APAU_II Practica 1

Usuarios de github:

@LucachuTW - Lucas González Fiz
@Minix16 - Javier Pérez Robles

1. Cálculo de la Distancia Euclidiana

Aunque no se ha implementado una función `calcular_distancia` propia, el código utiliza la función `pairwise_distances` de la librería `scikit-learn`. Esta función calcula eficientemente la distancia euclidiana entre todos los pares de puntos de dos matrices. La distancia euclidiana, como sabemos, es la raíz cuadrada de la suma de las diferencias al cuadrado entre las coordenadas de dos puntos. Es la medida de distancia estándar en K-Means.

2. Inicialización de los Centroides

La función `kmeans` incluye la lógica para inicializar los centroides (los puntos centrales de cada cluster). Se ofrecen dos métodos, seleccionables mediante el parámetro `init_method`:

- **Método 'random_points':** Se generan *k* centroides aleatorios. Las coordenadas de cada centroide se eligen uniformemente al azar dentro del rango de valores de los datos de entrada (mínimo y máximo de cada característica).

- **Método 'random_data':** Se seleccionan *k* puntos al azar *directamente del conjunto de datos* para que sirvan como centroides iniciales. Esto asegura que los centroides iniciales sean puntos "reales" y no valores fuera del rango de los datos.

3. Asignación de Puntos a Clusters

Dentro del bucle principal de `kmeans`, esta sección se encarga de asignar cada punto del conjunto de datos al cluster cuyo centroide esté más cercano:

1. **Cálculo de Distancias:** Se utiliza `pairwise_distances` para calcular la matriz de distancias entre todos los puntos de datos (`X`) y todos los centroides.
2. **Asignación:** Se usa `np.argmin(distances, axis=1)` para encontrar, para cada punto, el índice del centroide más cercano. Esto produce el vector `labels`, que contiene el índice del cluster asignado a cada punto.
3. **Equilibrio de clusters:**
    - Se itera sobre todos los puntos
    - Se comprueba si la distancia del punto a su centroide asignado, es igual a la distancia a otros centroides.
    - Si lo es, se cuentan cuantos puntos tiene asignado cada cluster.
    - Se asigna el punto en cuestión al cluster con menos puntos asignados.

4. Actualización de Centroides

Tras la asignación de puntos, se recalculan los centroides. El nuevo centroide de cada cluster se define como el *promedio* de todos los puntos que han sido asignados a ese cluster. La línea de código anterior realiza este cálculo de forma concisa y eficiente utilizando las capacidades de NumPy:

- `X[labels == j]`: Selecciona todos los puntos que pertenecen al cluster `j`.
- `.mean(axis=0)`: Calcula la media de esos puntos a lo largo de las columnas (es decir, la media de cada característica), obteniendo así las coordenadas del nuevo centroide.
- La comprensión de lista `[... for j in range(n_clusters)]` repite este proceso para cada cluster.

5. Implementación del Algoritmo K-Means (Función `kmeans`)

La función `kmeans` encapsula todo el algoritmo:

1. **Inicialización:** Prepara los centroides iniciales (según el `init_method` especificado).
2. **Bucle Principal:** Itera hasta un máximo de `max_iter` veces o hasta que los centroides dejen de cambiar (convergencia):
    - Asigna cada punto al cluster más cercano.
    - Actualiza los centroides como el promedio de los puntos asignados a cada cluster.
3. **Retorno:** Devuelve los centroides finales y las etiquetas de cluster asignadas a cada punto.

6. Generación de Datos Sintéticos

Se utiliza la función `make_blobs` de `scikit-learn` para generar un conjunto de datos de prueba. Esta función crea "manchas" de puntos, lo que es ideal para probar algoritmos de clustering como K-Means. Los parámetros controlan el número de puntos, el número de centros (clusters), la dispersión de los puntos y la reproducibilidad de los resultados.

7. Visualización de los Resultados

Se utiliza `matplotlib.pyplot` para visualizar los resultados: