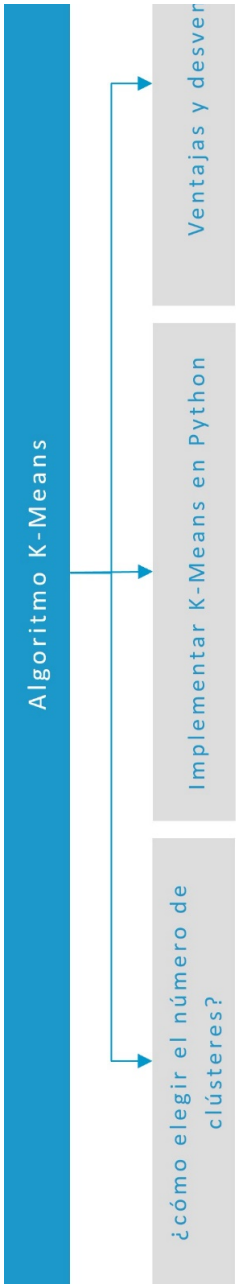
Aprendizaje Automático No Supervisado

Tema 2. Fundamentos y aplicaciones del agrupamiento K-Means

Índice

Esquema



2.1. Introducción y objetivos

En el universo del aprendizaje no supervisado, los **clústeres** son la piedra angular. La esencia del grupo está en la habilidad de proporcionar información sin necesidad de etiquetas. Es como contar un cuento, donde cada clúster narra lo que esconden sus instancias.

Hoy en día la cantidad de datos existentes son cada vez mayor, pero en muchas ocasiones contar con etiquetas es un lujo, siendo estas muy costosas de obtener.

Como ya se ha comentado, este tipo de técnicas se utilizan cuando se desconoce la estructura de los datos, puesto que no se tiene la variable objetivo de los datos. Por ejemplo, cuando se desconoce cuántos tipos de usuarios existen.

Estas técnicas dividen los datos en clústeres o grupos similares. Pero esta división se lleva a cabo sin la necesidad de indicar las características de cada uno de estos grupos. Para lograr la mejor división, las instancias de un grupo deben ser muy similares entre sí y muy distintas entre el resto de los grupos.

En este tema veremos los algoritmos de agrupación, y dentro de ellos nos centraremos en explicar el algoritmo K-Means, ya que es el principal exponente de esta técnica, gracias a su simplicidad y eficacia.

En este tema nos planteamos los siguientes objetivos:

- Presentar una explicación detallada del algoritmo K-Means y su funcionamiento interno.
- Describir el paso a paso de la asignación de centroides y su actualización para minimizar la suma de distancias cuadradas entre los puntos.
- Implementar el algoritmo de K-Means en un entorno de programación, utilizando Python.

Ideas clave

2.2. Algoritmo K-Means

El **algoritmo K-Means** particiona los datos en K clústeres distintos, cada uno de ellos representado por la media de los puntos; de ahí viene su nombre: K-Means o K-Medias. Su objetivo es minimizar la varianza intra-clúster y maximizar la varianza inter-clúster, haciendo que los grupos tengan datos lo más diferentes posibles entre sí y, al mismo tiempo, asegurar que los elementos dentro de cada grupo sean lo más similares posible entre sí (Van Der Post y Smith, 2021).

El algoritmo K-Means es una herramienta fundamental para los científicos de datos, gracias a su simplicidad y potencia. Es capaz de dividir un conjunto de datos complejo en particiones simples y digeribles.

Para entender mejor el algoritmo, introduciremos el concepto de los **diagramas de Voronoi.** Estos diagramas también conocidos como **polígonos de Thiessen** son una forma de dividir el espacio en regiones basadas en la proximidad a un conjunto específico de puntos. Cada región en el diagrama de Voronoi está asociado a uno de estos sitios y contiene todos los puntos del espacio que están más cerca de ese sitio que de cualquier otro. Son una herramienta importante en diversos campos, como la ciencia de datos, la geometría computacional y la visualización de datos.

Características clave:

- Cada región en un diagrama de Voronoi está delimitada por las líneas bisectrices, que son las líneas que están equidistantes entre dos sitios vecinos.
- Los vértices de los polígonos de Voronoi están ubicados en puntos donde tres o más sitios se encuentran equidistantes.
- Los diagramas de Voronoi pueden ser bidimensionales (2D) o tridimensionales (3D), dependiendo de la dimensión del espacio en el que se definen los puntos generadores.

- Los diagramas de Voronoi se utilizan comúnmente en aplicaciones de análisis espacial, como la planificación urbana, el diseño de redes de comunicación, la interpolación de datos y la visualización de datos geográficos.

A continuación, se puede ver un ejemplo en Python de la construcción de un diagrama de Voronoi.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import Voronoi, voronoi_plot_2d

# Generar puntos aleatorios
points = np.random.rand(5, 2)

# Calcular el diagrama de Voronoi
vor = Voronoi(points)

# Graficar el diagrama de Voronoi
voronoi_plot_2d(vor)
plt.plot(points[:,0], points[:,1], 'ro') # Plotear los puntos generados
plt.show()
```

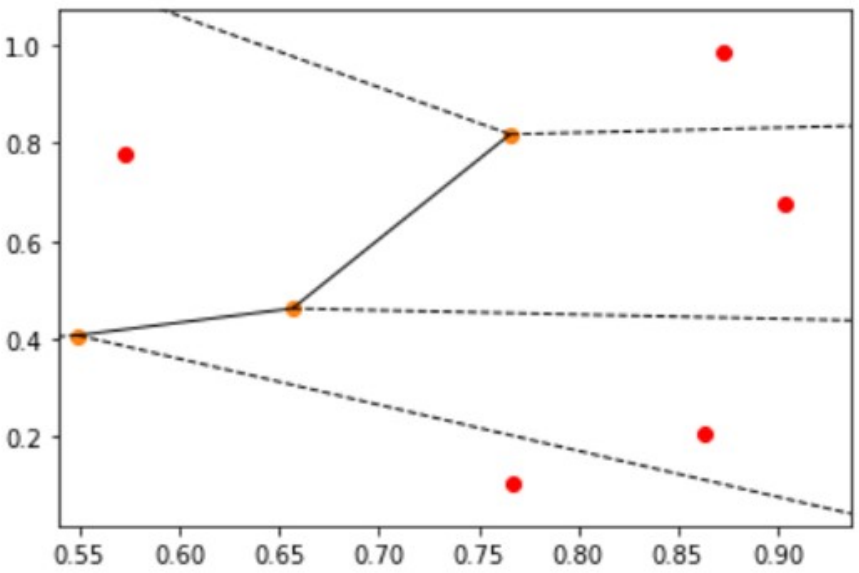


Figura 1. Diagrama de Voronoi generado con puntos aleatorios. Fuente: elaboración propia.

En el contexto de K-Means, estos diagramas se utilizan para ilustrar cómo se

agrupan los datos en torno a los centroides. Cada región de Voronoi corresponde a un clúster generado por K-Means, donde todos los puntos dentro de una región están más cerca de su respectivo centroide que de cualquier otro. Esto visualiza claramente la asignación de datos a clústeres y ayuda a comprender la eficiencia y la precisión del algoritmo en la partición del espacio de datos.

Una vez entendido el concepto de los diagramas de Voronoi, veremos en detalle el algoritmo K-Means.

El paso a paso del algoritmo es el siguiente:

- **Preprocesamiento de datos:** el preprocesamiento de datos en el aprendizaje no supervisado es importante para mejorar la calidad y precisión de los modelos, al igual que en el aprendizaje supervisado.
- **Inicialización de centroides:** el ciclo comienza seleccionando aleatoriamente K puntos como centroides iniciales. Los centroides son puntos que representan el centro de cada uno de los K grupos que se quieren identificar.

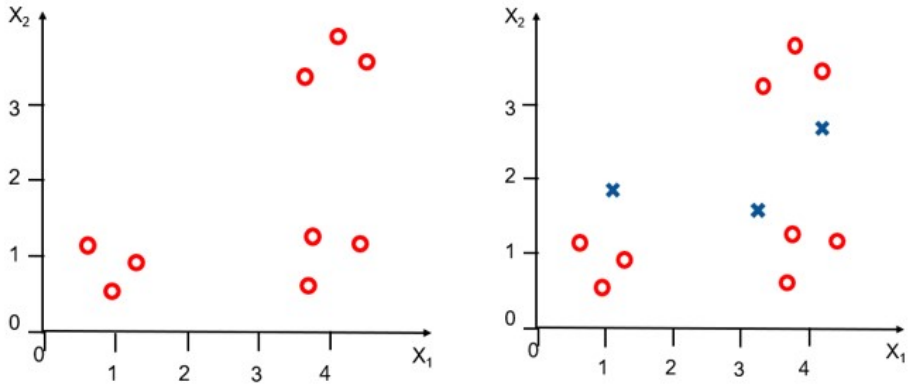


Figura 2. Inicialización de centroides. Fuente: elaboración propia.

- **Asignación de puntos a centroides:** para cada punto del conjunto de datos, se calcula la distancia entre ese punto y cada uno de los centroides. Luego, se asigna el punto al centroide más cercano.

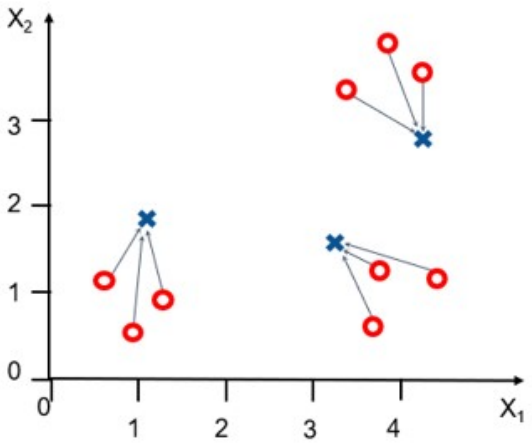


Figura 3. Asignación de puntos a los centroides. Fuente: elaboración propia.

- **Actualización de centroides:** después de asignar cada punto a un centroide, se recalcula la posición de cada centroide como el centroide del grupo de puntos asignados a él. Es decir, que el nuevo centroide es el promedio de todas las coordenadas de los puntos asignados al grupo.

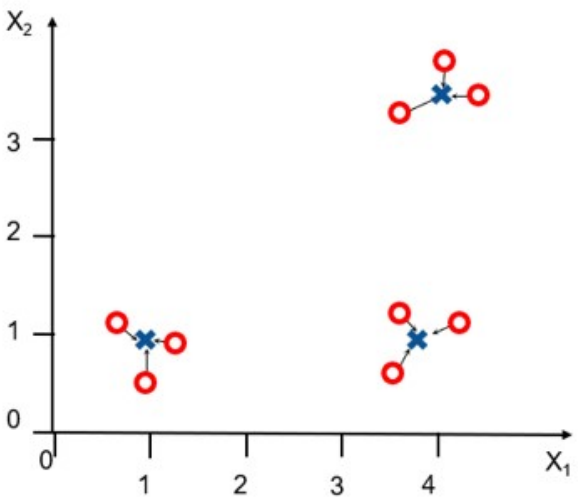


Figura 4. Actualización de centroides. Fuente: elaboración propia.

- **Repetición:** los pasos de asignación de centroides y de actualización de centroides se repiten iterativamente hasta que los centroides dejen de cambiar de manera significativa o hasta que se alcance un número máximo de iteraciones.

- **Convergencia:** cuando los centroides permanecen constantes entre iteraciones y los puntos están asignados a los grupos de manera estable, podemos hablar de convergencia.
- **Resultado final:** un conjunto de K grupos, donde cada grupo está representado por su centroide.

El poder de K-Means reside en su versatilidad, ya que se puede aplicar a una gran variedad de conjuntos de datos. Desde segmentar clientes hasta la compresión de imágenes. La necesidad de predefinir el número de grupos, K, puede ser un desafío. Veremos a continuación algunas de las técnicas existentes para facilitar la elección del valor de K.

2.3. ¿Cómo elegir el número de clústeres?

Elegir un valor de K erróneo implica que se tendrán grupos que no representan al conjunto de datos, lo que implica que no nos va a brindar información suficiente sobre cada uno de los grupos.

En muchos casos no conocemos ese número mágico que nos indica cuál es la cantidad de grupos que deben generarse. Si conocemos el dominio, debería ser sencillo saber cuántos grupos se desean crear. Pero, lo cierto, es que no es tan fácil de encontrar.

En la Figura 5 podemos ver una simulación del algoritmo K-Means con diferentes valores de K para un mismo conjunto de datos. El mejor valor de K es cuatro, se pueden ver los grupos o clústeres de datos bien diferenciados. Cualquier otro valor de K va a generar un modelo que no es adecuado para este conjunto de datos. En este caso hemos utilizado la fuerza bruta, intentando configurar el algoritmo con diferentes valores de K.

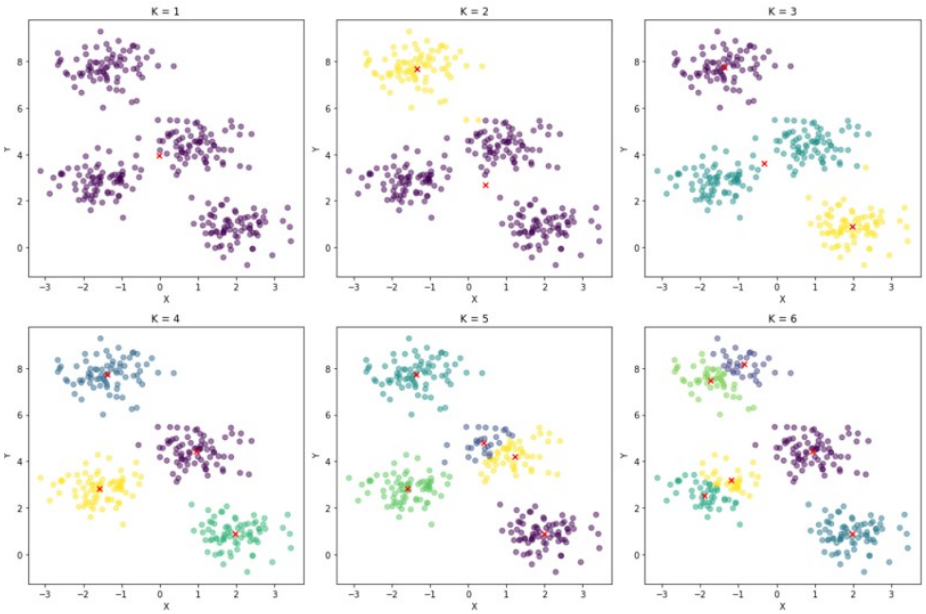


Figura 5. Diferentes valores de K para un mismo conjunto de datos. Fuente: elaboración propia.

Ideas clave

A continuación, veremos algunos métodos y/o métricas conocidas para encontrar el número óptimo de clústeres. Pero antes recordemos dos principios básicos:

- Los puntos dentro del grupo deben ser lo más similares posible.
- Los puntos que pertenecen a distintos grupos deben ser lo más distintos posible.

Métricas

A continuación, veremos dos métricas que se utilizarán posteriormente en los métodos para hallar el número ideal de clústeres o valores de K. Estamos hablando de la inercia y del coeficiente de la silueta.

Inercia

La primera métrica es la inercia. Se calcula como la suma de las distancias al cuadrado entre los puntos de datos y los centros de los grupos a los que pertenece. Este valor cuantifica la variación dentro de cada uno de los grupos. La siguiente ecuación muestra la fórmula de la inercia I.

$$I = \sum_{i=1}^{n} \left\| x_i - \mu_{c(i)} \right\|^2$$

Donde:

- n es el número total de puntos.
- x_i es el i-ésimo punto de datos.
- $\mu_{c(i)}$ es el centroide del clúster al que pertenece el punto x_i.
- $\left\| x_i - \mu_{c(i)} \right\|^2$ es la distancia euclidiana cuadrada entre el punto x_i y su centroide correspondiente $\mu_{c(i)}$.

En la Figura 6, se puede observar el cálculo de la inercia I para cada valor de k. La

inercia es una medida de que tan compactos están los clústeres. Generalmente disminuye a medida que aumenta k, pero una disminución de la inercia no garantiza obtener la mejor agrupación. Así lo podemos observar en la Figura 6, donde el mejor valor de k es 4 y el valor de la inercia para esta configuración no es el menor valor. Más adelante combinaremos la inercia con el método del codo, para poder decidir cuál es el valor óptimo de K.

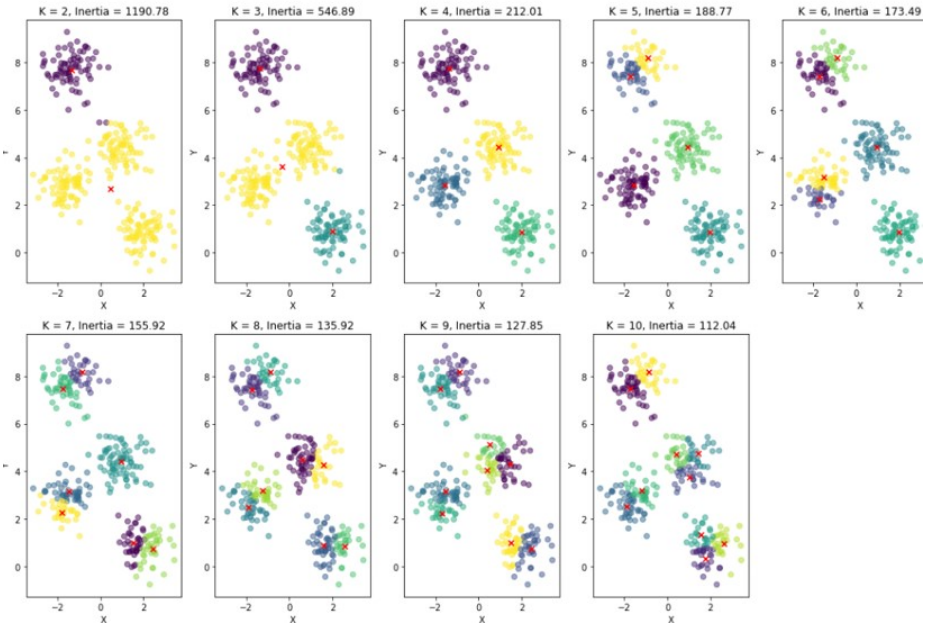


Figura 6. Diferentes valores de K con su respectivo valor de inercia para un mismo conjunto de datos.

Fuente: elaboración propia.

Como hemos visto anteriormente, la inercia es una función decreciente, a medida que aumenta k, la inercia disminuye. Asumiendo que K es el número óptimo de clústeres, tenemos que para k<K la inercia disminuye rápidamente; mientras que para un k>K la disminución de la inercia va más lenta.

El **método del codo** consiste en trazar la inercia en función del número de clústeres y observar el punto en el que la inercia comienza a desacelerarse, formando un codo. Este punto puede indicarnos que el valor óptimo para K es el punto donde comienza a formarse el codo (Dutta, 2020). Sin embargo, puede ser un método muy

subjetivo, sobre todo en conjuntos de datos complejos.

En la Figura 7 se puede ver cómo el valor de la inercia va disminuyendo rápidamente hasta que en k=4 empieza a disminuir, formando un codo en el gráfico.

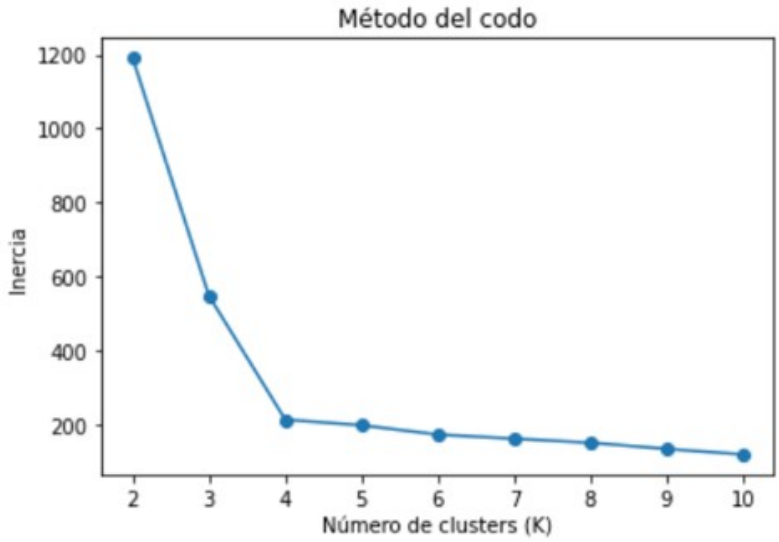


Figura 7. Método del codo utilizando el valor de la inercia. Fuente: elaboración propia.

Como dato interesante, el método del codo ha sido utilizado para ver en problemas de lenguaje natural la cantidad óptima de temas en una red social.

Coeficiente de la silueta

La segunda métrica es el **coeficiente de la silueta** *(silhouette coefficient).* Nos muestra un resumen de la variación dentro de un grupo y la variación entre grupos. Para cada instancia se calcula la distancia al centro del grupo al que pertenece (a este valor lo denominamos a). Además, calculamos la distancia al segundo mejor centro del grupo (a este valor lo denominamos b). Cuando hablamos del segundo mejor grupo hacemos referencia al grupo más cercano y que no es el grupo al que pertenece el dato. Con los valores a y b aplicamos la siguiente fórmula.

$$s = (b - a) / max\ (a,b)$$

En un escenario ideal, la distancia a es muy pequeña en comparación con la distancia b, y el valor de s es un valor cercano a 1. Si la distancia a y b son similares, entonces s tiende a 0. Si la agrupación no es lo suficientemente óptima para el conjunto de datos, la distancia a es mayor que la distancia b y el valor de s es negativo o cercano a -1.

En la Figura 8 podemos ver que el coeficiente de la silueta más alto es para un valor de K = 4. Y el valor más bajo lo tenemos para un valor de K = 9, donde a y b son similares.

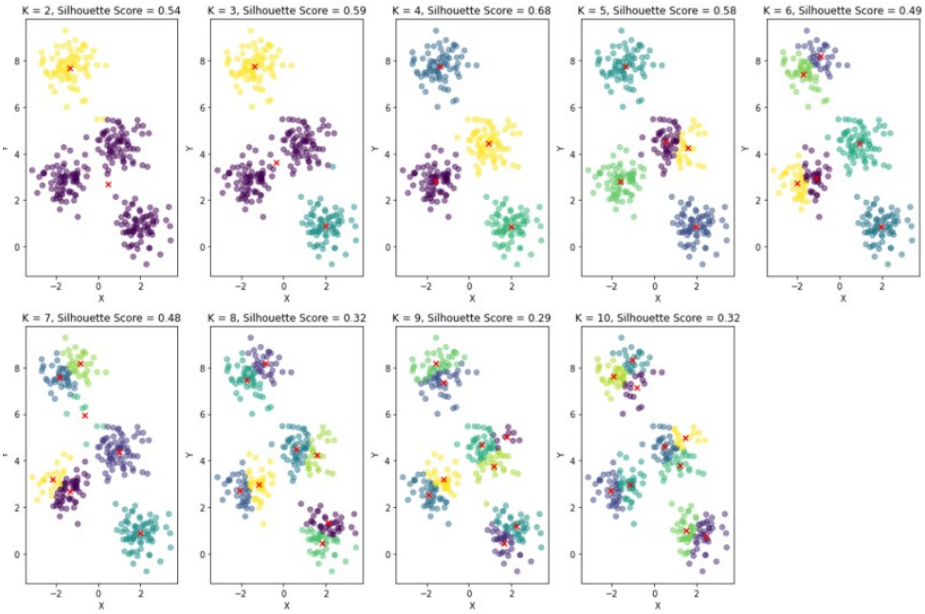


Figura 8. Diferentes valores de K con su respectivo coeficiente de la silueta para un mismo conjunto de datos. Fuente: elaboración propia.

Utilizando el coeficiente de la silueta, visto anteriormente, se utiliza el **método de la silueta** para hallar el valor óptimo de k:

- Se calcula la puntuación de la silueta para cada punto del conjunto de datos.
- Se promedian esos puntos para cada grupo.

- Se trazan los puntajes promedios para diferentes valores de k.
- El valor de K óptimo es aquel en el que la puntuación de la silueta es el promedio más alto.

En la Figura 9 se puede observar que el valor más alto es para un K=4. Coincide con el valor obtenido a través del método del codo.

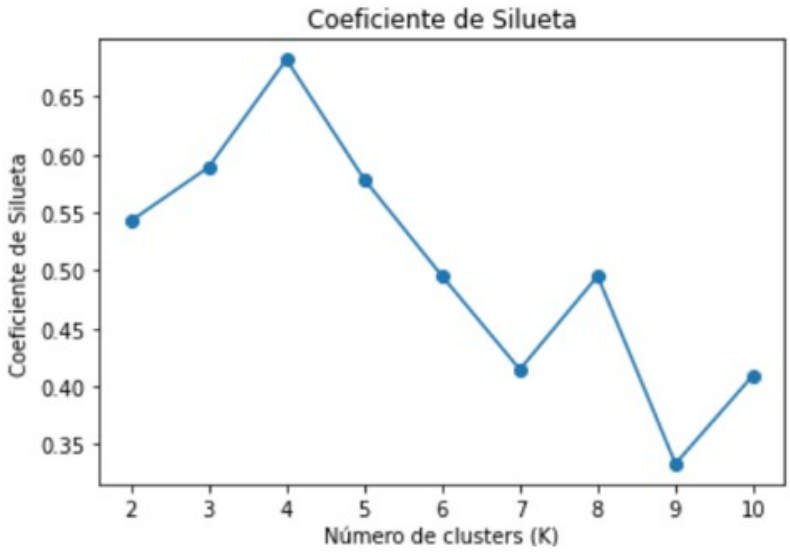


Figura 9. Método de la silueta para obtener el valor óptimo de K. Fuente: elaboración propia.

Estadística de brecha

Es una medida utilizada para evaluar la calidad de los clústeres generados por el algoritmo K-Means. Esta métrica compara la inercia intra-clúster de un conjunto de datos con la que se esperaría encontrar en un conjunto de datos aleatorios sin estructura de *clustering,* con la inercia de los clústeres generados con los datos originales (Dutta, 2020).

- **Dispersión intra-clúster con datos originales:** primero se calcula la inercia intra-clúster de los datos originales utilizando el algoritmo K-Means para diferentes valores de k. Este valor mide cuán compactos y cohesivos son los clústeres formados por el algoritmo para cada valor de k.

- **Dispersión intra-clúster con datos aleatorios:** se generan varios conjuntos de datos de referencia aleatorios (también conocidos como conjuntos de datos nulos) manteniendo la estructura de los datos originales, pero mezclando aleatoriamente las etiquetas de los puntos. Para cada conjunto de datos de referencia se calcula la inercia intra-clúster utilizando K-Means.

- **Cálculo de la brecha:** la brecha se calcula como la diferencia entre la inercia intra-clúster esperada en los datos aleatorios y la inercia intra-clúster observada en los datos originales. Se promedia esta diferencia sobre todos los conjuntos de datos de referencia.

- **Selección del número óptimo de clústeres:** el número óptimo de clústeres se elige el valor de K que maximiza la brecha. En otras palabras, se busca el punto donde la dispersión intra-clúster observada en los datos reales es menor que la dispersión de los datos aleatorios.

A pesar de la falta de organización, los datos aleatorios agrupados producen inercias decrecientes constantemente a medida que k aumenta. Esto es debido a que cuantos más centros de conglomerados hay, menor es la distancia entre los puntos de datos y los centroides, lo que produce una disminución de la inercia. Con los datos originales la inercia varía disminuyendo rápidamente para los valores de k que están por debajo del valor óptimo de K, después tiende a estabilizarse o disminuye en menor medida. Cuando se trazan juntas la inercia de los datos observados y aleatorios, la diferencia puede observarse (Figura 10). En este caso, el número óptimo de clústeres puede estar entre 3, 4 y 5.

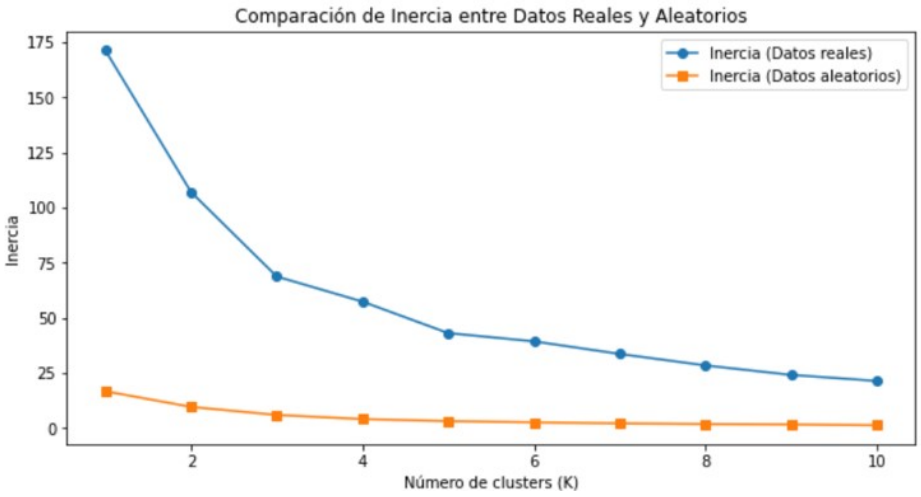


Figura 10. Estadística de brecha para obtener el valor óptimo de K. Fuente: elaboración propia.

2.4. Implementar K-Means en Python

En este apartado aplicaremos K-Means al conjunto de datos que podemos encontrar en https://www.kaggle.com/datasets/camnugent/california-housing-prices?resource=download, adaptación del *dataset* encontrado en https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html, cuya fuente original es Statlib (https://lib.stat.cmu.edu/).

El código que presentamos a continuación es una adaptación del artículo de blog: https://www.datacamp.com/tutorial/k-means-clustering-python.

```
import numpy as np # operaciones numéricas
import pandas as pd # manipulación y análisis de datos
import matplotlib.pyplot as plt #creación de gráficos
from sklearn.cluster import KMeans # módulo KMeans de scikit-learn
from sklearn.preprocessing import StandardScaler # preprocesamiento
from sklearn.metrics import silhouette_score #Coeficiente silhouette
```

Leeremos el conjunto de datos y tendremos en cuenta solo las columnas de latitud, longitud y valor medio del inmueble para nuestro ejercicio.

```
datos_inmuebles = pd.read_csv('housing.csv', usecols = ['longitude',
'latitude', 'median_house_value'])
```

Verificamos que no existan valores faltantes.

```
datos_faltantes = datos_inmuebles.isnull().sum()

# Mostrar el número de valores faltantes por columna
print("Valores faltantes por columna:")
print(datos_faltantes)

Valores faltantes por columna:
longitude              0
latitude               0
median_house_value     0
dtype: int64
```

Ahora utilizamos la biblioteca `seaborn` para hacer un gráfico de dispersión. Indicando los valores de los ejes, en este caso utilizaremos latitud y longitud.

Dentro del diagrama de dispersión configuraremos el atributo `hue`, para asignar una categoría a los puntos de acuerdo con su valor promedio. El grafico resultante mostrará los puntos de datos con etiquetas de color. Veamos el código y el resultado.

```
import seaborn as sns

sns.scatterplot(data = datos_inmuebles, x='longitude', y='latitude', hue =
'median_house_value')
```

```
<Axes: xlabel='longitude', ylabel='latitude'>
```

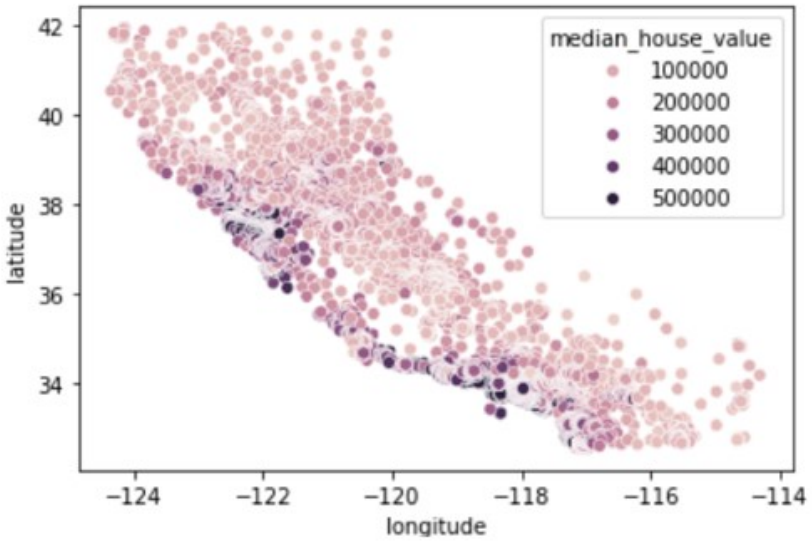


Figura 11. Gráfico de dispersión, indicando valores en los ejes y categoría según valor medio de la vivienda. Fuente: elaboración propia.

Existen valores de inmuebles de todas las categorías en todas las latitudes y longitudes. Usaremos normalización o estandarización de valores, debido a que hay valores como el valor medio de la vivienda que es muy alto y comparado con la latitud y la longitud.

Creamos una instancia de la clase `StandardScaler` del módulo de preprocesamiento de Scikit-learn. Este método calcula la media y la desviación estándar de cada una de

las características, estandariza las variables y garantiza que se tenga un rango y una varianza similar en todos los atributos. De esta manera evitamos que una de las características sea más dominante que otra en el algoritmo de K-Means.

```
#se crea una instancia de StandardScaler para la normalización

escalado=StandardScaler()

#Aplicamos el StandardScaler al DataFrame
#El método 'fit_transform' escala y estandariza los datos
datos_escalados = escalado.fit_transform(datos_inmuebles)

#Se calculan los coeficientes de la silueta para diferentes valores de k

coeficientes_silhoutte = []

#Se crea una iteración sobre cada valor de k de 2 hasta 10

for k in range(2,11):
    #Instancia de KMeans
    kmeans= KMeans(n_clusters=k)
    #trabajamos con los datos escalados
    kmeans.fit(datos_escalados)
    #Calcular el coeficiente de la silueta
    coeficiente=silhouette_score(datos_escalados, kmeans.labels_)
    #Agregar el puntaje al vector de coeficientes
    coeficientes_silhoutte.append(coeficiente)

#Graficar los coeficientes de la silueta para diferentes valores de k.
plt.plot(range(2,11), coeficientes_silhoutte, marker='o')
plt.xlabel('Número de clusters (k)')
plt.ylabel('Coeficiente de la silueta')
plt.title('Coeficiente de la silueta para diversos valores de k')
plt.show
```

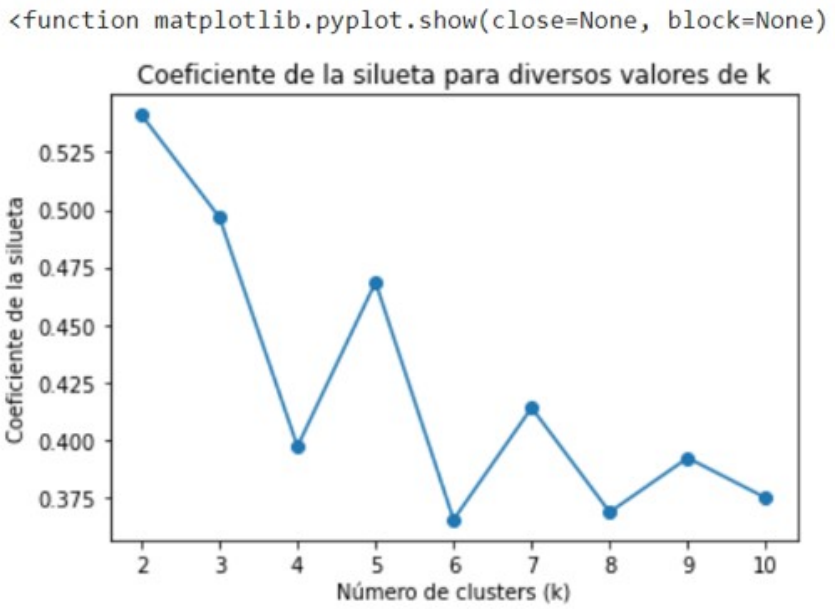


Figura 12. Coeficientes de la silueta para diferentes valores de k. Fuente: elaboración propia.

Analizando los resultados de la Figura 12, el número óptimo de clústeres es 2. Este es el valor más alto que tenemos y es el valor que nos va a permitir tener grupos bien diferenciados.

Entrenamos el algoritmo de K-Means con dos clústeres, calculamos los centroides y asignamos cada punto a un grupo.

```
#Establecemos el número de grupos (k) en 2.

k=2

#Utilizamos el algoritmo de KMeans con una semilla aleatoria para poder
reproducir el entrenamiento
kmeans1= KMeans(n_clusters=k, init='k-means++', random_state=42)

#Modelamos con los datos escalados
kmeans1.fit(datos_escalados)

# Creamos los grupos y las etiquetas de los centroides

labels= kmeans1.labels_
centroids= kmeans1.cluster_centers_
```

```
#Mostramos el gráfico de dispersión
plt.scatter(datos_escalados[:,0], datos_escalados[:, 1], c=labels, s=50,
cmap='viridis')
plt.scatter(centroids[:,0], centroids[:, 1], marker='x', s=200, c='red')
plt.xlabel('longitud')
plt.ylabel('latitud')
plt.title('Centroides de los grupos obtenidos por K-Means')
plt.show()
```

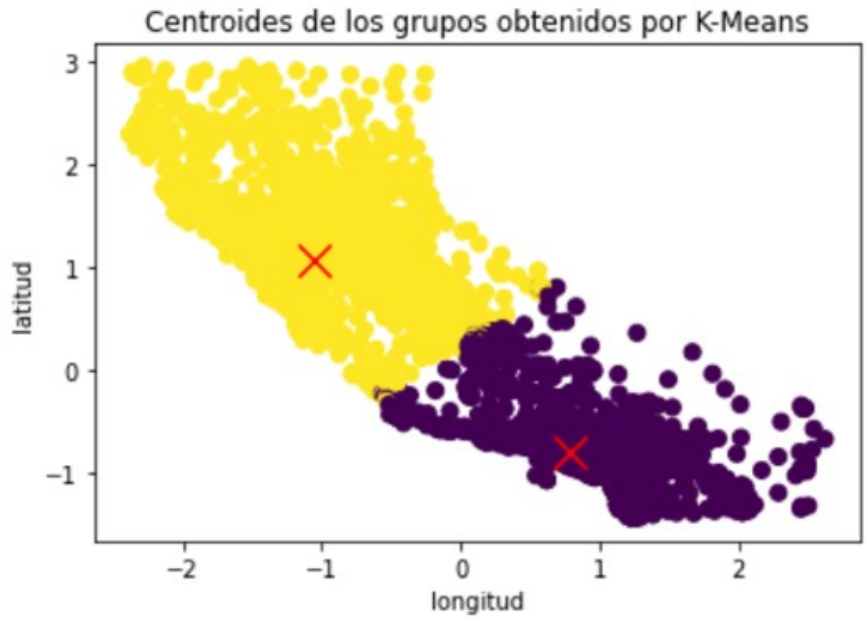


Figura 13. Clústeres obtenidos con K=2. Fuente: elaboración propia.

En la Figura 13 observamos 2 clústeres bien diferenciados, donde uno de ellos contiene los inmuebles con latitud alta y longitud baja, y al otro grupo pertenecen los inmuebles con longitudes altas y latitudes bajas. Ahora comparemos con el método del codo.

```
#Función que calcula la inercia para cada valor de k.
def plot_kmeans(dataset, max_k):
    inertias = []

    for i in range(2, max_k+1):
        # Entrenar el modelo de K-Means
        kmeans = KMeans(n_clusters=i)
        kmeans.fit(dataset)
```

```
        # Calcular la inercia
        inertia = kmeans.inertia_
        inertias.append(inertia)

    # Graficar el método del codo
    plt.plot(range(2, max_k+1), inertias, marker='o')
    plt.title('Método del codo')
    plt.xlabel('Número de clusters (K)')
    plt.ylabel('Inercia')
    plt.show()

# Visualizar el método del codo para diferentes valores de K
plot_kmeans(datos_escalados, max_k=10)
```

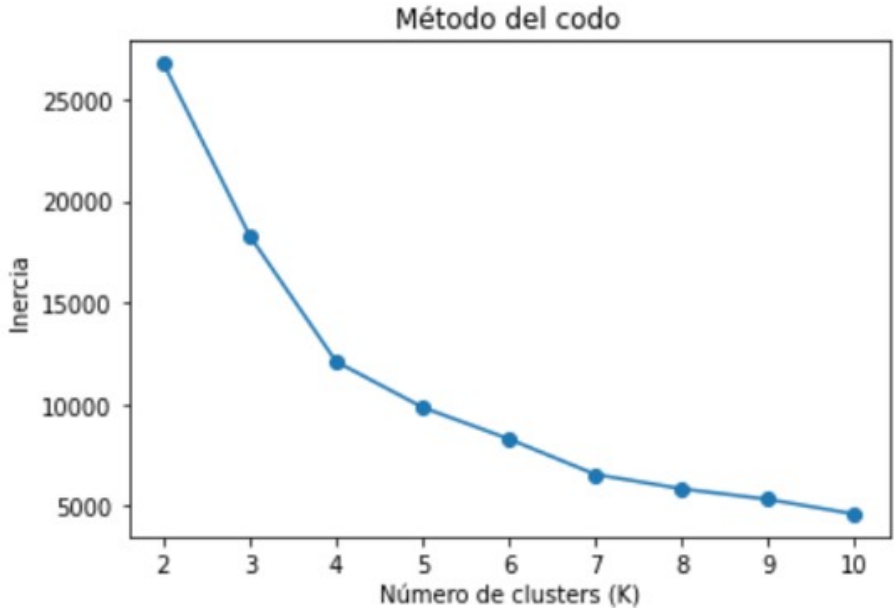


Figura 14. Valores de inercia vs. número de clústeres, método del codo. Fuente: elaboración propia.

En la Figura 14 podemos observar que el codo se crea en K=4. Vamos a entrenar un modelo con ese valor de k y comparamos los resultados.

```
#Establecemos el número de grupos (k) en 4.

k=4

#Utilizamos el algoritmo de KMeans con una semilla aleatoria para poder
reproducir el entrenamiento
```

```
kmeans2= KMeans(n_clusters=k, init='k-means++', random_state=42)

#Modelamos con los datos escalados
kmeans2.fit(datos_escalados)

# Creamos los grupos y las etiquetas de los centroides

labels= kmeans2.labels_
centroids= kmeans2.cluster_centers_

#Mostramos el gráfico de dispersión
plt.scatter(datos_escalados[:,0], datos_escalados[:, 1], c=labels, s=50,
cmap='viridis')
plt.scatter(centroids[:,0], centroids[:, 1], marker='x', s=200, c='red')
plt.xlabel('longitud')
plt.ylabel('latitud')
plt.title('Centroides de los grupos obtenidos por K-Means')
plt.show()
```

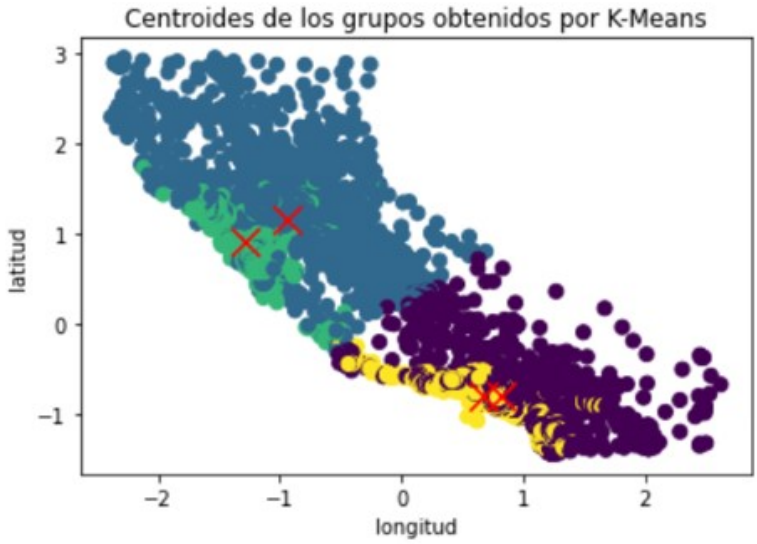
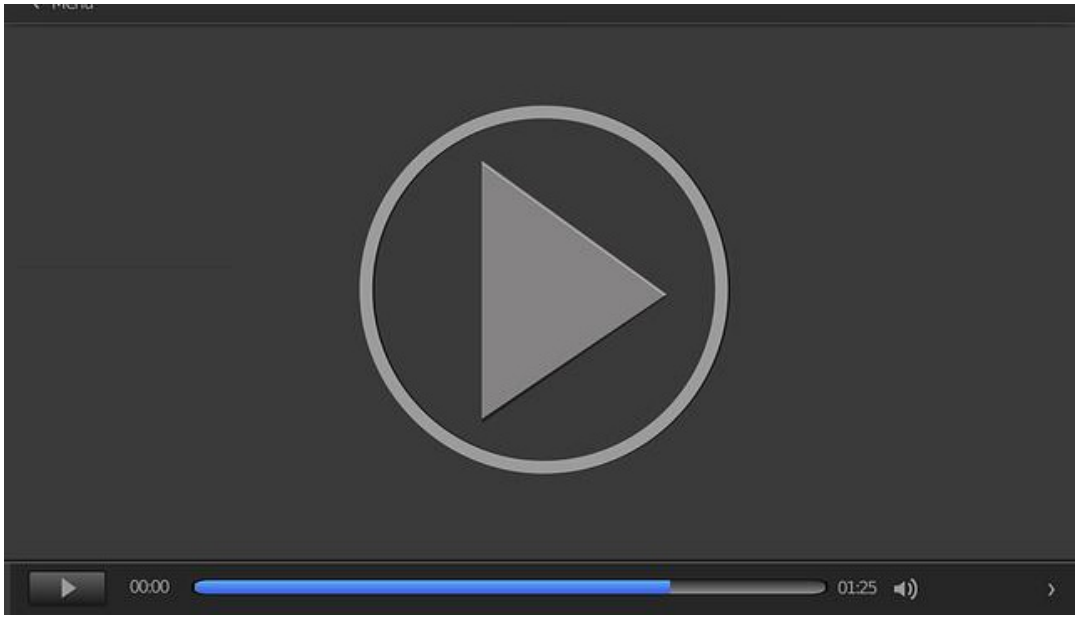


Figura 15. Clústeres obtenidos con K=4. Fuente: elaboración propia.

En la Figura 15 vemos 4 clústeres, pero no están bien diferenciados. Para este ejemplo el coeficiente de la silueta nos dio una mejor aproximación del número de clústeres óptimos.

A continuación, veremos el vídeo ***Implementación de K-Means en Python.***

Ideas clave



Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=e273ad18-e29d-4da6-8e23-b1bd018609c0

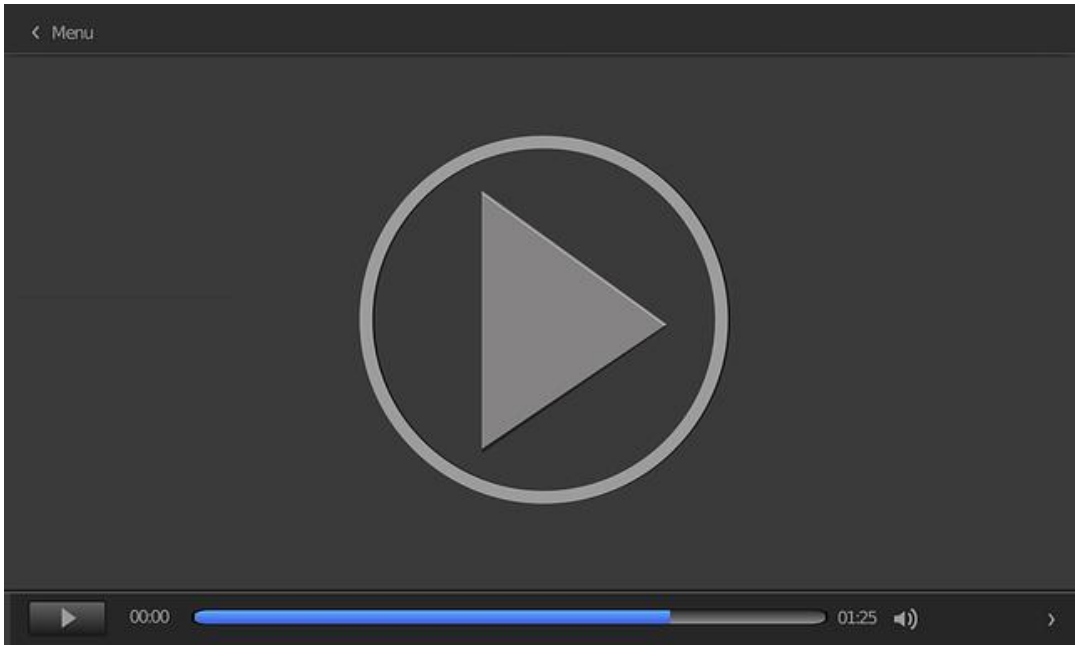
2.5. Ventajas y desventajas del K-Means

A continuación, se listan una serie de ventajas y desventajas que se tiene con este algoritmo.

VENTAJAS	DESVENTAJAS
Es un algoritmo: ▸ Muy popular. ▸ Sencillo de aplicar. ▸ Eficiencia computacional. ▸ Capacidad para manejar datos de alta dimensionalidad. ▸ Escalabilidad. ▸ Flexibilidad. ▸ Interpretabilidad.	▸ Debe definirse el número de grupos o clústeres. Puede ser difícil de encontrar el valor óptimo. ▸ Sensibilidad a los valores *outliers*.

Tabla 1. Ventajas y desventajas de K-Means. Fuente: elaboración propia.

A continuación, veremos el vídeo ***Aplicaciones y casos de estudio.***



Accede al vídeo:

Ideas clave

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=7a04fc7e-beeb-45c7-8cad-b1be00ed52d0

2.6. Cuaderno de ejercicios

Partiremos de este conjunto de datos de Github https://github.com/SooyeonWon/customer_analytics_fmcg/blob/main/segmentation%20data.csv.

Asumimos que somos científicos de datos y vamos a analizar este conjunto de datos de los clientes de una empresa de bienes de consumo que contiene las siguientes columnas: *sex, marital status, edad, education, income, occupation, settlement size.*

La empresa quiere segmentar a los clientes para ayudar al departamento de marketing para lanzar nuevos productos en función de la segmentación.

Ejercicio 1

Explora, describe y preprocesa los datos para poder aplicar el algoritmo de K-Means.

Solución

- Leer el conjunto de datos, explorarlo y describirlo:

```
import pandas as pd
df= pd.read_csv('segmentation data.csv', index_col = 0)
df.head()
df.describe()
df.info()
```

- Preprocesamiento:

```
scaler = StandardScaler()
df_std = scaler.fit_transform(df)
df_std = pd.DataFrame(data = df_std,columns = df.columns)
```

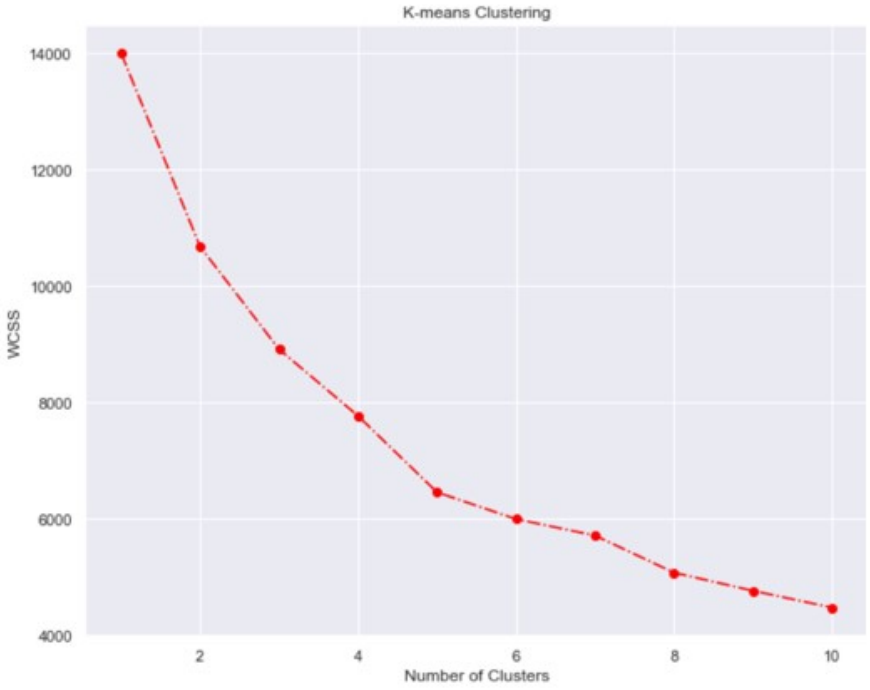
Ejercicio 2

Utiliza la técnica del codo para identificar el mejor número de clústeres o grupos. Entrena un algoritmo de K-Means con el valor de K encontrado.

Solución

```
wcss = []
for i in range(1,11):
    kmeans_pca = KMeans(n_clusters = i, init = 'k-means++', random_state =
42)
    kmeans_pca.fit(df_std)
    wcss.append(kmeans_pca.inertia_)

plt.figure(figsize = (10,8))
plt.plot(range(1, 11), wcss, marker = 'o', linestyle = '-.',color='red')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.title('K-means Clustering')
plt.show()
```




```
kmeans = KMeans(n_clusters = 4, init = 'k-means++', random_state = 42)
kmeans.fit(df_std)
df_segm_kmeans= df_std.copy()
df_std['Segment K-means'] = kmeans.labels_
df_std
```

Ejercicio 3

Utiliza la técnica de estadística de brecha para identificar el mejor número de clústeres o grupos. Entrena un algoritmo de K-Means con el valor de K encontrado.

Solución

- Función para calcular la estadística de la brecha:

```
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances
import matplotlib.pyplot as plt

def calcular_brecha(datos, n_refs=20, max_k=10):
    """
    Calcula la estadística de la brecha para un rango de valores de K.

    Parameters:
    - datos: matriz de datos normalizados.
    - n_refs: número de conjuntos de datos de referencia.
    - max_k: número máximo de clústeres a evaluar.

    Returns:
    - brechas: valores de la brecha para cada K.
    """
    shape = datos.shape
    # Generar datos de referencia aleatorios
    referencias = np.random.random_sample(size=(n_refs, shape[0],
shape[1]))

    brechas = []

    for k in range(1, max_k + 1):
        # Entrenar K-Means con datos originales
        kmeans = KMeans(n_clusters=k)
        kmeans.fit(datos)
        distancias_originales = pairwise_distances(datos,
kmeans.cluster_centers_)
        Wk_original = np.sum(np.min(distancias_originales, axis=1)) /
datos.shape[0]

        # Calcular Wk para datos de referencia
        Wk_referencia = []
        for ref in referencias:
            kmeans_ref = KMeans(n_clusters=k)
            kmeans_ref.fit(ref)
            distancias_ref = pairwise_distances(ref,
```

```
kmeans_ref.cluster_centers_)
            Wk_referencia.append(np.sum(np.min(distancias_ref, axis=1)) /
ref.shape[0])

        Wk_ref_promedio = np.mean(Wk_referencia)
        brecha = np.log(Wk_ref_promedio) - np.log(Wk_original)
        brechas.append(brecha)

    return brechas

# Calcular la brecha para valores de K de 1 a 10
brechas = calcular_brecha(datos_escalados, max_k=10)

# Graficar la estadística de la brecha
plt.plot(range(1, 11), brechas, marker='o')
plt.xlabel('Número de clusters (K)')
plt.ylabel('Estadística de la brecha')
plt.title('Estadística de la Brecha para diferentes valores de K')
plt.show()

# Determinar el mejor valor de K
mejor_k = np.argmax(brechas) + 1
print(f"El mejor número de clústeres según la estadística de la brecha es:
{mejor_k}")
```

Ejercicio 4

Utiliza la técnica de coeficientes de la silueta para identificar el mejor número de clústeres o grupos. Entrena un algoritmo de K-Means con el valor de K encontrado. Compara y saca conclusiones sobre cuál es el mejor método para este conjunto de datos.

Solución

- Método de la silueta:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

def plot_silhouette(dataset, max_k):
    silhouette_scores = []
```

```
    for i in range(2, max_k+1):
        # Entrenar el modelo de K-Means
        kmeans = KMeans(n_clusters=i)
        kmeans.fit(dataset)

        # Calcular el coeficiente de silueta
        silhouette_avg = silhouette_score(dataset, kmeans.labels_)
        silhouette_scores.append(silhouette_avg)

    # Graficar el coeficiente de silueta
    plt.plot(range(2, max_k+1), silhouette_scores, marker='o')
    plt.title('Coeficiente de Silueta')
    plt.xlabel('Número de clusters (K)')
    plt.ylabel('Coeficiente de Silueta')
    plt.show()

# Visualizar el coeficiente de silueta para diferentes valores de K
plot_silhouette(df_std, max_k=10)
```

- El mejor método es la técnica de la silueta.

2.7. Referencias bibliográficas

Dutta, I. (2020, octubre 25). *Cheat sheet for implementating 7 methods for selecting the optimal number of clusters in Python.* Medium. https://towardsdatascience.com/cheat-sheet-to-implementing-7-methods-for-selecting-optimal-number-of-clusters-in-python-898241e1d6ad

Van Der Post, H. y Smith, M. (2023). *Unsupervised Machine Learning: with Python.* Reactive Publishing.

Segmentación de clientes paso a paso

Raúl Valerio-Statistics. (2023, junio 30). *Segmentación de clientes paso a paso: descubre el RFM y K-Means en acción | Tutorial Python Cluster* [Vídeo]. YouTube. https://www.youtube.com/watch?v=CsXCeOiKRZw

Se detalla paso a paso, desde la preparación de los datos hasta la interpretación de los resultados de la segmentación.

K-Means clustering

Six Sigma Pro SMART. (2023, agosto 10). *K-Means Clustering | K-Means++ Clustering | Cluster Analysis | Data Science* [Vídeo]. YouTube. https://www.youtube.com/watch?v=KgJNIMe4yVc

Es una guía del algoritmo K-Means.

K-Means en R

Raúl Valerio. (2023, junio 23). *Clustering en Series de Tiempo: K-Means en R| Descubre número óptimo con Métodos Silhouette y Elbow* [Vídeo]. YouTube. https://www.youtube.com/watch?v=C64Xyp3X-0U

Explica de forma muy dinámica la diferencia entre estos conceptos.

Test

1. ¿Cuál es el objetivo principal del algoritmo K-Means?

   A. Reducir la dimensión de los datos.

   B. Dividir un conjunto de datos en grupos similares.

   C. Generar reglas de asociación entre variables.

   D. Clasificar datos en categorías predefinidas.

2. ¿Qué técnica se utiliza para encontrar el valor óptimo de K en K-Means?

   A. Coeficiente de Pearson.

   B. Análisis de componentes principales (PCA).

   C. Regresión lineal.

   D. Coeficiente de la silueta.

3. ¿Cuál es la base del método del codo para encontrar el valor óptimo de K?

   A. El análisis de correlación entre variables.

   B. La minimización de la distancia intra-clúster.

   C. La maximización del índice de Davies-Bouldin.

   D. La reducción de la inercia total del modelo.

4. ¿Qué compara la estadística de brecha en K-Means?

   A. La dispersión intra-clúster de los datos reales con la esperada en datos aleatorios.

   B. La dispersión inter-clúster de los datos reales con la esperada en datos aleatorios.

   C. La precisión del modelo en datos de entrenamiento con datos de prueba.

   D. La dispersión intra-clúster de los datos con la dispersión inter-clúster.

5. ¿Cuáles es una de las ventajas que tiene aplicar el modelo K-Means?

A. La interpretabilidad de los resultados.

B. El alto coste computacional en conjuntos de datos grandes.

C. No requiere especificar el número de clústeres.

D. No es adecuado para datos no lineales.

6. ¿Qué medida se utiliza en el coeficiente de la silueta para evaluar la cohesión y separación de los clústeres?

A. Entropía.

B. Distancia intra-clúster.

C. Coeficiente de correlación.

D. Distancia inter-clúster.

7. ¿Cuál es la desventaja común de K-Means en la práctica?

A. Sensible a la inicialización de los centroides.

B. Puede manejar fácilmente datos no lineales.

C. Requiere grandes cantidades de memoria.

D. No es escalable para grandes conjuntos de datos.

8. ¿Qué indica un valor del coeficiente de la silueta de -1 en el algoritmo K-Means?

A. Un clúster muy cohesionado.

B. Un punto de datos mal asignado a un clúster.

C. Un clúster con poca separación de otros clústeres.

D. Un clúster con mucha variabilidad.

9. ¿Cuál es el mejor método para hallar el valor óptimo de K?

A. El método del codo.

B. El coeficiente de la silueta.

C. La estadística de brecha.

D. Ninguno en particular, depende del conjunto de datos.

10. ¿K-Means y KNN son lo mismo?

A. No, pero K-Means y KNN se parecen en que son algoritmos de aprendizaje supervisado.

B. No, porque K-Means es un algoritmo de aprendizaje no supervisado y KNN es un algoritmo de aprendizaje supervisado.

C. No, pero K-Means y KNN se parecen en que son algoritmos de aprendizaje no supervisado.

D. Sí, son el mismo algoritmo.