Modelos no supervisados © EDICIONES ROBLE, S.L.

Indice

Modelos no supervisados

I. Introducción

En esta unidad se estudiará el análisis de componentes principales y algunos modelos principales de aprendizaje no supervisado. En este tipo de modelos, los conjuntos de datos usados para entrenar no disponen de —o no utilizan— la solución del problema, como sucede en los modelos supervisados, por lo que son especialmente eficaces para la identificación de patrones desconocidos.

La unidad comenzará con el repaso del análisis de componentes principales (PCA,*Principal Component Analysis*), los cuales pueden servir para reducir la dimensionalidad de los conjuntos de datos. Mediante esta técnica, se pueden encontrar las direcciones en el espacio en las que la variabilidad de los datos es máxima y usarlas para proyectar las variables.

Una vez realizado el estudio PCA, se presentarán los algoritmos de análisis de clúster k-means, agrupación jerárquica y DBSCAN. Estos modelos pueden identificar conjuntos de datos que son similares entre sí y diferentes al resto de datos.

En la unidad también se repasará el concepto de distancia. Este concepto es importante en los algoritmos de análisis de clúster, ya que es la forma de medir lo similares o disimilares que son dos registros.

Esta unidad se complementa con un notebook Python en el que se incluye todo el código utilizado y algunos ejemplos adicionales a los que se hace referencia en el texto. Es aconsejable seguir el texto con este notebook U4_códigos.

II. Objetivos

Los objetivos que se deberían alcanzar en esta unidad son:

- Saber aplicar análisis de componentes principales a un conjunto de datos.
- Conocer diferentes métricas para medir la similitud de los datos.
- Comprender y utilizar el algoritmo de k-means.
- Poder estimar el número de clústeres existentes en un conjunto de datos.
- Comprender y utilizar el proceso de clasificación mediante árbol jerárquico.
- Comprender y utilizar el algoritmo DBSCAN.

III. Análisis de componentes principales

El análisis de componentes principales (PCA, *Principal Component Analysis*) es una técnica utilizada para reducir la dimensionalidad de los conjuntos de datos en la que se proyectan las variables originales en unos nuevos ejes espaciales. Estos nuevos ejes son los que se conocen como componentes principales. El algoritmo de PCA busca los ejes espaciales (los componentes principales) que contengan la mayor cantidad de información (varianza) y estima la parte de la información que ha de recoger cada uno de estos. Una vez obtenidos estos nuevos datos, los datos originales se pueden proyectar sobre los nuevos ejes de coordenadas para seleccionar únicamente aquellos que recogen la parte de la información que se desea conservar.

Calcular Matriz de covarianza

La implementación del PCA requiere calcular la varianza y covarianza del conjunto de datos para construir la matriz de covarianza.

Se puede recordar que la varianza de una variable se define como:

$$var(x) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1} = \left(\frac{1}{n-1}\sum_{i=1}^{n} x_i^2\right) - \bar{x}^2$$

x_i representa el registro i de la variable$x$, $\bar{x}$ representa el valor promedio y n es el número de registros.

Por otro lado, la covarianza de un par de variables se define mediante la expresión:

$$cov(x,y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n-1}$$

y_i representa el registro i de la variable$y$ e $\bar{y}$ representa el valor promedio.

A partir de estas definiciones es fácil ver que varianza es igual a la covarianza de una variable consigo misma, es decir:

$$cov(x,x) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(x_i - \bar{x})}{n-1} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1} = var(x)$$

A partir de esto, se puede construir la matriz de covarianza:

$$\begin{bmatrix} cov(x_1,x_1) & cov(x_1,x_2) & cov(x_1,x_3) \\ cov(x_2,x_1) & cov(x_2,x_2) & cov(x_2,x_3) \\ cov(x_3,x_1) & cov(x_3,x_2) & cov(x_3,x_3) \end{bmatrix}$$

En Python, la forma de obtener esta matriz es mediante el comando np.cov, disponible en la librería numpy.

Calcular valores y vectores propios

Por otro lado, para la obtención de los componentes principales es necesario calcular los valores propios (o autovector) y vectores propios (o autovectores) de las matrices de correlación. Un escalar diferente de cero, λ, es el valor propio de una matriz cuadrada de orden m, A, si existe un vector de orden m no nulo, $\vec{v}$ tal que se verifique:

$$A\vec{v} = \lambda\vec{v}$$

En este caso, se dice que $\vec{v}$ es el vector propio de la matriz asociado al valor propio λ. Para estimar los vectores propios y los valores propios se puede multiplicar el valor propio por la matriz identidad, I, y despejar en la ecuación anterior:

$$(A - \lambda I)\vec{v} = 0$$

De este modo, como, por definición, el vector propio no puede ser nulo, para que se verifique esta expresión el determinante ha de ser nulo:

$$|A - \lambda I| = 0$$

A partir de aquí se obtiene un sistema de m ecuaciones con el que se pueden obtener los valores propios de la matriz.

En Python los valores propios y los vectores propios de una matriz se pueden obtener utilizando la función np.linalg.eig, disponible en la librería numpy.

Proceso para obtener los componentes principales de un conjunto de datos

El proceso para obtener los componentes principales de un conjunto de datos se resume en los siguientes pasos:

1. A cada una de las variables se le resta el valor medio de la misma.
2. Se construye la matriz de covarianza.
3. Se descompone la matriz es sus valores propios y vectores propios.
4. Se seleccionan los k vectores propios que se corresponden con los k valores propios más grandes, donde k es la dimensión del nuevo espacio de variables menor o igual al anterior.
5. Se proyectan los datos utilizando los k vectores propios para obtener los componentes principales.

La selección del número de vectores propios se realiza utilizando el porcentaje de la información que recoge cada uno de los componentes principales. Este porcentaje se obtiene como la fracción que representa cada uno de los valores propios del total, es decir:

$$\frac{\lambda_i}{\sum_{i=1}^{m} \lambda_i}$$

m es el número de variables en el conjunto de datos.

Este proceso se ha implementado en el código de la figura 3.1.

```
x = np.array([[0.9, 1],
              [2.4, 2.6],
              [1.2, 1.7],
              [0.5, 0.7],
              [0.3, 0.7],
              [1.8, 1.4],
              [0.5, 0.6],
              [0.3, 0.6],
              [2.5, 2.6],
              [1.3, 1.1]])

y = np.array([x.T[0] - mean(x.T[0]),
              x.T[1] - mean(x.T[1])])
c = np.cov(y)

l, v = np.linalg.eig(c)

print "Los vectores propios son: ", v[0], "y", v[1]
print "Los valores propios son: ", l

print "Primer componente: ", dot(y.T, v.T[0])
print "Segundo componente: ", -dot(y.T, v.T[1])
```

Figura 3.1. Obtención de los componentes principales del conjunto de datos.

En la figura 3.1. se define un conjunto de datos de forma manual, sobre el que se aplicarán los pasos del PCA definidos anteriormente. En primer lugar, se crea un nuevo conjunto de datos (y), substrayendo en cada una de las columnas su media. Posteriormente se calcula la matriz de covarianza (c), sus valores propios (l) y vectores propios (v). El primer valor propio es 1,25 y el segundo es 0,033, por lo que es el primer componente principal que se obtiene mediante la proyección de los datos sobre el primer vector propio, esto se realiza en las dos últimas líneas.

Estos resultados indican que el primer componente principal representa aproximadamente el 97% de la varianza total.

Análisis utilizando scikit-learn

El análisis se puede repetir utilizando scikit-learn. Para esto se ha de importar el constructor PCA, construir un objeto con las opciones y ejecutar la función fit_transform sobre el conjunto de datos. La opción más habitual en el constructor es el número máximo de componentes principales que se desean obtener, en caso de que no se indique un valor se obtendrán todos los componentes.

En la figura 3.2. se muestra un ejemplo de esto, en el cual se obtienen los mismos resultados que con el código de la figura 3.1.

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
x_pca = pca.fit_transform(x)

print x_pca

print "Varianza explicada con la primera componentes:", pca.explained_variance_ratio_[0]
print "Varianza explicada con la segunda componentes:", pca.explained_variance_ratio_[1]
```

Figura 3.2. PCA con scikit-learn.

Además de los componentes principales se puede obtener la varianza total de los datos que explica cada uno de los componentes utilizando la propiedad explained_variance_ratio_ de los objetos PCA, con lo que se puede validar el resultado obtenido previamente.

El PCA se puede utilizar para la representación de conjuntos de datos que tengan más de dos dimensiones. Por ejemplo, el conjunto de datos Iris disponible entre los ejemplos de scikit-learn. Este conjunto de datos dispone de 50 muestras de tres especies de flor, en los que se han medido cuatro características de cada flor: el largo y el ancho de los sépalos y pétalos. El código para la carga y la visualización de las dos primeras características se muestra en la figura 3.3., mientras que en la figura 3.4. se muestra la gráfica obtenida.

```
from sklearn import datasets

iris_names  = datasets.load_iris().target_names
iris_target = datasets.load_iris().target
iris_values = datasets.load_iris().data

scatter(iris_values[iris_target == 0, 0], iris_values[iris_target == 0, 1], c='r')
scatter(iris_values[iris_target == 1, 0], iris_values[iris_target == 1, 1], c='g')
scatter(iris_values[iris_target == 2, 0], iris_values[iris_target == 2, 1], c='b')
legend(iris_names)
```

Figura 3.3. Carga de los datos del conjunto de datos Iris.

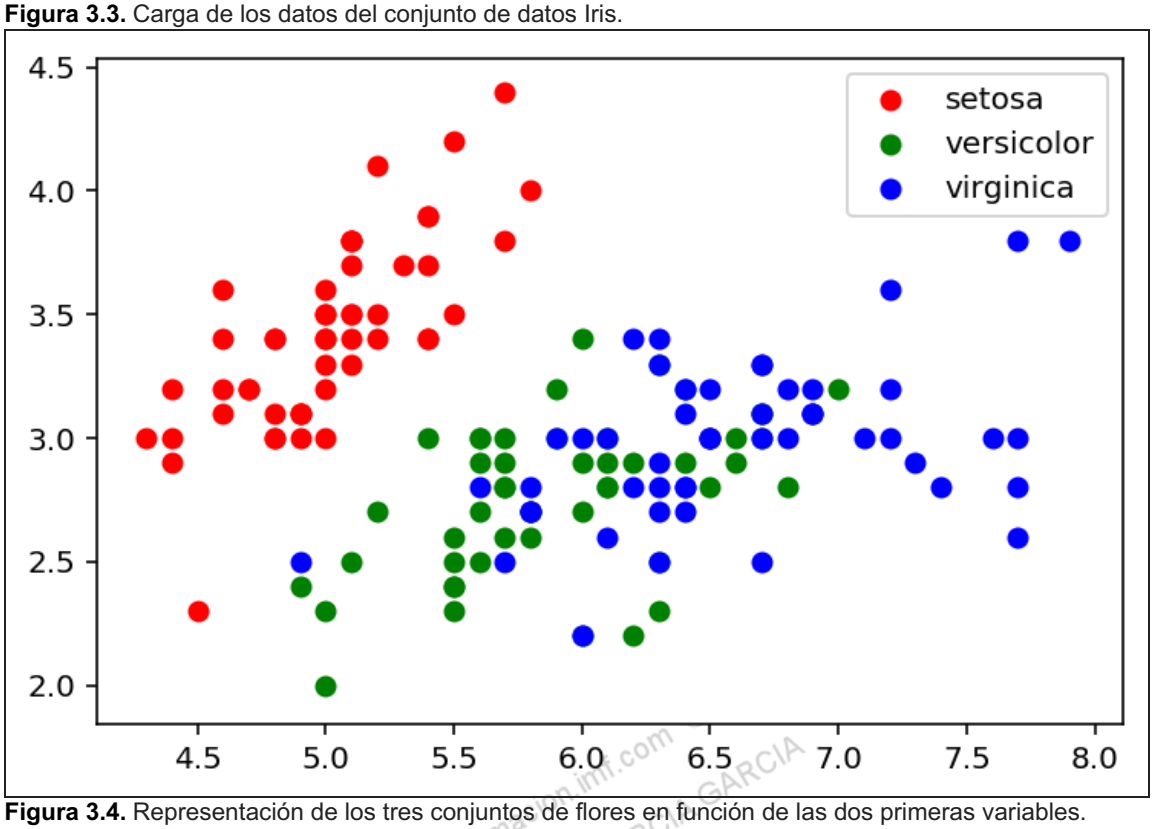


Figura 3.4. Representación de los tres conjuntos de flores en función de las dos primeras variables.

El análisis de componentes principales del conjunto Iris y la obtención de la varianza explicada por cada componente se pueden realizar utilizando el código de la figura 3.5.

```
pca      = PCA(n_components = None)
iris_pca = pca.fit_transform(iris_values)

for i in range(shape(iris_pca)[1]):
    explained_var = pca.explained_variance_ratio_[range(i + 1)].sum()
    print "Varianza explicada con", i + 1, "componentes:", explained_var
```

Figura 3.5. Obtención de los componentes principales del conjunto de datos Iris.

Al ejecutar el código de la figura 3.5. se comprueba que los dos primeros componentes principales llegan a explicar el 97 % de la varianza. El resultado de representar los dos primeros componentes principales se puede ver en la figura 3.6.

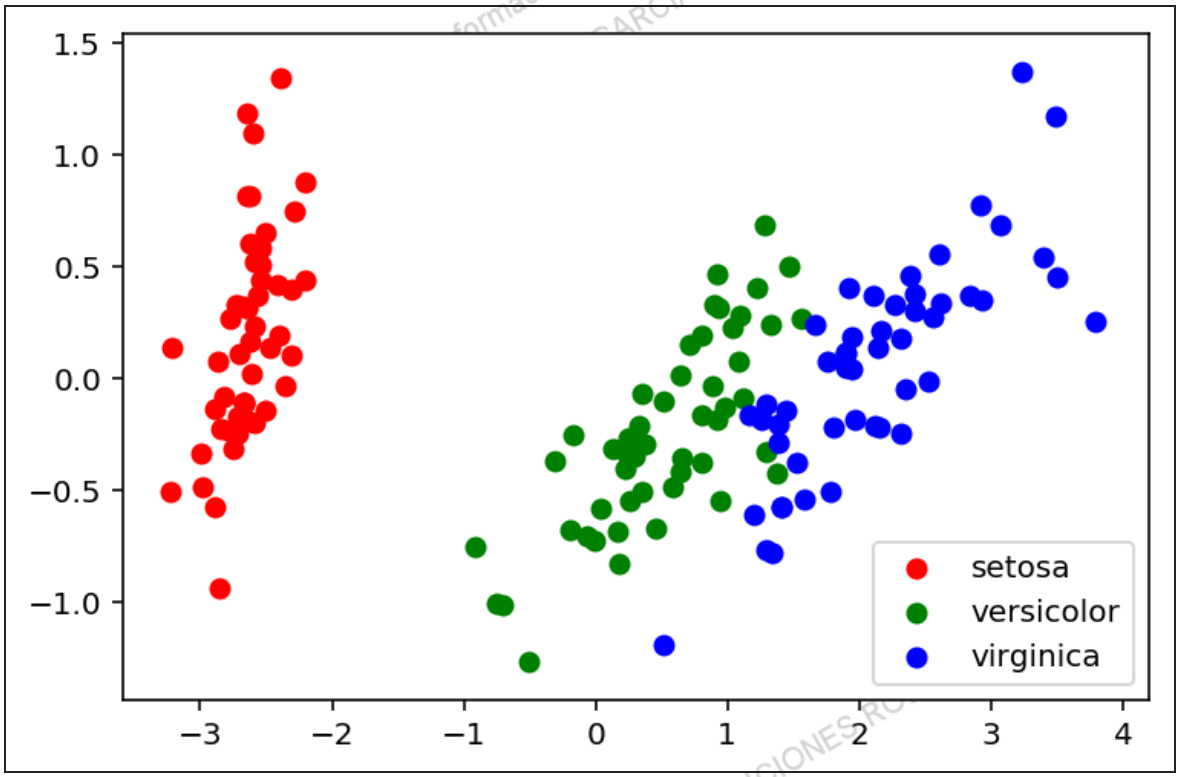


Figura 3.6. Representación de los dos primeros componentes principales del conjunto de datos Iris.

IV. Identificación de objetos similares con k-means

Uno de los problemas típicos que se puede resolver con técnicas de aprendizaje automático es la identificación de los grupos en los que se pueden agrupar los registros. Este es un problema no supervisado, debido a que, generalmente, los posibles grupos en los que se divide el conjunto de datos no son conocidos *a priori*, es decir, no existe una etiqueta previa que asignar. Este tipo de problemas se resuelve mediante el análisis de clúster o conglomerados. El análisis de clúster hace referencia a las familias de técnicas que permiten agrupar las observaciones de conjuntos de datos en clústeres, de tal manera que los miembros asignados a un mismo clúster muestren una mayor similitud entre sí que con los miembros de otros. Una de las técnicas más utilizadas para realizar el análisis de clúster es k-means, ya que resulta fácil de implementar y, al mismo tiempo, la interpretación de los resultados es sencilla.

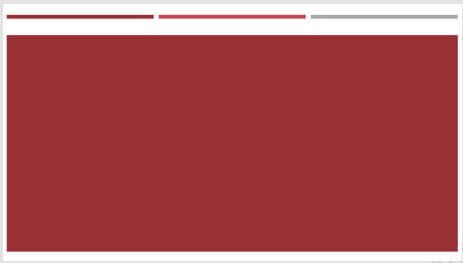
En esta sección se estudiará k-means, pero antes es necesario repasar el concepto de distancia para poder medir lo similar o disimilar que son dos registros en un conjunto de datos.

4.1. Midiendo la similitud entre dos registros

El método de k-means y, en general, los métodos de análisis de clúster requieren poder medir la similitud existente entre todos los registros del conjunto de datos. Una forma de hacer esto es asumir que los datos son puntos un espacio n-dimensional en el que se define una distancia con la que se mide la separación entre dos registros, de esta forma, la similitud de dos registros se relaciona con la inversa de su distancia.

La distancia más conocida y utilizada es la distancia euclídea, ya que es la que se utiliza en el día a día para medir la separación de dos puntos. Además, se puede obtener fácilmente mediante una regla. Matemáticamente, una distancia o métrica es una función, $d(a, b)$, que asigna un número a cada par de puntos de un espacio n-dimensional, $a = (a_1, a_2, \cdots, a_n)$), y verifica las siguientes propiedades:

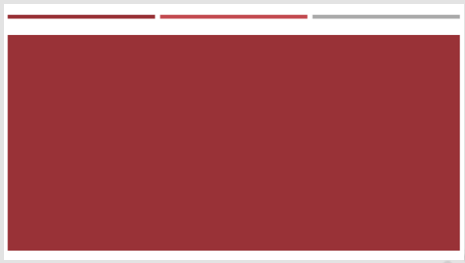
No negativa



El valor nunca puede ser menor de cero.

$$d(a, b) \geq 0$$

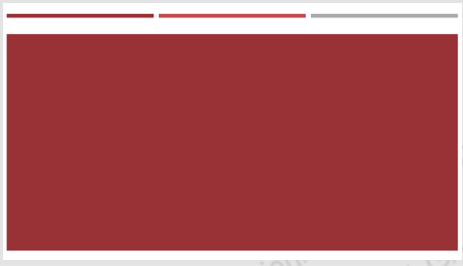
Simétrica



Garantizando que la distancia entre a y b sea la misma que entre b y a.

$$d(a, b) = d(b, a)$$

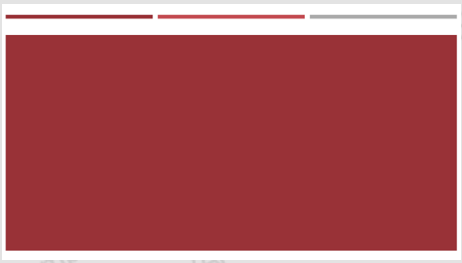
Desigualdad triangular



La distancia entre dos puntos ha de ser menor o igual que la suma de las distancias de los puntos originales a un punto intermedio. Es decir, la distancia que hay que recorrer en dos caras de un triángulo es siempre menor o igual que la de la otra cara.

$$d(\boldsymbol{a}, \boldsymbol{b}) \leq d(\boldsymbol{a}, \boldsymbol{c}) + d(\boldsymbol{c}, \boldsymbol{b})$$

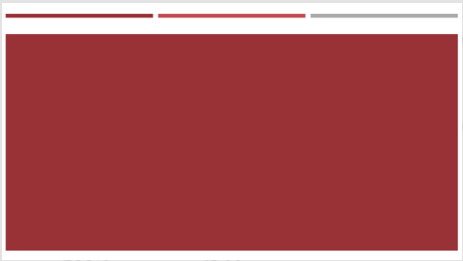
Distancia con el mismo punto



La distancia con el mismo punto es cero

$$d(a, a) = 0$$

Distancia entre dos puntos



Para que la distancia entre dos puntos sea cero, estos han de ser el mismo[1], es decir:

$$d(a, b) = 0 \Rightarrow a = b$$

[1]En una aplicación real, como la segmentación de clientes, esta última propiedad no indica que dos registros diferentes no puedan tener una distancia cero, sino que estos serían indistinguibles únicamente con las características utilizadas en el análisis de clúster. Por ejemplo: edad, antigüedad, número de hijos, consumo, etc.

Distancia euclídea

En general, la distancia euclídea entre dos puntos se define como:

$$d(a, b) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$$

Distancia euclídea normalizada

Aunque esta distancia es útil para medir separación en el mundo físico, muestra algunas desventajas cuando intentamos medir las distancias en un espacio de características, como es su dependencia con las unidades de las coordenadas. En un espacio físico, todas las coordenadas se miden en las mismas unidades (centímetros, pulgadas, etc.), pero esto no es así en un espacio de características, como puede ser la edad (años), ingresos (€, $, etc.) o número de hijos. Para solucionar este problema, se suele utilizar la distancia euclídea normalizada, que se define como:

$$d(a,b) = \sqrt{\sum_{i=1}^{n}(\frac{a_i}{\sigma_i} - \frac{b_i}{\sigma_i})} = \sqrt{\sum_{i=1}^{n}(\frac{(a_i - b_i)^2}{\sigma_i^2}}$$

σ_i^2 es la varianza de la coordenada i. El uso de esta distancia ofrece dos ventajas importantes frente a la euclídea. La primera es la independencia de los resultados de las unidades utilizadas en cada una de las coordenadas; la segunda es la de ajustar el peso de las coordenadas en función de su varianza, lo que hace que no pese igual una separación de un euro o dólar en el sueldo de un cliente que un hijo en la segmentación de clientes, como sucede en la euclídea.

A la hora de crear modelos utilizando la distancia euclídea, se ha de tener en cuenta que la varianza para cada una de las características en el conjunto de entrenamiento no tiene por qué ser igual a la del conjunto durante la evaluación, incluso puede ser que en producción solamente se evalúe un único dato cada vez. Para resolver este problema se suele calcular la varianza de cada una de las características y se normalizan para utilizar la distancia euclídea sobre estas.

Distancia Manhattan

Otra distancia de interés en algunos problemas es la Manhattan o geometría del taxista. El nombre alude a que el diseño cuadriculado de las calles de la isla de Manhattan hace que el camino más corto posible en taxi sea exactamente esta distancia. Matemáticamente, se define como:

$$d(a,b) = \sum_{i=1}^{n} |a_i - b_i|$$

La interpretación de esta distancia se puede ver en la figura 3.7., donde se muestra la distancia entre dos puntos. En esta figura la línea negra representa la distancia euclídea, mientras que la distancia Manhattan viene representada por los caminos verde y rojo. Es fácil ver que en estas dos últimas rutas se recorre la misma distancia.

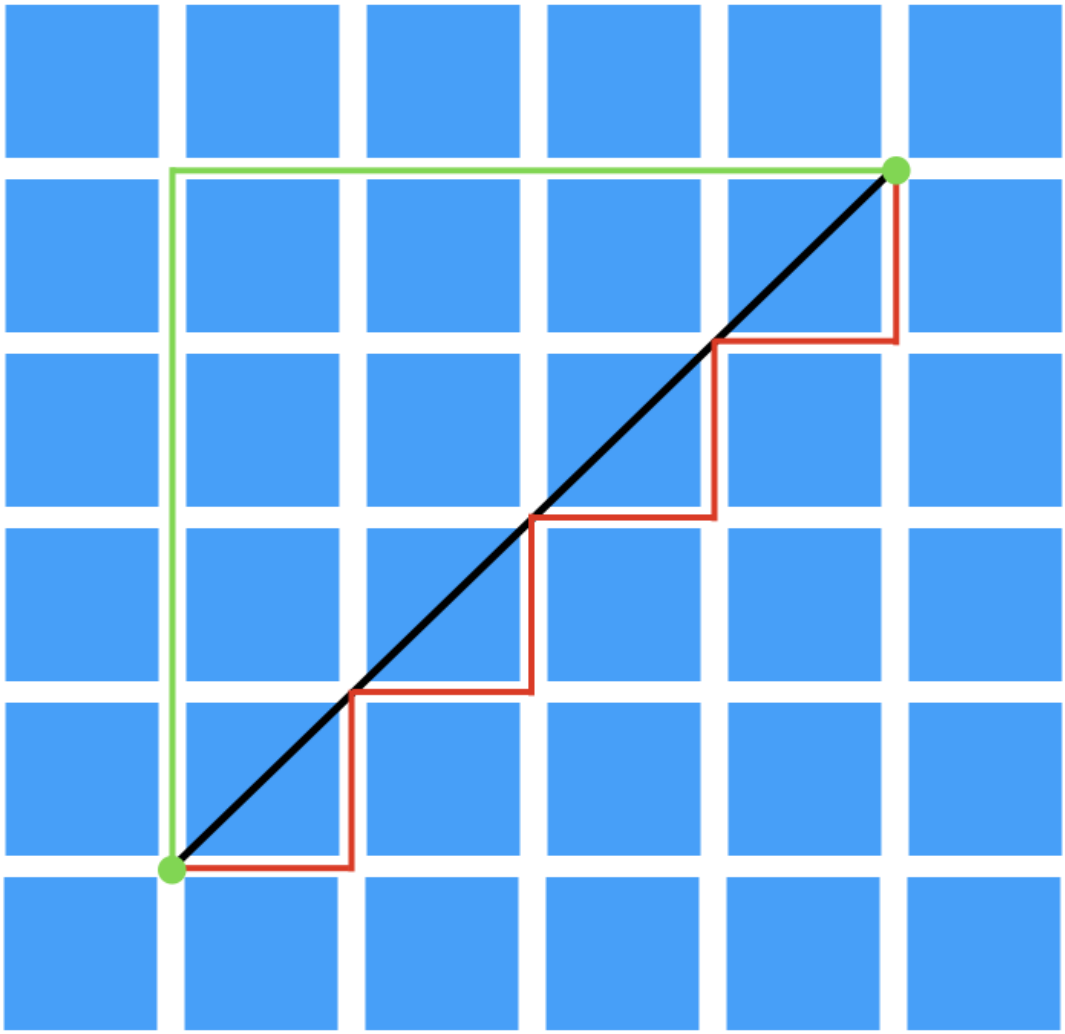


Figura 3.7. Representación de la distancia Manhattan. *Fuente:* elaboración propia.

Al igual que la distancia euclídea, la Manhattan también puede ser normalizada o permite utilizar características normalizadas para evitar los efectos de las dimensiones.

Anotación: posibles distancias utilizadas en problemas de clúster

En la tabla 3.1. se muestra un resumen de algunas distancias que se utilizan en problemas de clúster. En esta tabla **a·b** representa el producto a escalar entre los vectores a y b$|a|$ indica el modulo y$\bar{a}$ es la media.

Euclídea	$\boldsymbol{d(a,b)} = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$
Manhattan	$d(a,b) = \sum_{i=1}^{n}\lvert a_i - b_i\rvert$
Minkowsky	$d(a,b) = \sqrt[p]{\sum_{i=1}^{n}\lvert a_i - b_i\rvert^p}$
Coseno	$d(a,b) = 1 - \frac{a \cdot b}{\lvert a\rvert\lvert b\rvert}$
Correlación	$d(a,b) = 1 - \frac{(a-\bar{a})\cdot(b-\bar{b})}{\lvert(a-\bar{a})\rvert\lvert(b-\bar{b})\rvert}$

Tabla 3.1. Resumen de posibles distancias utilizadas en problemas de clúster.

Obtención de la distancia entre dos puntos en Python.

En Python se puede utilizar la función cdist, que se encuentra disponible en scipy.spatial.distance, para calcular la distancia existente entre dos vectores. El funcionamiento básico de la función es: cdist(A, B, metric)

A y B son dos vectores con los puntos entre los que se desea calcular la distancia y metric es el nombre de la distancia utilizada, en el caso de que no se indique el valor, por defecto, se utiliza la euclídea. Los nombres para las distancias que se han visto en la tabla 3.1. son:

- Euclídea: 'euclidean'
- Minkowsky: 'minkowski'
- Manhattan: 'cityblock'
- Coseno: 'cosine'
- Correlación: 'correlation'
- Euclídea normalizada: 'seuclidean'

Adicionalmente, para algunas distancias, puede ser necesario indicar otros parámetros opcionales:

- **p:** la norma en la métrica de Minkowsky, por defecto, utiliza 2, debido a lo cual se obtendrá el mismo valor que en la métrica euclídea. En el caso de que p sea 1 se obtendrá la distancia Manhattan.

- **V:** un vector con la varianza utilizada en la métrica euclídea normalizada, por defecto, calculará la varianza del conjunto de datos. En aplicaciones prácticas, la varianza de los conjuntos de datos de entrenamiento no tiene por qué coincidir con los de producción, por lo que es necesario indicar este valor.

Por ejemplo, se pueden comprobar los resultados que se obtienen entre el punto [0, 1] y [1, 2] con las diferentes distancias:

- Euclídea: 1,41421356
- Minkowsky (p = 3): 1.25992105
- Manhattan: 2
- Coseno: 0,10557281
- Correlación: 1e-16
- Euclídea normalizada (V = [0.25, 0,5]): 2,44948974

En la figura 3.8. se muestra el código utilizado para obtener estos valores.

```
from scipy.spatial.distance import cdist

print cdist([[0, 1]], [[1, 2]], 'euclidean')
print cdist([[0, 1]], [[1, 2]], 'minkowski', p = 3)
print cdist([[0, 1]], [[1, 2]], 'cityblock')
print cdist([[0, 1]], [[1, 2]], 'cosine')
print cdist([[0, 1]], [[1, 2]], 'correlation')
print cdist([[0, 1]], [[1, 2]], 'seuclidean', V =[0.25, 0.5])
```

Figura 3.8. Obtención de la distancia entre dos puntos en Python.

4.2. Algoritmo de k-means

El algoritmo de k-means es una de las técnicas de análisis de clúster más utilizadas debido a su sencillez. El algoritmo solamente necesita un conjunto de datos para ser entrenado y conocer el número de clústeres en los que se va a dividir. Los pasos en los que se basa este algoritmo son:

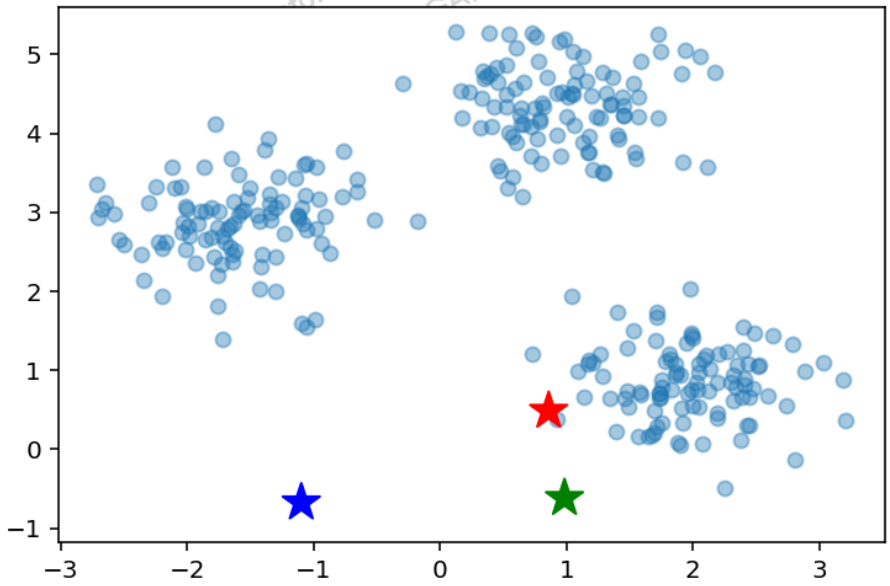
1. En el espacio n-dimensional se han de obtener tantas coordenadas, a las que se denominan centroides, como clústeres se van a crear. Se pueden seleccionar de forma aleatoria o mediante alguno de los algoritmos existentes.
2. Calcular un valor de distancia de cada uno de los puntos del conjunto de datos a todos los centroides. Cada punto se asignará al clúster del centroide que tenga más cerca.
3. Una vez asignados todos los puntos a un clúster, se calcula la posición media de sus componentes y esta pasa a ser el nuevo centroide.
4. Se ha de comprobar si la posición de los nuevos centroides se ha desplazado respecto a los anteriores menos de una cantidad prefijada. En caso afirmativo, se finaliza el proceso y, en caso negativo, se vuelve al punto 2.

La forma en la que trabaja el algoritmo se muestra desde la figura 3.9. a la figura 3.12.

Ejemplo

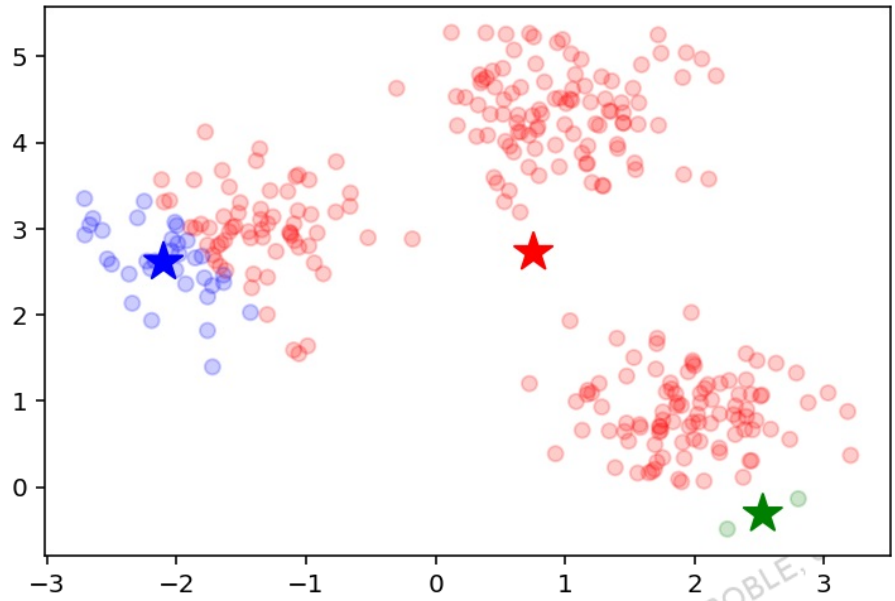
En este ejemplo se inicia el proceso con 100 puntos calculados aleatoriamente, los cuales se muestran como círculos azules en la figura 3.9. Se obtienen aleatoriamente 3 centroides, los cuales están representados como estrellas de color rojo, azul y verde. Como se inicia el proceso con 3 centroides se van a obtener tres clústeres. A continuación, se procede a asignar cada uno de los puntos a un clúster. El resultado se puede ver en la figura 3.10., donde se ha asignado un color a cada uno de los puntos, lo que permite ver que la separación actual no es óptima, ya que se pueden apreciar puntos del mismo color unos cerca de otros. Esto se corrige al actualizar el valor de los centroides con la media de cada uno de los clústeres actuales, como se observa en la figura 3.11. En este punto los centroides se encuentran en una situación cercana a la óptima, que se resuelve con una última iteración que se muestra en la figura 3.12.

Figura 3.9. Algoritmo de k-means. Paso 1.



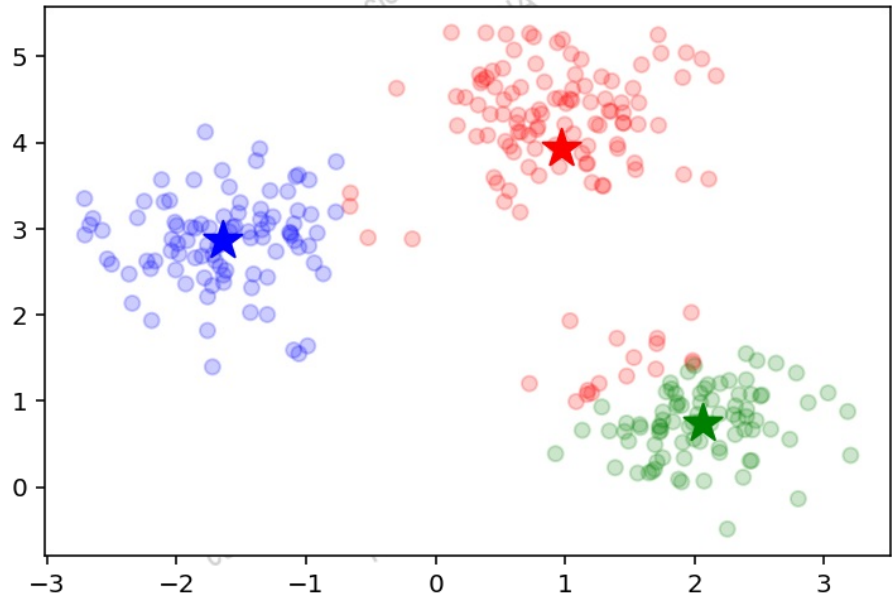
Fuente: elaboración propia.

Figura 3.10. Algoritmo de k-means. Paso 2.



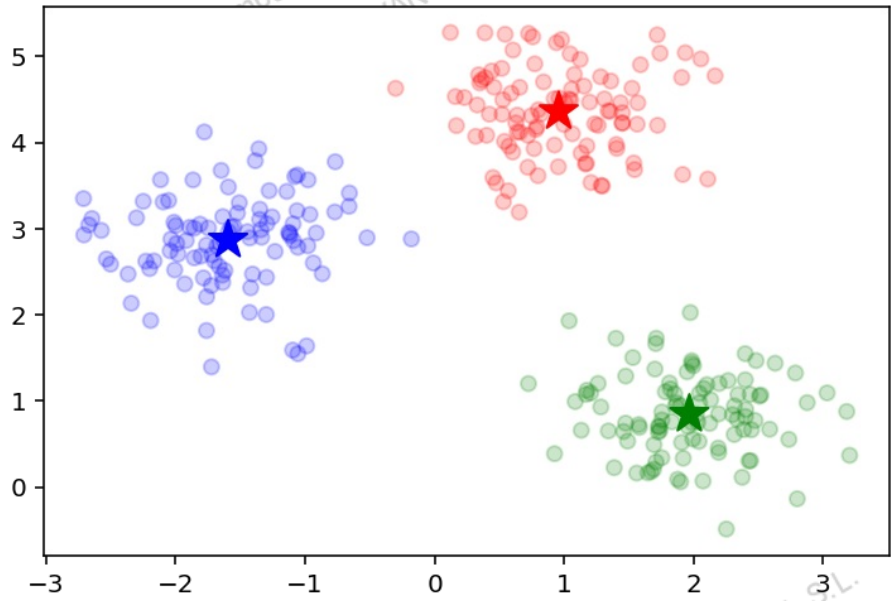
Fuente: elaboración propia.

Figura 3.11. Algoritmo de k-means. Paso 3.



Fuente: elaboración propia.

Figura 3.12. Algoritmo de k-means. Paso 4.



Fuente: elaboración propia.

En scikit-learn existen dos implementaciones del algoritmo, ambas disponibles en sklearn.cluster, que se diferencian en la cantidad de datos empleados durante el entrenamiento:

- **KMeans:** es la implementación estándar en la que se utilizan todos los datos.
- **MiniBatchKMeans:** utiliza un subconjunto de los datos para reducir el tiempo de cálculo, siendo adecuado para grandes muestras de datos.

Las principales opciones a la hora de crear un objeto con ambos constructores son:

- **n_clusters:** es el número de clústeres en los que se divide. En caso de no indicar el valor, usará 8.
- **n_init:** es el número de veces que se ejecutará el algoritmo de k-means, por defecto se ejecuta 10 veces.
- **random_state:** al igual que en otros métodos, es la semilla para el generador de números aleatorios.
- **init:** es el método para calcular los puntos de inicio, por defecto se utiliza el 'k-means++'.

Actualmente ni en KMeans ni en MiniBatchKMeans se puede fijar la métrica utilizada para calcular las distancias, por lo que siempre se utilizará la métrica euclídea.

1

Para utilizar los algoritmos de k-means es necesario disponer de un conjunto de datos de prueba, en el scikit-learn existen múltiples funciones con las que se pueden crear. En esta ocasión se utilizará la función make_blobs, que se encuentra en sklearn.datasets. Esta función genera aleatoriamente datos para un número de clases indicadas en torno a unos centroides. En la figura 3.13. se muestra un ejemplo de su utilización. En él, se crean dos conjuntos de 300 datos con 3 y 5 clases y una desviación estándar en los mismos de 0,5, la semilla es fijada en 1 para garantizar la reproducibilidad de los resultados. Esta función devuelve, como primera variable, el conjunto de datos y, como segunda, un vector con el identificador del clúster al que pertenece cada punto.

```
from sklearn.datasets import make_blobs

blobs_3, classes_3 = make_blobs(300,
                                centers      = 3,
                                cluster_std  = 0.5,
                                random_state = 1)
blobs_5, classes_5 = make_blobs(300,
                                centers      = 5,
                                cluster_std  = 0.5,
                                random_state = 1)
```

Figura 3.13. Ejemplo de datos para problemas de clúster.

2

A partir de estos datos se puede utilizar el código de la figura 3.14. para segmentar el conjunto de 3 clases en 3 clústeres. En este listado, lo primero que se hace es importar la clase KMeans. Para facilitar la presentación de los resultados se crea un vector con ocho colores que serán utilizados más adelante. Posteriormente, se crea un objeto kmeans al que se le indica que tendrá tres clústeres, se fija la semilla y se entrena con el conjunto de datos generado previamente. La clase de cada uno de los registros se obtiene utilizando el método predict del objeto entrenado en el conjunto de datos. Finalmente, se representa la muestra de datos estableciendo un color en función del clúster asignado en la predicción. Los resultados se muestran en la figura 3.15., donde se aprecia una separación adecuada de los datos.

```
from sklearn.cluster import KMeans

color_map = array(['b','g','r','c','m','y','k'])

kmeans = KMeans(n_clusters = 3,
                random_state = 1).fit(blobs_3)
classes = kmeans.predict(blobs_3)

scatter(blobs_3[:, 0], blobs_3[:, 1],
        color = color_map[classes])
scatter(kmeans.cluster_centers_[:, 0],
        kmeans.cluster_centers_[:, 1],
        marker = '*',
        s = 250,
        color = 'black')
```

Figura 3.14. Implementación de k-means en scikit-learn.

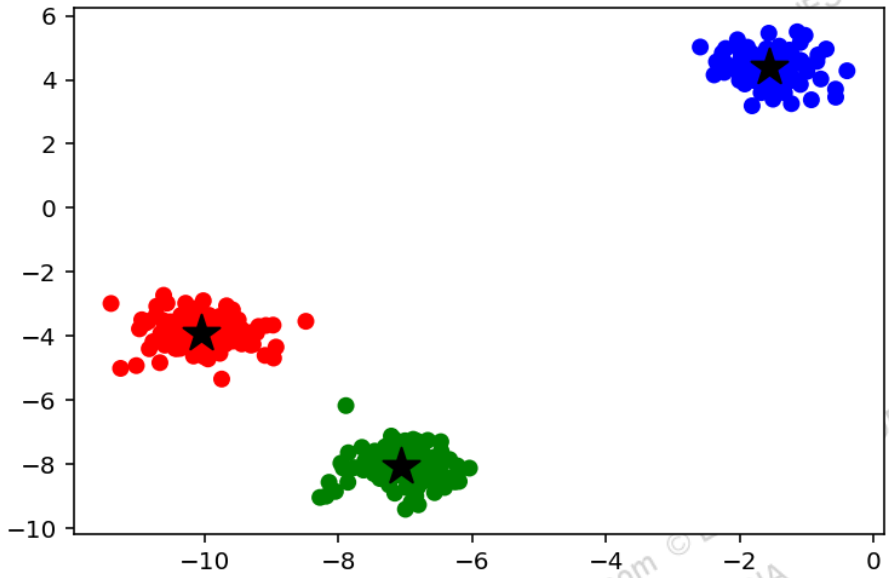


Figura 3.15. Resultados de k-means para 3 clústeres sobre el conjunto de datos de muestra blobs_3.*Fuente:* elaboración propia.

3

La función make_blobs devuelve una etiqueta numérica para cada uno de los registros. En este caso no tiene sentido comparar los valores con los que devuelve la función predict, ya que estos son un nombre y cambian en cada ejecución.

El constructor MiniBatchKMeans tiene sentido cuando el conjunto de datos de entrenamiento contiene cientos de miles de registros, en caso de que fuera necesario, el objeto creado se utilizaría como el creado con KMeans.

k-means solamente se puede usar con variables numéricas, en caso de que las variables disponibles sean categóricas se ha de utilizar el algoritmo de k-modes. Actualmente no existe una implementación de k-modes en scikit-learn, pero sí existe un paquete que lo implementa.

Página web de Python. *kmodes 0.7: Python implementations of the k-modes and k-prototypes clustering algorithms*. (https://pypi.python.org/pypi/kmodes/)

La importancia de seleccionar el número de clústeres correcto se puede ver si se ejecuta el código de la figura 3.14. sobre el conjunto de datos blobs_5 sin cambiar el número de clústeres. Los resultados que se obtienen en este caso se muestran en la figura 3.16. En ella se puede apreciar cómo el algoritmo ha asignado varios grupos, que deberían estar separados, en el mismo, lo que ha sido generado por indicarle que solamente existen tres grupos en el conjunto de datos.

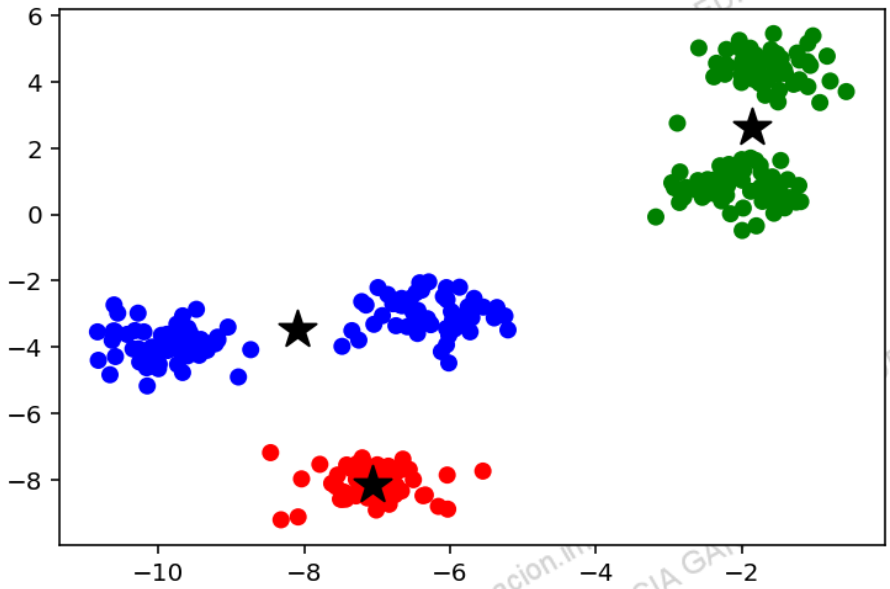


Figura 3.16. Resultados de KMeans para tres clústeres sobre el conjunto de datos de muestra blobs_5. *Fuente:* elaboración propia.

```
4
```

Otro posible error es hacer lo contrario: indicarle a k-means que busque más grupos de los que hay. En estos casos, lo que hace el algoritmo es dividir los grupos en subgrupos. Por ejemplo, cambiando n_clusters = 5 en el código de la figura 3.14. se obtendrían los resultados de la figura 3.17.

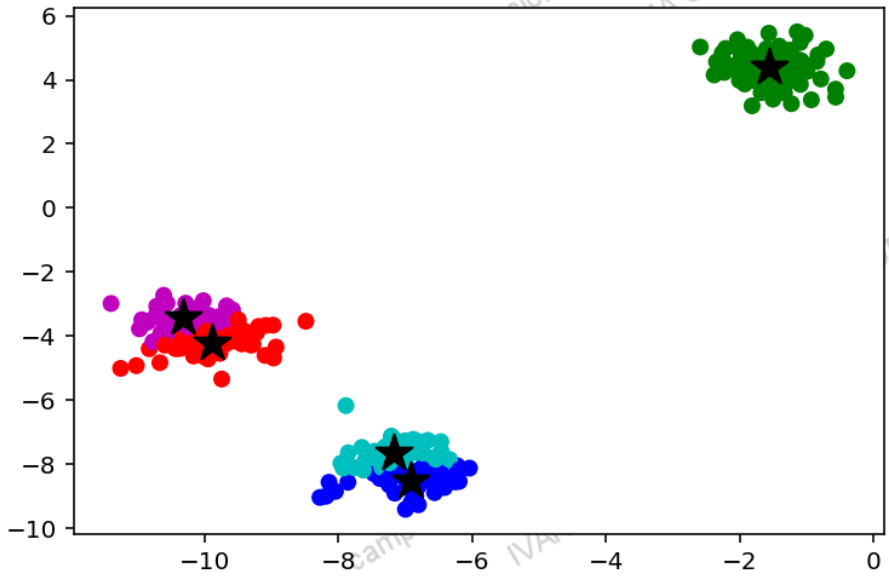


Figura 3.17. Resultados de KMeans para cinco clústeres sobre el conjunto de datos de muestra blobs_3.
Fuente: elaboración propia.

4.3. Identificación del número de clústeres en un conjunto de datos

Los resultados obtenidos en la sección anterior muestran la necesidad de seleccionar correctamente el número de clústeres. Cuando los problemas se encuentran en espacios bidimensionales o tridimensionales se puede realizar visualmente representando los datos, pero en problemas en más dimensiones es necesario utilizar otras técnicas, como son el método de la dispersión o el de la Silhouette.

1

Uno de los métodos más fáciles de implementar y de interpretar para la selección del número de clústeres es el de la distorsión o método del codo. En este se ha de calcular la distorsión promedia de los clústeres, que es la distancia promedia del centroide a todos los puntos del clúster y se obtiene con el algoritmo de k-means en función del número de clústeres. En esta situación, cuando se va de una situación como la que se muestra en la figura 3.16. —con un número inferior de clústeres al correcto—, a otra como la de la figura 3.15. —con el número adecuado—, el valor de la dispersión se reducirá. Por otro lado, cuando aumenta el número de clústeres por encima de este valor —como se muestra en la figura 3.17.—, el valor promedio de la dispersión se reducirá más lentamente, formándose un codo en la gráfica. Para la creación de estos gráficos se puede utilizar la función que se define en la figura 3.18. Esta función crea un objeto k_means y lo entrena con el conjunto de datos desde 1 clúster hasta un máximo determinado por max_k. En cada una de las iteraciones se obtiene el valor de la inercia del objeto y finalmente presenta los resultados en una gráfica.

```
def plot_dispersion(x, figure_name, max_k = 10, n_init = 10):
    inertia = []

    for k in range(1, max_k):
        kmeans = KMeans(n_clusters = k, n_init = n_init).fit(x)
        inertia.append(kmeans.inertia_)

    plot(range(1, max_k), inertia, 'bx-')
    xlabel('k')
    ylabel(u'Dispersión')
    title(figure_name)
```

Figura 3.18. Función para representar la gráfica de la dispersión en función del número de clústeres.

2

El resultado de ejecutar esta función en los conjuntos de datos creados anteriormente se muestra en la figura 3.19., donde se aprecia que la distorsión se reduce bruscamente en 3 clústeres para el primer conjunto de datos y en 5 para el segundo, como era de esperar, debido a la construcción de los mismos.

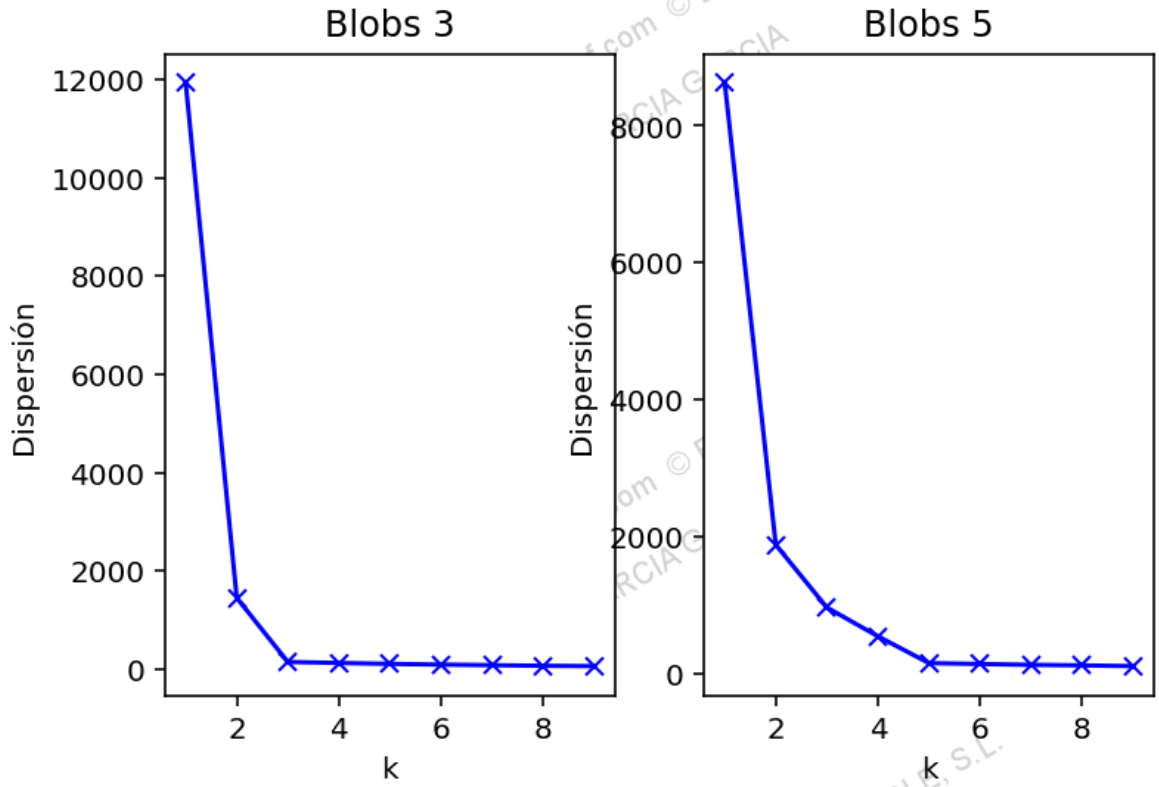


Figura 3.19. Gráfica de la distorsión obtenida mediante la función de la figura 3.18.*Fuente:* elaboración propia.

3

En algunas ocasiones, el método de la distorsión no ofrece resultados porque los clústeres no se encuentran tan claramente separados como en el ejemplo. En estos casos se puede utilizar el método de la Silhouette. El coeficiente de la Silhouette se define como la diferencia entre la distancia media a los elementos del clúster más cercano (b) y a distancia intra-clúster media de los elementos de un clúster (a) dividido por el máximo de los dos, es decir:

$$s = \frac{b - a}{max(a, b)}$$

Al dividir por el máximo los dos valores, la Silhouette solamente puede tomar valores entre -1 y 1. El concepto de Silhouette se muestra en la figura 3.20., donde se representa la diferencia entre la distancie intra-clúster (a) y la distancia al clúster más cercano (b).

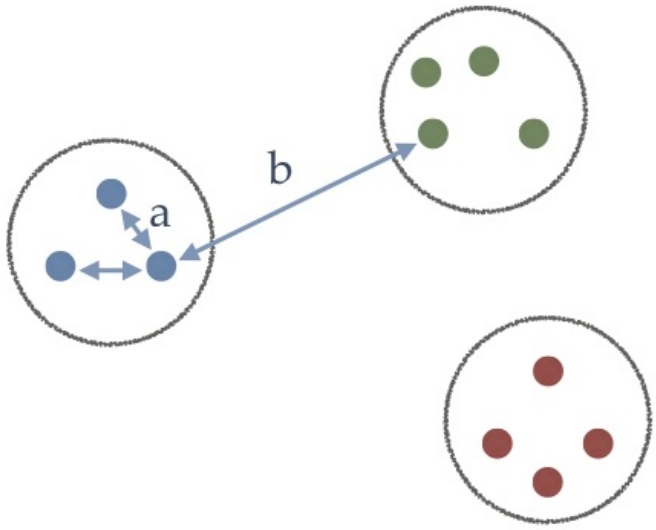


Figura 3.20. Concepto de Silhoette. *Fuente:* elaboración propia.

4

En el momento que se alcance el número de clústeres óptimos para un conjunto de datos, la Silhouette, en esta situación, se maximiza. Para obtener este valor se implementa una función, como en el caso de la distorsión, para representar la Silhouette según el número de clústeres. En la figura 3.21. se muestra un ejemplo de una función que realiza estas gráficas. En esta figura, lo primero que se hace es importar la métrica silhouette_score, que implementa el cálculo de esta métrica. Posteriormente, de forma similar al caso de la distorsión, se itera desde 2 clústeres hasta una cantidad determinada. Es importante destacar que en este caso no se puede operar desde un clúster, ya que en esta situación no se puede calcular el valor de la distancia entre clústeres y, por tanto, el de la Silhouette.

```
from sklearn.metrics import silhouette_score

def plot_sillhouette(blobs, figure_name, max_k = 10, n_init = 10):
    sillhouette_avgs = []

    for k in range(2, max_k):
        kmean = KMeans(n_clusters = k, n_init = n_init).fit(blobs)
        sillhouette_avgs.append(silhouette_score(blobs, kmean.labels_))

    plot(range(2, max_k), sillhouette_avgs)
    title(figure_name)
```

Figura 3.21. Función para representar la gráfica Silhouette en función del número de clústeres.

```
5
```

El resultado de ejecutar esta función en los conjuntos de datos creados anteriormente se puede ver en la figura 3.22. En este caso, se observa que la Silhouette toma un valor máximo para 3 clústeres en el primer conjunto de datos y para 5 en el segundo, como se esperaba por la forma en la que fueron creados estos datos.

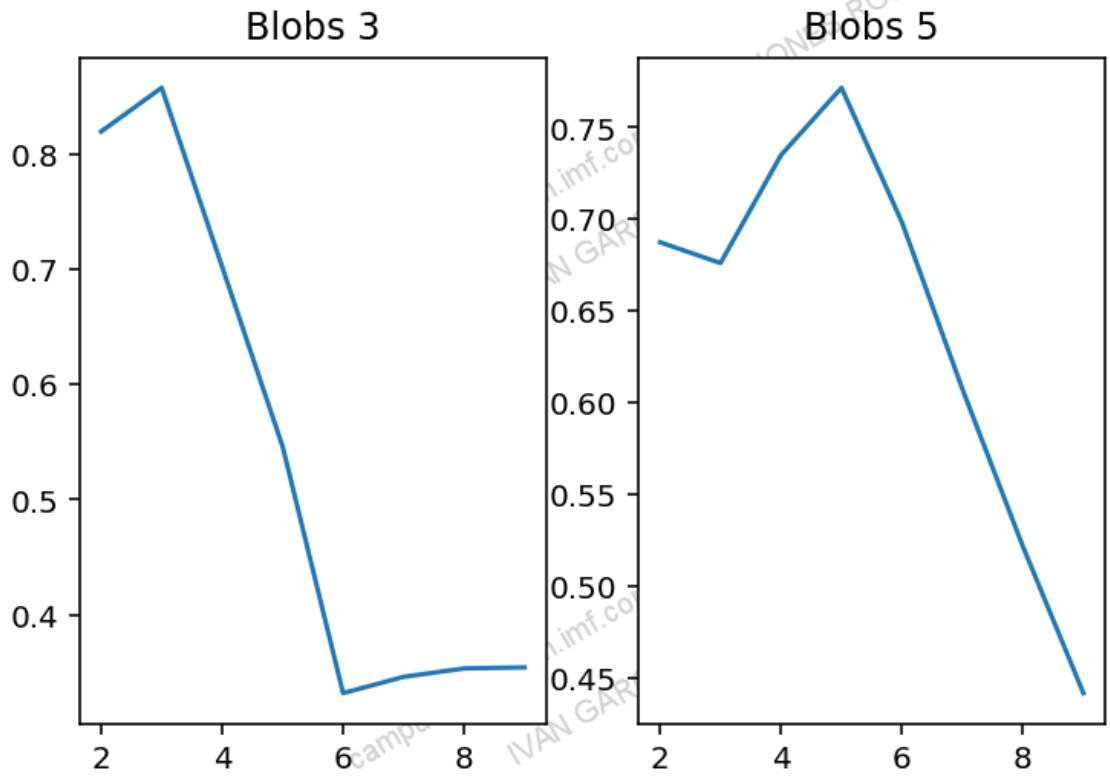


Figura 3.22. Gráfica de la Silhouette obtenida mediante la función de la figura 3.21.*Fuente:* elaboración propia.

V. Organización de clústeres como árbol jerárquico

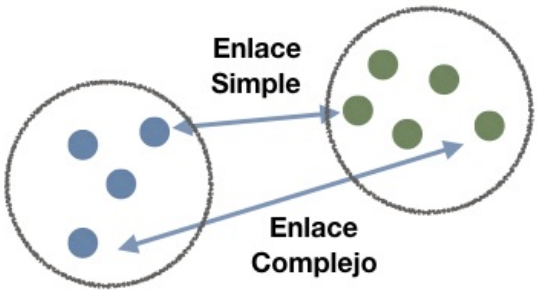
Entre las técnicas de análisis de clúster se encuentra la agrupación jerárquica (*hierarchical clustering*). Estos algoritmos muestran ventajas como la posibilidad de generar dendrogramas (visualizaciones de una agrupación jerárquica binaria) y, a diferencia de k-means, no es necesario conocer el número de clústeres para ejecutar el análisis. Dentro de la agrupación jerárquica existen dos enfoques:

La agrupación se realiza mediante un enfoque ascendente, cada observación comienza en su propio grupo y estos se agruparán mientras se asciende en la jerarquía.

La agrupación comienza con un único clúster que abarca todas las muestras disponibles, y se divide iterativamente en clústeres más pequeños hasta que cada uno de ellos solo contenga una muestra.

En el enfoque aglomerativo, los algoritmos más habituales para el análisis son el de enlace simple (*single linkage*) y el de enlace complejo (*complete linkage*). Para utilizar el de enlace simple se ha de calcular la distancia entre los miembros más próximos para cada par de clústeres y agrupar aquellos que muestren la menor distancia. Para el de enlace complejo se ha de calcular la distancia entre los miembros más distantes de cada par de clústeres. La diferencia entre estos dos algoritmos se comprende observando la figura 3.23.

Figura 3.23. Comparación entre el enlace simple y el enlace complejo.



Fuente: elaboración propia.

En esta sección únicamente se va a estudiar el enfoque aglomerativo de enlace complejo, siendo este un procedimiento iterativo que se puede resumir en los siguientes pasos:

1. Calcular la matriz de distancias entre todos los registros.
2. Representar cada punto como un clúster único.
3. Combinar los dos clústeres más cercanos en función de la distancia de los miembros más separados.
4. Actualizar la distancia entre todos los nuevos clústeres.
5. Repetir los pasos de 3 a 5 hasta que quede un único clúster.

1

En la implementación del algoritmo, el primer paso es la construcción de la matriz de distancias. Para facilitar la comprensión de los resultados, se utilizarán únicamente los cinco primeros registros del conjunto de datos blobs_3 utilizado previamente. Para el cálculo se utiliza el código de la figura 3.24.

```
import pandas as pd
from scipy.spatial.distance import pdist, squareform

df = pd.DataFrame(blobs_3[0:5, :])
row_dist = pd.DataFrame(squareform(pdist(df, metric = 'euclidean')))
row_dist
```

Figura 3.24. Construcción de la matriz de distancias.

2

En el código de la figura 3.24. se ha empleado la función pdist de la librería SciPy. Esta función es similar a cdist, pero, en lugar de calcular la distancia entre dos vectores, la calcula entre todos los puntos de un vector. Para presentar el resultado como una matriz se ha utilizado la función squareform. El resultado del código anterior se observa en la figura 3.25.

	0	1	2	3	4
0	0.000000	6.092759	0.833313	11.845115	12.017689
1	6.092759	0.000000	6.816534	13.609028	13.798075
2	0.833313	6.816534	0.000000	11.391524	11.559173
3	11.845115	13.609028	11.391524	0.000000	0.189181
4	12.017689	13.798075	11.559173	0.189181	0.000000

Figura 3.25. Matriz de distancias.

3

A continuación, se utiliza la función linkage de SciPy para calcular la agrupación jerárquica aglomerativa de enlace complejo. La forma de llamarla es mediante la matriz condesada (la salida de la función pdist) e indicando el método de uso. Esto se muestra en el código de la figura 3.26.

```
from scipy.cluster.hierarchy import linkage

row_clusters = linkage(pdist(df, metric = 'euclidean'),
                       method = 'complete')

pd.DataFrame(row_clusters,
             columns = ['row 1', 'row 2', 'distance', 'items in cluster'],
             index = ['cluster %d' %(i) for i in range(row_clusters.shape[0])])
```

Figura 3.26. Aplicación del método aglomerativo de enlace complejo.

4

En el código de la figura 3.26., los resultados de los enlaces se han presentado en una tabla en cuya primera y segunda columnas se muestran los miembros más separados de cada grupo, en la tercera, la distancia entre estos y en la última columna, el número de miembros de cada uno de los clústeres. Esta tabla se muestra en la figura 3.27.

	row 1	row 2	distance	items in cluster
cluster 0	3.0	4.0	0.189181	2.0
cluster 1	0.0	2.0	0.833313	2.0
cluster 2	1.0	6.0	6.816534	3.0
cluster 3	5.0	7.0	13.798075	5.0

Figura 3.27. Resultados del método aglomerativo de enlace complejo.

5

Una vez calculados los valores de los enlaces pueden ser representados en un dendrograma. Para esto solamente se ha de importar la función dendrogram, disponible en scipy.cluster.hierarchy, con lo que se obtiene la figura 3.28. En esta figura se aprecia la formación de los clústeres y que los registros más similares, al emplear la distancia euclídea, son el 3 y 4; posteriormente, los más cercanos son el 0 y 1.

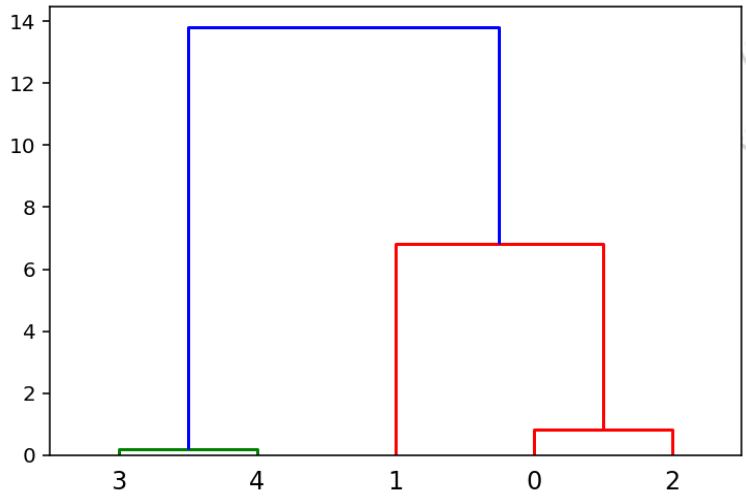


Figura 3.28. Dendrograma de los clústeres utilizados en el ejemplo. *Fuente:* elaboración propia.

6

Hasta ahora se ha estudiado la forma en la que funciona el algoritmo de agrupación aglomerativo de enlace complejo. No es habitual que esta familia de algoritmos se emplee de este modo, ya que se encuentra disponible en scikit-learn, concretamente a través del constructor AgglomerativeClustering. Una forma de repetir el análisis realizado previamente es mediante el código de la figura 3.29. En este código se crea un objeto AgglomerativeClustering para separar la muestra de datos en 2 clústeres, utilizando la distancia euclídea y enlace complejo. El objeto así creado se utiliza para segmentar la muestra de datos.

```
from sklearn.cluster import AgglomerativeClustering

ac = AgglomerativeClustering(n_clusters = 2,
                             affinity = 'euclidean',
                             linkage = 'complete')
ac.fit_predict(df)
```

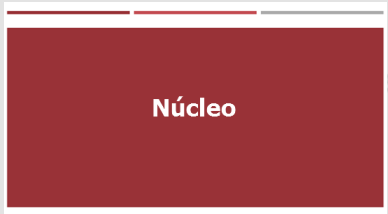
Figura 3.29. Utilización de AgglomerativeClustering en scikit-learn.

Una vez ejecutado el código la figura 3.29. se obtiene un vector en el cual los tres primeros registros pertenecen a un clúster y los 2 últimos a otro. Esto es justamente lo que se ha visto anteriormente en el dendrograma de la figura 3.28., ya que los registros 3 y 4 son los más próximos. A partir del dendrograma, se deduce que, en caso de volver a realizar el análisis buscando 3 clústeres, se obtendría uno con los registros 3 y 4, otro con el 0 y 2 y otro solamente con el registro 1.

VI. Localización de regiones a través de DBSCAN

Finalmente, se va a estudiar otro algoritmo existente dentro de la familia de análisis de clúster: el agrupamiento espacial basado en densidad de aplicaciones con ruido (DBSCAN, *Density-Based Spatial Clustering of Applications with Noise*). La densidad es el número de puntos que se encuentran en un radio ϵ del punto.

El primer paso para implementar este algoritmo es calificar todos los puntos con una de las siguientes tipologías: núcleo, alcanzable o ruido. La definición de cada tipo de punto es:



Son los puntos que contienen, por lo menos, un número mínimo de puntos en el entorno de radio ϵ.

Alcanzable

Son los puntos que no contienen el número mínimo de puntos en el entorno de radio ϵ, pero existe, por lo menos, un punto núcleo dentro de este.

Ruido

Son el resto de puntos del conjunto de datos.

Una vez realizada la clasificación, se puede aplicar el algoritmo DBSCAN realizando los siguientes pasos:

1. Alrededor de cada punto núcleo o conjunto concertado de puntos núcleo (estos son aquellos puntos núcleos que se encuentran más próximos que ϵ) se forma un grupo.
2. Se ha de asignar cada punto alcanzable a un clúster correspondiente.

Estos conceptos han sido representados en la figura 3.30. En esta se puede apreciar que los puntos tipo núcleo son de color azul, los verdes son los alcanzables y los rojos son el ruido. En esta figura también se observa que el número mínimo en el entorno de radio ϵ para que el punto sea nuclear es 4.

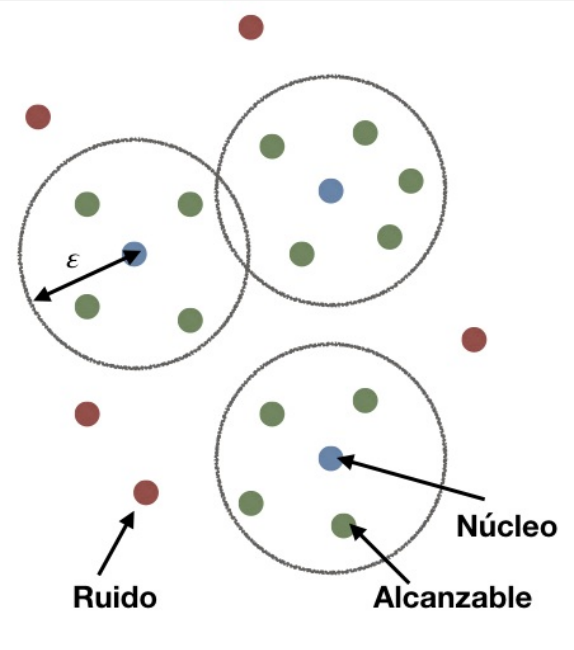


Figura 3.30. Esquema de los conceptos de DBSCAN. *Fuente:* elaboración propia.

1

Una de las principales ventajas de DBSCAN es que no asume que los clústeres tienen que ser necesariamente esféricos, como en el caso de k-means, por lo que se puede utilizar para identificar objetos con formas irregulares. Esto procede del hecho de que se pueden unir varios puntos núcleo en el mismo clúster, como se muestra en la figura 3.31. En esta figura se ven dos clústeres: uno con la frontera en gris, donde solamente hay un punto núcleo, y otro formado por varios puntos nucleares con la frontera dibujada en rojo.

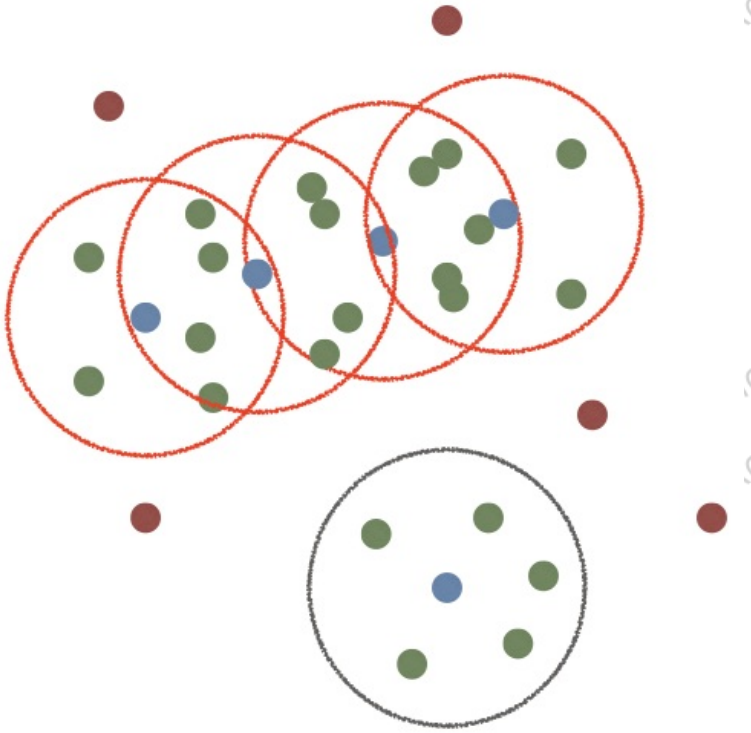


Figura 3.31. Ejemplo de un clúster con varios puntos núcleo de DBSCAN. *Fuente:* elaboración propia.

2

El constructor para la creación de modelos DBSCAN de scikit-learn se llama justamente DBSCAN y se encuentra disponible dentro de los modelos de clúster. Para la construcción de un modelo se ha de indicar el radio (eps), la cantidad mínima de muestras para considerar que un punto es el núcleo de un clúster (min_samples) y la métrica utilizada (metric). Una vez creado el objeto, las predicciones se realizan con el comando predict, como en los modelos estudiados anteriormente.

Ahora se puede comparar el funcionamiento de los tres algoritmos de análisis de clúster que se han visto en esta unidad. Para ello, en primer lugar, se compara el funcionamiento de los tres algoritmos con el conjunto de datos blobs_3 creado y utilizado previamente. En la figura 3.32. se muestra el código para realizar esto.

```
from sklearn.cluster import DBSCAN

km = KMeans(n_clusters = 3,
            random_state = 1).fit_predict(blobs_3)

ac = AgglomerativeClustering(n_clusters = 3,
                             affinity = 'euclidean',
                             linkage = 'complete').fit_predict(blobs_3)

db = DBSCAN(eps = 0.6,
            min_samples = 5,
            metric = 'euclidean').fit_predict(blobs_3)
```

Figura 3.32. Análisis de clúster del conjunto de datos blobs_3 mediante los algoritmos KMeans, AgglomerativeClustering y DBSCAN

3

Tal y como indica la figura 3.32., inicialmente se carga el constructor DBSCAN, ya que es el único que falta por importar. Luego se construye un objeto KMeans con modelos de tres clústeres y se fija la semilla a 0. Después se realiza una división en clústeres del conjunto de datos. Posteriormente, se realiza un análisis similar, pero con un objeto AgglomerativeClustering, en el que se han seleccionado tres clústeres, la distancia euclídea y enlace complejo. Finalmente, el análisis se realiza utilizado DBSCAN con una distancia de 0,6, una cantidad mínima de 5 elementos para considerar un clúster y utilizando la distancia euclídea. Los resultados obtenidos se pueden ver en la figura 3.33.

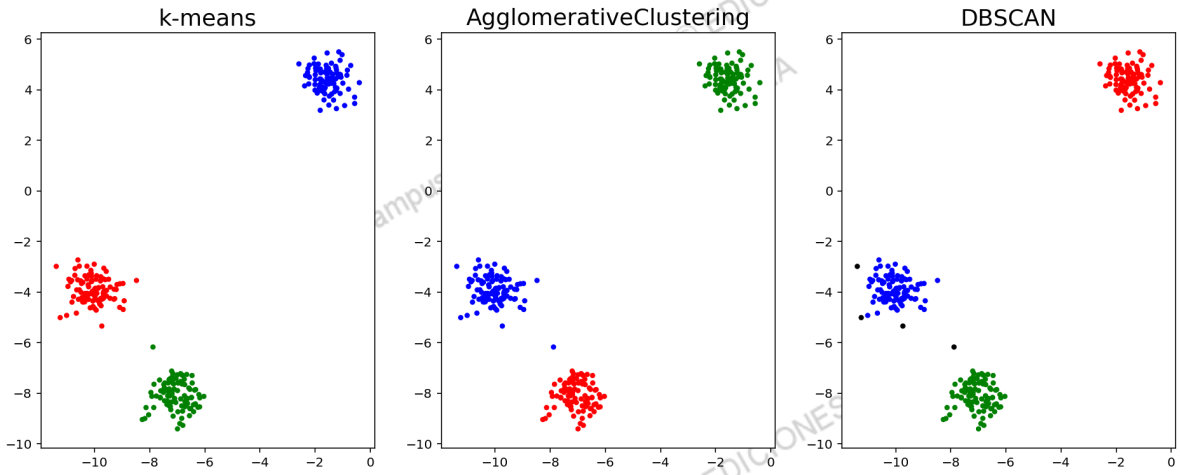


Figura 3.33. Representación gráfica de análisis de clúster del conjunto de datos blobs_3 mediante los algoritmos KMeans, AgglomerativeClustering y DBSCAN. *Fuente:* elaboración propia.

4

En la figura 3.33. se puede apreciar cómo los tres algoritmos han dividido el conjunto de datos en 3. En el caso de KMeans y AgglomerativeClustering este fue el valor que se indicó cuando se construyeron los modelos, mientras que en DBSCAN, el número de clúster se deduce del radio y del número mínimo de elementos. Aun así, existen diferencias en los resultados. En los dos primeros casos, todos los puntos se asocian a un clúster, mientras que en DBSCAN existen puntos, los identificados con el color negro, situados entre los dos clústeres de la parte inferior de la gráfica, que no han sido asociados a ningún clúster.

Para este tipo de problema, los tres algoritmos ofrecen resultados aceptables. Se probará a continuación su uso en otro tipo de problemas, como los objetos en forma de luna que se pueden construir con la función make_moons, disponible en sklearn.datasets. Esta función es la que se utiliza en el código de la figura 3.34. para crear el conjunto de datos que se analiza utilizando los tres algoritmos.

```
from sklearn.datasets import make_moons

moons, moons_classes = make_moons(n_samples = 200,
                                  noise = 0.05,
                                  random_state = 1)

km = KMeans(n_clusters = 2,
            random_state = 1).fit_predict(moons)

ac = AgglomerativeClustering(n_clusters = 2,
                             affinity = 'euclidean',
                             linkage = 'complete').fit_predict(moons)

db = DBSCAN(eps = 0.2,
            min_samples = 5,
            metric = 'euclidean').fit_predict(moons)
```

Figura 3.34. Análisis de clúster de un conjunto de datos generados con make_moons mediante los algoritmos KMeans, AgglomerativeClustering y DBSCAN. *Fuente:* elaboración propia.

5

Los resultados obtenidos mediante el análisis de la figura 3.34 se muestran en la figura 3.35. En esta última figura se puede observar que el método de k-means y el de agrupamiento jerárquico no son adecuados para identificar zonas de irregularidades, como las lunas, mientras que el método DBSCAN sí que es adecuado.

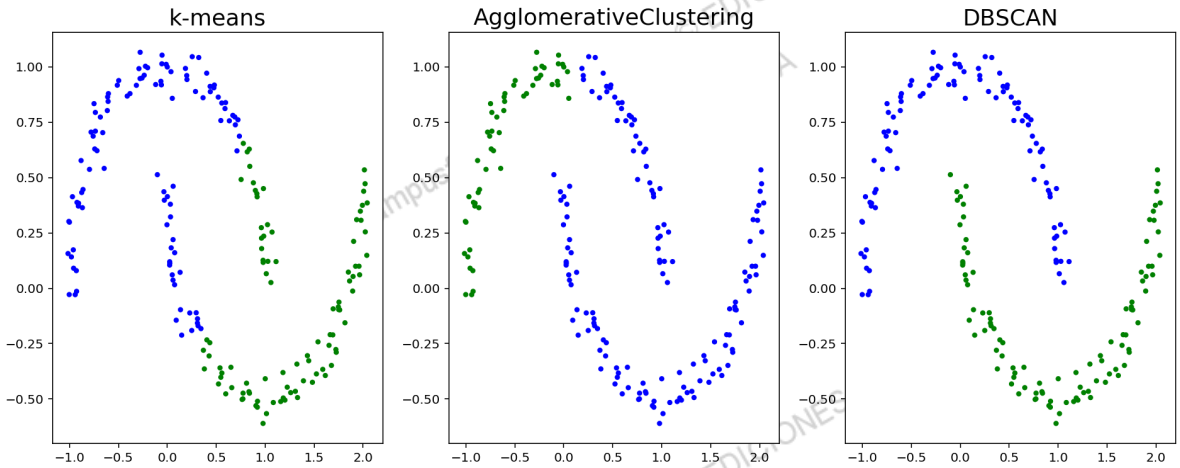


Figura 3.35. Representación gráfica de análisis de clúster del conjunto de datos generados con make_moons mediante los algoritmos KMeans, AgglomerativeClustering y DBSCAN. *Fuente:* elaboración propia.

6

Lo visto en esta sección no indica que el método DBSCAN sea el más adecuado para todo tipo de problemas de análisis de clúster. Por ejemplo, en la figura 3.36. se presenta una comparación basada en la existente en la web de scikit-learn, donde se cotejan los tres métodos analizados y se exponen los diferentes resultados que se obtienen con cada método.

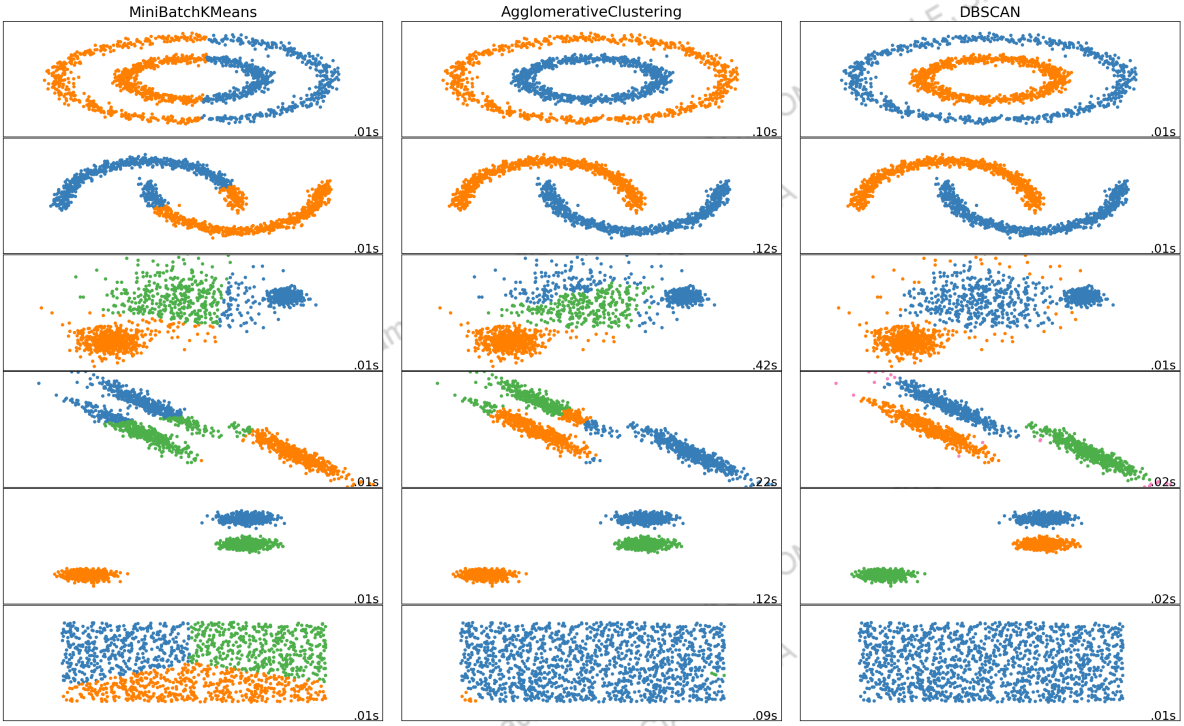


Figura 3.36. Comparación de los algoritmos utilizados en esta unidad. *Fuente:* http://scikit-learn.org/stable/modules/clustering.html#clustering.

VII. Resumen

En esta unidad, se han visto los siguientes métodos que permiten analizar datos en los que no se busca una solución conocida:

- Análisis de componentes principales.
- k-means.
- Agrupación jerárquica.
- DBSCAN.

El análisis de componentes principales permite reducir la dimensionalidad de los conjuntos de datos y comprimirlos con una pérdida de la información original conocida.

El resto de métodos permite la identificación de patrones desconocidos en los conjuntos de datos.

Ejercicios

Caso práctico

En el archivo “mammals.csv” se encuentra una lista de mamíferos y los constituyentes de su leche. A partir de esta información, segmenta los mamíferos sobre la base de los constituyentes de la leche y obtén los valores promedios de la leche para cada grupo de animales.

Solución

En este caso práctico, lo primero que se ha de realizar es importar el archivo con los datos, por lo que se han de utilizar pandas. Para importar los datos se puede utilizar el comando:

```
mammals = pd.read_csv('mammals.csv', sep = ',')
```

Al cargar el conjunto de datos se aprecia que existen seis variables: el nombre del animal, el agua, la proteína, la grasa, la lactosa y la ceniza. Solamente las cinco últimas variables son numéricas, por lo que son las únicas que se utilizarán para realizar la segmentación.

Para identificar el número de clústeres en los que se divide la leche de los mamíferos, se utilizara el método de la silueta. Para ello se puede emplear el código de la figura 3.21. Al ejecutar este aplicando los datos de la leche de los mamíferos se obtiene una gráfica como la que muestra la figura 3.37.

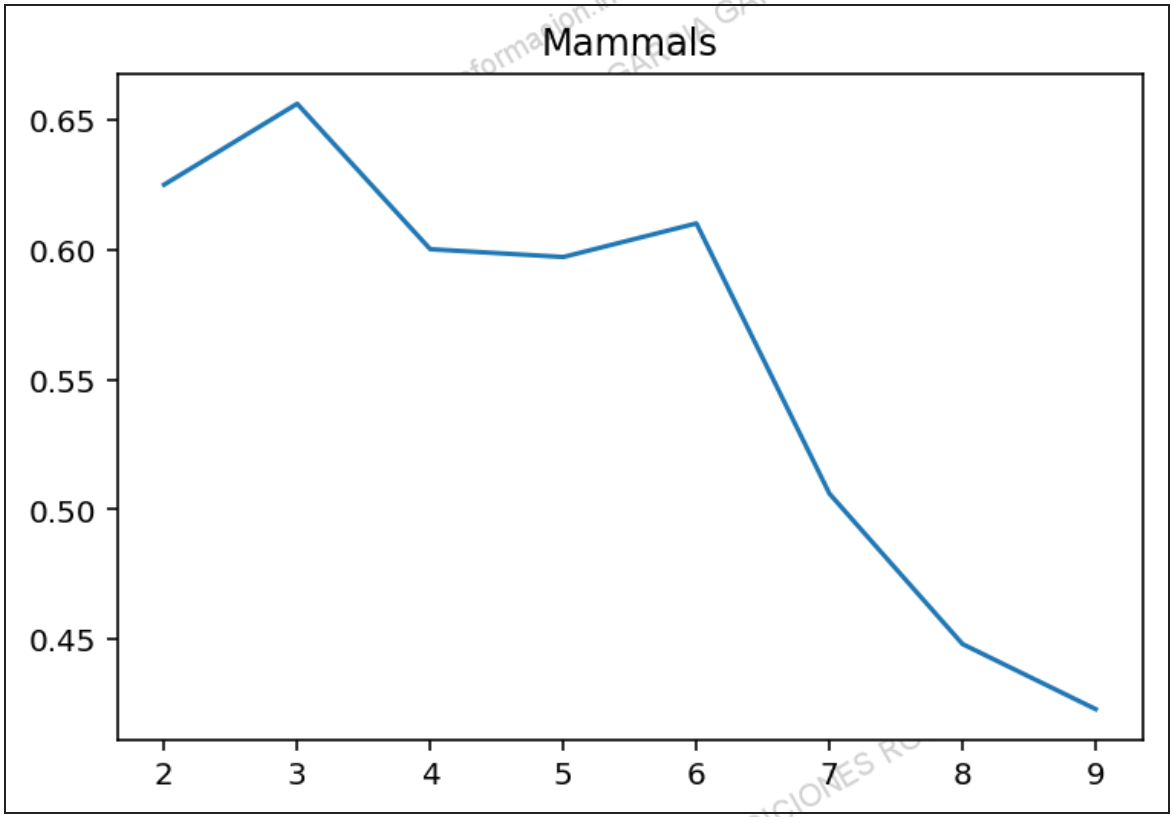


Figura 3.37. Resultado de aplicar el método de la silueta a los datos de la leche de los mamíferos.

Atendiendo a los resultados de la figura 3.37. se observa que el número de clústeres es 3. Con esta información ya se puede aplicar el método de k-means a los datos para obtener los centroides, lo que se puede realizar con el comando:

```
kmeans = KMeans(n_clusters = 3, n_init = 10).fit(mammals_data)
```

La posición de los centroides se puede localizar en la propiedadkmeans.cluster_centers_.

El código completo del caso se encuentra en elnotebook U4_Caso.

Recursos

Enlaces de Interés

Manual de scikit-learn. Clustering.
http://scikit-learn.org/stable/modules/clustering.html#clustering

kmodes. Python implementations of the k-modes and k-prototypes clustering algorithms.
https://pypi.python.org/pypi/kmodes/

Bibliografía

- ***Bootstrapping Machine Learning***. : Dorard L. [En línea] URL disponible en: http://www.louisdorard.com/machine-learning-book/
- ***Designing Machine Learning Systems with Python***. : Julian D. Birmingham: Packt Publishing; 2016.
- ***Mastering Machine Learning With scikit-learn***. : Hackeling G. Birmingham: Packt Publishing; 2014.
- ***scikit-learn Cookbook***. : Hauck T. Birmingham: Packt Publishing; 2014.

Glosario.

- **Dendrograma**: Visualización de una agrupación jerárquica binaria.
- **Distancia**: Función matemática que asocia un valor a un par de puntos.
- **Valor propio**: Valor escalar no nulo asociado a un vector propio.
- **Vector propio**: Vectores no nulos que al ser multiplicados por una matriz dan lugar a un múltiplo escalar de sí mismos.