Grado en Matemáticas
Trabajo de Fin de Grado

Métodos de agrupamiento de datos: convergencia y aplicaciones

César Hernández Fogued

Director:
Ricardo López Ruiz

Curso académico 2023-2024

Resumen

En un mundo cada vez más impulsado por los datos, la inteligencia artificial (IA) se ha convertido en una herramienta esencial para resolver problemas complejos y mejorar la toma de decisiones en diversos sectores. Desde la predicción de tendencias de mercado hasta la optimización de recursos, la IA permite analizar grandes volúmenes de datos de manera eficiente y extraer patrones útiles. Dentro de este panorama, el aprendizaje automático destaca como una técnica clave que permite a los sistemas adaptarse y mejorar su rendimiento automáticamente, abriendo la puerta a innovaciones que transforman industrias enteras. Estas tecnologías no solo ayudan a identificar oportunidades ocultas, sino que también permiten diseñar estrategias más personalizadas y efectivas en campos como la medicina, el transporte y la economía.

El trabajo de fin de grado se enfoca en los métodos de agrupamiento en el contexto del aprendizaje no supervisado, explorando especialmente los algoritmos k-medias y DBSCAN. Estos son comparados por sus características, ventajas y limitaciones, destacando sus aplicaciones en conjuntos de datos reales. El enfoque del trabajo incluye tanto fundamentos teóricos sobre la convergencia como implementaciones prácticas en Python, con el objetivo de estudiar la eficacia de los métodos.

El agrupamiento tiene como objetivo identificar patrones en los datos sin etiquetas, formando subconjuntos similares. K-medias es un método basado en la partición de los datos en clústeres esféricos, minimizando la distancia entre los datos y los centros de los clústeres. DBSCAN, en cambio, es un algoritmo que se basa en la densidad, detectando clústeres de formas arbitrarias así como puntos de ruido, siendo, por tanto, útil en datos con ruido o distribuciones irregulares.

Para verificar la aplicabilidad del agrupamiento, se utiliza el índice de Hopkins, que mide la tendencia de los datos a formar grupos. Además, se explora la reducción de dimensionalidad con análisis de componentes principales (PCA), permitiendo simplificar el procesamiento y la representación de los datos, manteniendo la mayor parte de la información.

En términos de aplicaciones, se utilizan datos de países, con variables como mortalidad infantil, esperanza de vida y PIB per cápita. K-medias logra agrupar países en tres categorías según su nivel de desarrollo, mientras que DBSCAN identifica patrones menos rígidos, detectando países atípicos. También se aplica k-medias a la compresión de imágenes, mostrando su utilidad en tareas de reducción de información.

Finalmente, con la intención de añadir un toque personal, se implementaron ambos algoritmos en Python, manteniendo la notación y algunas mejoras que se implementan usualmente en las librerías. Se concluye que ambos métodos son valiosos dependiendo del contexto, pero su eficacia depende de la elección adecuada de parámetros y de las características del conjunto de datos.

Abstract

In an increasingly data-driven world, artificial intelligence (AI) has become an essential tool to solve complex problems and improve decision making in diverse sectors. From predicting market trends to resource optimization, AI allows one to analyze huge data volumes in a more efficient way and extract useful patterns. Within this framework, machine learning stands out as a key technique that allows systems to automatically adapt and improve their performance, letting those innovations transform entire industries. These technologies not only help identify hidden opportunities, but also enable the design of more personalized and effective strategies in fields such as medicine, transportation, and economics.

This final degree project focuses on clustering methods in the context of unsupervised learning, particularly exploring k-means and DBSCAN algorithms. These are compared in terms of their characteristics, advantages and limitations, emphasizing their applications in real databases. The project combines theoretical foundations on convergence with practical implementations in Python, in order to study the effectiveness of these methods.

The aim of clustering is to identify patterns in unlabeled data by forming similar subsets. K-means tends to partition data into spherical clusters by minimizing the distance between data points and cluster centers. DBSCAN, on the other hand, is a density-based algorithm that detects clusters of arbitrary shapes and noise points, making it particularly useful for datasets with noise or irregular distributions.

To verify the applicability of clustering, the Hopkins index is used to measure the tendency of the data to form groups. Moreover, dimensionality reduction is explored using principal component analysis (PCA), enabling the simplification of data processing and representation while retaining most of the information.

In terms of applications, country data is used, including variables such as infant mortality, life expectancy, and GDP per capita. K-means manages to form three country clusters by their level of development, while DBSCAN identifies more flexible patterns and detects outlier countries. K-means is also applied to image compression, demonstrating its usefulness in information reduction tasks.

Finally, in order to include a personal touch, both algorithms were implemented in Python, maintaining notation and incorporating some commonly used enhancements from libraries. We conclude that both methods are valuable depending on the context, but their effectiveness depends on the correct fixation of the parameters and the characteristics of the data set.

Índice general

Capítulo 1

Introducción al aprendizaje automático

En la era del big data y la inteligencia artificial, la capacidad de procesar y analizar grandes volúmenes de información ha cobrado una relevancia sin precedentes. Los datos son generados cada vez a mayor velocidad, provenientes de una gran variedad de fuentes, como redes sociales, dispositivos IoT, transacciones comerciales, entre otras. Sin embargo, la verdadera utilidad de estos datos no radica en su mera recolección, sino en la capacidad de extraer conocimiento valioso de ellos. Es en este contexto donde el aprendizaje automático o machine learning desempeña un papel crucial, proporcionando técnicas y algoritmos que permiten descubrir patrones ocultos y generar modelos predictivos a partir de los datos.

En el aprendizaje automático, se distinguen varios enfoques según el tipo de datos y el objetivo del aprendizaje. Los más comunes son el aprendizaje supervisado y el aprendizaje no supervisado. El primero requiere datos etiquetados y se utiliza en tareas como la clasificación y regresión, donde el modelo aprende a partir de ejemplos conocidos. El aprendizaje no supervisado, en el que se basa el agrupamiento, no necesita etiquetas previas. Aquí, los algoritmos buscan identificar patrones o estructuras ocultas en los datos, agrupando elementos similares sin una respuesta correcta predeterminada. También tienen importancia otras ramas como el aprendizaje por refuerzo, que entrena un agente recibiendo recompensas o castigos; el aprendizaje semi-supervisado, que combina datos etiquetados y no etiquetados; y el aprendizaje profundo, en el que se utilizan redes neuronales.

El agrupamiento de datos o clústering tiene gran importancia dentro de la rama de aprendizaje automático no supervisado, junto a la reducción de dimensionalidad. La finalidad de los procedimientos de agrupamiento es encontrar una estructura o patrones en unos datos sin etiquetas. Un grupo o clúster es, por tanto, un subconjunto de los datos similares entre sí y poco similares a los del resto de clústeres.

En general, se puede usar métodos de agrupamiento para compresión de imágenes [1], segmentación de clientes en marketing [2], identificación de comunidades en redes sociales [3], categorizar el espectro de observaciones astronómicas [4] y análisis de datos biológicos y médicos [5, 6], entre otras muchas aplicaciones. Su versatilidad y capacidad para proporcionar tendencias y perspectivas sin necesidad de supervisión directa lo convierten en una herramienta fundamental para la exploración de datos.

El principal objetivo de este trabajo es estudiar a fondo dos de los procedimientos de agrupamiento (clustering) más utilizados [7]: k-medias (k-means) y el agrupamiento espacial basado en densidad de aplicaciones con ruido o Density-based spatial clustering of applications with noise, que a partir de ahora se denominará por sus siglas en inglés DBSCAN. Este trabajo comenzará mencionando algunos detalles sobre los métodos de agrupamiento, centrándose más en los dos mencionados en las secciones siguientes, hablando de sus características y ventajas, así como ofreciendo algunos resultados de convergencia. Posteriormente, se aplicarán los algoritmos a unos datos de ejemplo sobre los países del mundo para ilustrar su funcionamiento y eficacia. Para finalizar, se expondrán unas conclusiones sobre el trabajo.

1.1. Introducción al aprendizaje no supervisado

A lo largo de este apartado, se introducirán algunos conceptos sobre el aprendizaje no supervisado, agrupamiento de datos y reducción de la dimensionalidad, ofreciendo contexto sobre lo que posteriormente se desarrollará más en profundidad.

Para un problema normal de agrupamiento, los datos se organizarán generalmente en tablas. En ellas, por filas se distribuirán las m observaciones y en las columnas se mostrarán los diferentes atributos o variables de estas (llamemos n al cardinal). Cada fila será por tanto un vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. Un ejemplo típico, usando datos de un banco, por filas se distribuirán los clientes (observaciones) y por columnas sus atributos, como ingresos, número de tarjetas, etc. En este caso, un clúster es un subconjunto de estas observaciones o clientes. También se creará un nuevo atributo indicando a qué clúster pertenece cada observación. En secciones posteriores, los datos de ejemplo a los que se les aplicarán los algoritmos, se organizarán de este modo. En ellas, a las observaciones se les podrá llamar también puntos.

Otro aspecto de gran importancia es que los datos deben estar dotados de una medida que cuantifique la similitud entre ellos. Diferentes elecciones de esta medida podrán conducir a diferentes soluciones, para el mismo conjunto de datos, siendo por tanto importante escoger la óptima para cada problema. La forma más cómoda y común de elegir esta medida es elegir la distancia euclídea (o su cuadrado), aunque también pueden usarse otras como la de Minkowski para algún p [8, 9]:

$$d(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

Asimismo, también pueden usarse formas cuadráticas para definir medidas de disimilitud. Sin embargo, la distancia euclídea ($p = 2$) será la que se empleará a lo largo de este trabajo. Se pueden definir también medidas de similitud para variables cualitativas, pero en este caso, conviene tener conocimiento adicional sobre la variable y crear de manera manual la matriz de distancias [10]. Sin embargo, no se tratarán variables de este tipo a lo largo del trabajo.

Por otro lado, para que todas las variables tomen una importancia por igual en este tipo de medidas, es necesario aplicar la tipificación o z-score (o alguna similar), haciendo que cada variable siga una distribución de media 0 y desviación estándar 1. Además, en general, los conjuntos de datos pueden tener algunas variables con fuerte correlación entre ellas, generando redundancia y perjudicando el rendimiento de los algoritmos posteriores. Por ello, adquiere gran importancia el concepto de reducción de la dimensionalidad, determinando las variables que ofrecen mayor información sobre las observaciones.

Existen varios métodos muy usados para realizar agrupamiento, desde el k-medias o DBSCAN hasta métodos jerárquicos acumulativos y otros métodos probabilísticos que usan el algoritmo esperanza-maximización (expectation maximization, EM). Otros métodos ofrecen además la posibilidad de tener grados de permanencia a varios clústeres como el k-medias difuso (fuzzy k-means).

Como resultado del procesado de los datos, se presentan los diferentes clústeres obtenidos, con los números de elementos que tiene cada uno, sus centroides, varianza intra-clúster, entre otros parámetros. Además, a cada observación se añade una variable extra indicando el índice del clúster al que pertenece, o tantas variables como clústeres para métodos difusos, en la que cada una guarda el grado de permanencia a un determinado clúster.

Capítulo 2

Métodos de agrupamiento de datos

2.1. Consideraciones previas al agrupamiento

2.1.1. Índice de Hopkins

Antes de aplicar una técnica de agrupamiento a un conjunto de datos, conviene comprobar si los datos están organizados de manera que se puedan inferir grupos de ellos, y no sean una nube uniforme de puntos. Para ello, existen estadísticos como el índice de Hopkins, propuesto para comprobar la aleatoriedad de un conjunto de datos. Consiste en un test de hipótesis en el que la hipótesis nula H_0 asume que los datos se distribuyen de manera aleatoria mientras que la alternativa H_1 propone que tienen cierta tendencia a formar grupos [11]. Para realizar este test, se definen dos subconjuntos de datos de tamaño k, con $k \leq 0{,}1m$, es decir, como mucho con un tamaño del 10 % de los datos originales, para asegurar que las $2m$ distancias son independientes y un mismo elemento no es el vecino más cercano de más de un elemento. El primero de los conjuntos anteriores, $\hat{X}$ proviene de los propios datos originales X; y el otro, Y se obtiene de manera uniformemente aleatoria sobre el espacio de valores que definen los datos X. Además, se definen las distancias u_i del elemento i de $\hat{X}$ al vecino más cercano en X; y w_i entre el elemento i de Y de nuevo al vecino más cercano en X.

Definición 2.1.1 (Estadístico de Hopkins)**.** *Sea d la dimensión (número de columnas o atributos) de los datos, el estadístico de Hopkins se define por:*

$$H = \frac{\sum_{i=1}^{k} u_i^d}{\sum_{i=1}^{k} u_i^d + \sum_{i=1}^{k} w_i^d} \tag{2.1.1}$$

Lema 2.1.2 (Distribución del estadístico de Hopkins [11–14])**.** *El estadístico de Hopkins para una muestra de tamaño k de los datos, sigue una distribución $Beta(k, k)$.*

Aunque no se añade la demostración, en el Anexo A se realizan simulaciones para diferentes grupos de datos y comprobar que se cumple el lema 2.1.2 evaluando la distribución que siguen los estadísticos de Hopkins.

Es necesario recalcar que, aunque se obtengan valores de este índice muy cercanos a 1, no se asegura que la forma de los clústeres vaya a ser clara, simplemente implica que los datos no son completamente uniformemente aleatorios.

2.1.2. Reducción de la dimensionalidad

La identificación de patrones y agrupaciones es crucial en el aprendizaje no supervisado, especialmente si se tratan datos con muchas variables. Aunque la intuición humana puede detectar grupos en pocas dimensiones, esto se complica enormemente en el resto de casos. Por ello, reducir la dimensionalidad de forma efectiva es esencial para preservar las relaciones clave entre los datos y facilitar el entendimiento y procesado de los mismos. Un método común para representar datos de muchas dimensiones en menos (por ejemplo en dos o tres dimensiones) se denomina análisis de componentes principales (ACP o PCA en inglés).

El objetivo principal de este proceso es ser capaz de representar la mayor parte de la información de los datos, utilizando un número menor de variables (componentes) nuevas. Una vez calculadas, las componentes se ordenan según la cantidad de información que ofrecen sobre los datos, lo cual es equivalente a ordenarlas según la cantidad de varianza original que describen. Esto es así porque se relaciona una mayor cantidad de información que ofrece una componente con un aumento en la variabilidad de los datos en tal componente. Estas nuevas variables se construyen mediante una transformación lineal de las primeras y de tal manera que no tendrán correlación lineal entre ellas.

Las componentes principales se calcularán a partir de la matriz de covarianzas, obteniendo sus vectores y valores propios. Alternativamente, si los datos no son dimensionalmente homogéneos o presentan diferentes órdenes de magnitud, también se puede proceder mediante la matriz de correlación. Los vectores propios determinarán las componentes mientras que valores propios determinarán la fracción de varianza que es explicada por ella [15,16], por lo que se tomarán las que presenten este valor mayor.

Teniendo en mente lo anterior y usando el teorema espectral, se puede construir una base de vectores propios ortogonales. El teorema espectral [17] garantiza que cualquier matriz cuadrada simétrica con coeficientes reales es diagonalizable y además pueden elegirse los vectores propios de forma que sean ortonormales. De esta manera, se pueden representar los datos en las coordenadas de estos vectores propios, mediante un cambio de base $\{p_1, \ldots, p_n\}$. Llamando Y a las coordenadas de los datos en tal base, los datos se pueden poner de la forma descrita en la ecuación 2.1.2, donde P guarda los vectores propios por filas. En otras palabras, se realiza el producto escalar entre tales vectores y los datos, lo que es equivalente a proyectar. Notar que se interpreta que X son las coordenadas de los datos en la base canónica. Por otro lado, después de la flecha en la ecuación 2.1.2 se representan los datos mediante las coordenadas en la base $\{p_1, \ldots, p_n\}$ y se muestra cómo al elegir solamente r vectores propios (componentes) en la proyección, se introduce un término del error correspondiente a la proyección en el resto de componentes.

$$Y_{coord} = X_{coord}P \quad \longrightarrow \quad X = \sum_{i=0}^{n} y_i \cdot p_i^T = \sum_{i=0}^{r} y_i \cdot p_i^T + E \tag{2.1.2}$$

Tomando la última igualdad de la ecuación 2.1.2 y se elegirán los r vectores propios de tal forma que sus valores propios sean los mayores y por tanto representen la mayor información de los datos. Así se procederá en las secciones posteriores.

2.2. K-medias (K-means)

K-medias es un método de agrupamiento de datos cuyo objetivo es realizar una partición de un conjunto de observaciones en exactamente k clústeres. Esta partición se realiza de forma se minimice cierta función que representa las disimilitudes entre las observaciones y los centros de los respectivos clústeres. El número de clústeres k queda fijado antes de realizar la partición.

El algoritmo parte de un conjunto de observaciones $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m)$ donde $\mathbf{x}_i$ es un vector de $\mathbb{R}^n$ y un entero k con $2 \leq k \leq m$. El objetivo será encontrar una partición $\mathbf{C} = \{C_1, C_2, \ldots, C_k\}$ que cumpla:

$$\arg\min_{\mathbf{C}} \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in C_i} d(\mathbf{x}_j, \mu_i) \tag{2.2.1}$$

donde μ_i representa el centroide (punto medio) del clúster C_i y d es una medida de disimilitud entre dos puntos. De esta manera, se pretenderá que cada dato pertenezca al clúster cuyo valor medio es el más cercano. Usualmente, como función de disimilitud se toma la distancia euclídeas al cuadrado, debido a su sencillez computacional [9]. En ese caso, la expresión 2.2.1 coincide

con la suma de las varianzas intra-clúster, es por ello que cobre sentido que este método busque minimizarla.

El algoritmo de k-medias [9, 18] parte de k centroides inciales (posteriormente se hablará sobre cómo optimizar este paso) y la misma cantidad de clústeres. Tras sucesivas reasignaciones de estos con el fin de minimizar la función de la ecuación 2.2.1, se alcanza finalmente un conjunto de clústeres y centroides que no cambia en la siguiente iteración. Se considera entonces que el algoritmo ha convergido. En algunas implementaciones del algoritmo, como el realizado por la librería Scikit-learn [19], se dice que ha convergido cuando la suma de distancias entre los centroides de dos iteraciones consecutivas es menor que cierta tolerancia. La definición 2.2.1 describe el algoritmo usualmente usado en el caso de que la distancia sea la euclídea. Para el caso más general se tiene la reformulación del mismo en la definición 3.1.12.

Definición 2.2.1 (Algoritmo de K-medias)**.** *Se denomina* ***algoritmo de K-medias*** *a:*

1. *Se parte de unos centroides iniciales* $\{\mu_i^{(0)}\}_{i=1}^k$, $t = 0$.

2. *En cada iteración* t *se asignan las observaciones a los clústeres cuyo centroide sea el más cercano:*
$$S_j^{(t)} = \{x_i : d(x_i, \mu_j^{(t)}) \leq d(x_i, \mu_p^{(t)}) \quad \forall p = 1, \ldots, k\}$$

3. *Se actualizan los centroides como el punto medio de cada clúster. Si los centroides no cambian, finaliza el algoritmo, en otro caso, se vuelve al paso 2 con la siguiente iteración* $t + 1$.
$$m_j^{(t+1)} = \frac{1}{|S_j^{(t)}|} \sum_{x_i \in S_j^{(t)}} x_i$$

La selección de los centroides iniciales (inicialización de k-medias) es determinante en la calidad de los clústeres y velocidad de convergencia posterior, por ello conviene optimizar este paso. Históricamente, se han usado varios procedimientos para realizar esta inicialización [20]: el método de Forgy directamente elige aleatoriamente k observaciones del conjunto de datos como centroides iniciales. Otros métodos como el de partición aleatoria primero asigna aleatoriamente un clúster para cada observación y después calcula los centroides a partir de ellos.

No obstante, la inicialización k-medias++ [18] es el método más comúnmente utilizado en la actualidad. Este procedimiento también elige los centroides iniciales del conjunto de datos pero maximizando la distancia entre ellos, no aleatoriamente como el caso de Forgy. Esto se consigue partiendo de una observación inicial y eligiendo una observación como siguiente centroide con probabilidad proporcional a la distancia al cuadrado del centroide ya asignado más cercano. K-medias++ es la inicialización que utiliza por defecto Scikit-learn [19].

Aunque se profundizará en la convergencia de este algoritmo en la sección 3.1, al ser de carácter heurístico, no hay garantías de la convergencia al óptimo global. Por ello, dada la rapidez de ejecución (en general) del algoritmo conviene ejecutarlo varias veces, tomando el conjunto de clústeres de mayor calidad. Si como medida de disimilitud se toma la distancia euclídea, se suele minimizar la suma residual de cuadrados o inercia para cuantificar la bondad de los clústeres.

Una limitación importante de k-medias es que busca y genera clústeres esféricos. De esta manera, estará especialmente indicado para ciertos conjuntos de datos y no para otros, como se podrá ver más adelante en la imagen 2.3.1. Además, el número de clústeres k es un parámetro elegido de antemano por lo que será necesario tomarlo adecuadamente, usando métodos como el comentado en la sección 2.4.1.

2.2.1. K-medias difuso (Fuzzy K-means)

K-medias difuso pertenece a una clase de métodos de agrupamiento en los que la definición de pertenencia a un clúster se ve ampliada: una misma observación puede estar contenida parcialmente en varios clústeres. Se definen funciones de pertenencia u_{ij} representando la fracción de pertenencia de la observación $\mathbf{x}_i$ al clúster C_j. En este caso, la función objetivo a minimizar $J_{general}$ también se ve ampliada de la forma de la ecuación 2.2.2, donde r es el parámetro de difusión [21] [22]. Se añade el parámetro r porque si $m \to 1$ el conjunto de clústeres óptimo es cada vez más cercano a una partición exclusiva. Es decir, si no se introduce el parámetro, se tiene un resultado similar al k-medias usual. Por otro lado, cuando $m \to \infty$ la partición óptima se aproxima a la matriz de pertenencia con todos sus valores iguales a $1/k$.

$$J_{general} = \sum_{j=1}^{k}\sum_{i=1}^{m} u_{ij}^{r}\ d(\mathbf{x}_i, \mu_j) \qquad J_{entropía} = \sum_{j=1}^{k}\sum_{i=1}^{m} u_{ij}\ d(\mathbf{x}_i, \mu_j) + \lambda \sum_{j=1}^{k}\sum_{i=1}^{m} u_{ij}\log(u_{ij}) \quad (2.2.2)$$

Otra opción [23] [24] es introducir un sumando extra proporcional a la entropía cambiada de signo ($\lambda > 0$) en el caso de un sistema clásico y discreto [25]. En este caso discreto la entropía se maximiza cuando todas las probabilidades son iguales, por lo que el segundo sumando de $J_{entropía}$ se minimizará cuando la pertenencia a los clústeres es la misma. Es por ello que para minimizar la función completa $J_{entropía}$ se obtendrá una opción intermedia entre la pertenencia a un único clúster y la pertenencia a todos por igual.

El procedimiento que realiza el algoritmo de k-medias difuso en ambos casos tiene una estructura similar al de k-medias (definición 2.2.1), aunque es necesario un paso intermedio extra para calcular la matriz de pertenencias, además de una variación en el cálculo de los centroides.

Finalmente, existen también versiones para datos que varían con el tiempo, en las que las funciones de disimilitud se integran en función del tiempo [26]. En estos casos, la pertenencia a los clústeres no cambia con el tiempo aunque sí la disimilitud entre las observaciones o entre ellas y los centroides. Es por ello que es necesario integrar esta cantidad en el tiempo para obtener una disimilitud promedio.

2.2.2. Implementación personal de k-medias en Python

K-medias, de la misma forma que DBSCAN, es ampliamente utilizado en la comunidad científica y del análisis de datos. Debido a esto, varias librerías se han encargado de su correcta y eficiente implementación, como es el caso de la librería Scikit-learn. En concreto, los códigos en python de la implementación son abiertos, tanto para k-medias [19] como para DBSCAN [27]. K-medias lo implementa en unas 2300 líneas, mientras que DBSCAN lo hace en algo menos de 500 líneas. Es por ello que, para facilitar el entendimiento del código, se ha realizado una implementación personal del mismo, también en python, en el anexo E. Aunque claramente este código no será óptimo ni cubrirá gran cantidad de casos diferentes como el proporcionado por la librería, sí funcionará correctamente.

El procedimiento necesario para aplicar el algoritmo k-medias a un conjunto de datos 2.2.1 se mencionó en la sección 2.2 y su implementación directa en python no es de gran complejidad, como se ve en el anexo E.1. Se ha utilizado la misma nomenclatura que el código de Scikit-learn [19] para los nombres de las variables y parámetros (max_iter, n_clusters, ...) así como para varias funciones similares, aunque notablemente simplificadas.

Cesar_Simple_KMeans, al igual que el código de Scikit-learn [19], se definirá como clase. Esta poseerá tres funciones principales: *_compute_distances* que, para cada observación, de-

vuelve las distancias a los k centroides; *fit* que aplica el algoritmo 2.2.1 al conjunto de datos, devolviendo las etiquetas de los clústeres a cada punto, usando la inicialización de Forgy. Finalmente, *predict* se encarga de devolver etiquetas también, pero pensada para datos nuevos, una vez fijados los clústeres para los anteriores. Los parámetros de entrada son el número de clústeres, el máximo de iteraciones a realizar y el estado aleatorio, por si se quiere fijar la semilla al incializar los centroides.

Con lo anterior, el código de *Cesar_Simple_KMeans* parece suficiente para funcionar correctamente y así lo hace. Sin embargo, se pueden añadir algunos cambios para agilizar y facilitar la convergencia como se hace en *Cesar_Improved_KMeans*. Esta clase extiende a *Cesar_Simple_KMeans* añadiendo dos parámetros más: tolerancia para suponer que ha convergido cuando los centroides difieren menos que ella y agilizar la convergencia; y número de inicializaciones, de manera que el algoritmo elige como ajuste el mejor de todos ellos. Para cuantificar qué ajuste es el mejor se introduce la variable inercia, también presente en Scikit-learn [19], representando la suma residual de cuadrados, de manera que escogerá el ajuste con menor inercia. Además, se crea una nueva función: *_init_centroids* que sustituirá la inicialización de Forgy de *Cesar_Simple_KMeans* por k-medias++. Utilizando este método se consiguen elegir centroides más alejados, eligiéndolos con probabilidades proporcionales al cuadrado de la distancia a los ya elegidos, como se mencionó en la sección 2.2.

2.3. DBSCAN

DBSCAN (Density-Based Spatial clustering of Applications with Noise, o agrupamiento espacial basado en densidad de aplicaciones con ruido, en español) es un algoritmo de agrupamiento que tiene en cuenta la densidad de los puntos. Es especialmente útil para identificar grupos de datos de forma arbitraria y también para detectar ruido (puntos que no pertenecen claramente a ningún clúster). El principio fundamental de DBSCAN es que los clústeres son definidos como regiones con alta densidad de puntos en el espacio de las variables, mientras que las áreas con baja densidad de puntos se consideran ruido. Este algoritmo de agrupamiento, a diferencia de muchos otros, introduce el concepto de ruido que asigna a los datos que no pertenecen a ningún clúster. De esta manera, los datos atípicos no afectan a los parámetros de los clústeres resultantes, no como ocurre en otros métodos como el k-medias. Es por ello que este algoritmo está especialmente recomendado a conjuntos de datos en los que la probabilidad de contener este tipo de datos es alta. A continuación, se detallan los conceptos clave y más adelante se explicará el funcionamiento del algoritmo [28–31].

Definición 2.3.1 (Radio y MinPts)**.** ***Eps (ϵ o radio)*** *es el radio que define el vecindario de un punto. Dos puntos se consideran vecinos si la distancia entre ellos es menor o igual a ϵ. Por otro lado,* ***MinPts (mínimo de puntos)*** *es el número mínimo de puntos requeridos dentro de un radio ϵ para que un punto se considere un core point (punto núcleo).*

Tras definir los dos parámetros clave de este algoritmo se pueden definir también las tres clases de puntos que se asignarán tras la aplicación del método de agrupamiento.

Definición 2.3.2 (Puntos núcleo, borde y ruido)**.** *Los* ***puntos núcleo (core points)*** *serán aquellos puntos que tienen al menos MinPts puntos en su vecindario dentro del radio ϵ. De forma análoga, se dirá que son* ***puntos frontera (border points)*** *los puntos que no cumplen con la condición anterior sobre MinPts por sí mismos, pero que están dentro del vecindario de un punto núcleo. Finalmente, los* ***puntos de ruido (outliers)*** *acaban siendo puntos que no son ni puntos núcleo ni de borde. Estos puntos se consideran ruido y no se asignan a ningún clúster.*

Es decir, según la definición 2.3.2, si un punto tiene suficientes vecinos es punto núcleo, pero si no los tiene depende de si este es vecino de otro punto núcleo o no. Si es vecino de un núcleo, el punto sería asignado como punto borde del clúster correspondiente al punto núcleo, mientras que en otro caso sería considerado como ruido.

Definición 2.3.3 (Algoritmo de DBSCAN)**.** *Se denomina* ***algoritmo de DBSCAN*** *a:*

1. *Seleccionar un punto no visitado del conjunto de datos.*
2. *Buscar todos los puntos dentro de un radio ϵ alrededor del punto seleccionado y comprobar si es un* ***punto núcleo****. En ese caso se inicia un nuevo clúster con sus vecinos.*
3. *Analizar cada vecino no visitado del clúster de la misma manera. Si un vecino es también un* ***punto núcleo****, el clúster se expande a través de él, añadiendo a sus vecinos. Si no lo es, el punto igualmente pertenecerá al clúster pero como un* ***punto frontera*** *del mismo.*
4. *Si el punto inicialmente seleccionado no cumple la condición del mínimo de vecinos (MinPts), se clasifica como* ***ruido****. Esta asignación podrá cambiar si en un paso posterior se descubre que el punto es vecino de un núcleo, pasando a ser un punto frontera.*
5. *Repetir el proceso con el siguiente punto no visitado hasta que todos los puntos hayan sido procesados.*

Este tipo de algoritmos basados en densidad de puntos tienen la ventaja de que detecta clústeres de formas arbitrarias, al no asumir ninguna estructura de clúster (como esferas o elipses, como el k-medias). Además, es un algoritmo robusto frente a ruido, ya que identifica y aísla puntos atípicos. A esto se le suma que no es necesario especificar previamente el número de clústeres al comenzar el procedimiento. Sin embargo, presenta el inconveniente de que la elección de los parámetros ϵ y *MinPts* es fundamental y puede ser difícil de ajustar para diferentes conjuntos de datos. A esto se le suma que puede tener problemas para identificar clústeres con densidades muy variadas. Además, no es un método completamente determinista, ya que los puntos frontera que son alcanzables desde más de un clúster pueden asignarse a cualquiera de ellos, dependiendo del orden de creación de los clústeres. No obstante, existen variantes del método, como DBSCAN* [32], que toman los puntos borde como ruido, consiguiendo que el algoritmo sea completamente determinista.

Por último, en la figura 2.3.1 se representan los clústeres formados por los algoritmos protagonistas de este trabajo: DBSCAN y k-medias, sobre tres conjuntos de datos simulados de una forma especialmente pensada para mostrar las limitaciones de k-medias frente a DBSCAN. Para el conjunto de datos intermedio (nubes de puntos esféricas y separadas) ambos algoritmos obtienen de manera correcta los clústeres. No obstante, no ocurre lo mismo en los otros dos conjuntos de datos, en los que la necesidad de k-medias de generar clústeres esféricos, provoca que el ajuste no sea el óptimo en estos tipos de datos. Sin embargo, en conjuntos de datos reales, no salen a la luz estas limitaciones de manera tan clara.

2.3.1. Implementación personal de DBSCAN en Python

La implementación del algoritmo de DBSCAN de la definición 2.3.3 es más directa que para el caso de k-medias. El código creado se muestra en el anexo E.3, con los nombres de las variables y algunas funciones similares (aunque simplificadas) al código de Scikit-learn [27] para facilitar el entendimiento.

Cesar_DBSCAN, al igual que *Cesar_Simple_KMeans* y el código de Scikit-learn, se definirá como clase. Esta poseerá tres funciones principales: *_region_query* que devuelve la

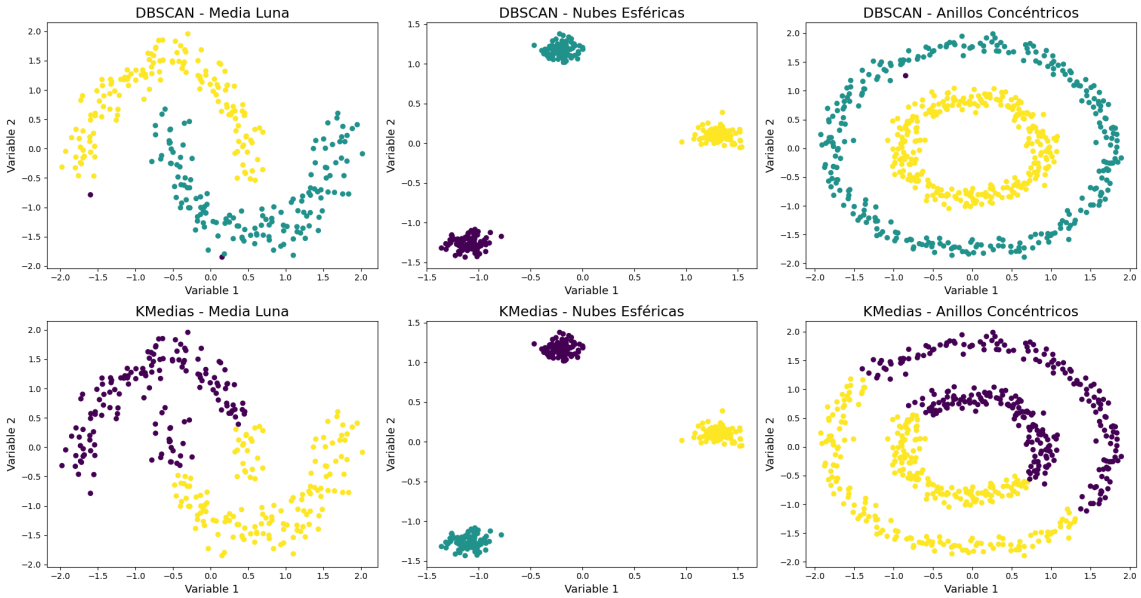


Figura 2.3.1: Datos simulados en python con una distribución en el plano especialmente elegida para comparar los clústeres que generan k-medias y DBSCAN.

vecindad-Eps de un punto; *_expand_cluster* que expande el clúster dado un punto núcleo y *fit* que crea un nuevo atributo en la clase con las etiquetas de los clústeres o de ruido.

Para aplicar el algoritmo hace falta especificar *Eps, MinPts* y el conjunto de datos. Notar que al comprobar si un punto es núcleo, se calcula el cardinal de la vecindad-Eps del mismo, pero hay que tener en cuenta que en la vecindad está el propio punto. Si se establece *MinPts*=1 todos puntos serían núcleos.

Como se comentó en la sección 3.2, no hay ambigüedad en la convergencia (excepto en los puntos frontera) de DBSCAN. Además, tanto el código de Scikit-learn [27] como el implementado en el anexo E.3 visitan los puntos en el orden del propio conjunto de datos, se obtienen exactamente las mismas etiquetas de los clústeres para varios datos de ejemplo, en particular para el usado en la sección 4.1. Si se cambiase el orden, algunos puntos frontera podrían cambiar de clúster (se quedarían en el primer clúster creado tal que un núcleo sea su vecino) y, además, podría hacer una permutación en las etiquetas de los clústeres.

2.4. Evaluación de los resultados

El principal desafío del aprendizaje no supervisado radica en la ausencia de un subconjunto de datos correctamente etiquetados para evaluar la calidad del ajuste. Es por ello que es necesario introducir estadísticos que pretendan cuantificar la calidad de los clústeres o si el número de ellos es el adecuado. Para el primer aspecto se utilizará el método de la silueta (silhouette) y para el segundo se utilizará el método del codo.

2.4.1. Método del codo

Dos buenos candidatos para cuantificar la dispersión dentro de los clústeres son la suma residual de cuadrados y el error cuadrático medio, cuya única diferencia es dividir por el número de observaciones. Estos estadísticos son usados ampliamente en la literatura científica [33, 34]. También es usado por ejemplo en la implementación de k-medias en la librería Scikit-learn [19] para comprobar qué inicialización genera mejores clústeres. Se verá cómo va-

ría este error cuadrático medio al aumentar el número de clústeres, siguiendo una tendencia claramente decreciente. Sin embargo, interesará tomar el número de clústeres tal que tal decrecimiento del error cuadrático medio se frena claramente, quedando en la gráfica 2.4.1 una forma similar a un codo. El punto marcado en rojo es el que se tomaría como óptimo, siendo un número de clústeres no demasiado alto y un error cuadrático medio suficientemente bajo.

2.4.2. Método de la silueta (silhouette method)

Como se ha mencionado anteriormente, es necesario cuantificar la calidad y coherencia dentro de los clústeres. Para ello, se introducen conceptos como el método de la silueta [35]. En él se compara la similitud entre un objeto y los elementos de su clúster con los elementos del resto de clústeres.

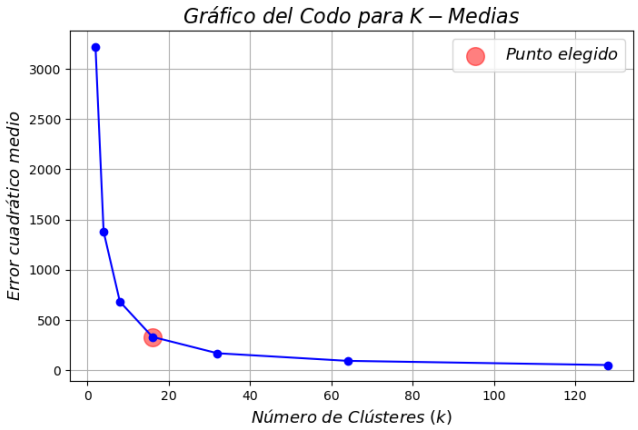


Figura 2.4.1: Aplicación del método del codo. Se ha utilizado el conjunto de datos de los píxeles de la imagen de la sección 4.2.

A partir de ahora, se tomará un conjunto de datos en el que se haya aplicado un método para formar clústeres, si hay puntos etiquetados como ruido se omiten en este proceso. Sea $x_i \in C_I$ (una observación en el clúster I), se define:

$$a_i = \frac{1}{|C_I| - 1} \sum_{x_j \in C_I, i \neq j} d(x_i, x_j) \qquad b_i = \min_{K \neq I} \frac{1}{|C_K|} \sum_{x_j \in C_K} d(x_i, x_j)$$

a_i representa la distancia media entre x_i y los demás puntos contenidos en su propio clúster. Se divide por $|C_I| - 1$ porque no se incluye la distancia $d(x_i, x_i)$ en la suma. Por otro lado, b_i cuantifica la distancia media entre x_i y el clúster más cercano al que no pertenece. Con estos dos parámetros se puede definir el coeficiente de la silueta.

Definición 2.4.1 (Coeficiente de la silueta de una observación)**.** *Se define el* ***coeficiente de la silueta para una observación*** x_i *como la diferencia relativa entre las dos cantidades* a_i *y* b_i*, es decir:*

$$s_i = \frac{b_i - a_i}{\max\{a_i, b_i\}}$$

en el caso de que $|C_I| < 1$*. Si* $|C_I| = 1$ *se define* $s_i = 0$ *para evitar que el número de clústeres aumente demasiado.*

Este estadístico toma valores entre -1 y +1 para cada observación, donde un valor cercano a 1 ($a_i \ll b_i$) indica que este punto está correctamente asignado a su clúster y claramente separado de otros. Si la mayoría de las observaciones tienen un valor cercano a 1, entonces se considera que la configuración del clúster es apropiada. Si s_i está cerca de -1, entonces de la misma manera se ve que x_i estaría mejor agrupado en su clúster vecino. Un s_i cercano a cero significa que el dato está en el borde de dos clústeres.

También es frecuente utilizar la media de s_i sobre todo el conjunto de datos. Este estadístico cuantifica lo mismo que en el caso de una única observación, pero tomando una visión global sobre todos los datos.

Capítulo 3

Convergencia de los algoritmos

A lo largo de este apartado, se centrará el estudio de convergencia en los algoritmos k-medias [9] y DBSCAN [28].

3.1. K-medias

Para abordar esta sección, se dará una versión más precisa del problema que pretende resolver el algoritmo, mediante la definición 3.1.1.

Definición 3.1.1 (Problema P)**.** *Sean $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m \in \mathbb{R}^n$ un número finito de observaciones y sea k con $2 \leq k \leq m$ el número de clústeres, entonces el problema P viene dado por:*

$$\begin{aligned} P: \textit{minimizar} \quad & f(W,Z) = \sum_{i=1}^{k}\sum_{j=1}^{m} w_{ij} D(x_j, z_i) \\ \textit{sujeto a} \quad & \sum_{i=1}^{k} w_{ij} = 1, \qquad j = 1, \ldots m \\ & w_{ij} \in \{0,1\}, \quad i = 1, \ldots k,\ j = 1, \ldots m \end{aligned} \tag{3.1.1}$$

Donde $W = [w_{ij}]$ es una matriz real de tamaño $k \times m$, $Z = [z_1, z_2, \ldots, z_k] \in \mathbb{R}^{nk}$ son los centros de los clústeres (centroides), y $D(x_j, z_i)$ es una medida de disimilitud entre x_j y z_i.

Notar que las condiciones $w_{ij} \in \{0,1\}$ y $\sum_{i=1}^{k} w_{ij} = 1$ se incluyen para que cada observación pertenezca exactamente a un clúster, concretamente si $w_{ij} = 1$, la observación j pertenece al clúster i. En otras versiones de k-medias, como el fuzzy k-means o k-medias difuso (sección 2.2.1), se relaja la condición $w_{ij} \in \{0,1\}$, quedando $w_{ij} \in [0,1]$. De esta manera, cada observación puede pertenecer parcialmente a varios clústeres y w_{ij} toma el papel de función de permanencia de la observación j al clúster i. Los valores cercanos a uno indican una mayor similitud, mientras que los cercanos a cero indican una menor similitud. El problema P, en general, no cumple las condiciones de convexidad por lo que un mínimo local no tiene por qué ser global.

El algoritmo 2.2.1 de k-medias se podría interpretar en términos de la notación introducida anteriormente, para una función de disimilitud cualquiera, de la siguiente manera:

1. Comenzar con un conjunto de centroides iniciales $z^{[1]}$, $i = 1, 2, \ldots, k$ $(l = 1)$.
2. Asignar cada observación x_j, $j = 1, 2, \ldots, m$, a su centroide más cercano, lo que equivale a fijar los valores de w_{ij}.
3. Recalcular los centroides $z_i^{[l+1]}$, $i = 1, 2, \ldots, k$, minimizando $F(W, Z)$. Si $z^{[l+1]} = z^{[l]}$, $i = 1, 2, \ldots, k$, finaliza el algoritmo; de lo contrario, se vuelve al paso 2.

Definición 3.1.2 (Función Reducida)**.** *La función reducida $F(W)$ del Problema P se define como:*

$$F(W) = \text{mín}\{f(W,Z) : Z \in \mathbb{R}^{nk}\}$$

donde W es una matriz real de tamaño $k \times m$.

Lema 3.1.3 (Concavidad de la Función Objetivo Reducida)**.** *La función objetivo reducida F es cóncava.*

Demostración: Sean W_1 y W_2 dos puntos cualesquiera y sea γ un escalar tal que $0 \leq \gamma \leq 1$. Entonces:

$$\begin{aligned} F(\gamma W_1 + (1-\gamma)W_2) &= \min\{f(\gamma W_1 + (1-\gamma)W_2, Z) : Z \in \mathbb{R}^{nk}\} \\ &= \min\{\gamma f(W_1, Z) + (1-\gamma) f(W_2, Z) : Z \in \mathbb{R}^{nk}\} \\ &\geq \gamma \min\{f(W_1, Z) : Z \in \mathbb{R}^{nk}\} + (1-\gamma)\min\{f(W_2, Z) : Z \in \mathbb{R}^{nk}\} \\ &= \gamma F(W_1) + (1-\gamma)F(W_2) \end{aligned}$$

En la segunda igualdad se ha usado que $f(W, Z)$ es lineal. Por lo tanto, F es cóncava. □

Teorema 3.1.4. *Consideremos el conjunto S dado por:*

$$S = \left\{ W : \sum_{i=1}^{k} w_{ij} = 1,\ j = 1, 2, \ldots, m,\ w_{ij} \geq 0 \right\}$$

Un punto es extremo de S si y solo si satisface las restricciones del problema P.

Demostración: Usando notación de programación lineal, cada punto extremo de S está asociado a una base. Cualquier base de las restricciones en S es una matriz identidad. Por lo tanto, cada variable básica tendrá valor 1 y las variables no básicas tendrán valor 0. Recíprocamente, si un punto satisface las restricciones de P, tendrá exactamente m variables w_{ij} con valor 1 y el resto 0, de forma que se puede poner de la forma de punto extremo de S. □

Definición 3.1.5 (Problema Reducido)**.** *El problema reducido RP del problema P se define como:*

$$\textit{minimizar } F(W), \textit{ sujeto a } W \in S.$$

Lema 3.1.6. *Los problemas RP y P son equivalentes.*

Demostración: Dado que la función F es cóncava, existe una solución en un punto extremo de S, que satisface las restricciones del problema P. Además, claramente minimizar $f(W, Z)$ en ambos argumentos es equivalente a calcular para cada W el valor mínimo de $f(W, Z)$ variando Z (calcular $F(W)$) y luego calcular el mínimo de ellas (resolver el problema RP). □

Con el teorema anterior se tiene una forma más sencilla En la siguiente sección, se introducirán los puntos que son soluciones parciales óptimas y se mostrará que son puntos de Karush-Kuhn-Tucker [36–38].

3.1.1. Soluciones Parcialmente Óptimas y Puntos de Karush-Kuhn-Tucker

Definición 3.1.7 (Puntos de Karush-Kuhn-Tucker)**.** *Sea el problema:* $\min_{\mathbf{x}\in\mathbb{R}} f(\mathbf{x})$ *sujeto a* $h_i(\mathbf{x}) \leq 0 \quad i = 1, \ldots, m \quad y \quad g_j(\mathbf{x}) = 0 \quad j = 1, \ldots, r$, *con r igualdades y m desigualdades. Los puntos de Karush-Kuhn-Tucker son aquellos en los que existen constantes u_i, v_j tales que:*

$$\textit{(Cond. estacionariedad)} \qquad \nabla f(\mathbf{x}) + \sum_{i=1}^{m} u_i \nabla h_i(\mathbf{x}) + \sum_{j=1}^{r} v_j \nabla g_j(\mathbf{x}) = 0 \tag{3.1.2}$$

$$\textit{(Factibilidad primal)} \qquad h_i(\mathbf{x}) \leq 0 \quad y \quad g_j(\mathbf{x}) = 0 \quad \forall i, j \tag{3.1.3}$$

$$\textit{(Factibilidad dual)} \qquad u_i \geq 0 \quad \forall i \tag{3.1.4}$$

$$\textit{(Holgura complementaria)} \qquad u_i \cdot h_i(\mathbf{x}) = 0 \quad \forall i \tag{3.1.5}$$

Definición 3.1.8 (Soluciones Parcialmente Óptimas)**.** *Un punto (W^*, Z^*) es una solución parcialmente óptima del problema P si satisface:*

$$f(W^*, Z^*) \leq f(W, Z^*) \quad \textit{para todo } W \in S,$$

$$f(W^*, Z^*) \leq f(W^*, Z) \quad \textit{para todo } Z \in \mathbb{R}^{nk}.$$

Definición 3.1.9 (Problemas parciales)**.** *Para obtener una solución parcialmente óptima, se definen los siguientes dos problemas:*

- ***Problema P1:*** *Dado $Z \in \mathbb{R}^{nk}$, minimizar $f(W, Z)$ sujeto a $W \in S$.*
- ***Problema P2:*** *Dado $W \in S$, minimizar $f(W, Z)$ sujeto a $Z \in \mathbb{R}^{nk}$.*

La solución del Problema P1 es trivial, ya que para un j específico, $w_{rj} = 1 \quad \text{si} \quad D(x_j, z_r) \leq D(x_j, z_l), \quad l = 1, 2, \ldots, k$, y $w_{ij} = 0$ para $i \neq r$. Sin embargo, la solución del Problema P2 no es tan directa como la de P1.

Teorema 3.1.10 (Puntos de Karush-Kuhn-Tucker)**.** *Supongamos que $f(W^*, Z)$ es diferenciable en $Z = Z^*$. Entonces, una solución parcialmente óptima (W^*, Z^*) del problema P es un punto de Karush-Kuhn-Tucker.*

Demostración: El vector $\mathbf{x}$ de la definición 3.1.7 se puede descomponer en la parte de las pertenencias W y la parte de los centroides Z. El problema P no presenta restricciones sobre la variable Z, por lo que las condiciones 3.1.7 sobre ellas se limitan a que $\nabla_Z f(W^*, Z^*) = 0$. Esto se cumple porque $f(W^*, Z)$ es diferenciable en $Z = Z^*$ y Z^* minimiza $f(W^*, Z)$ por ser solución de P2. Para la parte de las pertenencias W, las funciones son lineales en W, tratándose por tanto de un problema de programación lineal con restricciones de igualdad, además de la condición de positividad. Por lo anterior, se cumplen las condiciones de factibilidad primal y dual, así como la de holgura complementaria de la definición 3.1.7. Por el teorema B.0.1 del anexo B, existen vectores $\mathbf{u}$, $\mathbf{v}$, con $\mathbf{u} \geq 0$ tales que cumplen la condición de estacionariedad de la definición 3.1.7. En definitiva, el punto (W^*, Z^*) es de Karush-Kuhn-Tucker. □

Tomando el caso de distancia euclídea como medida de disimilitud (la que se usa normalmente), en el anexo B se añade un contraejemplo de convergencia a mínimo local. Consiste en un caso en dos dimensiones en el que el algoritmo de k-medias converge a un punto que no es un mínimo local.

Aplicando los lemas y teoremas sobre el problema RP junto a algún resultado extra [9], se puede demostrar el siguiente teorema para el caso particular de función de disimilitud de forma cuadrática, aunque la llegada al mínimo global en ningún caso está asegurada. Para el resto de funciones de disimilitud no hay resultados tan interesantes.

Teorema 3.1.11. *Considerando el problema P donde $D(x_j, z_i) = (x_j - z_i)'(x_j - z_i)$; entonces, las soluciones parciales óptimas de P son mínimos locales.*

Definición 3.1.12 (Algoritmo 1)**.** *Pretendiendo reformular de algoritmo de k-medias de manera general, se define el* ***algoritmo 1*** *como sigue:*

1. *Elegir un punto inicial $Z^0 \in \mathbb{R}^{nk}$ $(r = 0)$.*
2. *Calcular la solución básica W^r resolviendo $P1$ con $Z = Z^r$.*
3. *Calcular la solución Z^{r+1} resolviendo $P2$ con $W = W^r$. Si $f(W, Z^{r+1}) = f(W, Z^r)$, finaliza el algoritmo y se toma $(W^*, Z^*) = (W^r, Z^{r+1})$. De lo contrario, se sigue con el paso 2.*

Teorema 3.1.13. *El algoritmo 1 dado por la definición 3.1.12 converge a una solución parcialmente óptima del problema P en un número finito de iteraciones. Es esencialmente una reformulación del algoritmo k-medias.*

Demostración: Se mostrará que un punto extremo de S es visitado a lo sumo una vez por el algoritmo antes de detenerse. Supongamos que esto no es cierto, es decir, $W^{r_1} = W^{r_2}$ para algunos r_1 y r_2 donde $r_1 \neq r_2$. Al aplicar el paso 2), se obtienen Z^{r_1+1} y Z^{r_2+1} como soluciones óptimas para $W = W^{r_1}$ y $W = W^{r_2}$, respectivamente, lo que implica que $f(W^{r_1}, Z^{r_1+1}) = f(W^{r_2}, Z^{r_2+1})$. Se ha usado que, dado que $W^{r_1} = W^{r_2}$, aplicar el paso 2) en cada caso obtendrá el mismo valor. Sin embargo, la secuencia $f(\cdot\,,\cdot)$ generada por el algoritmo es estrictamente decreciente, lo que es una contradicción. Por lo tanto, $W^{r_1} \neq W^{r_2}$ y, dado que hay un número finito de puntos extremos en S, el algoritmo alcanza una solución parcialmente óptima en un número finito de iteraciones. □

El hecho de que haya un número finito de puntos extremos en S equivale a que el número de formas de asignar observaciones a clústeres es finita, ya que por el teorema 3.1.4 hay una biyección entre las asignaciones de observaciones con los extremos de S. Además, el cardinal de estos conjuntos lo determina el número de Stirling de segunda especie $S(m, k)$.

3.2. DBSCAN

Debido a la estructura del algoritmo DBSCAN (definición 2.3.3), no se requieren teoremas como el 3.1.13 para garantizar la convergencia en un número finito de iteraciones. Esto es así porque el algoritmo visita los puntos uno a uno y una única vez. Esto último ocurre a menos que en la primera visita se le asigne como punto ruido y en la segunda se le asigne a un clúster por ser vecino de un punto núcleo. En cualquier caso, el número de pasos realizados por el algoritmo es finito.

Por otro lado, en el caso de DBSCAN no existe una función objetivo a minimizar, por lo que tampoco se hablará de conceptos de mínimo local o similares. Sin embargo, sí que se introducirán algunas definiciones frecuentemente usadas en la literatura científica [28–31] así como algunos resultados sobre la independencia de los resultados con el orden con el que se inicia y aplica el algoritmo sobre los puntos.

Definición 3.2.1 (Vecindad-Eps de un punto [28])**.** *La **vecindad-Eps** de un punto p, denotada por $N_{Eps}(p)$, se define como:*

$$N_{Eps}(p) = \{q \in D \mid dist(p, q) \leq Eps\}.$$

Definición 3.2.2 (Nociones basadas en densidad)**.** *Un punto p es **directamente alcanzable por densidad** desde un punto q si q es punto núcleo ($|N_{Eps}(q)| \geq MinPts$) y $p \in N_{Eps}(q)$. Análogamente, un punto p es **alcanzable por densidad** desde un punto q si existe una cadena de puntos $p_1, \ldots, p_n$, con $p_1 = q$, $p_n = p$ tal que p_{i+1} es directamente alcanzable por densidad desde p_i. Finalmente, un punto p está **conectado por densidad** a un punto q si existe un punto o tal que ambos, p y q, son alcanzables por densidad desde o.*

Notar que la relación de ser alcanzable es transitiva pero no es simétrica: ningún punto puede ser alcanzado desde un punto que no sea núcleo, aunque cualquier punto puede ser alcanzable. Por lo tanto, es necesaria la noción de conectividad por densidad para definir formalmente la extensión de un clúster dada por DBSCAN. Esta relación de estar conectados por densidad es simétrica.

Definición 3.2.3 (Clúster)**.** *Sea D una base de datos de puntos. Un clúster C es un subconjunto no vacío de D que satisface las siguientes condiciones:*

1. *$\forall p, q$: si $p \in C$ y q es alcanzable por densidad desde p, entonces $q \in C$. (Maximalidad)*

2. *$\forall p, q \in C$: p está conectado por densidad a q. (Conectividad)*

La definición 3.2.3 formaliza las condiciones que deben cumplir los clústeres que genere el algoritmo, por lo que estos conjuntos deben ajustarse a dicha definición. Por otro lado, si p es un punto núcleo, este forma un clúster junto a otros puntos (núcleo o no) que sean alcanzables desde él. Cada clúster contiene al menos un punto núcleo y por tanto al menos MinPts puntos. Los puntos no núcleos alcanzables pueden pertenecer a un clúster pero actúan como una barrera puesto que no es posible alcanzar más puntos desde estos.

Lema 3.2.4. *Sea p un punto en D y $|N_{Eps}(p)| \geq MinPts$. Entonces, el conjunto $O = \{o \mid o \in D$ y o es alcanzable por densidad desde $p\}$ es un clúster. Recíprocamente, sea C un clúster y sea p cualquier punto núcleo en C ($|N_{Eps}(p)| \geq MinPts$). Entonces, C es igual al conjunto O.*

Demostración: $\Longrightarrow$) por definición, el conjunto O cumple la condición 1. de 3.2.3. Además, cualquier par de puntos $o, q \in O$ son alcanzables por densidad desde p y como el punto p es núcleo, o y q están conectados por densidad. $\Longleftarrow$) sea $p \in C$ núcleo, si C no fuera exactamente O, tendría algún punto extra y/o algún punto menos. En el primer caso, tal punto extra no sería alcanzable por densidad desde p y por lo que no estarían conectados por densidad, incumpliendo la condición 2. de 3.2.3. En el segundo caso, se incumpliría la condición 1., porque tal punto que falta sería alcanzable por densidad desde p y no estaría en C. □

El lema anterior confirma que el algoritmo de la definición 2.3.3 se presenta de forma correcta. Dados los parámetros Eps y MinPts, se puede construir un clúster mediante un enfoque de dos pasos. Primero, se elige un punto arbitrario de la base de datos que satisfaga la condición de punto núcleo como semilla. Después, se guardan todos los puntos que son alcanzables por densidad desde la semilla, obteniendo así el clúster que contiene al punto inicial.

Teorema 3.2.5. *Exceptuando los puntos frontera, las asignaciones de los clústeres creados y los puntos ruido son independientes del orden con el que se aplica el algoritmo sobre el conjunto de datos.*

Demostración: Gracias al lema 3.2.4 se tiene que los conjuntos de la definición 3.2.3 de clúster, dado cualquier punto p núcleo contenido en ellos, coinciden con los conjuntos O del lema 3.2.4. Por tanto, buscando puntos núcleo en los datos y expandiendo el clúster como se propone en el algoritmo 2.3.3, se generarán todos los conjuntos O y por tanto, todos los clústeres según la definición 3.2.3. Por consiguiente, da igual el punto núcleo por el que se inicie cada clúster y, en definitiva, da igual el orden con el que se aplica el algoritmo sobre los datos. Por otro lado, los puntos ruido no pertenecen a ningún clúster, y dado que estos no dependen del orden con el que se aplica el algoritmo (excepto los frontera), el ruido tampoco dependerá. □

Como se ha mencionado en el teorema 3.2.5, los puntos frontera sí que pueden depender del orden en el que se aplica el algoritmo de 2.3.3 sobre los datos, en particular, dependen del orden de creación de los clústeres. Esto es porque si un punto está sin etiquetar y es vecino de un punto núcleo de un clúster, se asigna a ese clúster. Sin embargo, si también es vecino de un punto núcleo de otro clúster, al ya estar etiquetado, este no se reasigna al segundo clúster. Rigurosamente, siguiendo la definición 3.2.3, estos puntos deberían pertenecer a todos los clústeres tales que un punto núcleo en ellos sea su vecino y no solamente al primer clúster que lo etiquete. Sin embargo, la implementación del algoritmo de 2.3.3 etiquetando los puntos frontera solo 1 vez es más sencilla y genera resultados más fácilmente de analizables. Además esta forma de proceder es la que se usa en general, incluyendo la librería Scikit-learn [27]. Concluyendo, el hecho de que este tipo de puntos se asignen a unos clústeres o a otros realmente es insignificante, ya que están en regiones intermedias entre los clústeres.

Capítulo 4

Aplicaciones

En esta sección se utilizarán los algoritmos de agrupamiento k-medias y DBSCAN sobre conjuntos de datos reales con el fin de poder compararlos y comprobar su rendimiento. Por un lado, se tomará una base de datos sobre los países del mundo, que será fácilmente interpretable y sencilla de extraer conclusiones. Por otro lado, utilizará una imagen con varios tonos de colores diferentes y se verá como utilizando k-medias se puede reducir la cantidad de estos, reduciendo la cantidad de información necesaria para almacenar la imagen.

4.1. Conjunto de datos sobre los países del mundo

Se tomará la base de datos 'Unsupervised Learning on Country Data' ('Aprendizaje no supervisado en datos sobre paises') en Kaggle. Estos datos miden nueve atributos de 167 países: mortalidad infantil (por cada 1.000 nacimientos), exportaciones (como porcentaje del PIB), salud (gasto en salud como porcentaje del PIB), importaciones (como porcentaje del PIB), ingresos (en dólares per cápita), inflación (variación porcentual anual), esperanza de vida (en años), fertilidad total (número de hijos por mujer), y PIB per cápita (en dólares). Estos datos no tienen ningún valor nulo y sus variables son nueve numéricas (coma flotante) y una que indica el nombre. En el Anexo C se muestran histogramas para cada una de las variables numéricas en la figura C.0.1. Además, se añade un diagrama de cajas en la figura C.0.2 de estas variables, pero tras tipificarlas. Esto se hace así para que se puedan compartir los ejes y no se comparen unidades diferentes, sino número de desviaciones típicas sobre la media. Finalmente, en la figura C.0.3 se muestra también la matriz de correlaciones entre los atributos, para comprobar si pudieran existir relaciones lineales entre ellos. Excepto en la esperanza de vida, el resto de distribuciones tienen colas derechas mayores que las izquierdas, teniendo más datos atípicos en esa zona.

Por otro lado, se calculó $N_{sim} = 10{,}000$ veces el índice de Hopkins tomando muestras de 17 (un 10 %) elementos, representados en la gráfica A.0.2 del Anexo A. Se obtiene una media del estadístico de 0,9964, equivaliendo a un p-valor de 0 (menor que la precisión de los coma flotante). Es decir, si los datos estuviesen distribuidos de manera uniforme y por tanto los estadísticos de Hopkins siguieran una $Beta(17, 17)$, la probabilidad de haber obtenido un valor de 0,9964 o mayor es 0. En el anexo se debate por qué no se toma también la cola izquierda de la distribución para calcular el p-valor. Con lo anterior se concluye que los datos siguen cierta distribución no uniforme en el espacio.

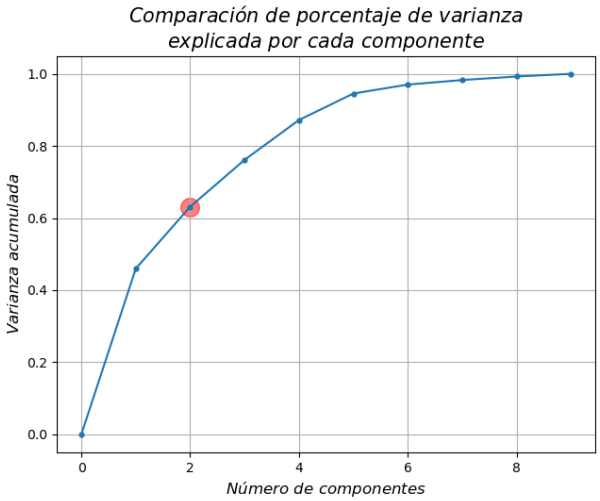
Para facilitar el procesado y sobre todo la representación de los datos, se aplicará el análisis de componentes principales basado en la matriz de covarianzas, comentado en la sección 2.1.2. En la tabla 4.1.1 se muestran los porcentajes de varianza que explican cada componente principal por orden, además de la varianza explicada acumulada hasta esa componente. Se representa además esto último en la figura 4.1.1 (a), donde se ve que si se toman dos componentes principales, ya se explica suficiente varianza (un 63 %), además de ser más sencillo de representar por ser bidimensional. En la tabla 4.1.2 se muestran las coordenadas de estas dos componentes en las variables originales.

La primera de ellas (CP1), muestra coordenadas positivas en las variables que se podrían categorizar como buenas: exportaciones, salud, importaciones, ingresos, esperanza de vida y PIB per cápita. Asimismo, presenta contribución negativa la fertilidad, que objetivamente es un

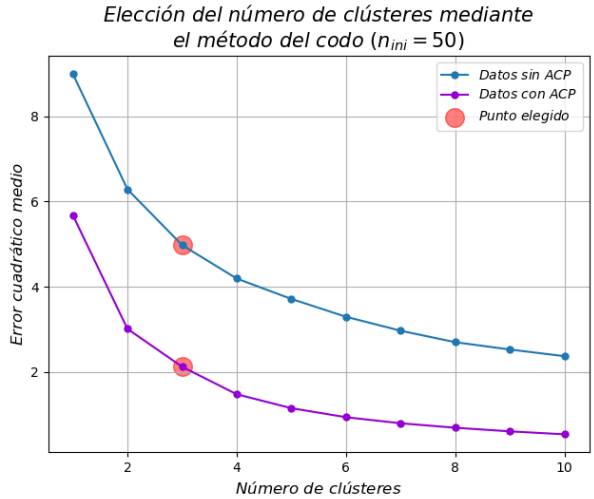
atributo bueno, pero es característico de países desarrollados. Por tanto, los países más desarrollados tendrán una coordenada en CP1 elevada mientras que los subdesarrollados la tendrán menor (incluso negativa). La segunda componente principal no tiene una interpretación tan clara, principalmente será alta si el país tiene un gran volumen de comercio: muchas importaciones y exportaciones. Una vez realizado el análisis previo, es hora de aplicar los métodos de agrupa-

Componente principal	**PC1**	**PC2**	**PC3**	**PC4**	**PC5**	**PC6**	**PC7**	**PC8**	**PC9**
Varianza explicada	0.46	0.17	0.13	0.11	0.07	0.02	0.01	0.01	0.01
Varianza acumulada	0.46	0.63	0.76	0.87	0.95	0.97	0.98	0.99	1.00

Tabla 4.1.1: Varianza explicada y acumulada de cada una de las componentes principales.



(a) Varianza acumulada en cada componente.



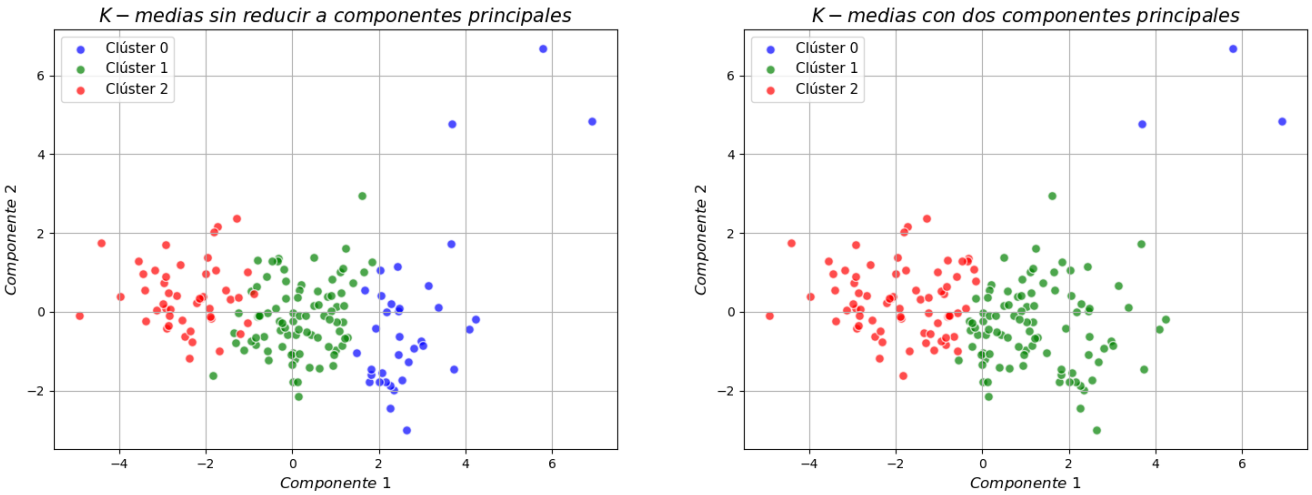
(b) Método del codo.

Figura 4.1.1: Representación de la segunda fila de la tabla 4.1.1 (a) junto a la aplicación del método del codo para k-medias en (b). Se compara este método tanto para los datos (azul) como para las dos primeras componentes (morado) y en ambos se concluye que el número óptimo de clústeres es 3.

CP	**Mort.**	**Exp.**	**Salud**	**Imp.**	**Ing.**	**Inf.**	**Esp.**	**Fert.**	**PIB/c**
CP 1	-0.42	0.28	0.15	0.16	0.40	-0.19	0.43	-0.40	0.39
CP 2	0.19	0.61	-0.24	0.67	0.02	-0.01	-0.22	0.16	-0.05

Tabla 4.1.2: Coordenadas de las dos componentes principales sobre los atributos. Los atributos son, respectivamente, mortalidad infantil, exportaciones, salud, importaciones, ingresos, inflación, esperanza de vida, fertilidad total y PIB per cápita.

miento. Se aplicará el algoritmo k-medias de dos formas: utilizando los nueve atributos originales y tomando solamente dos componentes principales. Se utilizarán en ambos tres clústeres, según sugiere el método del codo de la figura 4.1.1 (b). Los resultados en ambos casos son ligeramente diferentes, según se ve en la figura 4.1.2. Usando los nueve atributos originales, los clústeres generados tienen tamaños de 36, 84 y 47, respectivamente, como se muestra en la figura 4.1.2 (a). El primer clúster (clúster 0) recoge los países desarrollados como España, Canadá, Estados Unidos, etcétera. El clúster 1 agrupa a países en vías de desarrollo, como Jamaica, República Dominicana y Marruecos. Finalmente el tercer clúster (clúster 2) está formado por los países subdesarrollados como Mozambique, Nigeria y Namibia.



(a) Usando los datos con todas las variables.

(b) Tomando únicamente dos componentes principales.

Figura 4.1.2: Diferentes resultados de aplicar el algoritmo k-medias sobre el conjunto de datos.

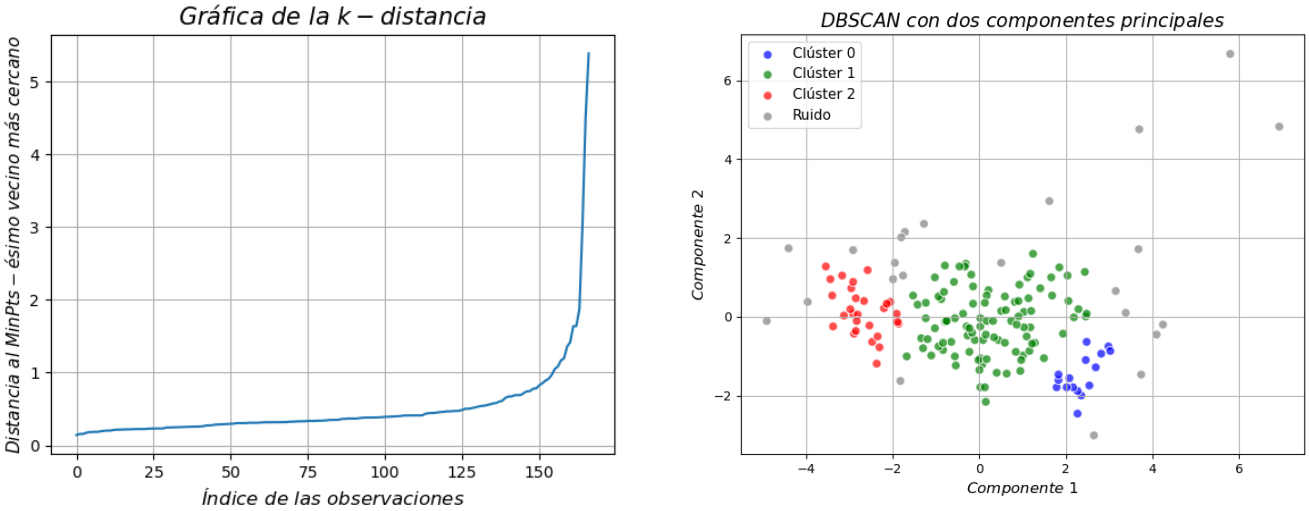
Los clústeres formados tomando dos componentes principales agrupan los países de manera diferente (figura 4.1.2 (b)), forma únicamente dos grupos de países desarrollados con 93 países y otro con 71 países subdesarrollados, eliminando el grupo intermedio. Este se sustituye por un grupo formado por 3 países que se encuentran más alejados del conjunto principal de países, cuya diferencia principal es un volumen de comercio mayor al usual. Este último grupo está formado por Luxemburgo, Singapur y Malta. Este ejemplo permite visualizar el hecho de que no hay un conjunto de clústeres claramente mejor que otro, sino que cada resultado será mas apropiado que otro según las necesidades del análisis. Para estudiar más a fondo los resultados, se tomarán los clústeres creados a partir de todas las variables de los datos, mostrados en la figura 4.1.2 (a), por ser grupos de tamaños similares. La tabla 4.1.3 muestra las medias de los diferentes atributos separadas por clústeres, donde se ve que cuanto mayor es el índice del clúster, más similares son las medias de los atributos a los de un país peor desarrollado.

Clúster	**Mort.**	**Exp.**	**Salud**	**Imp.**	**Ing.**	**Inf.**	**Esp.**	**Fert.**	**PIB/c**
0	5.0	58.74	8.81	51.49	45672	2.67	80.13	1.75	42494
1	21.93	40.24	6.20	47.47	12306	7.60	72.81	2.31	6486
2	92.96	29.15	6.39	42.32	3942	12.02	59.19	5.01	1922

Tabla 4.1.3: Medias de las variables agrupadas por clúster. Clúster 0: países desarrollados; clúster 1: países en vías de desarrollo; y clúster 2: países subdesarrollados. Las abreviaciones son las mismas que en la tabla 4.1.2.

En la imagen C.0.4 (b) del anexo C se hace especial énfasis en la variable PIB per cápita, buen indicador del desarrollo del país, realizando un diagrama de cajas separado por clústeres. De nuevo, menor es cuanto mayor es el índice del clúster. De la misma manera que para el PIB per cápita, se hicieron diagramas de caja para el resto de variables en la figura C.0.5. También se realiza un estudio de la silueta de cada observación en la figura C.0.4 (a). Esta figura separa en el eje Y las observaciones por clúster y las ordena dentro de cada grupo por valor de la silueta. De esta forma se puede visualizar sus valores con mayor facilidad. Se aprecia que en algunas observaciones de los clústeres 0 y 2 el valor de la silueta es incluso negativo. Estas observaciones se encontrarán en las zonas que se encuentren más cercanas al clúster 1.

K-medias es un algoritmo especialmente indicado para ser aplicado sobre este conjunto de datos porque se quiere formar clústeres por similitud entre países y no por densidad de ellos en el espacio de atributos. Sin embargo, también se puede aplicar DBSCAN en dos componentes principales para ver qué resultados se tienen. Para establecer el parámetro $MinPts$, se suele tomar $MinPts \geq dim + 1$ aunque $MinPts = 4$ suele ser suficiente para dimensión 2 [28] y eps se puede estimar mediante la gráfica de la k-distacia (figura 4.1.3 (a)). En ella se muestra la distancia al $MinPts$-ésimo vecino más cercano. Se buscará el codo en la curva que separará las observaciones pertenecientes a clústeres del ruido. Este valor óptimo se encuentra para $eps = 0{,}9$, aunque si se toma eps de esa cantidad se forma un único clúster central rodeado de ruido. La propia distribución de las observaciones hace que ese sea el resultado de buscar clústeres por densidad. Sin embargo, si se disminuye eps hasta $eps = 0{,}5$ se forman 3 clústeres (figura 4.1.3 (b)) similares a los de k-medias (figura 4.1.2 (a)), exceptuando el hecho de que los países más alejados los etiqueta como ruido. Los clústeres obtenidos de esta manera no son tan útiles en este problema en específico como los de k-medias pero pueden extraerse conclusiones igualmente.



(a) Gráfico de la k-distancia.

(b) Resultado de aplicar DBSCAN sobre los datos.

Figura 4.1.3: Estimación de eps con el gráfico de la k-distancia (a) junto a los resultados de aplicar DBSCAN con $MinPts = 4$, $eps = 05$ al conjunto de datos de los países.

4.2. Compresión de imágenes

Cuando se habló de las aplicaciones del agrupamiento de datos en la sección 1, se mencionó que una de ellas era la compresión de imágenes [1]. Una imagen sin comprimir, puede tener del orden de unidades o decenas de millón de colores. La idea es reducir esta cantidad drásticamente para disminuir la información necesaria para guardar la imagen, perdiendo la menor calidad posible. Sin embargo, el método más usado para la compresión de imágenes no usa la idea anterior sino la transformación discreta de coseno [39], utilizada en extensiones como jpeg.

El conjunto de datos de la sección anterior 4.1 las observaciones se organizan por filas y los atributos por columnas, siguiendo el formato estándar. En una imagen RBG de $N_{ancho} \times N_{alto}$ píxeles, los colores se guardan en una hipermatriz de tamaño $N_{ancho} \times N_{alto} \times 3$, al tener cada píxel una contribución en cada uno de los 3 colores. Para organizar estos datos de forma estándar, los píxeles se ordenarán una manera concreta, se pondrán por filas y la contribución a cada píxel por columnas (serán los atributos). El orden de los píxeles puede construirse, por

ejemplo, poniendo todas filas seguidas, quedando la matriz en 1 dimensión.

A partir de ahora, se escogió una imagen para poner en práctica lo anterior. Esta se tomó de 'Applied Machine Learning' y se muestra en el anexo D, así como todos los resultados de las compresiones. Tomando los datos de la imagen organizados de la forma anterior, se puede aplicar cualquier algoritmo de agrupamiento, aunque algunos de ellos estarán especialmente indicados para este ejemplo. Por ejemplo, los algoritmos que introducen el concepto de ruido no interesan porque se quiere que todos los píxeles se asignen a un clúster y por tanto a un color, aunque el clúster más cercano esté a una cierta distancia. Además algoritmos como DBSCAN pueden generar clústeres de formas muy variada (véase 2.3.1) y por tanto con zonas de ellos alejadas de sus centroides considerablemente. En este tipo de aplicaciones, interesa que a cada píxel se le asocie el centroide con el color más cercano y estas asignaciones las generan automáticamente algoritmos del tipo de k-medias, por la suposición de clústeres esféricos. En ejemplos como este se demuestra cómo la elección del algoritmo de agrupamiento es crucial.

En la imagen D.0.1 se muestra el procedimiento mostrado en los párrafos anteriores para diferente cantidad de clústeres (colores), en potencias de 2 desde 2 colores hasta 128. Visualmente, apenas se aprecia diferencia a partir de los 16 colores excepto en la zona verde y en menor medida el tronco. Para cuantificar la bondad del ajuste, se muestra también el error cuadrático medio para cada imagen, mostrado anteriormente en la imagen 2.4.1 para explicar el método del codo.

El error cuadrático medio toma valores relativamente altos, del orden de mil, para los primeros casos. Esto es porque las contribuciones de cada color toman valores entre 0 y 255 y el error cuadrático medio toma diferencias entre esas cantidades al cuadrado. No se han tipificado las variables porque tomaban valores de magnitudes similares.

Otro aspecto interesante a mencionar es que se ha usado la inicialización de centroides k-medias++ y se ha ejecutado el algoritmo 10 veces para cada imagen. Con lo anterior se toma el resultado de menor inercia (RSS) y por tanto, el mejor ajuste, tal y como hace el código del anexo E.2. La imagen D.0.2 se introduce para observar cómo, aunque se utilice k-medias++, conviene realizar varias inicializaciones diferentes para asegurar que el resultado se acerque al óptimo global. Se aprecian notables diferencias entre las imágenes de D.0.2 en la zona a la izquierda de la imagen, donde el color es verde en la imagen original. Tomando la semilla como 8 se obtiene este color y es, de las cuatro, la que menor error cuadrático medio (y por tanto, inercia) tiene. Por esta razón, es la que se asemeja en mayor medida a la imagen de D.0.1 correspondiente a $k = 8$. Los valores que toman la semilla no son interpretables, simplemente es el número sobre el que se genera la elección pseudo-aleatoria de los centroides.

Capítulo 5
Conclusiones

Como se ha comentado en la introducción, tener la capacidad de analizar y extraer conclusiones de grandes volúmenes de datos es una gran ventaja, sobre todo en la actualidad. A lo largo de este trabajo se han mencionado varias formas de formar grupos o clústeres a partir de un conjunto de datos distribuidos por observaciones y atributos. Junto a ello, se ha hecho especial énfasis en dos de los métodos más utilizados: k-medias y DBSCAN extrayendo conclusiones sobre su convergencia y realizando versiones de estos en código propio. Finalmente, utilizando este código propio, se han aplicado los algoritmos sobre dos conjuntos de datos: sobre los países del mundo y los píxeles de una imagen.

En la sección 3 se discute la convergencia de los algoritmos. Para el caso de k-medias, se tienen resultados de convergencia del algoritmo a puntos de Karush-Kuhn-Tucker y además en un número finito de iteraciones. Sin embargo, no se asegura la convergencia un óptimo local, según el contraejemplo del anexo B. No ocurre lo mismo para DBSCAN, en el que la convergencia está asegurada salvo por posibles cambios en la asignación de puntos frontera de dos clústeres. También es cierto que, en el caso de este segundo algoritmo, la propia forma de su construcción hace que la convergencia no sea un gran problema, como sí es el caso de k-medias.

Para afrontar la ineficiente convergencia de k-medias, surgen varios métodos comentados durante el trabajo. El más común consiste en combinar varias inicializaciones de los centroides con realizar cada una de ellas mediante el procedimiento k-medias++. Todo esto se comentó e introdujo en el código de k-medias de la sección 2.2.2, además de los propios algoritmos de k-medias y DBSCAN.

Posteriormente, con el fin de añadir aplicaciones prácticas, se han aplicado los algoritmos sobre conjuntos de datos. Se ha elegido una base de datos sobre los países del mundo (sección 4.1) para que sus resultados fueran fácilmente interpretables. Se han aplicado los algoritmos k-medias y DBSCAN obteniendo sus correspondientes clústeres, concluyendo ser más apropiado el primer método para ese tipo de datos, aunque obteniendo en ambos resultados similares. Las figuras C.0.4 y C.0.5, así como en la tabla 4.1.3 confirman que los resultados obtenidos eran los esperados. Para empezar, según el método del codo (figura 4.1.1 (b)) se ha tomado la decisión de tomar 3 clústeres, generando una partición similar a la que usualmente se suele realizar sobre los países: países desarrollados, países en vías de desarrollo y países subdesarrollados. Los valores medios de los atributos en cada clúster concuerdan también con la clasificación anterior, destacando las diferencias en atributos como la mortalidad infantil, inflación, esperanza de vida o fertilidad, en la tabla 4.1.3 y figura C.0.5; además del PIB per cápita (figura C.0.4), atributo en el que más diferencias se encuentran. Otro resultado a destacar es el hecho de que los valores de silueta son razonablemente buenos, indicando la buena forma de los clústeres.

Algoritmos basados en densidad como DBSCAN son más apropiados en conjuntos de datos con alta frecuencia de datos atípicos, ya que pueden identificar grupos de forma más flexible y manejar mejor la presencia de ruido en los datos. DBSCAN, a diferencia de k-medias, es más apropiado para datos en los que no se quiera especificar el número de clústeres previamente. También es útil en aquellos en los que se busque formar clústeres de formas arbitrarias, permitiendo detectar estructuras no esféricas. DBSCAN, por tanto, es útil en diversas aplicaciones: detección de fraudes, permite identificar transacciones anómalas fuera de los patrones comunes, redes sociales. Es capaz de detectar agrupaciones de puntos de interés cercanos y señala áreas aisladas o de baja densidad.

También se ha aplicado el análisis de componentes principales, permitiendo representar las asignaciones de las observaciones a los clústeres en dos dimensiones, en las figuras 4.1.2 y 4.1.3 (b). Se ha utilizado también para aplicar DBSCAN al conjunto de datos de los países, aunque no para k-medias. Esto se ha hecho así porque al aplicar ACP, usar únicamente el 63 % de información daba demasiada importancia a atributos relacionados con el comercio (importaciones y exportaciones), generando un clúster de tres países más alejado del resto (figura 4.1.2 (b)).

Finalmente, para incluir otra aplicación de k-medias, se ha aplicado este algoritmo para reducir el número de colores de una imagen, mostrado en la figura D.0.1. Esto ha permitido concluir que 16 colores eran suficientes para representar la imagen, salvo quizá, pequeños detalles. Además, se ha podido visualizar cómo el algoritmo puede converger a diferentes conjuntos de clústeres en la imagen D.0.2.Este resultado confirma la importancia de ejecutar el algoritmo k-medias varias veces para aproximarse al óptimo global.

En la sección 2.2.2 se compararon las líneas de código, concluyendo que los algoritmos personales las reducían notablemente. Para realizar la comparación de manera más clara, se añade la tabla 4.1.3 junto a los tiempos de ejecución de cada código para el conjunto de datos de los países. Se concluye que el más rápido es DBSCAN en la librería [27] seguido por la versión simple de k-medias. Esto último afirma que la inicialización k-medias++ junto al cálculo de la inercia ralentiza la ejecución de los otros códigos de k-medias. El hecho de que el algoritmo de DBSCAN sea tan lento comparado con el de la librería es porque este último utiliza algoritmos como árbol de bolas (ball tree) o árbol k-d (k-d tree) para optimizar la búsqueda de vecinos. Si se escoge el algoritmo de búsqueda sin optimizar ('brute') se multiplica el tiempo de ejecución en un factor 100, acercándose en gran medida al tiempo del código personal. Para el caso de k-medias, el código de la librería [19] tarda ligeramente más que el personal mejorado. Esto podría deberse a la ausencia de comprobaciones adicionales para manejar casos extremos como clusters vacíos o valores NaN.

Código	**Tiempos (ms)**	**Líneas de código**
$DBSCAN$	1,17	476
$Cesar\_DBSCAN$	97,21	100
$KMeans$	156,79	2301
$Cesar\_Improved\_KMeans$	116,73	125
$Cesar\_Simple\_KMeans$	36,32	79

Tabla 5.0.1: Comparación de los tiempos de ejecución y líneas de código entre los algoritmos implementados por scikit-learn [27] [19] y los personales, sobre el conjunto de datos de los países.

Como una posible mejora del trabajo, se podría considerar aplicar este tipo de procedimientos a bases de datos reales, con miles o millones de observaciones y varias decenas de variables. Bases de datos de este estilo podrían ser proporcionadas por ciertas empresas del sector tecnológico o de análisis de datos, por ejemplo.

A lo largo de este trabajo se ha comprobado la capacidad de los algoritmos de agrupamiento de datos para generar grupos en estos. No son necesarias unidades, contexto o explicación de las variables para que este tipo de procedimientos funcionen correctamente. Además, aumentar las dimensiones de la base de datos no aumenta la dificultad de su resolución en la misma cantidad, simplemente aumenta el tiempo de ejecución (pequeño para las bases de datos del trabajo) y posiblemente el coste de extraer conclusiones de los resultados.

En conclusión, el trabajo realizado ha permitido llevar a cabo un análisis detallado de los métodos de agrupamiento de datos k-medias y DBSCAN, comprobando su convergencia y ofreciendo soluciones en el caso de que esta no esté asegurada. Finalmente, se comprobaron tanto la convergencia como sus soluciones, aplicándolo a dos conjuntos de datos.

Bibliografía

[1] M. Kaya, "An algorithm for image clustering and compression," *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 13, no. 1, pp. 79–92, 2005.

[2] M. P. Bach, S. Juković, K. Dumičić, and N. Šarlija, "Business client segmentation in banking using self-organizing maps," *South East European Journal of Economics and Business*, vol. 8, no. 2, pp. 32–41, 2013.

[3] A. Alsayat and H. El-Sayed, "Social media analysis using optimized k-means clustering," in *2016 IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA)*, pp. 61–66, 2016.

[4] W. Jang and M. Hendry, "Cluster analysis of massive datasets in astronomy," *Statistics and Computing*, vol. 17, pp. 253–262, 2007.

[5] Q. Zou, G. Lin, X. Jiang, X. Liu, and X. Zeng, "Sequence clustering in bioinformatics: an empirical study," *Briefings in bioinformatics*, vol. 21, no. 1, pp. 1–10, 2020.

[6] S. Khanmohammadi, N. Adibeig, and S. Shanehbandy, "An improved overlapping k-means clustering method for medical applications," *Expert Systems with Applications*, vol. 67, pp. 12–18, 2017.

[7] T. S. Madhulatha, "An overview on clustering methods," 2012.

[8] J. Irani, N. Pise, and M. Phatak, "Clustering techniques and the similarity measures used in clustering: A survey," *International journal of computer applications*, vol. 134, no. 7, pp. 9–14, 2016.

[9] S. Z. Selim and M. A. Ismail, "K-means-type algorithms: A generalized convergence theorem and characterization of local optimality," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 1, pp. 81–87, 1984.

[10] B. S. Everitt, S. Landau, M. Leese, and D. Stahl, "Chapter 3," in *Cluster Analysis*, pp. 44–69, Chichester, UK: Wiley, 5th ed., 2011.

[11] K. Wright, "Will the real hopkins statistic please stand up?," *R Journal*, vol. 14, no. 3, 2022.

[12] A. Banerjee and R. Dave, "Validating clusters using the hopkins statistic," in *2004 IEEE International Conference on Fuzzy Systems (IEEE Cat. No.04CH37542)*, vol. 1, pp. 149–153 vol.1, 2004.

[13] R. G. Lawson and P. C. Jurs, "New index for clustering tendency and its application to chemical problems," *Journal of chemical information and computer sciences*, vol. 30, no. 1, pp. 36–41, 1990.

[14] A. Adolfsson, M. Ackerman, and N. C. Brownstein, "To cluster, or not to cluster: An analysis of clusterability methods," *Pattern Recognition*, vol. 88, pp. 13–26, 2019.

[15] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[16] J. Shlens, "A tutorial on principal component analysis," *CoRR*, vol. abs/1404.1100, 2014.

[17] P. R. Halmos, "What does the spectral theorem say?," *The American Mathematical Monthly*, vol. 70, no. 3, pp. 241–247, 1963.

[18] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," tech. rep., Stanford, 2006.

[19] scikit-learn developers, "Kmeans algorithm implementation." `https://github.com/scikit-learn/scikit-learn/blob/f5aac2173/sklearn/cluster/_kmeans.py#L1196`, 2023. Accessed: 2024-09-13.

[20] G. Hamerly and C. Elkan, "Alternatives to the k-means algorithm that find better clusterings," in *Proceedings of the eleventh international conference on Information and knowledge management*, pp. 600–607, 2002.

[21] C.-T. Chang, J. Z. Lai, and M.-D. Jeng, "A fuzzy k-means clustering algorithm using cluster center displacement.," *J. Inf. Sci. Eng.*, vol. 27, no. 3, pp. 995–1009, 2011.

[22] C. Zhen, "Using big data fuzzy k-means clustering and information fusion algorithm in english teaching ability evaluation," *Complexity*, vol. 2021, no. 1, p. 5554444, 2021.

[23] M. J. Li, M. K. Ng, Y.-m. Cheung, and J. Z. Huang, "Agglomerative fuzzy k-means clustering algorithm with selection of number of clusters," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 11, pp. 1519–1534, 2008.

[24] J. Xu, J. Han, K. Xiong, and F. Nie, "Robust and sparse fuzzy k-means clustering.," in *IJCAI*, pp. 2224–2230, 2016.

[25] A. Wehrl, "General properties of entropy," *Rev. Mod. Phys.*, vol. 50, pp. 221–260, Apr 1978.

[26] S. Tokushige, H. Yadohisa, and K. Inada, "Crisp and fuzzy k-means clustering algorithms for multivariate functional data," *Computational Statistics*, vol. 22, no. 1, pp. 1–16, 2007.

[27] S.-L. Developers, "scikit-learn: Machine learning in python - dbscan implementation." `https://github.com/scikit-learn/scikit-learn/blob/04fbe04fe/sklearn/cluster/_dbscan.py#L181`, 2023. Accessed: 2024-10-12.

[28] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)* (E. Simoudis, J. Han, and U. M. Fayyad, eds.), pp. 226–231, AAAI Press, 1996.

[29] X. Zhang, H. Liu, and X. Zhang, "Novel density-based and hierarchical density-based clustering algorithms for uncertain data," *Neural Networks*, vol. 93, pp. 240–255, 2017.

[30] A. Dockhorn, C. Braune, and R. Kruse, "Variable density based clustering," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, IEEE, 2016.

[31] S. Brecheisen, H.-P. Kriegel, and M. Pfeifle, "Efficient density-based clustering of complex objects," in *Fourth IEEE International Conference on Data Mining (ICDM'04)*, pp. 43–50, 2004.

[32] R. J. Campello, D. Moulavi, and J. Sander, "Density-based clustering based on hierarchical density estimates," in *Pacific-Asia conference on knowledge discovery and data mining*, pp. 160–172, Springer, 2013.

[33] M. Cui *et al.*, "Introduction to the k-means clustering algorithm based on the elbow method," *Accounting, Auditing and Finance*, vol. 1, no. 1, pp. 5–8, 2020.

[34] M. A. Syakur, B. K. Khotimah, E. Rochman, and B. D. Satoto, "Integration k-means clustering method and elbow method for identification of the best customer profile cluster," in *IOP conference series: materials science and engineering*, vol. 336, p. 012017, IOP Publishing, 2018.

[35] H. Řezanková, "Different approaches to the silhouette coefficient calculation in cluster evaluation," in *21st international scientific conference AMSE applications of mathematics and statistics in economics*, pp. 1–10, 2018.

[36] M. A. Hanson, "On sufficiency of the kuhn-tucker conditions," *J. Math. Anal. Appl*, vol. 80, no. 2, pp. 545–550, 1981.

[37] G. Gordon and R. Tibshirani, "Karush-kuhn-tucker conditions," *Optimization*, vol. 10, no. 725/36, p. 725, 2012.

[38] J. R. Berrendero, "Apuntes de matemáticas para la investigación operativa: Tema 5 - dualidad y condiciones kkt." `https://verso.mat.uam.es/~joser.berrendero/cursos/Matematicas-IO/io-tema5-16.pdf`, 2016.

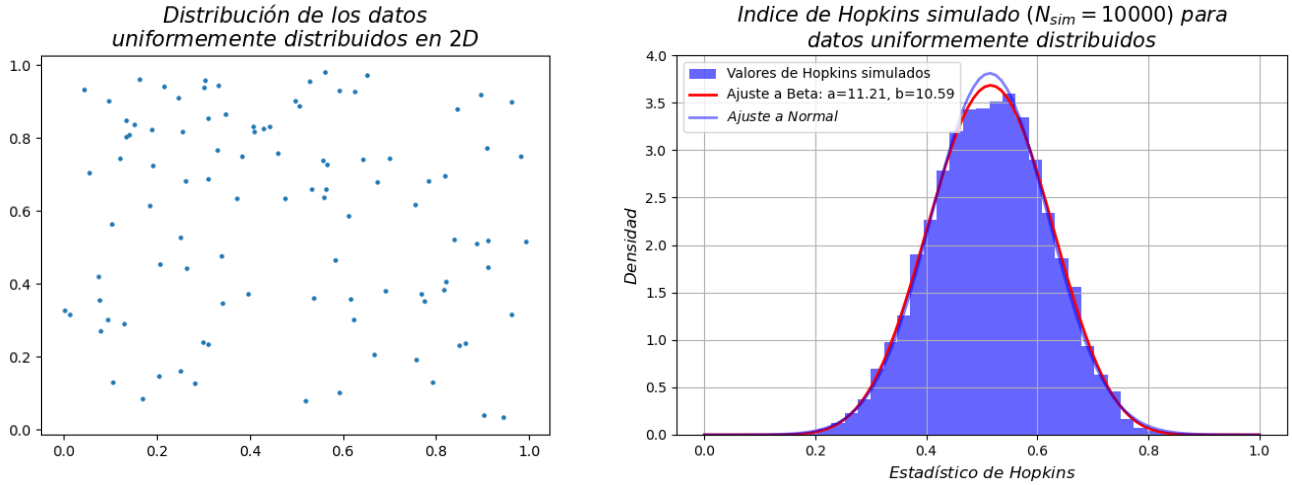
[39] A. Raid, W. Khedr, M. A. El-Dosuky, and W. Ahmed, "Jpeg image compression using discrete cosine transform-a survey," *arXiv preprint arXiv:1405.6147*, 2014.

[40] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley, "Kkt conditions, first-order and second-order optimization, and distributed optimization: Tutorial and survey," 2021. Section 4: Karush-Kuhn-Tucker (KKT) Conditions.

Anexo A

Comprobación del lema 2.1.2: el estadístico de Hopkins sigue una distribución $Beta(k, k)$

Se ha mencionado en la sección 2.1.1 que, aunque en ningún caso sirva como una demostración, se simularían varios conjuntos de datos completamente aleatorios, con densidad de probabilidad uniforme, para evaluar la distribución de los valores que toma el estadístico de Hopkins en ellos. Para ello, por simplicidad, se eligen datos en dos dimensiones, en el espacio de parámetros $[0, 1] \times [0, 1]$ como en la figura A.0.1 (a). Se toman $m = 100$ datos totales y muestras de $k = 10$ elementos. Se calcula el índice de Hopkins $N_{sim} = 10000$ veces, con muestras diferentes y se muestra el histograma de ellos en la figura A.0.1 (b) junto al ajuste a una función de distribución beta. Se puede observar que el ajuste es muy bueno y los parámetros se acercan a $(10, 10)$, como debería ocurrir. La pequeña discrepancia se debe a la elección aleatoria de los datos, si se eligiesen varios conjuntos de datos, el ajuste total se acercaría en mayor medida a $(10, 10)$. También se ajusta a una distribución normal, aunque el ajuste es ligeramente menos preciso.



(a) Datos uniformemente simulados.

(b) Histograma de los estadísticos de Hopkins.

Figura A.0.1: Ejemplo de los datos tomados (a) junto al histograma del estadístico de Hopkins y ajuste a una función de distribución beta y normal (b). Los 100 puntos totales se escogen en grupos de 10 para calcular el índice. Los valores del ajuste a labeta son $a = 11{,}21$ $b = 10{,}59$ y a la normal $\mu = 0{,}483$ $\sigma = 0{,}111$, con errores menores al 1 %. Los estadísticos de Hopkins se han calculado tomando grupos de $k = 10$ elementos.

Para poder comparar con conjuntos de datos reales, se calcularán también $N_{sim} = 10000$ índices de Hopkins para el conjunto de datos sobre los países, mostrándose en las figuras A.0.2. Se aprecia en la figura A.0.2 que todos los estadísticos de Hopkins toman valores mayores a 0.8 y casi todas iteraciones (casi dos órdenes de magnitud de diferencia) caen en el último bin. A partir de valores de 0,8 del índice de Hopkins, el p-valor es del orden de 10^{-3}, quedando en 0 para los bines cercanos a 1. Por tanto, se rechaza H_0 a favor de H_1 y se propone por ello que tienen cierta estructura los datos.

Por otro lado, si se generan conjuntos de datos de forma específica, se pueden conseguir

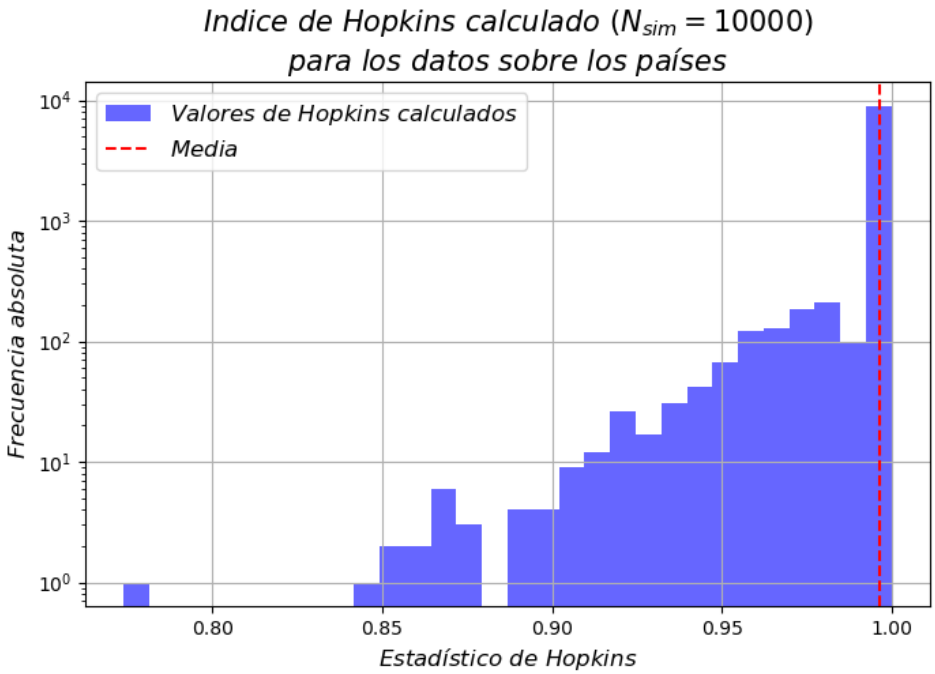
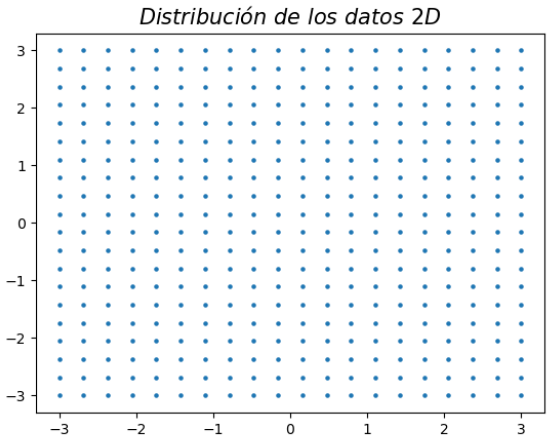


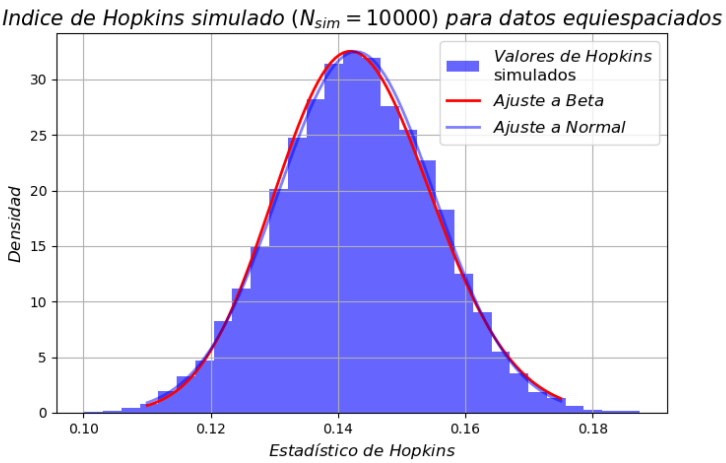
Figura A.0.2: Histograma de los estadísticos de Hopkins de los datos sobre países, en escala logarítmica. Se añade también la media con una línea roja para un índice de Hopkins de 0,996. Los estadísticos de Hopkins se han calculado tomando grupos de $k = 17$ elementos.

valores del índice de Hopkins menores a 0,5. Este es el caso de datos repartidos en el espacio de parámetros de forma equiespaciada (figura A.0.3 (a)), en los que claramente tampoco tienen tendencia a formar clústers. En la figura A.0.3 (b) se ajusta el histograma de este conjunto de datos específico a distribuciones beta y normal, teniendo ajustes idénticos y muy cercanos al histograma. Se obtiene una media del índice de Hopkins en torno a $0{,}14 < 0{,}5$ que no depende de la distancia entre estos puntos, sino de la propia forma de red cuadrada de distribuirse. En los datos uniformemente, el histograma también parecía tener tendencia gaussiana pero el ajuste no era idéntico a la beta.

En resumen, los conjuntos de datos que obtienen estadísticos de Hopkins entorno a 0,5 o menores no permiten la formación de clústeres de forma clara. Por tanto, a la hora de hacer el test de hipótesis, se calculará el p-valor considerando que solamente cobran interés los valores más extremos al obtenido en la rama derecha de la distribución, y no a ambos lados. Es por esto que el p-valor se asociará con la probabilidad de obtener valores del índice mayores al obtenido, suponiendo que la distribución es $Beta(k, k)$.



(a) Datos equiespaciados.



(b) Histograma de los estadísticos de Hopkins.

Figura A.0.3: Datos equiespaciados tomados (a) junto al histograma del estadístico de Hopkins y ajuste a una función de distribución beta y normal (b). Los valores del ajuste a la beta son $a = 763{,}34$ $b = 4577{,}99$ y a la normal $\mu = 0{,}1429$ $\sigma = 0{,}0048$, con errores menores al 1 %. Los estadísticos de Hopkins se han calculado tomando grupos de $k = 10$ elementos.

Anexo B

Contraejemplo de convergencia de k-medias a un óptimo local. Teorema sobre KKT.

Aunque una solución parcialmente óptima es un punto de Karush-Kuhn-Tucker del Problema P, puede que ni siquiera sea un mínimo local. Lo anterior se comenta en la sección 3, justo después del teorema 3.1.10. El siguiente ejemplo ilustra esta posibilidad [9].

Sean las observaciones $X_1 = (0,0)$, $X_2 = (4,0)$, $X_3 = (4,2)$ y $X_4 = (8,2)$, y $k = 2$, utilizando la distancia euclídea. Si se utiliza el algoritmo k-medias con centroides iniciales $z_1 = (2,0)$ y $z_2 = (6,2)$, el algoritmo se detendrá después de una iteración con los clústeres (X_1, X_2) y (X_3, X_4) y

$$W = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

y $f(W,Z) = 8$. Sin embargo, este punto no es un mínimo local, ya que si se define

$$W = \begin{bmatrix} 1 & 1-\epsilon & 0 & 0 \\ 0 & \epsilon & 1 & 1 \end{bmatrix}$$

con $\epsilon > 0$ todo lo pequeño que se quiera. En este caso, los nuevos centroides se actualizarán en función de ϵ. El centroide del clúster 1 tenderá a situarse sobre el punto X_1 y el del clúster 2 sobre X_3. Si los centroides se colocasen justamente sobre ellos, es decir, en $(0,0)$ y $(4,2)$; la función a minimizar ya toma un valor menor $f(W,Z) = 8 - 2\,\epsilon$. Por tanto, los centroides que se obtendrían al aplicar el paso 3 de k-medias (definición 3.1.12) tendrían un valor de la función menor o igual a $8 - 2\,\epsilon$. Es decir, variando W de la forma anterior con $\epsilon > 0$ todo lo pequeño que se quiera, se tiene un valor de la función objetivo. El algoritmo finaliza con los centroides $(0,0)$ y (4,51, 1,39) como se ve en la figura B.0.1.

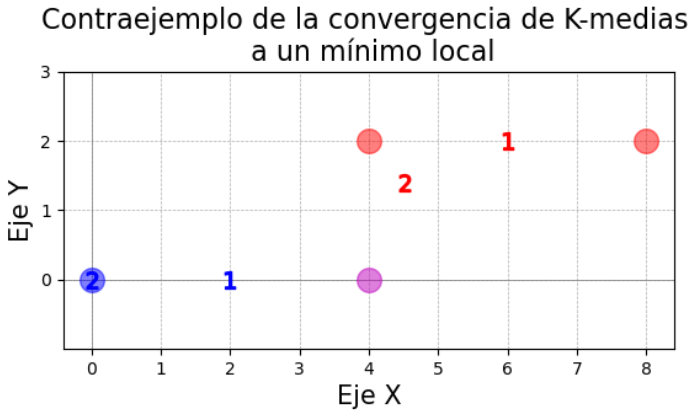


Figura B.0.1: Visualización del contraejemplo comentado en el párrafo a la izquierda. Los datos son los círculos de colores: azul, para el clúster 1 y rojo para el 2. Los centroides son los marcadores '1' y '2' indicando la inicialización.

En la figura B.0.1, los datos son los círculos de colores: azul, para el clúster 1 y rojo para el 2. El punto X_2 se muestra en magenta porque pasa de ser azul con la primera inicialización a

rojo en la segunda. También se muestran los centroides resultantes de la primera inicialización con el marcador '1' y los de la segunda con '2'.

Teorema B.0.1. *Sea el problema PI dado por:*

$$\begin{aligned} PI: minimizar \quad & f(x) \\ sujeto\ a \quad & f_i(x) \leq 0, \qquad i = 1, \dots m \\ & h_i(x) = 0, \qquad i = 1, \dots r \end{aligned} \tag{B.0.1}$$

donde todas las funciones son diferenciables. Sea $\hat{x}$ un mínimo local de (PI) que verifica la cualificación de las restricciones anteriores. Entonces, existen vectores $\hat{u} \geq 0$ y $\hat{v}$ tales que $\hat{x}$, $\hat{u}$ y $\hat{v}$ verifican las condiciones KKT dadas en la definición 3.1.7.

Demostración: Teorema 2 de [38] (más información en [40]). □

Anexo C

Información sobre la base de datos de los países

Histogramas de cada variable

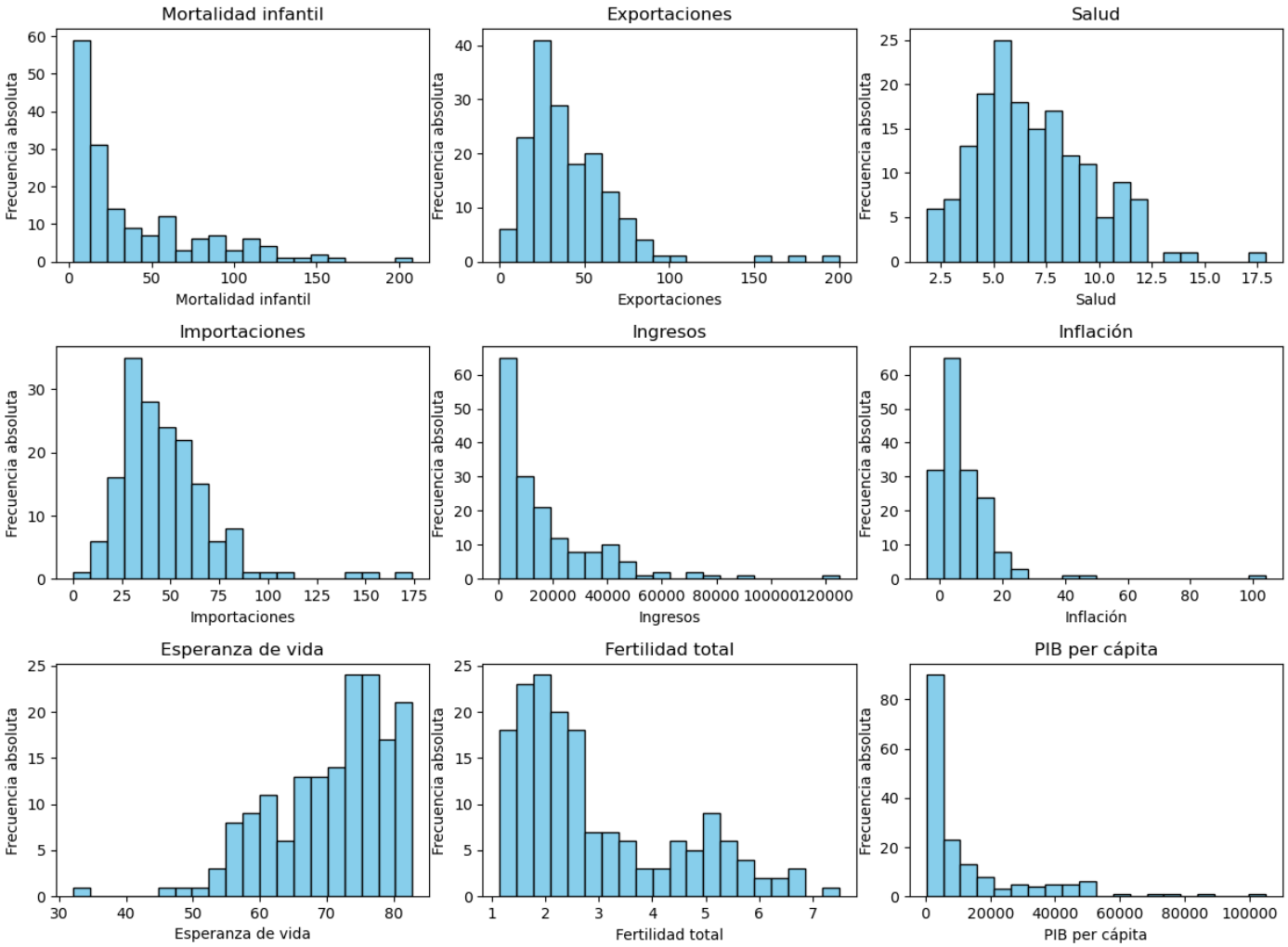


Figura C.0.1: Histogramas sobre los 9 atributos de los 167 países: Mortalidad infantil (por cada 1,000 nacimientos), Exportaciones (como porcentaje del PIB), Salud (gasto en salud como porcentaje del PIB), Importaciones (como porcentaje del PIB), Ingresos (en dólares per cápita), Inflación (variación porcentual anual), Esperanza de vida (en años), Fertilidad total (número de hijos por mujer), y PIB per cápita (en dólares).

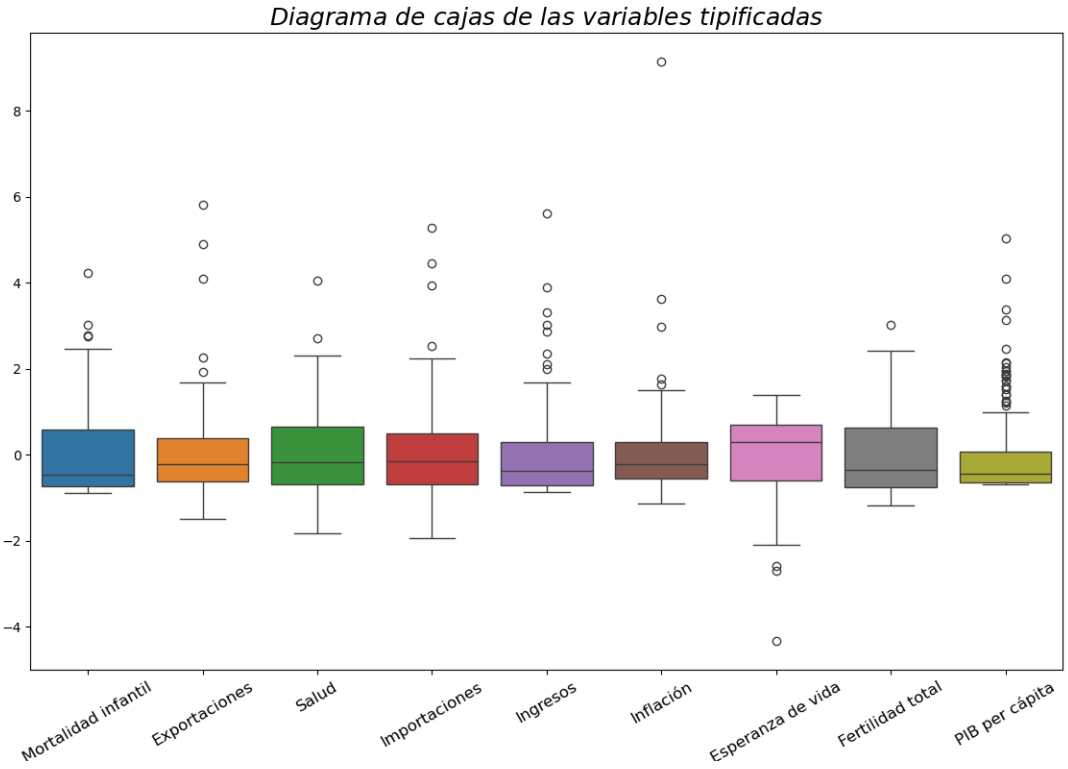


Figura C.0.2: Diagrama de cajas de las nueve variables numéricas tipificadas.

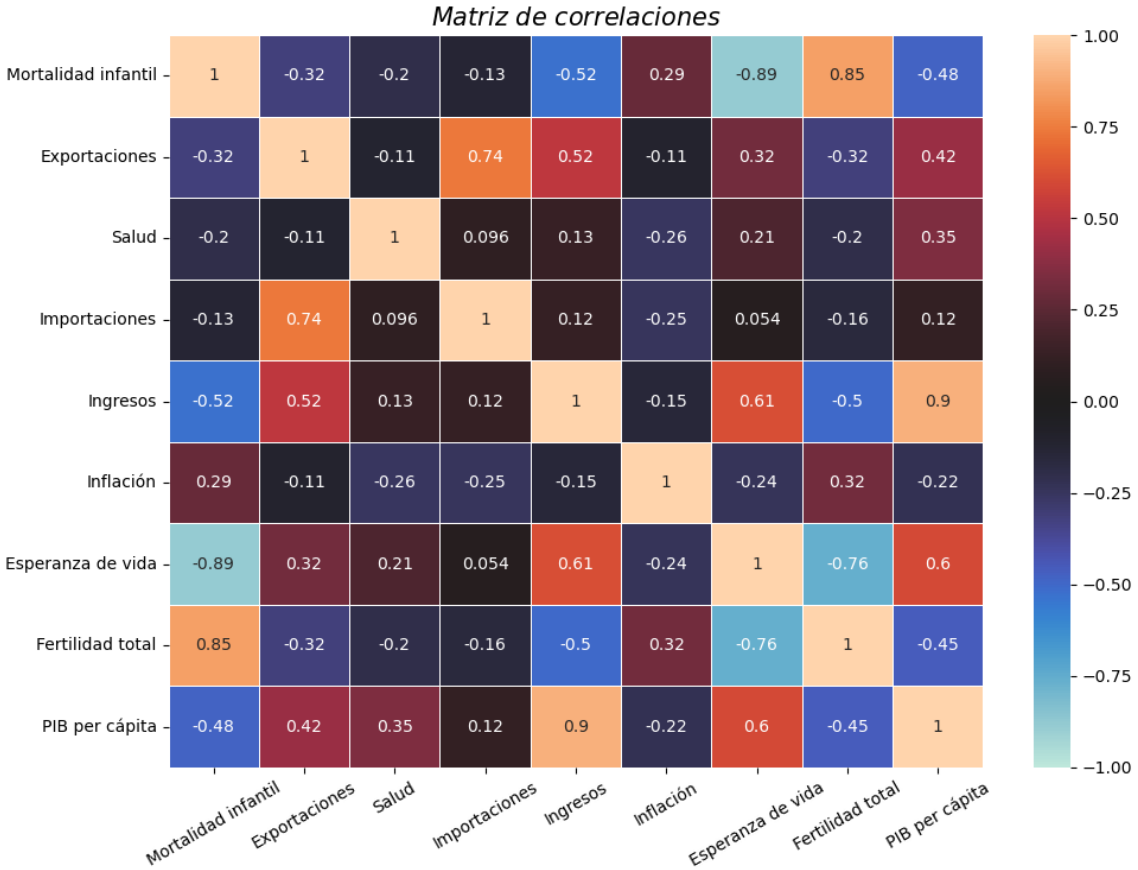
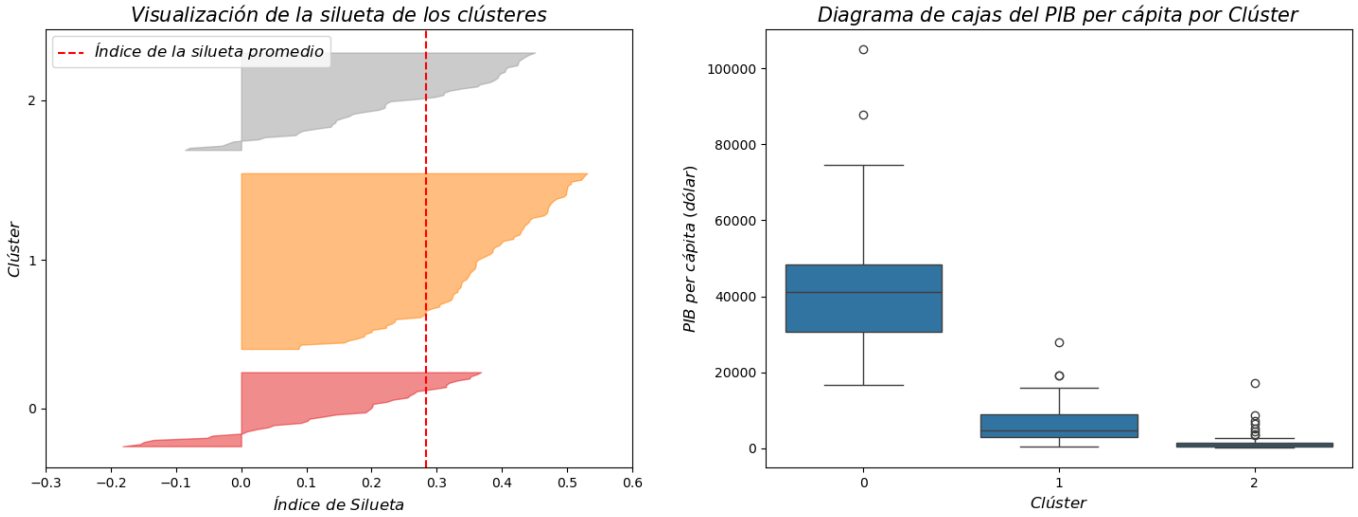


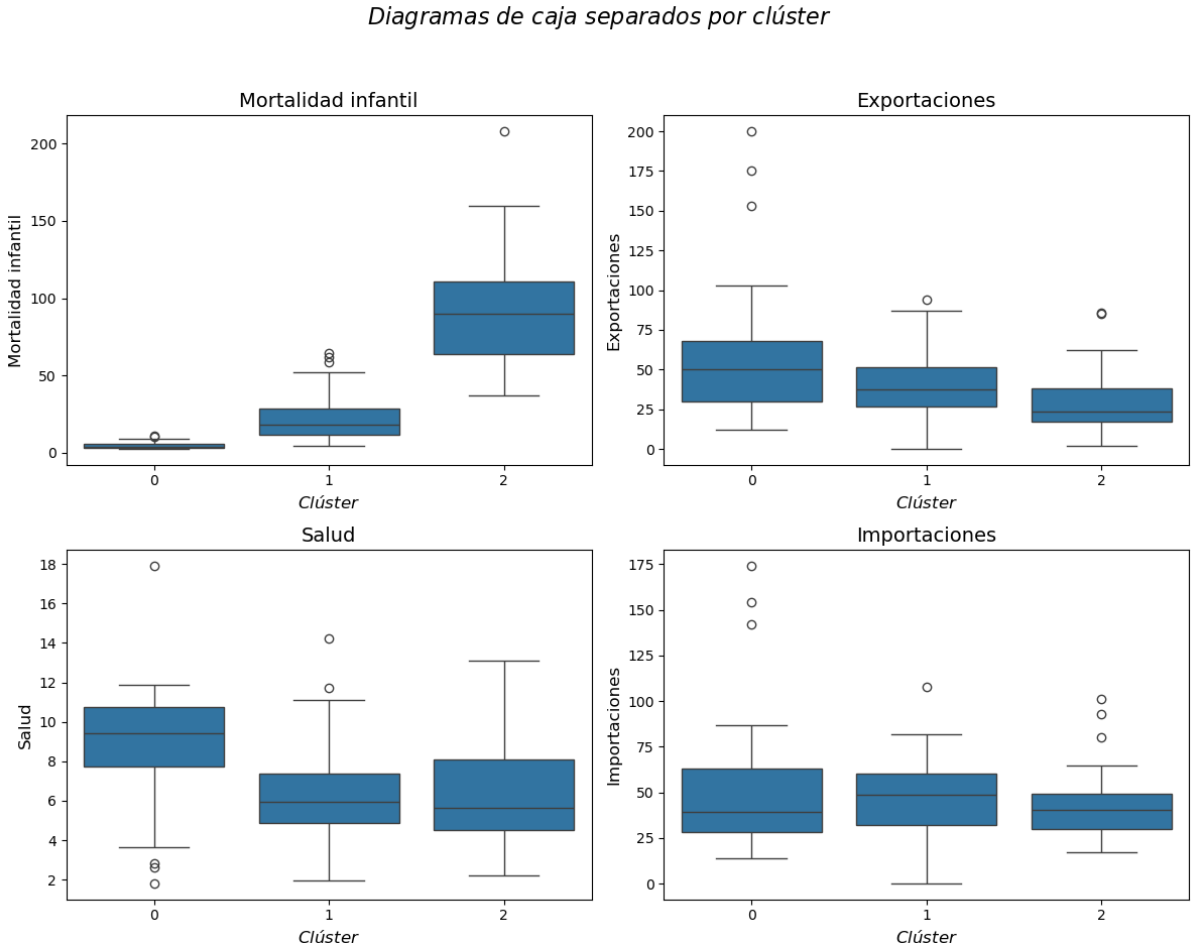
Figura C.0.3: Matriz de correlaciones de los nueve atributos numéricos de los países.



(a) Indice de la silueta.

(b) Diagrama de cajas del PIB per cápita.

Figura C.0.4: Evaluación de los resultados de aplicar k-medias sobre el conjunto de datos de los países de la sección 4.1. Índice de la silueta para cada observación separadas por clústeres (a) y diagrama de cajas del PIB per cápita separado los clústeres (b).



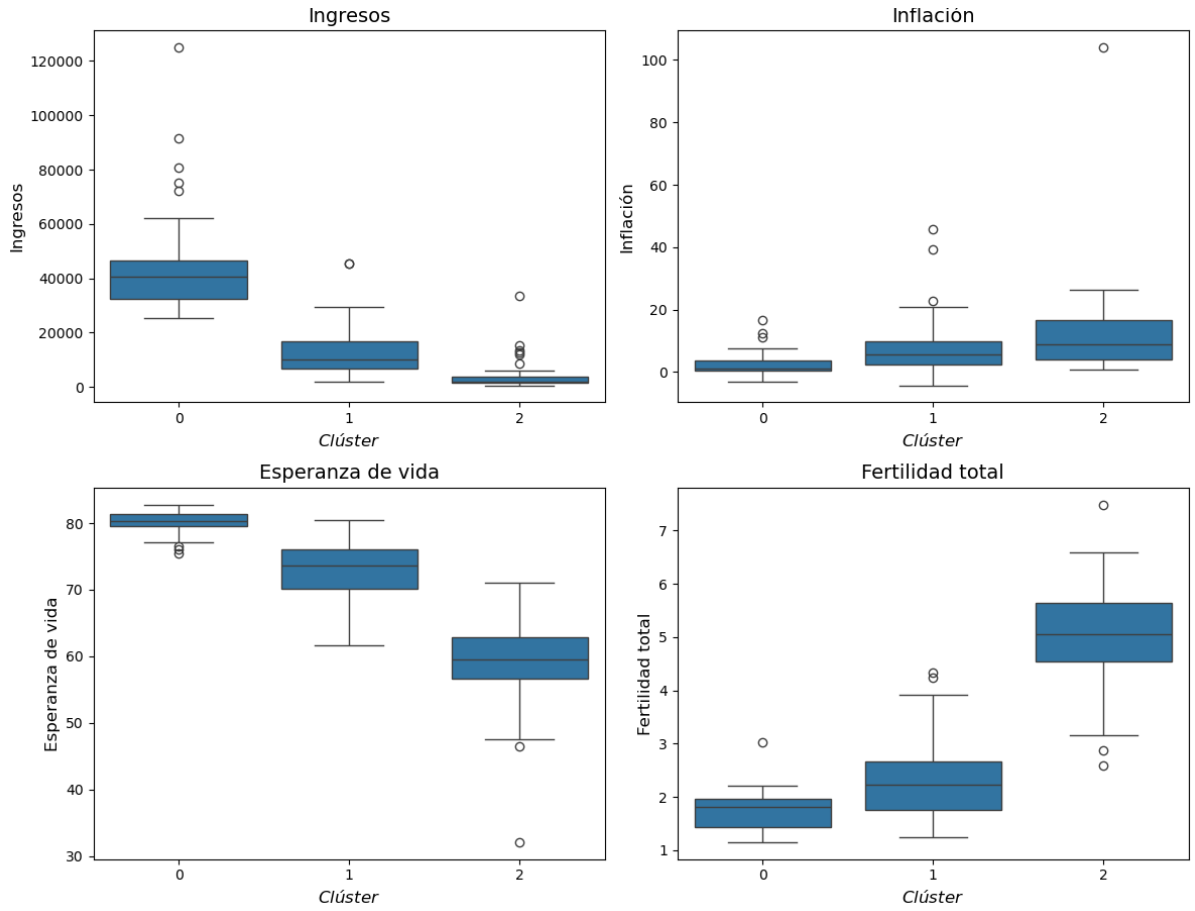


Figura C.0.5: Diagramas de caja para el resto de variables, separando las observaciones por grupos. Los grupos son los clústeres que formó k-medias en la figura 4.1.2 (a).

Anexo D

Resultados de la compresión de una imagen

En este anexo se incluyen los resultados de comprimir una imagen reduciendo el número de colores mostrados en esta. Esta imagen se representa en la última fila, en la columna derecha, de la figura D.0.1. Los comentarios acerca de las figuras de este anexo se encuentran en la sección 4.2.

Figura D.0.1: Aplicación de los métodos de agrupamiento a la compresión de imágenes, para varios números de clústeres, junto a la imagen original (fuente de la imagen: 'Applied Machine Learning'). En este caso cada clúster representa un color diferente. En los títulos de las imágenes se muestra el número de clústeres o colores y el error cuadrático medio.

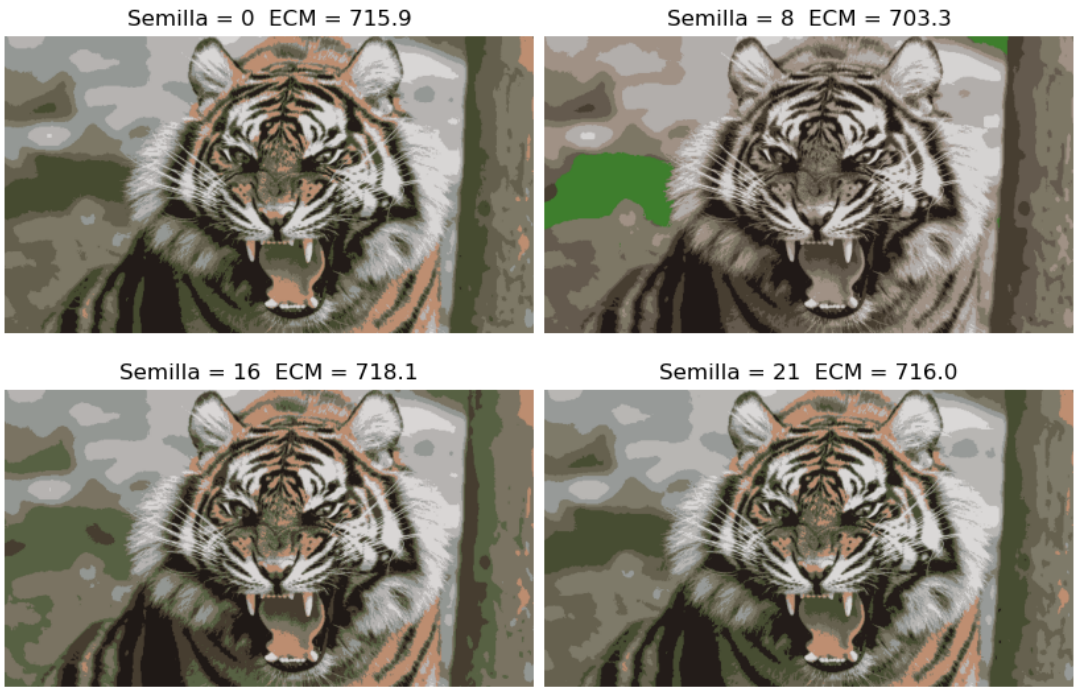


Figura D.0.2: Visualización de cómo el algoritmo k-medias suele no converger a óptimos globales. El resultado final depende de la semilla que determina la elección de los centroides iniciales. Se toma como número de clústeres o colores $k = 8$.

Anexo E

Implementaciones en Python de los algoritmos de agrupamiento

E.1. K-medias directo

```
import numpy as np

class Cesar_Simple_KMeans:
    def __init__(self, n_clusters=8, max_iter=300, random_state=None):
        """
        Initializes the SimpleKMeans class.

        Parameters:
        ----------
        n_clusters (int, default = 8): The number of clusters for the algorithm.
        max_iter (int, default = 300): The maximum number of iterations for
            convergence.
        random_state: (int on None) it fixes the random state in the function np
            .random.seed. It determines random number generation for centroid
            initialization.

        Returns:
        ----------
        None
        """
        self.n_clusters = n_clusters
        self.max_iter = max_iter
        self.random_state = random_state

    def fit(self, X):
        """
        Fits the K-means model to the input data X. Forgy method is used for the
             initialization of the centroids.

        Parameters:
        ----------
        X: The dataset (array-like) of shape (n_samples, n_features).

        Returns:
        ----------
        self: (object) returns the model with computed labels for each point.
        """
        # Forgy initialization: choose k centroids randomly
        np.random.seed(self.random_state)
        random_indices = np.random.permutation(X.shape[0])
        self.centroids = X[random_indices[0:self.n_clusters]]

        for _ in range(self.max_iter):
            # Cluster assignment: find the nearest centroid for each point
            distances = self._compute_distances(X)
            self.labels = np.argmin(distances, axis=1)

            # Recompute centroids: update centroids based on assigned points
            new_centroids = np.array([X[self.labels == i].mean(axis=0) for i in
                range(self.n_clusters)])
```

```
            # Stop iteration if centroids do not change
            if np.all(self.centroids == new_centroids):
                break

            self.centroids = new_centroids

        return self

    def predict(self, X):
        """
        Predicts the cluster labels for new data points.

        Parameters:
        ----------
        - X: The (new) dataset (array-like) of shape (n_samples, n_features) to
            be predicted.

        Returns:
        ----------
        - labels: (array-like) of shape (n_samples) of cluster indices
            corresponding to the input data.
        """
        distances = self._compute_distances(X)
        return np.argmin(distances, axis=1)

    def _compute_distances(self, X):
        """
        Computes the distances between each data point and each centroid.

        Parameters:
        ----------
        X (ndarray): A 2D array where each row represents a data point.

        Returns:
        ----------
        distances: (array-like) of shape (n_samples, n_clusters) where each
            element (i, j) represents the distance from point i to centroid j.
        """
        return np.array([[np.linalg.norm(x - centroid) for centroid in self.
            centroids] for x in X])
```

E.2. K-medias mejorado

```
import numpy as np

class Cesar_Improved_KMeans:
    def __init__(self, n_clusters=8, max_iter=300, tol=1e-4, n_init=5,
        random_state=None):
        """
        Initializes the Cesar_Simple_KMeans class.

        Parameters:
        ----------
        n_clusters (int, default = 8): The number of clusters for the algorithm.
        max_iter (int, default = 300): The maximum number of iterations for
            convergence.
        tol (float, default = 1e-4): The tolerance to declare convergence. The
            iteration stops when the change in centroids is less than this value
            .
        n_init (int, default = 5): The number of times the algorithm will run
            with different centroid seeds.
        random_state: (int on None, default = None) it fixes the random state in
             the function np.random.seed. It determines random number generation
             for centroid initialization.

        Returns:
        ----------
        None
        """
        self.n_clusters = n_clusters
        self.max_iter = max_iter
        self.tol = tol
        self.n_init = n_init
        self.random_state = random_state

    def fit(self, X):
        """
        Fits the K-means model to the input data X. Forgy method is used for the
             initialization of the centroids.

        Parameters:
        ----------
        X: The dataset (array-like) of shape (n_samples, n_features).

        Returns:
        ----------
        self: (object) returns the model with computed labels for each point.
        """
        best_inertia = np.inf  # Store the best result
        best_centroids = None
        best_labels = None

        for _ in range(self.n_init):
            # Initialize centroids using k-means++
            centroids = self._init_centroids(X)

            for _ in range(self.max_iter):
                # Assign each point to the nearest centroid
                distances = self._compute_distances(X, centroids)
                labels = np.argmin(distances, axis=1)

                # Compute new centroids
                new_centroids = np.array([X[labels == i].mean(axis=0) for i in
                    range(self.n_clusters)])
```

```
                # Check if centroids have converged
                centroid_shift = np.linalg.norm(new_centroids - centroids)
                if centroid_shift <= self.tol:
                    break

                centroids = new_centroids

            # Compute inertia (sum of squared distances to centroids)
            inertia = np.sum(np.min(self._compute_distances(X, centroids), axis
                =1) ** 2)

            # Save the best result (with the lowest inertia)
            if inertia < best_inertia:
                best_inertia = inertia
                best_centroids = centroids
                best_labels = labels

        self.centroids = best_centroids
        self.labels = best_labels
        self.inertia = best_inertia

    def _init_centroids(self, X):
        """
        Initializes centroids using the k-means++ method.

        Parameters:
        ----------
        X (ndarray): A 2D array where each row represents a data point.

        Returns:
        ----------
        centroids: (array-like) of shape (n_clusters, n_features), the initial
            centroids.
        """
        np.random.seed(self.random_state)
        centroids = []
        centroids.append(X[np.random.choice(X.shape[0])])

        for _ in range(1, self.n_clusters):
            distances = np.min(self._compute_distances(X, np.array(centroids))
                ** 2, axis=1)
            probabilities = distances / np.sum(distances)
            cumulative_probabilities = np.cumsum(probabilities)
            r = np.random.rand()
            next_centroid = X[np.searchsorted(cumulative_probabilities, r)]
            centroids.append(next_centroid)

        return np.array(centroids)

    def _compute_distances(self, X, centroids):
        """
        Computes the distances between each data point and each centroid.

        Parameters:
        ----------
        X (ndarray): A 2D array where each row represents a data point.
        centroids: (array-like) of shape (n_clusters, n_features).

        Returns:
        ----------
        distances: (array-like) of shape (n_samples, n_clusters) where each
            element (i, j) represents the distance from point i to centroid j.
```

```
        """
        return np.array([[np.linalg.norm(x - centroid) for centroid in centroids
            ] for x in X])

    def predict(self, X):
        """
        Predicts the cluster labels for new data points.

        Parameters:
        ----------
        - X: The (new) dataset (array-like) of shape (n_samples, n_features) to
            be predicted.

        Returns:
        ----------
        - labels: (array-like) of shape (n_samples) of cluster indices
            corresponding to the input data.
        """
        distances = self._compute_distances(X, self.centroids)
        return np.argmin(distances, axis=1)
```

E.3. DBSCAN

```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 28 13:49:45 2024

@author: usuario
"""
import numpy as np
import pandas as pd
class Cesar_DBSCAN:
    def __init__(self, eps=0.5, min_samples=5):
        """
        Initialize the DBSCAN algorithm with the given parameters.

        Parameters:
        ----------
        - eps: (float, default=0.5) The maximum distance between two points for
            them to be considered neighbors.
        - min_samples: (int, default=5) The minimum number of points required to
             form a dense region (core point).
        """
        self.eps = eps
        self.min_samples = min_samples

    def fit(self, X):
        """
        Fit the DBSCAN model to the data.

        Parameters:
        ----------
        - X: The dataset (pd.dataframe or array-like) of shape (n_samples,
            n_features) where each element is a point in the space.

        Returns:
        ----------
        - self: (object) Returns the model with computed labels for each point.
        """
        if isinstance(X, pd.DataFrame):
            X = X.values

        # Initialize labels (-1 for unvisited points)
        labels = [-1] * len(X)
        cluster_id = 0

        # Iterate through each point in the dataset
        for i in range(len(X)):
            if labels[i] == -1:  # Skip if the point has already been visited
                # Find the neighbors within eps distance
                neighbors = self._region_query(X, i)

                if len(neighbors) >= self.min_samples:
                    # If it meets the condition, expand the cluster from this
                        point
                    self._expand_cluster(X, labels, i, neighbors, cluster_id)
                    cluster_id += 1

        # Store the resulting labels as a numpy array
        self.labels_ = labels
        return self

    def _expand_cluster(self, X, labels, point_idx, neighbors, cluster_id):
        """
```

```
        Expand the cluster starting from the given point.

        Parameters:
        ----------
        - X: The dataset (pd.dataframe or array-like) of shape (n_samples,
            n_features).
        - labels: (ndarray of int) The list of labels assigned to each point.
        - point_idx: (int) The index of the starting point for the cluster
            expansion.
        - neighbors: (ndarray of int) List of neighboring point indices for the
            starting point.
        - cluster_id: (int) The ID of the current cluster being expanded.
        """
        if isinstance(X, pd.DataFrame):
            X = X.values

        # Assign the starting point to the current cluster
        labels[point_idx] = cluster_id

        i = 0
        while i < len(neighbors):
            neighbor_idx = neighbors[i]

            if labels[neighbor_idx] == -1:
                # If the point has not been assigned yet, add it to the current
                    cluster
                labels[neighbor_idx] = cluster_id
                new_neighbors = self._region_query(X, neighbor_idx)

                if len(new_neighbors) >= self.min_samples:
                    # Add new neighbors to the list
                    neighbors += new_neighbors

            i += 1

    def _region_query(self, X, point_idx):
        """
        Find all neighbors within the eps distance of a given point.

        Parameters:
        ----------
        - X: The dataset (pd.dataframe or array-like) of shape (n_samples,
            n_features).
        - point_idx: (int) The index of the point to query for neighbors.

        Returns:
        ----------
        - neighbors: (ndarray of int) A list of indices of all points within the
             eps distance of the given point.
        """
        if isinstance(X, pd.DataFrame):
            X = X.values

        neighbors = []
        for i in range(len(X)):
            if np.linalg.norm(X[point_idx] - X[i]) <= self.eps:
                neighbors.append(i)
        return neighbors
```

E.4. Cálculo del índice de Hopkins

```
import numpy as np
from sklearn.neighbors import NearestNeighbors
import pandas as pd

def hopkins(X, portion=0.1, seed=None):
    """
    Given a dataframe, it calculates its Hopkin index

    Parameters
    ----------
    X: the dataset (pd.dataframe or array-like) of shape (n_samples, n_features)
    portion (float, default=0.1): portion of the total patterns used to
        calculate the index
    seed (int or None): fixes the random state to choose the uniform sample and
        the choice of the data sample

    Returns
    -------
    H (float): Hopkins index

    """
    if isinstance(X, pd.DataFrame):
        X = X.values

        # X: numpy array of shape (n_samples, n_features)
    n = X.shape[0]
    d = X.shape[1]
    m = int(portion * n)

    np.random.seed(seed)
    nbrs = NearestNeighbors(n_neighbors=1).fit(X)
    # u_dist
    rand_X = np.random.uniform(X.min(axis=0), X.max(axis=0), size=(m,d))
    u_dist = nbrs.kneighbors(rand_X, return_distance=True)[0]
    # w_dist
    idx = np.random.choice(n, size=m, replace=False)
    w_dist = nbrs.kneighbors(X[idx,:], 2, return_distance=True)[0][:,1]

    U = (u_dist**d).sum()
    W = (w_dist**d).sum()
    H = U / (U + W)
    return H
```

E.5. Cálculo de componentes principales

```
import numpy as np
# The data are assumed to be standardized.
# Calculate the covariance matrix
cov_matrix = np.cov(X_centered, rowvar=False)

# Decompose into eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

# Sort the eigenvalues and eigenvectors in descending order
sorted_indices = np.argsort(eigenvalues)[::-1]  # Change the order
sorted_eigenvalues = eigenvalues[sorted_indices]
sorted_eigenvectors = eigenvectors[:, sorted_indices]

# Select the first k components (e.g., k=2)
k = 2
W = sorted_eigenvectors[:, :k]

# Project the data onto the new components
Z = X_centered @ W

print("Principal components matrix (W):\n", W)
print("Projected data (Z):\n", Z)
```