Means y reducir el número de cálculos de distancia necesarios, haciéndolo más eficiente.

Principios del algoritmo de Elkan

- **Reducción de cálculos de distancia:** Elkan reduce el número de cálculos de distancia necesarios utilizando límites superiores e inferiores. Esto se basa en la observación de que, si sabemos que un punto de datos está cerca de un centroide particular, podemos evitar calcular la distancia de ese punto a centros de clústeres lejanos.
- ***Triangle inequality:*** aprovecha la desigualdad triangular para evitar calcular distancias innecesarias. La desigualdad triangular establece que, para cualquier triángulo, la suma de las longitudes de dos lados siempre será mayor o igual que la longitud del tercer lado.
- **Actualización eficiente:** cada vez que los centroides se actualizan, también se actualizan los límites superiores e inferiores de las distancias, minimizando los cálculos futuros.

Pasos del algoritmo de Elkan

- Inicializar los centroides de los k clústeres.
- Mantener un límite superior y un límite inferior de las distancias de cada punto a los centroides. Usar estos límites para evitar calcular distancias cuando no es necesario.
- Usar los límites para determinar el clúster más cercano a cada punto de datos sin calcular todas las distancias.

- Asignar cada punto de datos al clúster más cercano determinado.
- Actualización de centroides: calcular los nuevos centroides basados en las asignaciones actuales de los puntos a los clústeres.
- Actualización de límites: actualizar los límites superiores e inferiores de las distancias después de mover los centroides.
- Iteración: repetir los pasos 2 a 5 hasta que los centroides no cambien o se alcance un número máximo de iteraciones.

Ventajas del algoritmo de Elkan

- Eficiencia: reduce el número de cálculos de distancia, lo que puede acelerar el proceso de convergencia.
- Escalabilidad: funciona mejor en grandes conjuntos de datos debido a su optimización en cálculos de distancia.

Limitaciones

- **Complejidad adicional:** implementar el algoritmo de Elkan es más complejo que el K-Means estándar.
- **Sensibilidad a la inicialización:** como el K-Means tradicional, la calidad de los resultados puede depender de la inicialización de los centroides.

Veamos un ejemplo de uso de K-Means para comprimir una imagen.

```
# Módulos
import matplotlib.pyplot as plt
from matplotlib.image import imread
import pandas as pd
import seaborn as sns
import numpy as np
#from sklearn.datasets.samples_generator import
(make_blobs,make_circles,make_moons)
```

```
from sklearn.datasets import make_blobs

X, y = make_blobs(n_samples=100, centers=3, n_features=2, random_state=0)
from sklearn.cluster import KMeans, SpectralClustering
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_samples, silhouette_score

# Leer la imagen
img = imread('DALL·E 2024-06-11 17.26.23Robot.jpg')
img_size = img.shape

# Reshape para que sea bidimensional
X = img.reshape(img_size[0] * img_size[1], img_size[2])

# Correr el algoritmo K-Means
km = KMeans(n_clusters=30)
km.fit(X)

# Use los centroides para comprimir la imagen

X_compressed = km.cluster_centers_[km.labels_]
X_compressed = np.clip(X_compressed.astype('uint8'), 0, 255)

# Reshape X_recovered para tener la misma dimensión que la original

X_compressed = X_compressed.reshape(img_size[0], img_size[1], img_size[2])

# Dibujar la imagen original y la comprimida

fig, ax = plt.subplots(1, 2, figsize = (12, 8))
ax[0].imshow(img)
ax[0].set_title('Original Image')
ax[1].imshow(X_compressed)
ax[1].set_title('Comprimir la imagen a 30 colores')
for ax in fig.axes:
    ax.axis('off')
plt.tight_layout();
```

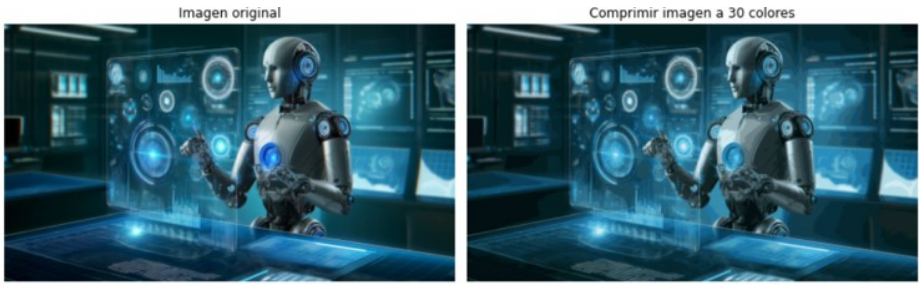


Figura 1. Compresión de imagen a 30 colores con K-Means. Fuente: elaboración propia.

```
# Entrenar con el algoritmo Elkan K-Means

km = KMeans(n_clusters=40, algorithm='elkan')
km.fit(X)

# Usar los centroides para comprimir la imagen
X_compressed1 = km.cluster_centers_[km.labels_]
X_compressed1 = np.clip(X_compressed1.astype('uint8'), 0, 255)

X_compressed1 = X_compressed1.reshape(img_size[0], img_size[1],
img_size[2])

# Dibujar la imagen original y la comprimida

fig, ax = plt.subplots(1, 2, figsize = (12, 8))
ax[0].imshow(img)
ax[0].set_title('Imagen original')
ax[1].imshow(X_compressed1)
ax[1].set_title('Imagen comprimida a 40 colores')
for ax in fig.axes:
    ax.axis('off')
plt.tight_layout();
```

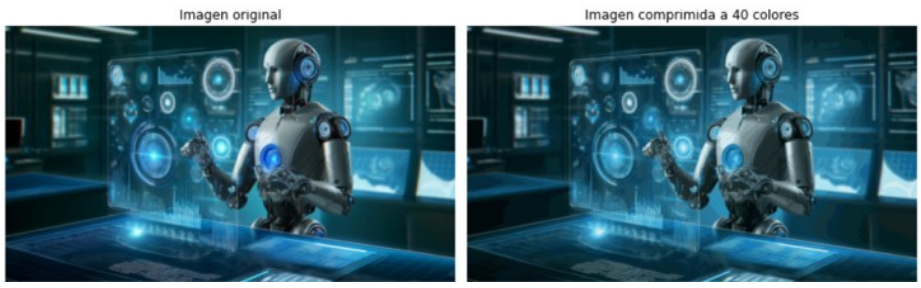
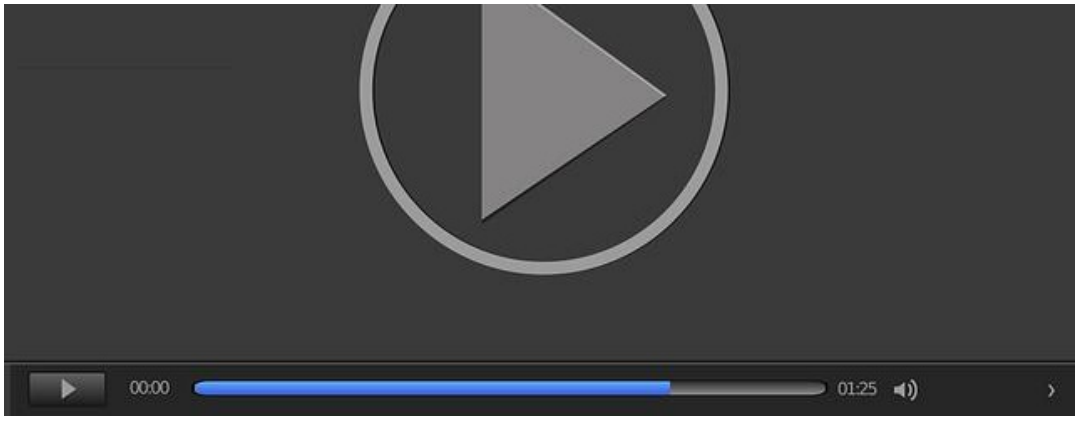


Figura 2. Compresión de imagen a 40 colores con Elkan K-Means. Fuente: elaboración propia.

A continuación, veremos el vídeo de ***Optimización y mejores prácticas en clustering.***

Ideas clave



Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=46b607a1-f531-4aec-b5a1-b1be00e3fb82

3.6. Soft K-Means / Fuzzy K-Means

Uno de los problemas que tiene el K-Means y de la mayoría de sus implementaciones, es la sensibilidad a la hora de inicializar los centroides. Una forma de minimizar este riesgo es inicializando K-Means varias veces y utilizar el valor que mejor separe a los grupos (Lazyprogrammer.me, 2016). Computacionalmente es más fácil crear límites difusos que establecer un solo grupo para un punto. Un punto puede pertenecer a uno o más grupos.

Teniendo en cuenta que la función de coste es sensible al mínimo local, se puede minimizar la función utilizando membresía difusa para cada clase. Es decir, cada punto tiene un % de membresía a cada grupo. Por ejemplo, puede ser un 70 % de un grupo y un 30 % de otro grupo. A continuación, explicamos el algoritmo de agrupamiento difuso clásico.

Agrupamiento difuso clásico

1. El primer paso es el mismo que el algoritmo clásico de K-Means, **inicializar los K clústeres al azar.**

2. Cada punto puede pertenecer a todos los grupos. Mediante el uso de una función de membresía que está entre 0 y 1, donde 0 indica que el punto está muy alejado del clúster y 1 está muy cerca al centroide del grupo.

Suponiendo que tenemos dos grupos en los que se van a dividir los datos, si se inicializa al azar, cada punto se encuentra en ambos grupos con un valor de membresía. En la siguiente tabla podemos ver los diferentes valores de membresía:

Clúster	(1,5)	(2,7)	(5,8)	(6,9)
1)	0.9	0.6	0.2	0.5
2)	0.1	0.4	0.8	0.5

Tabla 1. Valores de la membresía. Fuente: elaboración propia.

3. Encontrar el centroide: se utiliza la siguiente fórmula para calcular cada uno de los centroides:

$$V_{ij} = \frac{\sum_1^m (\Upsilon * x_k)}{\sum_1^n \Upsilon_{ik}^m}$$

Donde Υ es el valor de pertenencia difusa del punto de datos y m es el parámetro de borrosidad al conjunto difuso del dato que generalmente es 2 y xk es el punto.

$$V11 = \frac{(0.9^2 * 1 + 0.6^2 * 2 + 0.2^2 * 5 + 0.5^2 * 6)}{0.9^2 + 0.6^2 + 0.2^2 + 0.5^2} = 2.21$$

$$V12 = \frac{(0.9^2 * 5 + 0.6^2 * 7 + 0.2^2 * 8 + 0.5^2 * 9)}{0.9^2 + 0.6^2 + 0.2^2 + 0.5^2} = 6.26$$

$$V21 = \frac{(0.1^2 * 1 + 0.4^2 * 2 + 0.8^2 * 5 + 0.5^2 * 6)}{0.1^2 + 0.4^2 + 0.8^2 + 0.5^2} = 4.75$$

$$V22 = \frac{(0.1^2 * 5 + 0.4^2 * 7 + 0.8^2 * 8 + 0.5^2 * 9)}{0.1^2 + 0.4^2 + 0.8^2 + 0.5^2} = 8.06$$

Los centroides son: (2.21, 6.26) y (4.75,8.06).

4. Calcular la distancia de cada punto a cada uno de los centroides. Calculemos la distancia del punto (1,5) a los dos centroides:

$$D11 = ((1 - 2.21)^2 + (5 - 6.26)^2)^{0.5} = 1.75$$

$$D12 = ((1 - 4.75)^2 + (5 - 8.06)^2)^{0.5} = 4.84$$

5. Actualizar el valor de membresía utilizando la siguiente ecuación.

$$\Upsilon = \sum_1^n (d_{ki}^2 / d_{kj}^2)^{\frac{1}{m-1}}{}^{-1}$$

Calculamos el valor de membresía a cada centroide del punto (1,5).

$$\gamma 11 = \left[\left\{ \left[(1.75)^2 / (1.75)^2 \right] + \left[(1.75)^2 / (4.84)^2 \right] \right\}^{\left(\frac{1}{2-1}\right)} \right]^{-1} = 0.88$$

$$\gamma 12 = \left[\left\{ \left[(4.84)^2 / (4.84)^2 \right] + \left[(4.84)^2 / (1.75)^2 \right] \right\}^{\left(\frac{1}{2-1}\right)} \right]^{-1} = 0.12$$

6. Repetir los pasos de encontrar el centroide, calcular la distancia y actualizar el valor de la membresía. Se repite hasta que se obtengan valores constantes para los valores de membresía o la diferencia sea menor a un valor de tolerancia.

7. Desfuzzificar los valores de membresía obtenidos.

Aplicaciones

La aplicación del clúster difuso es muy amplia. Algunas de sus aplicaciones son las siguientes:

- En **segmentación de imágenes** ayuda a agrupar imágenes por píxeles con propiedades similares, como color y textura.
- En **marketing** se pueden identificar para segmentar a los clientes en función de sus preferencias y comportamientos de compra, permitiendo campañas más personalizadas.
- En **diagnóstico médico** se pueden agrupar pacientes con síntomas similares y que pueden tener varios diagnósticos a la vez.
- En **monitoreo ambiental** ayuda a identificar áreas de preocupación ambiental por contaminación.
- En **análisis de tráfico** identifica áreas donde el flujo de tráfico es similar ayudando a gestionar y planificar el tráfico.

- En **evaluación de riesgo** se utiliza para cuantificar riesgos de campos como finanzas, seguros e ingeniería.

Ventajas

El algoritmo de agrupación difusa cuenta con varias ventajas; entre ellas tenemos:

- **Flexibilidad:** permite superponer agrupaciones. Es muy útil cuando los datos tienen una estructura compleja o cuando los límites de las clases son ambiguas.
- **Robustez:** puede ser más robusta su implementación ante valores atípicos y ruido de los datos, al permitir una transición gradual entre grupos.
- **Interpretabilidad:** proporciona una comprensión más específica de los datos, y una representación más detallada de las relaciones entre los puntos de datos.

Desventajas

Encontramos algunos inconvenientes a la hora de implementar los algoritmos de agrupación difusa:

- **Complejidad:** computacionalmente son más costosos, ya que tiene que calcular el grado de membresía a cada grupo.
- Elegir el número correcto de clústeres y funciones de membresía puede ser todo un desafío y requerir conocimientos de un experto, además de mucha prueba y error.

A continuación, se puede ver un ejemplo en Python, utilizando la librería de Scikit-learn y la librería fcmeans:

```
from fcmeans import FCM
from sklearn.datasets import make_blobs
from matplotlib import pyplot as plt
import seaborn as sns
from seaborn import scatterplot as scatter
import numpy as np
import pandas as pd
from sklearn import datasets
```

```
from sklearn.datasets import load_iris

iris=load_iris()
X=iris['data']
y=iris['target']
#Dibujar los datos originales
plt.scatter(X[y==0,0], X[y==0,1], s=80, c='orange', label = 'Iris-setosa')
plt.scatter(X[y==1,0], X[y==1,1], s=80, c='yellow', label = 'Iris-
versicolour')
plt.scatter(X[y==2,0], X[y==2,1], s=80, c='green', label = 'Iris-
virginica')
plt.xlabel('Sepal length)
plt.ylabel('Sepal width')
plt.title('Datos originales')
plt.legend()
```

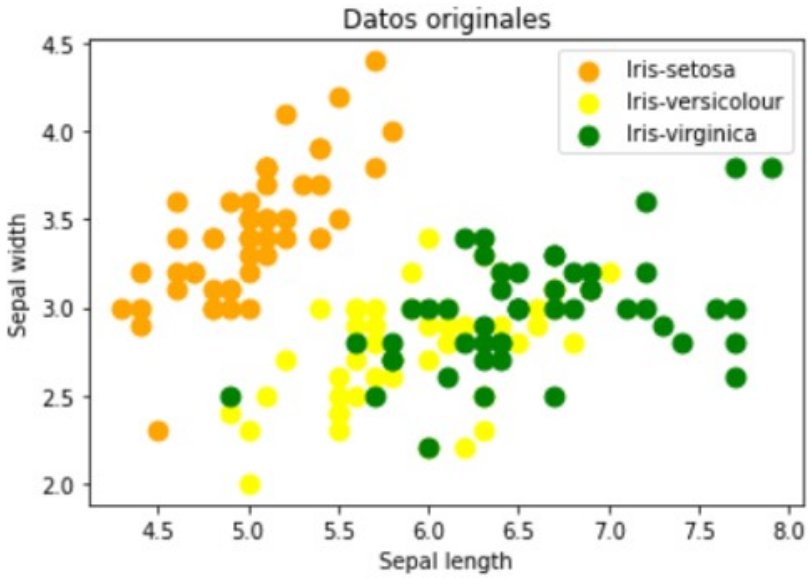


Figura 3. Datos del *dataset* Iris con su clasificación original. Fuente: elaboración propia.

```
#Entrenar el algoritmo fuzzy K-Means
fcm=FCM(n_clusters=3)
fcm.fit(X)
fcm_centers=fcm.centers
fcm_labels=fcm.u.argmax(axis=1)
f,axes=plt.subplots(1,2,figsize=(11,5))

#Dibujar los puntos y los centroides resultantes.
sns.scatterplot(x=X[:,0], y=X[:,1], ax= axes[0])
sns.scatterplot(x=X[:,0], y=X[:,1], ax= axes[1], hue=fcm_labels)
```

```
sns.scatterplot(x=fcm_centers[:,0], y=fcm_centers[:,1], ax=axes[1], marker=
"s", s=200)
plt.show()
```

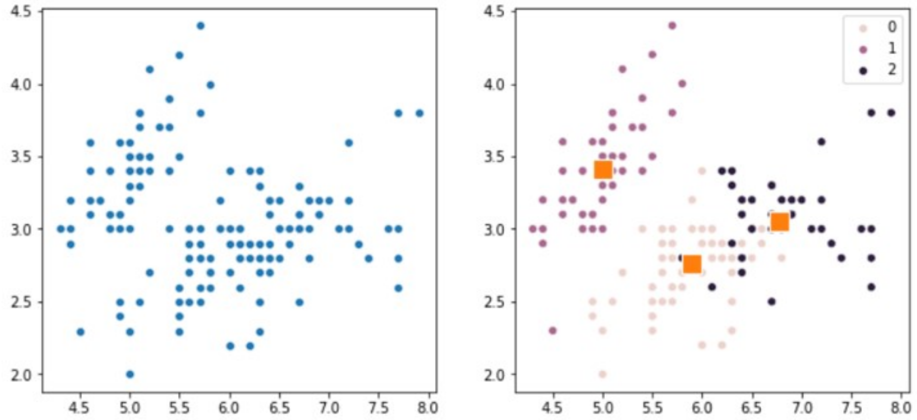


Figura 4. Aplicaciones de los clústeres difusos sobre el *dataset* de Iris. Fuente: elaboración propia.

```
# Predecir la membresía del clúster para cada punto de datos
cluster_membership = fcm.u.argmax(axis=1)

# Imprimir los centros de los clústeres
print('Centro de clústeres:', fcm_centers)

# Imprima la membresía del clúster para cada punto de datos
print('Membresía de clúster:', cluster_membership)
```

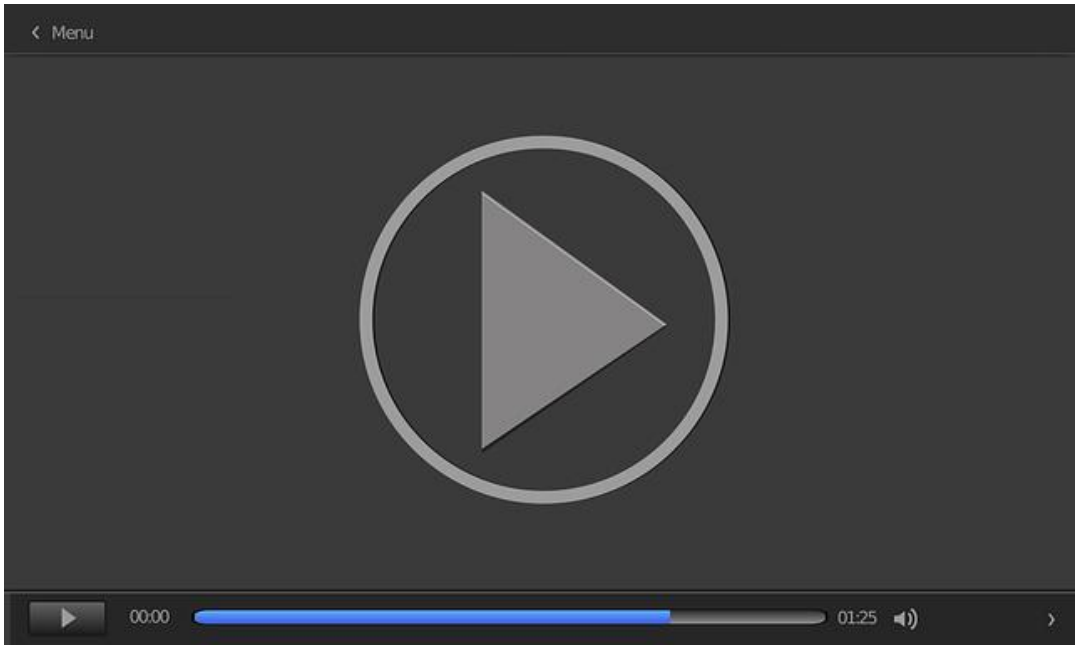
```
Centro de clústeres: [[5.88893094 2.76106885 4.36394956 1.39731397]
 [6.7750095  3.05238177 5.64677964 2.05354583]
 [5.00396596 3.41408893 1.48281542 0.25354627]]
Membresía de clúster: [2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 0 1 1 1 1
 1 1 0 1 1 1 1 1 0 1 0 1 0 1 1 0 0 1 1 1 1 1 0 1 1 1 1 0 1 1 1 0 1 1 1 0 1
 1 0]
```

Figura 5. Centroides y etiquetas dadas por la membresía después de aplicar Fuzzy K-Means. Fuente: elaboración propia.

En el código primero se importa el *dataset* de Iris. Después se dibujan los puntos originales (Figura 3), identificando cada clase con un color. Sin tener en cuenta la etiqueta original se le aplica un algoritmo de Fuzzy K-Means y posteriormente se muestran los centroides resultantes (Figura 4). La clasificación realizada no difiere mucho de la clasificación real. Y, finalmente, imprimimos los puntos donde están

ubicados los centroides y las etiquetas asignadas a cada punto después de aplicar Fuzzy K-Means (Figura 5).

A continuación, veremos el vídeo ***Aplicaciones avanzadas de K-Means.***



Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=04202076-ca31-40b9-b608-b1be00da113d

3.7. Cuaderno de ejercicios

Ejercicio 1: exploración y descripción de datos

Implementa el algoritmo Lloyd's K-Means utilizando la librería Scikit-learn en Python. Usa el *dataset* de `make_blobs` con 300 muestras, 3 centros y 2 características. Grafica los datos originales y los clústeres formados.

Solución

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# Generar datos
X, y = make_blobs(n_samples=300, centers=3, n_features=2, random_state=0)

# Implementar K-Means
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)

# Graficar resultados
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.75)
plt.show()
```

Ejercicio 2: flujo de trabajo y retos

Explica las diferencias clave entre los algoritmos Lloyd's y MacQueen's K-Means. ¿En qué escenarios sería preferible usar MacQueen's K-Means?

Solución

Lloyd's K-Means actualiza los centroides después de asignar todos los puntos, mientras que MacQueen's K-Means actualiza los centroides continuamente con cada nueva asignación. Lloyd's es más eficiente computacionalmente para grandes conjuntos de datos estáticos, ya que requiere menos recalculaciones por iteración.

MacQueen's es preferible en escenarios de datos de flujo continuo, ya que ofrece una convergencia más rápida al ajustar los centroides de inmediato con cada nuevo punto de datos.

Ejercicio 3: preprocesamiento de datos

Implementa el algoritmo Elkan's K-Means usando Scikit-learn y compara su eficiencia con el K-Means estándar en el *dataset* make_blobs de 1000 muestras, 5 centros y 2 características.

Solución

```
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
import time

# Generar datos
X, _ = make_blobs(n_samples=1000, centers=5, n_features=2, random_state=0)

# K-Means estándar
start = time.time()
kmeans = KMeans(n_clusters=5, algorithm='auto').fit(X)
end = time.time()
print(f"K-Means estándar: {end - start} segundos")

# Elkan's K-Means
start = time.time()
elkan_kmeans = KMeans(n_clusters=5, algorithm='elkan').fit(X)
end = time.time()
print(f"Elkan's K-Means: {end - start} segundos")
```

Este código muestra que Elkan's K-Means puede ser más eficiente al reducir los cálculos de distancia, especialmente en conjuntos de datos más grandes.

Ejercicio 4: discretización y codificación de datos categóricos

Describe cómo se podría aplicar Soft K-Means para la segmentación de una imagen y menciona una ventaja y una desventaja de este método.

Solución

Ideas clave

Soft K-Means se aplica a la segmentación de imágenes asignando a cada píxel una pertenencia difusa a varios clústeres en lugar de una asignación rígida. Esto permite que los píxeles pertenezcan parcialmente a múltiples clústeres, manejando mejor las transiciones suaves entre regiones de la imagen.

- **Ventaja:** flexibilidad en la asignación de píxeles, lo que resulta en transiciones más suaves entre regiones.
- **Desventaja:** computacionalmente más costoso debido al cálculo continuo de pertenencias difusas.

3.8. Referencias bibliográficas

Lazyprogrammer.me. (2016). *Unsupervised Machine Learning in Python: Master Data Science and Machine Learning with Cluster Analysis, Gaussian Mixture Models, and Principal Components Analysis.* Amazon.

Slonim, N., Aharoni, E. y Crammer, K. (2013). Hartigan's K-means versus Lloyd's K-means: is it time for a change? *Proceedings of the 23rd International Joint Conference on Artificial Intelligence,* 1677-1684.

Wohlenberg, J. (2021, junio 21). 3 versions of k-Means. Medium. https://towardsdatascience.com/three-versions-of-k-means-cf939b65f4ea

Ejemplo resuelto del algoritmo de agrupamiento Fuzzy C-Means

Mahesh Huddar. (2023, julio 4). *Fuzzy C Means Clustering Algorithm Solved Example | Clustering Algorithm in ML & DL by Mahesh Huddar* [Vídeo]. YouTube. https://www.youtube.com/watch?v=X7co6-U4BJY

El vídeo discute conceptos clave como el *clustering* difuso, el algoritmo Fuzzy C-Means y su aplicación en *machine learning* y minería de datos. También se explica el algoritmo C-Means y se compara con el K-Means mediante ejemplos.

Algoritmo de agrupamiento C-Means

Data Science with Sharan. (2021, septiembre 8). *Day 70 - Fuzzy C-Means Clustering Algorithm* [Vídeo]. YouTube. https://www.youtube.com/watch?v=VhYt7nxOKKs

Explica en detalle el algoritmo de Fuzzy C-Means, un método de *clustering* difuso que permite que un punto de datos pertenezca a más de un grupo con diferentes grados de pertenencia.

1. ¿Cuál es la principal diferencia entre los algoritmos de Lloyd MacQueen?

A. Lloyd actualiza los centroides de manera continua, mientras que MacQueen lo hace en pasos discretos.

B. MacQueen actualiza los centroides de manera continua, mientras que Lloyd lo hace en pasos discretos.

C. Lloyd y MacQueen actualizan los centroides de la misma manera.

D. Lloyd y MacQueen no recalculan los centroides.

2. ¿Cuál de las siguientes afirmaciones es correcta sobre el algoritmo de Hartigan-Wong?

A. No realiza actualizaciones de centroides.

B. Es menos eficiente que Lloyd en términos de cálculo.

C. Es más simple de implementar que Lloyd.

D. Realiza actualizaciones de centroides más frecuentes y locales.

3. ¿Qué ventaja tiene el algoritmo de Elkan sobre K-Means estándar?

A. No necesita calcular distancias.

B. No necesita inicializar centroides.

C. Usa límites superiores e inferiores para reducir cálculos de distancia.

D. Siempre converge al óptimo global.

4. ¿Cuál es la desventaja del algoritmo de Fuzzy K-Means?

A. No permite superponer agrupaciones.

B. Es menos robusto ante valores atípicos.

C. Computacionalmente es más costoso.

D. No puede manejar conjuntos de datos grandes.

5. ¿Cuál es una ventaja del Fuzzy K-Means respecto al K-Means clásico?

A. Mayor robustez ante valores atípicos y ruido.

B. Menor flexibilidad en la asignación de puntos.

C. Mayor sensibilidad a la inicialización de centroides.

D. Menor interpretación de datos.

6. En el algoritmo de Elkan, ¿qué se actualiza cada vez que se recalculan los centroides?

A. Solo los límites superiores.

B. Solo los límites inferiores.

C. Los límites superiores e inferiores de las distancias.

D. Ninguna de las anteriores.

7. ¿Cuál es el objetivo principal de la inicialización en los algoritmos de K-Means?

A. Seleccionar puntos que representen el centro de cada clúster.

B. Evitar la convergencia.

C. Asegurar que todos los puntos estén en el mismo clúster.

D. Reducir el número de iteraciones.

8. ¿Qué técnica se utiliza el algoritmo de Elkan para evitar calcular distancias innecesarias?

A. Inicialización aleatoria.

B. Distancia euclidiana.

C. Desigualdad triangular.

D. *Clustering* jerárquico.

9. ¿Cuál es un problema común en la mayoría de las implementaciones de K-Means?

A. La sensibilidad a la inicialización de los centroides.

B. La interpretación de los resultados obtenidos.

C. La convergencia rápida.

D. La sobreestimación de la cantidad de clústeres.

10. ¿Qué tipo de datos se agrupan en la segmentación de imágenes utilizando Fuzzy K-Means?

A. Textos con propiedades similares.

B. Píxeles con propiedades similares como color y textura.

C. Puntos de datos sin relación.

D. Palabras con significados similares.