Capítulo 8

Aprendizaje No Supervisado

Clustering o el análisis de clustering es un problema de aprendizaje no supervisado (ULA, por sus siglas en inglés). A menudo se usa como una técnica de análisis de datos para descubrir patrones interesantes en los datos, como grupos de clientes en función de su comportamiento.

8.1. Aprendizaje No Supervisado

El clustering (agrupamiento) consiste en dividir un conjunto de elementos heterogéneos en clústers o grupos homogéneos.

Se considera un paradigma de clasificación no supervisada, ya que asigna una clase a cada elemento (clúster al que pertenece), pero dichas clases no son conocidas durante el proceso de aprendizaje de el modelo.

- Tenemos que descubrir y, ver Figura 8.1.

8.1.1. Clustering

Clustering implica descubrir automáticamente la agrupación natural en los datos. A diferencia del aprendizaje supervisado (como el modelado predictivo), los algoritmos de clustering solo interpretan los datos de entrada y encuentran grupos o agrupaciones naturales en el espacio de características. Por tanto, las técnicas de clustering se aplican cuando no hay una clase para predecir, sino cuando las instancias se dividen en grupos naturales.

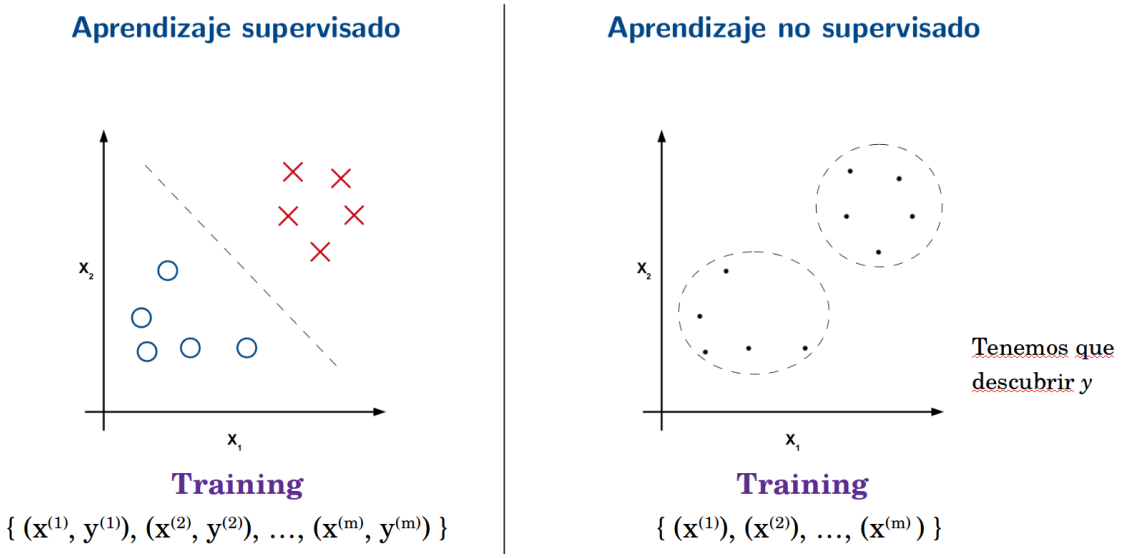


FIGURA 8.1: Aprendizaje Supervisado Vs. Aprendizaje No Supervisado.

Un grupo es a menudo un área de densidad en el espacio de características donde los ejemplos del dominio (observaciones o filas de datos) están más cerca del grupo que otros grupos. El grupo puede tener un centro (el centroide) que es una muestra o un espacio de entidades de puntos y puede tener un límite o extensión.

El objetivo del agrupamiento es encontrar una división de los datos en la que se dé:

- **Alta similaridad intra-cluster** (entre los elementos de un mismo clúster).
- **Baja similaridad inter-cluster** (entre los elementos de distinto clústers).

La similaridad se suele (veremos también que puede considerarse la conectividad) basar en una medida de distancia. Esta medida depende de la representación de los datos (discretos, numéricos, booleanos, etc). Además de la similaridad, se suelen usar medidas para medir la calidad del clúster, y que contemplan tanto la similaridad/similaridad intra/inter clúster, como el número de clústers.

8.1.2. Dominio del problema

La agrupación en clúster puede ser útil como actividad de análisis de datos para obtener más información sobre el dominio del problema, el llamado descubrimiento de patrones o descubrimiento de conocimiento. Por ejemplo:

- El árbol filogenético podría considerarse el resultado de un análisis de clustering manual.
- Separar datos normales de valores atípicos o anomalías puede considerarse un problema de clustering.
- La separación de los clústeres en función de su comportamiento natural es un problema de clustering, denominado segmentación del mercado.

La agrupación también puede ser útil como un tipo de ingeniería de características, donde los ejemplos existentes y nuevos se pueden mapear y etiquetar como pertenecientes a uno de los grupos identificados en los datos.

La evaluación de los grupos identificados es subjetiva y puede requerir un experto en el dominio, aunque existen muchas medidas cuantitativas específicas del grupo. Por lo general, los algoritmos de clustering se comparan académicamente en conjuntos de datos sintéticos con grupos predefinidos, que se espera que descubra un algoritmo.

8.1.3. Métodos de clustering

Existen una gran cantidad de algortimos de Aprendizaje No Supervisado que, según la naturaleza de nuestros datos, pueden acomodarse de mejor o peor problema en cuanto a la obtención de los resultados finales. Así, pueden diferenciarse las siguientes categorías como métodos de clustering:

- **Basados en centroides**: Construyen distintas particiones y las evalúan en función de algún criterio (generalmente basados en distancia). Ejemplos de algoritmos son: k-means, k-medoids, etc.

- **Jerárquicos**: Crean una descomposición jerárquica del conjunto de datos (objetos) usando algún criterio. Ejemplos de algoritmos son: Diana, Agnes, BIRCH, ROCK, etc.
- **Métodos basados en densidad**: Se basan en conectividad y en funciones de densidad de puntos en el espacio. Ejemplos de algoritmos son: DBSCAN, OPTICS, DenClue, etc.
- **Basados en modelos**: Se propone un modelo hipótesis para cada uno de los clústers y se trata de encontrar el mejor ajuste de ese modelo a otros. EM, COBWEB, etc.
- Otros métodos.

En la actualidad existen muchas aplicaciones en las que se pueden utilizar este tipo de algoritmos. Algunas de ellas son:

- Biología: Detección y agrupación de secuencias similares de genes.
- Marketing: descubrimiento de distintos grupos de clientes para personalización de ofertas, etc.
- Análisis de redes sociales: Detección de comunidades.
- Búsqueda de información: Agrupación de documentos o noticias similares.
- etc.

8.2. Algoritmos de aprendizaje no Supervisado

Hay muchos algoritmos de clustering para elegir y no existe el mejor algoritmo de clustering para todos los casos. En cambio, es una buena idea explorar una variedad de algoritmos de clustering y diferentes configuraciones para cada algoritmo.

- Clustering es un problema no supervisado que trata de encontrar grupos naturales en el espacio de características de los datos de entrada.
- Hay muchos algoritmos de clustering diferentes y no hay un método único mejor para todos los conjuntos de datos.
- Cómo implementar, ajustar y usar los mejores algoritmos de clustering en Python con la biblioteca de aprendizaje automático scikit-learn.

En este sentido nos encontramos con una gran cantidad de algoritmos que, dependiendo de la naturaleza de nuestros datos, cada uno converger de una manera diferente en la clusterización. Algunos de estos algoritmos pueden verse en la Figura 8.2.

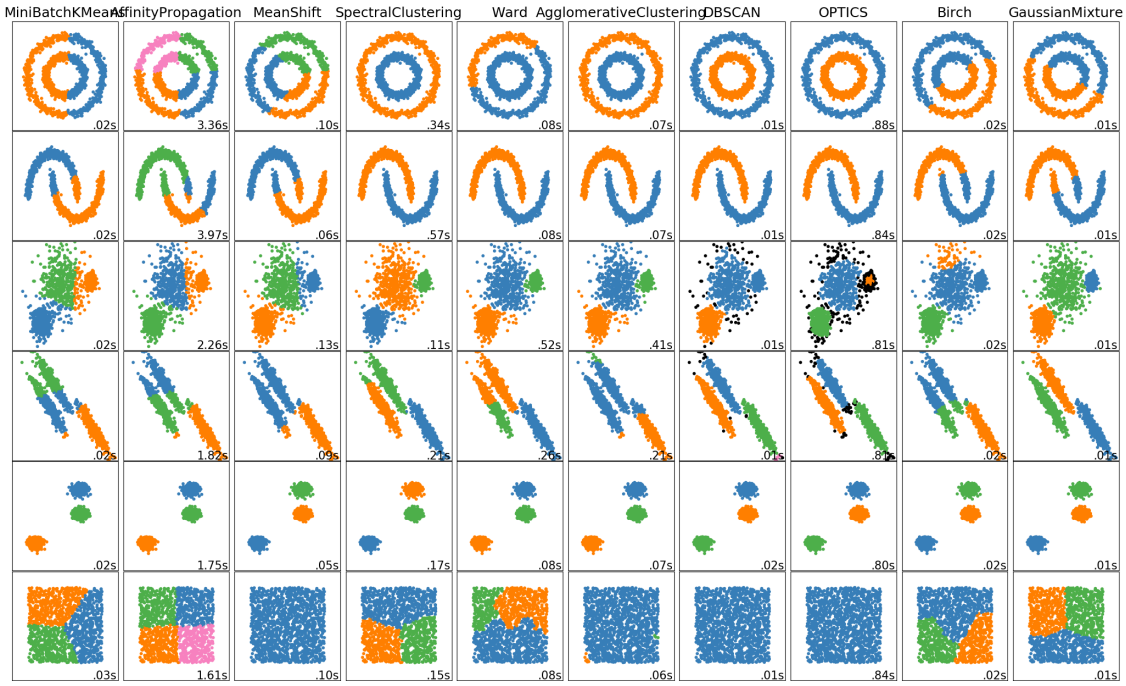


FIGURA 8.2: Diferentes algoritmos ULAs en Scikit Learn.

Ver Vídeo de este apartado

```
8.1. Aprendizaje No Supervisado.
```

Dataset

Utilizaremos la función `make_classification()`[1] para crear un conjunto de datos de clasificación binaria de prueba.

El conjunto de datos tendrá $1,000$ ejemplos, con dos características de entrada y un clúster por clase. Los grupos son visualmente obvios en dos dimensiones para que podamos trazar los datos con un diagrama de dispersión y colorear los puntos en el diagrama por el grupo asignado. Esto ayudará a ver, al menos en el problema de la prueba, qué tan bien se identificaron los grupos.

Los grupos en este problema de prueba se basan en un gaussiano multivariado, y no todos los algoritmos de agrupamiento serán efectivos para identificar estos tipos de grupos. Como tal, los resultados en este tutorial no deben usarse como base para comparar los métodos en general, ver Código 8.1.

```
> # Dataset
> from numpy import where
> from sklearn.datasets import make_classification
> from matplotlib import pyplot
> # Definicion del dataset
> X, y = make_classification(n_samples=1000, n_features=2,
        n_informative=2, n_redundant=0, n_clusters_per_class=1,
        random_state=4)
> # Crear un diagrama de dispersion con cada clase
> for class_value in range(2):
>     # obtener indices de fila para muestras con esta clase
>     row_ix = where(y == class_value)
>     # Crear la dispersion de esas muestras
>     pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
> pyplot.show()
```

CÓDIGO 8.1: Crear un dataset en dos dimensiones para clustering

[1] https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html

Ejecutar el ejemplo crea el conjunto de datos de agrupación sintética, luego crea un diagrama de dispersión de los datos de entrada con puntos coloreados por etiqueta de clase (agrupaciones idealizadas).

Podemos ver claramente dos grupos distintos de datos en dos dimensiones y la esperanza sería que un algoritmo de agrupación automática pueda detectar estas agrupaciones, ver Figura 8.3.

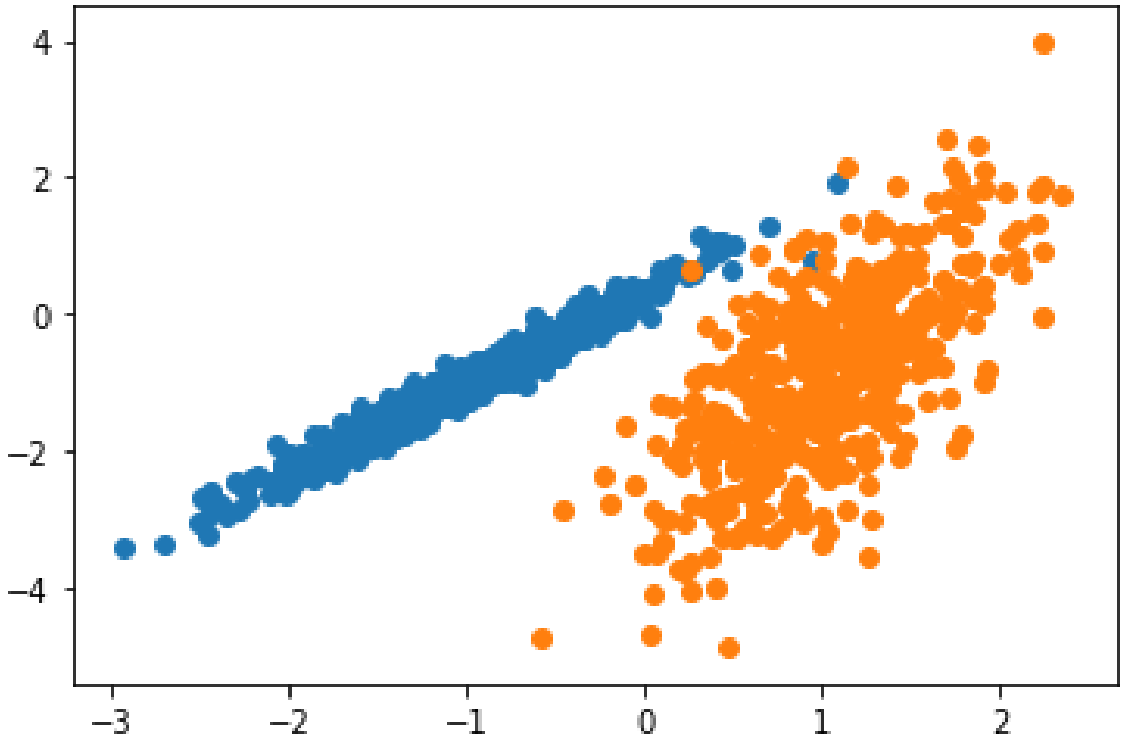


FIGURA 8.3: Dataset de dos dimensiones para clustering.

8.2.1. Affinity Propagation

Affinity Propagation implica encontrar un conjunto de ejemplos que mejor resuman los datos.

> *"Diseñamos un método llamado Affinity Propagation, que toma como entrada medidas de similitud entre pares de puntos de datos. Los mensajes de valor real se intercambian entre puntos de datos hasta que emerge gradualmente un conjunto de ejemplos de alta calidad y grupos correspondientes."*

– Clustering by Passing Messages Between Data Points[2], *2007.*

Se implementa a través de la clase `AffinityPropagation`[3] y la configuración principal para ajustar es el conjunto de `dumping` entre 0.5 y 1, y tal vez `preference`, ver Código 8.2.

```
> # Algoritmo clustering Affinity Propagation
> from numpy import unique
> from sklearn.cluster import AffinityPropagation
> # Definir el modelo
> model = AffinityPropagation(damping=0.9)
> # Ajustar el modelo
> model.fit(X)
> # Asiganar un cluster a cada ejemplo
> yhat = model.predict(X)
> # recuperar grupos unicos
> clusters = unique(yhat)
> # Crear el diagrama de dispersion por clusters
> for cluster in clusters:
>     row_ix = where(yhat == cluster)
>     pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
> pyplot.show()
```

CÓDIGO 8.2: Algoritmo Affinity Propagation

La ejecución del ejemplo se ajusta al modelo en el conjunto de datos de entrenamiento y predice un clúster para cada ejemplo en el conjunto de datos. Luego se crea un diagrama de dispersión con puntos coloreados por su grupo asignado. En este caso, podemos observar que este algoritmo no pude lograr un buen resultado, ver Figura 8.4.

8.2.2. Agglomeative

Agglomeative implica fusionar ejemplos hasta que se alcanza el número deseado de agrupaciones.

[2] https://science.sciencemag.org/content/315/5814/972

[3] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AffinityPropagation.html

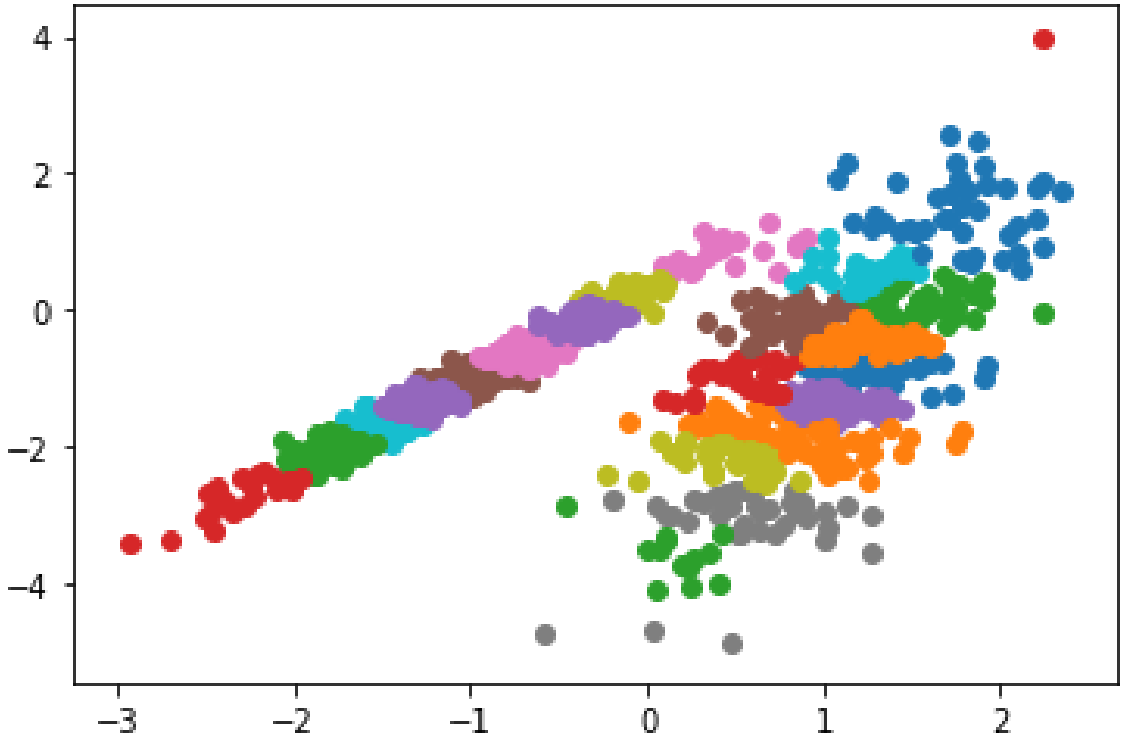


FIGURA 8.4: Algoritmo Affinity Propagation.

En la minería de datos y las estadísticas, la agrupación jerárquica (también llamada análisis de agrupación jerárquica o HCA) es un método de análisis de agrupación que busca construir una jerarquía de agrupaciones. Las estrategias para la agrupación jerárquica generalmente se dividen en dos tipos:

- **Aglomerativo**: este es un enfoque ”de abajo hacia arriba”: cada observación comienza en su propio grupo, y los pares de grupos se fusionan a medida que uno avanza en la jerarquía.
- **Divisivo**: este es un enfoque ”de arriba hacia abajo”: todas las observaciones comienzan en un grupo y las divisiones se realizan de forma recursiva a medida que uno se mueve hacia abajo en la jerarquía.

En general, las fusiones y divisiones se determinan de manera codiciosa (greedy). Los resultados del agrupamiento jerárquico generalmente se presentan en un *dendrograma*. Estos algoritmos son muy lentos y requieren una capacidad computacional alta, hay que tener en cuenta que tiene una complejidad de $\mathcal{O}(n)^3$, requiriendo $\mathcal{O}(n)^2$ de memoria.

Por ejemplo, suponga que estos datos deben agruparse y que la distancia euclidiana es la métrica de la distancia. El dendrograma de agrupamiento jerárquico sería como tal: Cortar el árbol a una altura dada dará una agrupación de partición con una precisión seleccionada. En este ejemplo, cortar después de la segunda fila (desde la parte superior) del dendrograma generará los grupos *"{a}–{b c}–{d e}–{f}"*. Cortar después de la tercera fila generará grupos *"{a}–{b c}–{d e f}"*, que es un grupo más grueso, con un número menor pero grupos más grandes, ver Figura 8.5.

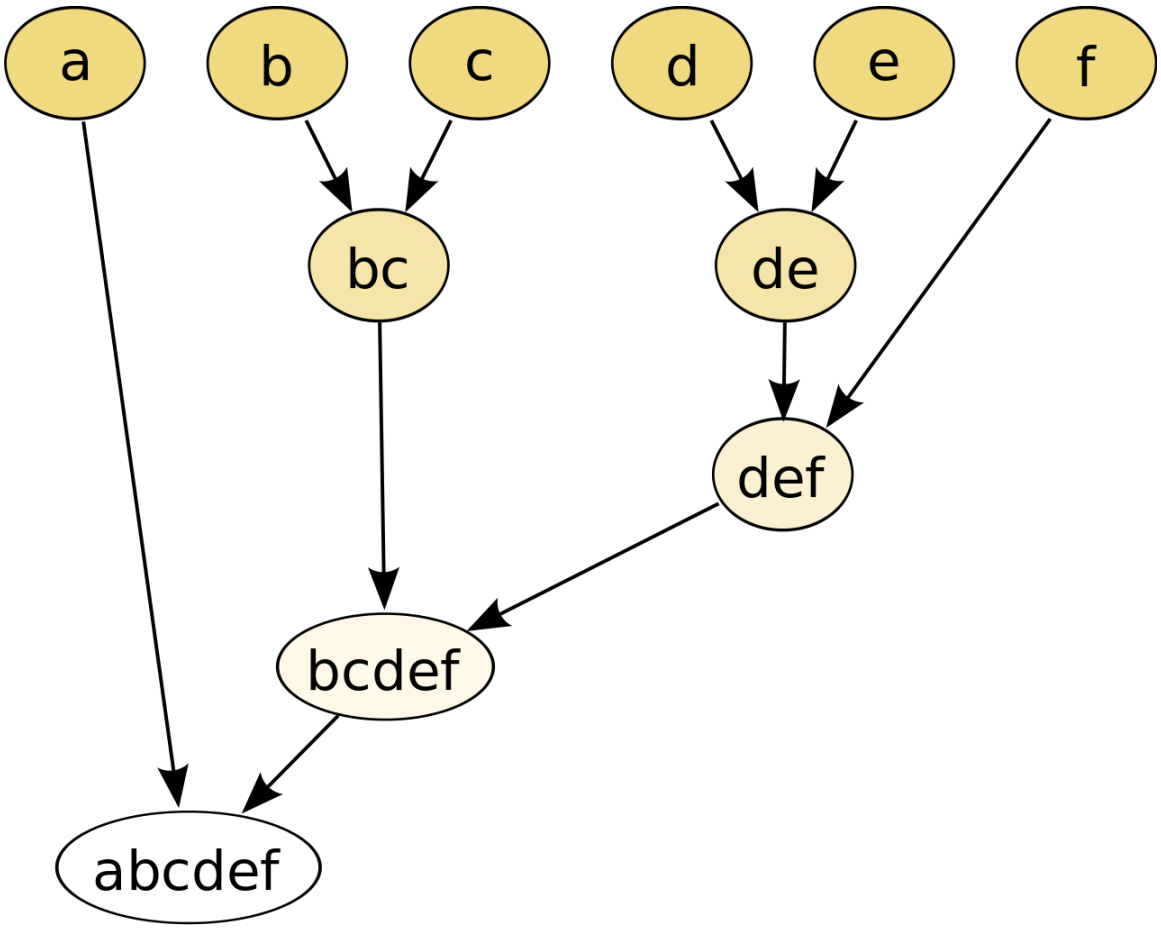


FIGURA 8.5: Dendograma de un algoritmo Agglomerative.

Se implementa a través de la clase `AgglomerativeClustering`[4] y la configuración principal para ajustar es el conjunto de `dumping` entre 0.5 y 1, y tal vez `preference`, ver Código 8.3.

[4] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html

```
> # Agglomerative clustering
> from numpy import unique
> from sklearn.cluster import AgglomerativeClustering
> # Definir el modelo
> model = AgglomerativeClustering(n_clusters=2)
> # Ajustamos y predecimos la clase
> yhat = model.fit_predict(X)
> clusters = unique(yhat)
> for cluster in clusters:
>     row_ix = where(yhat == cluster)
>     pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
> pyplot.show()
```

CÓDIGO 8.3: Algoritmo Agglomerative

La ejecución del ejemplo se ajusta al modelo en el conjunto de datos de entrenamiento y predice un clúster para cada ejemplo en el conjunto de datos. Luego se crea un diagrama de dispersión con puntos coloreados por su grupo asignado. En este caso, se encuentra una agrupación razonable, ver Figura 8.6.

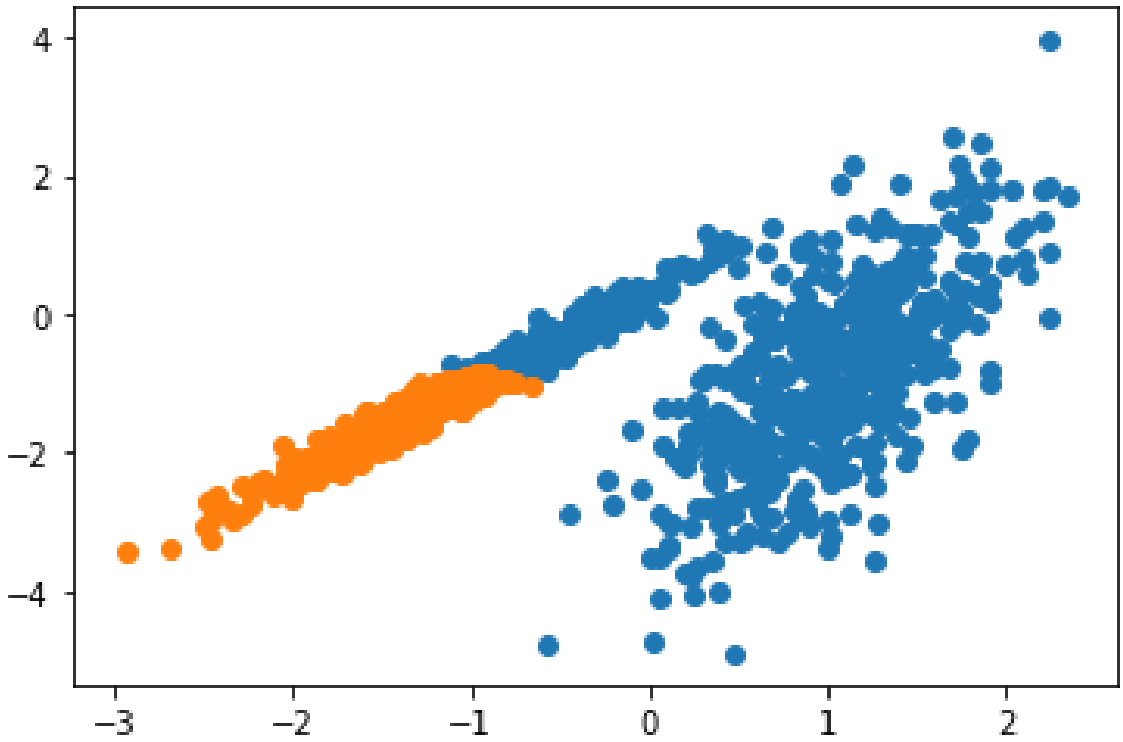


FIGURA 8.6: Algoritmo Agglomerative.

8.2.3. BIRCH

BIRCH Clustering (BIRCH es la abreviatura de Balanced Iterative Reducing and Clustering usando Hierarchies) implica la construcción de una estructura de árbol a partir de la cual se extraen los centroides del clúster.

Es un algoritmo de minería de datos sin supervisión que se utiliza para realizar agrupamientos jerárquicos en conjuntos de datos particularmente grandes. Una ventaja de BIRCH es su capacidad de agrupar de forma incremental y dinámica los puntos de datos métricos multidimensionales entrantes en un intento de producir la agrupación de la mejor calidad para un conjunto dado de recursos (limitaciones de memoria y tiempo). En la mayoría de los casos, BIRCH solo requiere un solo escaneo de la base de datos.

> *"BIRCH agrupa de forma incremental y dinámica los puntos de datos métricos multidimensionales entrantes para tratar de producir la mejor calidad de agrupación con los recursos disponibles (es decir, memoria disponible y limitaciones de tiempo)"*
>
> – *BIRCH: An efficient data clustering method for large databases*[5], *1996.*

Se implementa a través de la clase `Birch`[6] y la configuración principal para sintonizar es los hiperparámetros `umbral` y `n_clusters'`, el último de los cuales proporciona una estimación del número de clústeres, ver Código 8.4.

```
> # Algoritmo clustering Birch
> numpy import unique
> from sklearn.cluster import Birch
> # Definir el modelo
> model = Birch(threshold=0.01, n_clusters=2)
> # Ajustar el modelo
> model.fit(X)
> # Asignar el cluster a cada ejemplo
> yhat = model.predict(X)
```

[5] https://dl.acm.org/doi/10.1145/235968.233324

[6] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.Birch.html

```
> clusters = unique(yhat)
> for cluster in clusters:
>     row_ix = where(yhat == cluster)
>     pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
> pyplot.show()
```

CÓDIGO 8.4: Algoritmo BIRCH

La ejecución del ejemplo se ajusta al modelo en el conjunto de datos de entrenamiento y predice un clúster para cada ejemplo en el conjunto de datos. Luego se crea un diagrama de dispersión con puntos coloreados por su grupo asignado. En este caso, se encuentra una agrupación excelente, ver Figura 8.7.

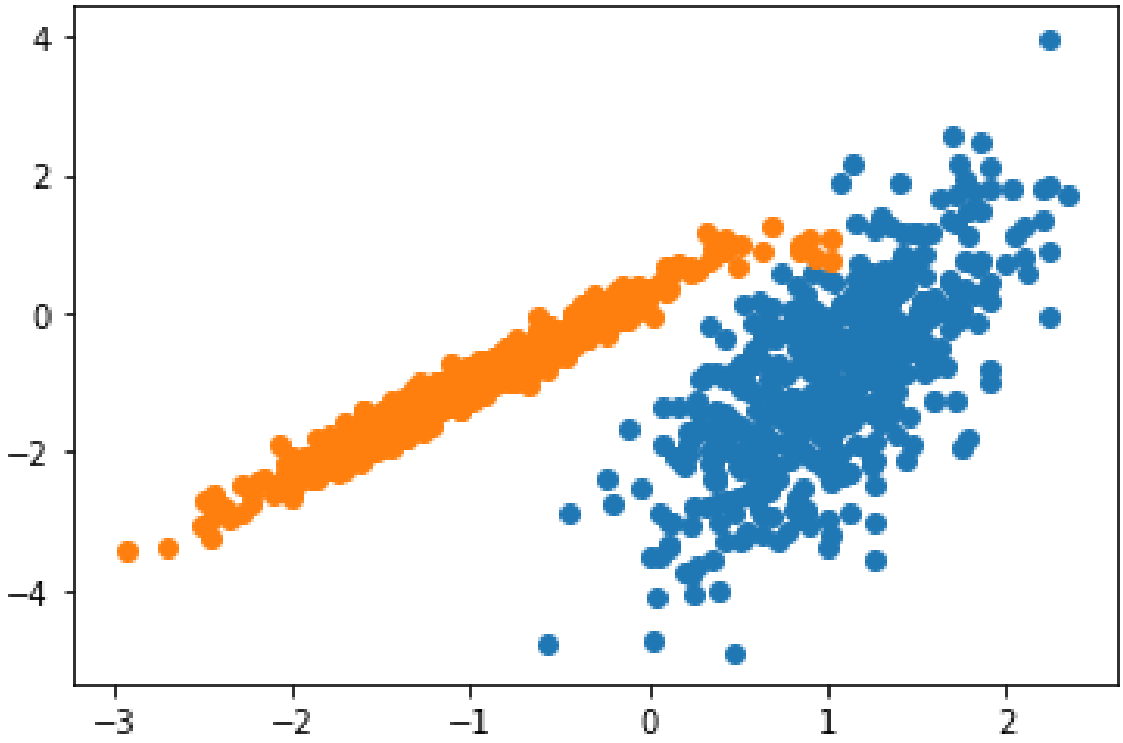


FIGURA 8.7: Algoritmo BIRCH.

8.2.4. DBSCAN

DBSCAN (abreviatura de Density-Based Spatial Clustering of Applications with Noise) es un algoritmo no paramétrico de agrupamiento basado en la densidad: dado un conjunto de puntos en algún espacio, agrupa los puntos

que están muy juntos (puntos con muchos vecinos cercanos), marcando como puntos atípicos que se encuentran solos en regiones de baja densidad (cuyos vecinos más cercanos están demasiado lejos). DBSCAN es uno de los algoritmos de agrupamiento más comunes y también más citado en la literatura científica.

DBSCAN implica encontrar áreas de alta densidad en el dominio y expandir esas áreas del espacio de características a su alrededor como clústeres.

> *"...presentamos el nuevo algoritmo de agrupación DBSCAN que se basa en una noción basada en densidad de agrupaciones que está diseñada para descubrir agrupaciones de forma arbitraria. DBSCAN requiere solo un parámetro de entrada y ayuda al usuario a determinar un valor apropiado para él".*

– *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*[7], *1996.*

Se implementa a través de la clase `DBSCAN`[8] y la configuración principal para ajustar son los hiperparámetros `eps` y `min_samples`, ver Código 8.5.

```
> #DBSCAN Clustering
> from numpy import unique
> from sklearn.cluster import DBSCAN
> # Definir el modelo
> model = DBSCAN(eps=0.30, min_samples=9)
> # Asignar el cluster a cada ejemplo
> yhat = model.fit_predict(X)
> clusters = unique(yhat)
> for cluster in clusters:
>     row_ix = where(yhat == cluster)
>     pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
> pyplot.show()
```

CÓDIGO 8.5: Algoritmo DBSCAN

La ejecución del ejemplo se ajusta al modelo en el conjunto de datos de entrenamiento y predice un clúster para cada ejemplo en el conjunto de datos.

[7] https://www.osti.gov/biblio/421283

[8] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html

Luego se crea un diagrama de dispersión con puntos coloreados por su grupo asignado. En este caso, se encuentra una agrupación razonable, aunque se requiere más ajuste, ver Figura 8.8.

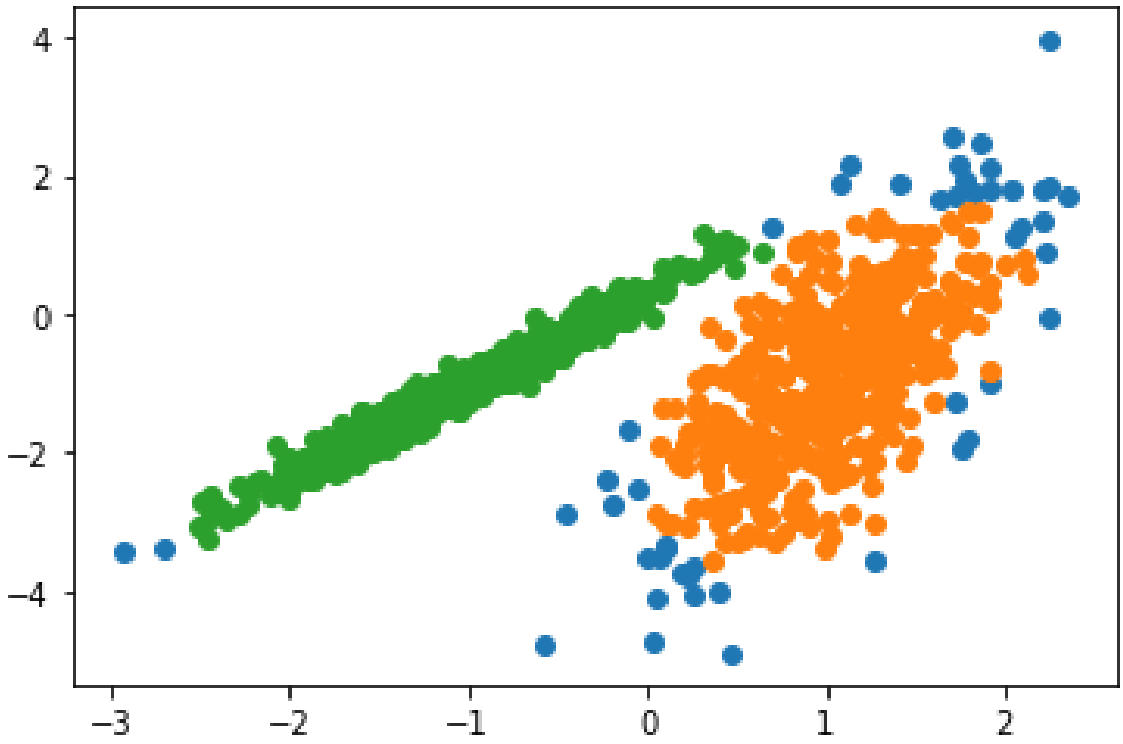


FIGURA 8.8: Algoritmo DBSCAN.

Ver Vídeo de este apartado

```
8.2. Algoritmos clustering (I).
```

8.2.5. K-Means

K-Means puede ser el algoritmo de agrupación más conocido e implica asignar ejemplos a los grupos en un esfuerzo por minimizar la variación dentro de cada grupo.

> *"El objetivo principal de este trabajo es describir un proceso para dividir una población N-dimensional en k conjuntos sobre la base de una muestra. El proceso, que se llama k-means, parece dar particiones que son razonablemente eficientes en el sentido de la variación dentro de la clase.".*

– Some methods for classification and analysis of multivariate observations[9], *1967.*

Se implementa a través de la clase `KMeans`[10] y la configuración principal para ajustar es el hiperparámetro `n_clusters` establecido en el número estimado de clústeres en los datos, ver Código 8.6.

```
> # k–Means Clustering
> from numpy import unique
> from sklearn.cluster import KMeans
> # Definir el modelo
> model = KMeans(n_clusters=2)
> # Asignar el cluster a cada ejemplo
> yhat = model.fit_predict(X)
> clusters = unique(yhat)
> for cluster in clusters:
>     row_ix = where(yhat == cluster)
>     pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
> pyplot.show()
```

CÓDIGO 8.6: Algoritmo k-Means

La ejecución del ejemplo se ajusta al modelo en el conjunto de datos de entrenamiento y predice un clúster para cada ejemplo en el conjunto de datos. Luego se crea un diagrama de dispersión con puntos coloreados por su grupo asignado. En este caso, se encuentra una agrupación razonable, aunque la varianza desigual en cada dimensión hace que el método sea menos adecuado para este conjunto de datos, ver Figura 8.9.

8.2.6. Mini-Batch K-Means

Mini-Batch K-Means es una versión modificada de *k*-means que realiza actualizaciones a los centroides del clúster utilizando mini-lotes de muestras en lugar de todo el conjunto de datos, lo que puede hacerlo más rápido para grandes conjuntos de datos y quizás más robusto al ruido estadístico.

[9] https://projecteuclid.org/euclid.bsmsp/1200512992
[10] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

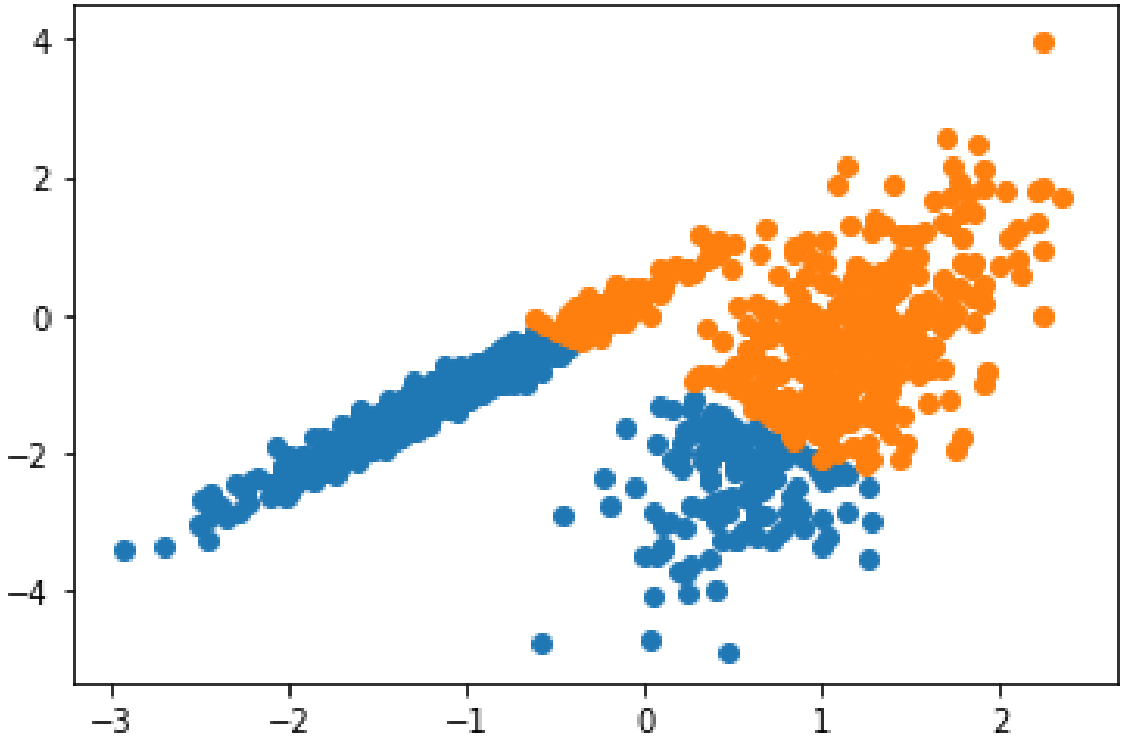


FIGURA 8.9: Algoritmo k-Means.

> *"...proponemos el uso de la optimización de mini lotes para la agrupación de k-means. Esto reduce el costo de cómputo en órdenes de magnitud en comparación con el algoritmo de lote clásico al tiempo que proporciona soluciones significativamente mejores que el descenso de gradiente estocástico en línea.".*
>
> – *Web-Scale K-Means Clustering*[11], *2010.*

Se implementa a través de la clase `MiniBatchKMeans`[12] y la configuración principal para ajustar es el hiperparámetro `n_clusters` establecido en el número estimado de clústeres en los datos, ver Código 8.7.

```
> # Algortimo Mini-Batch K-Means
> from numpy import unique
> from sklearn.cluster import MiniBatchKMeans
> # Definir el modelo
```

[11] https://dl.acm.org/doi/10.1145/1772690.1772862
[12] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MiniBatchKMeans.html

```
> model = MiniBatchKMeans(n_clusters=2)
> # Ajustar el modelo
> model.fit(X)
> # Asignar el cluster a cada ejemplo
> yhat = model.predict(X)
> clusters = unique(yhat)
> for cluster in clusters:
>     row_ix = where(yhat == cluster)
>     pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
> pyplot.show()
```

CÓDIGO 8.7: Algoritmo Mini-Batch K-Means

En este caso, se encuentra un resultado equivalente al algoritmo estándar *k*-means, ver Figura 8.10.

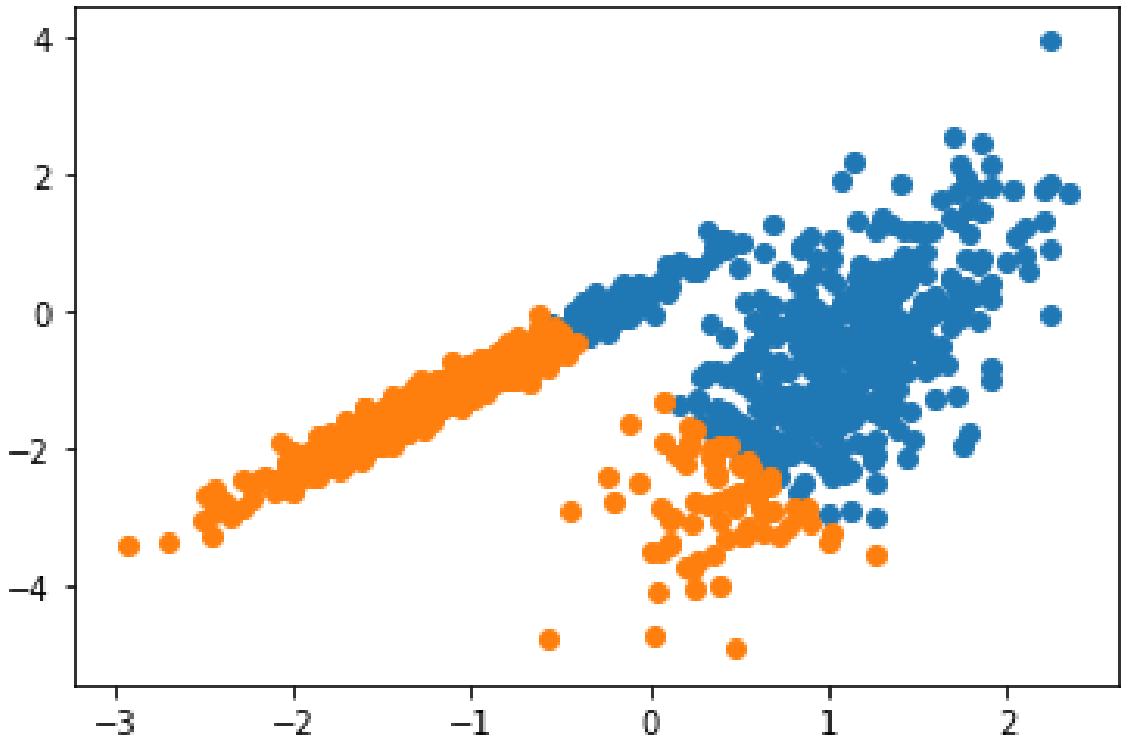


FIGURA 8.10: Algoritmo Mini-Batch K-Means.

8.2.7. Mean Shift

Mean Shift es una técnica de análisis de espacio de características no paramétrica para localizar los máximos de una función de densidad, un denominado algoritmo de búsqueda de modo. Los dominios de aplicación incluyen análisis de conglomerados en visión por computadora y procesamiento de imágenes.

La agrupación Mean Shift implica encontrar y adaptar los centroides en función de la densidad de ejemplos en el espacio de características.

> *"Probamos para datos discretos la convergencia de un procedimiento de desplazamiento medio recursivo al punto estacionario más cercano de la función de densidad subyacente y, por lo tanto, su utilidad para detectar los modos de la densidad."*
>
> – *Mean Shift: A robust approach toward feature space analysis*[13], *2002.*

Se implementa a través de la clase `MeanShift`[14] y la configuración principal para ajustar es el hiperparámetro `bandwith`, ver Código 8.8.

```
> # Algoritmo Mean Shift
> from numpy import unique
> from sklearn.cluster import MeanShift
> # Definir el modelo
> model = MeanShift()
> # Ajustar y predecir los clusters
> yhat = model.fit_predict(X)
> clusters = unique(yhat)
> for cluster in clusters:
>     row_ix = where(yhat == cluster)
>     pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
> pyplot.show()
```

CÓDIGO 8.8: Algoritmo Mean Shift

[13] https://www.computer.org/csdl/journal/tp/2002/05/i0603/13rRUxYrbVE

[14] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MeanShift.html

En este caso, se encuentra un conjunto razonable de clústeres en los datos, ver Figura 8.11.

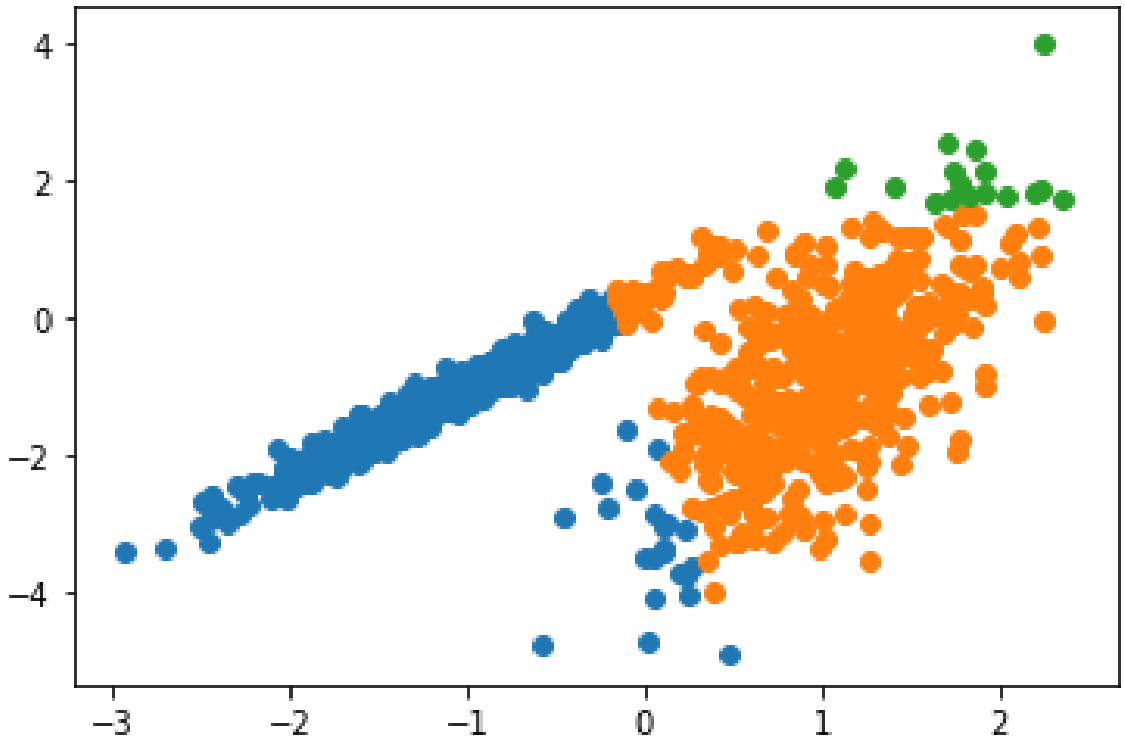


FIGURA 8.11: Algoritmo Mean Shift.

8.2.8. OPTICS

La agrupación de OPTICS (Ordering Points To Identify the Clustering Structure) es una versión modificada de DBSCAN, pero aborda una de las principales debilidades de DBSCAN: el problema de detectar grupos significativos en datos de densidad variable. Para hacerlo, los puntos de la base de datos se ordenan (linealmente) de modo que los puntos espacialmente más cercanos se conviertan en vecinos en el orden. Además, se almacena una distancia especial para cada punto que representa la densidad que se debe aceptar para un grupo para que ambos puntos pertenezcan al mismo grupo. Esto se representa como un dendrograma.

> *"Introducimos un nuevo algoritmo para el análisis de conglomerados que no produce una agrupación de un conjunto de datos explícitamente; pero*

en su lugar crea un orden aumentado de la base de datos que representa su estructura de agrupamiento basada en densidad. Este orden de agrupación contiene información que es equivalente a las agrupaciones basadas en densidad correspondientes a una amplia gama de configuraciones de parámetros."

– OPTICS: ordering points to identify the clustering structure[15]*, 1999.*

Se implementa a través de la clase `OPTICS`[16] y la configuración principal para ajustar son los hiperparámetros `eps` y `min_samples`, ver Código 8.9.

```
> # Algoritmo OPTICS
> from numpy import unique
> from sklearn.cluster import OPTICS
> # Definir el modelo
> model = OPTICS(eps=0.8, min_samples=10)
> # Ajustar y predecir los clusters
> yhat = model.fit_predict(X)
> clusters = unique(yhat)
> for cluster in clusters:
>     row_ix = where(yhat == cluster)
>     pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
> pyplot.show()
```

CÓDIGO 8.9: Algoritmo OPTICS

En este caso, no pude lograr un resultado razonable en este conjunto de datos, ver Figura 8.12.

8.2.9. Spectral Clustering

Spectral Clustering es una clase general de métodos de agrupación, extraída del álgebra lineal.

"Una alternativa prometedora que ha surgido recientemente en varios campos es utilizar métodos espectrales para la agrupación. Aquí, uno usa

[15] https://dl.acm.org/doi/10.1145/304182.304187
[16] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.OPTICS.html

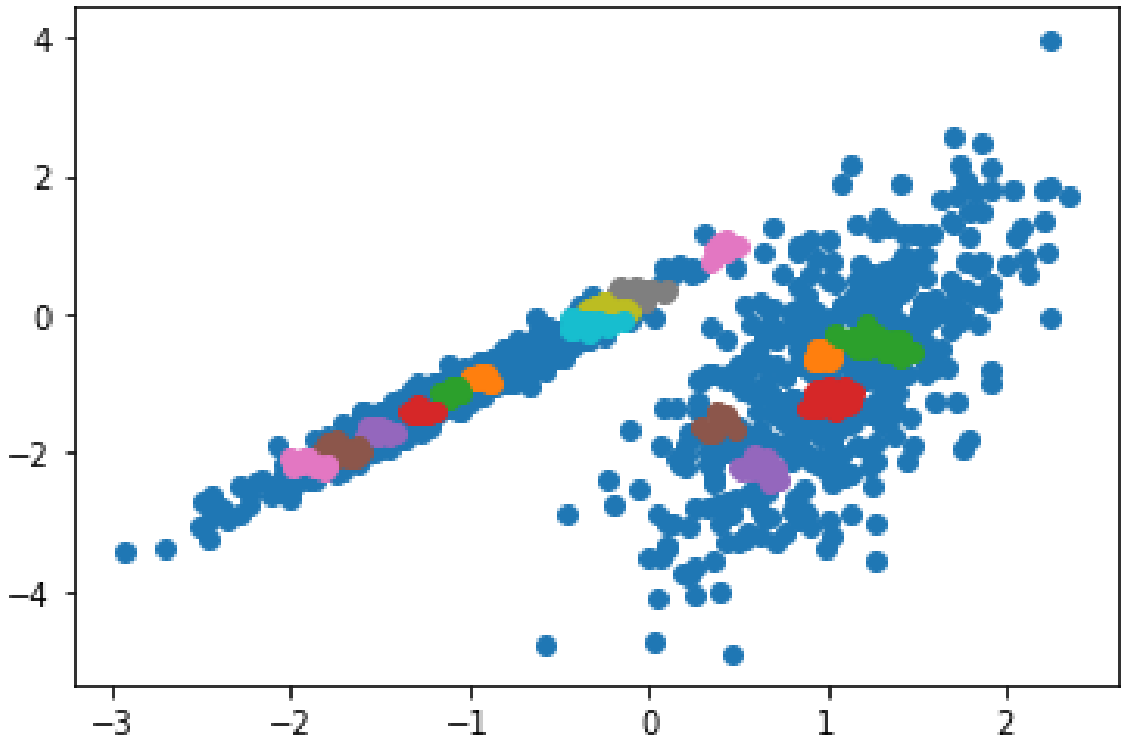


FIGURA 8.12: Algoritmo OPTICS.

> *los vectores propios superiores de una matriz derivada de la distancia entre puntos."*
>
> *– On Spectral Clustering: Analysis and an algorithm*[17]*, 2002.*

Se implementa a través de la clase `SpectralClustering`[18] y el principal Spectral Clustering es una clase general de métodos de agrupamiento, extraídos del álgebra lineal. El principal hiperparámetro es `n_clusters` utilizado para especificar el número estimado de clústeres en los datos, ver Código 8.10.

```
> # Algoritmo Spectral Clustering
> from numpy import unique
> from sklearn.cluster import SpectralClustering
> # Definir el modelo
> model = SpectralClustering(n_clusters=2)
```

[17] https://papers.nips.cc/paper/2092-on-spectral-clustering-analysis-and-an-algorithm.pdf

[18] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html

```
> # Ajustar y predecir los clusters
> yhat = model.fit_predict(X)
> clusters = unique(yhat)
> for cluster in clusters:
>     row_ix = where(yhat == cluster)
>     pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
> pyplot.show()
```

CÓDIGO 8.10: Algoritmo Spectral Clustering

En este caso, observamos un resultado razonable en este conjunto de datos, ver Figura 8.13.

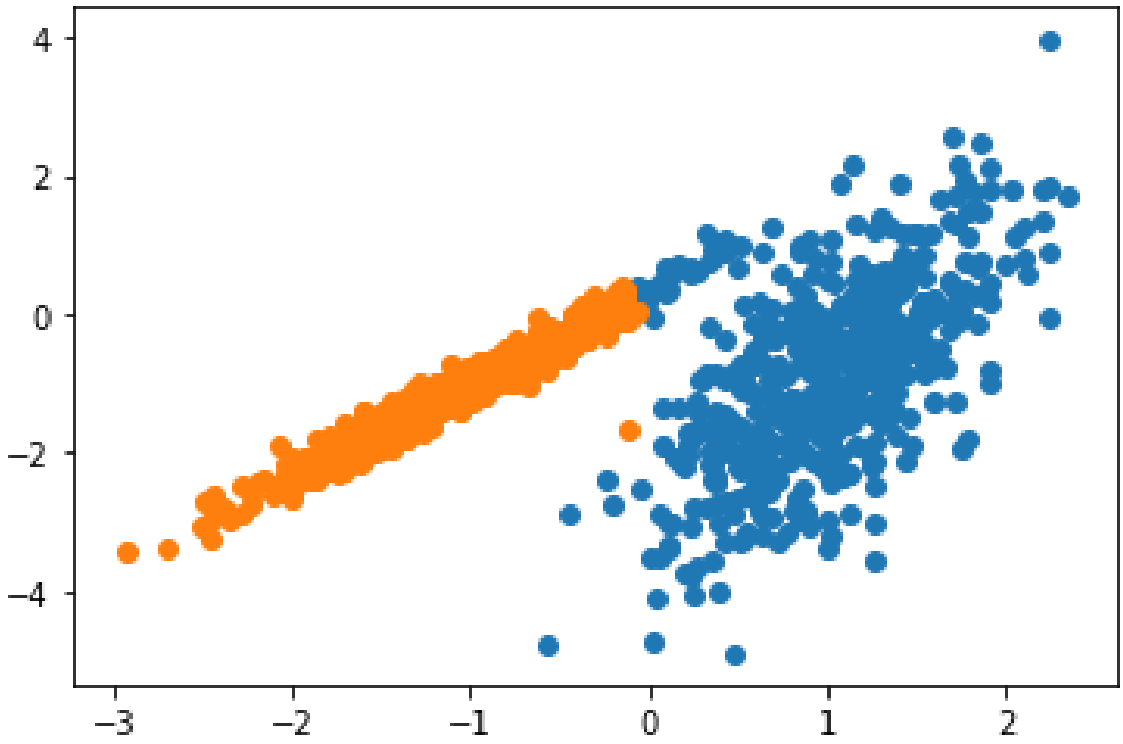


FIGURA 8.13: Algoritmo Spectral Clustering.

8.2.10. Gaussian Mixture Model

Un modelo de mezcla gaussiana resume una función de densidad de probabilidad multivariada con una mezcla de distribuciones de probabilidad gaussianas como su nombre indica.

Se implementa a través de la clase `GaussianMixture`[19] y la configuración principal para ajustar es el hiperparámetro 'n_clusters' utilizado para especificar el número estimado de clústeres en los datos, ver Código 8.11.

```
> # Algortimo Gaussian Mixture Model
> from numpy import unique
> from sklearn.mixture import GaussianMixture
> # Definir el modelo
> model = GaussianMixture(n_components=2)
> # Ajustar el modelo
> model.fit(X)
> # Asignar el cluster a cada ejemplo
> yhat = model.predict(X)
> clusters = unique(yhat)
> for cluster in clusters:
>     row_ix = where(yhat == cluster)
>     pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
> pyplot.show()
```

CÓDIGO 8.11: Algoritmo Gaussian Mixture Model

En este caso, podemos ver que los grupos se identificaron perfectamente. Esto no es sorprendente dado que el conjunto de datos se generó como una mezcla de gaussianos, ver Figura 8.14.

Ver Vídeo de este apartado

```
8.3. Algoritmos clustering (II).
```

Abrir Jupyter Notebook

```
8.2. Algoritmos ULAs.
```

Ver Vídeo de este apartado

```
8.4. Implementación Algoritmos clustering (I).
```

[19] https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html

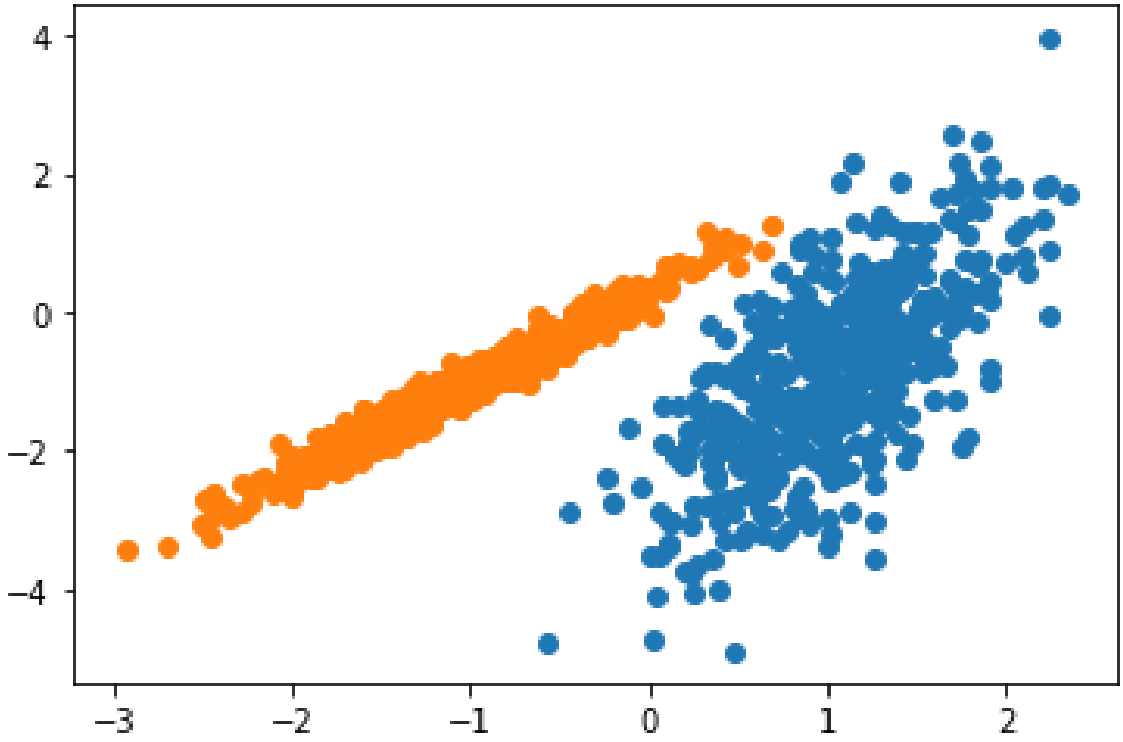


FIGURA 8.14: Algoritmo Gaussian Mixture Model.

Ver Vídeo de este apartado

```
8.5. Implementación Algoritmos clustering (II).
```

Ver Vídeo de este apartado

```
8.6. k-Means.
```

Ver Vídeo de este apartado

```
8.7. Clustering Jerárquico.
```

Ver Vídeo de este apartado

```
8.8. Métodos basado en densidad.
```

8.3. Determinar número óptimo de clústers

Uno de los problemas que nos encontramos a la hora de aplicar alguno de los métodos de Clustering es la elección del número de Clusters. No existe un criterio objetivo ni ampliamente válido para la elección de un número óptimo de Clusters; pero tenemos que tener en cuenta, que una mala elección de los mismos puede dar lugar a realizar agrupaciones de datos muy heterogéneos (pocos Clusters); o datos, que siendo muy similares unos a otros los agrupemos en Clusters diferentes (muchos Clusters).

Aunque no exista un criterio objetivo para la selección del número de Clusters, si que se han implementado diferentes métodos que nos ayudan a elegir un número apropiado de Clusters para agrupar los datos; como son, el método del codo (*elbow method*), el criterio de Calinsky, el Affinity Propagation (AP), el GAP (también con su versión estadística), Dendrogramas, etc. Dada la complejidad de alguno de estos métodos, vamos a explicar aquellos que son más sencillos y que nos dan; para la mayoría de los casos, unos resultados que nos permiten tomar la decisión de cuál será el número óptimo de Clusters para el conjunto de datos.

8.3.1. Método del codo

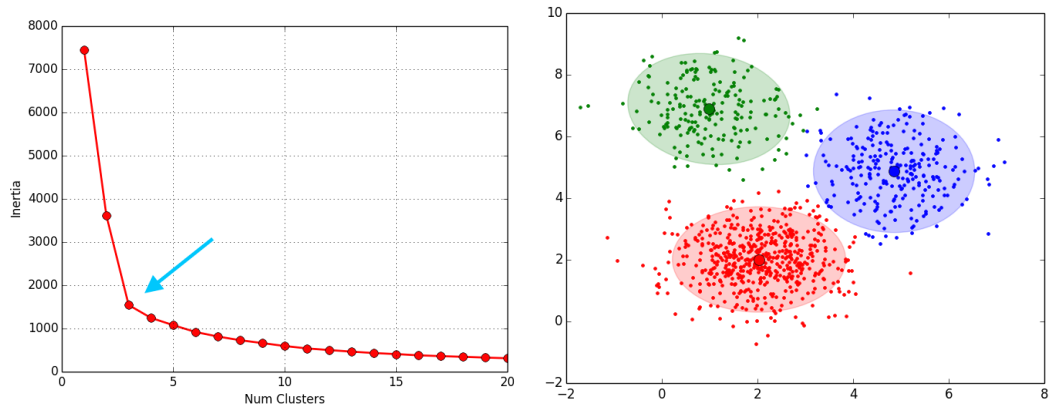
Este método utiliza los valores de la inercia obtenidos tras aplicar el algoritmo a diferentes números de Clusters (desde 1 a N Clusters), siendo la inercia la suma de las distancias al cuadrado de cada objeto del Cluster a su centroide:

$$Inercia = \sum_{i=0}^{N} ||x_i - \mu||^2 \tag{8.1}$$

Una vez obtenidos los valores de la inercia tras aplicar el algoritmo de 1 a N Clusters, representamos en una gráfica lineal la inercia respecto del número de Clusters. En esta gráfica se debería de apreciar un cambio brusco en la evolución

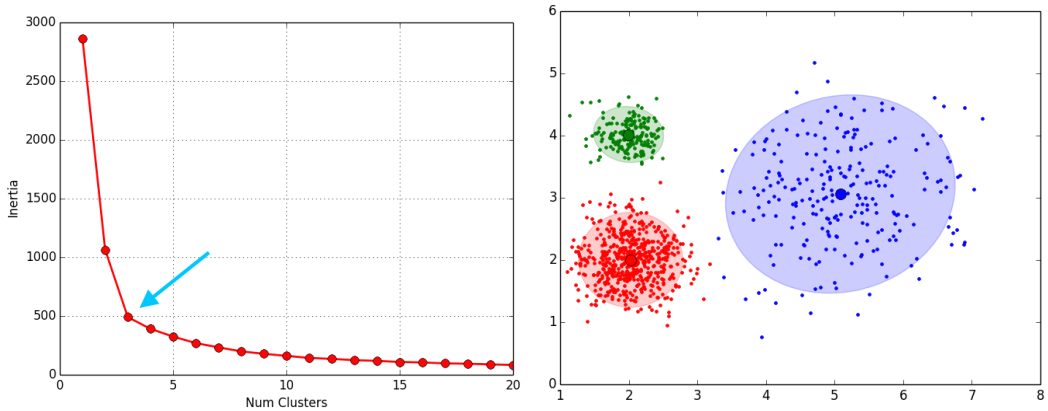
de la inercia, teniendo la línea representada una forma similar a la de un brazo y su codo. El punto en el que se observa ese cambio brusco en la inercia nos dirá el número óptimo de Clusters a seleccionar para ese conjunto de datos; o dicho de otra manera: el punto que representaría al codo del brazo será el número óptimo de Clusters para ese data set.

El script que se muestra a continuación, calcula los valores de la inercia tras aplicar el el algoritmo (*k*-means) de 1 a 20 Clusters (en tres dataset de prueba) y los pinta en una gráfica lineal (número de Clusters respecto a la inercia) para poder apreciar "el codo" y por tanto determinar el número óptimo de Clusters para el dataset, ver pares de Figuras 8.15, 8.16 y 8.17.



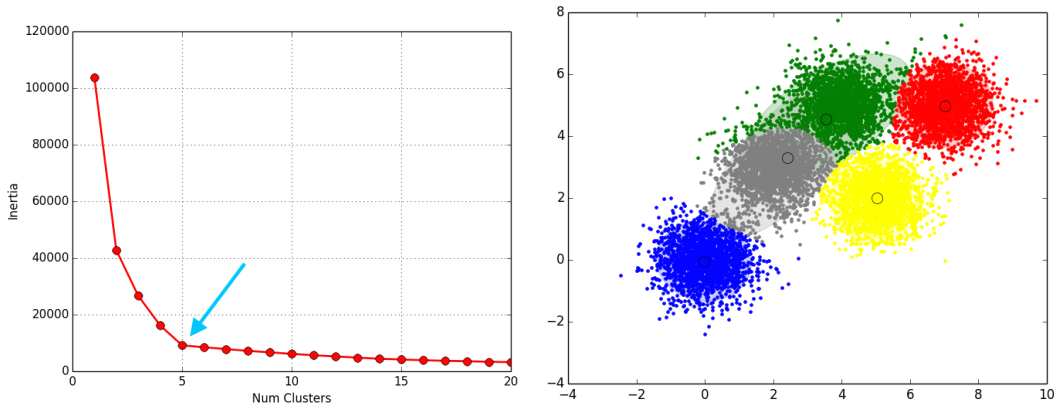
(a) Gráfica del codo. (b) Clasificación de los clúster.

FIGURA 8.15: Conjunto de datos de prueba 1.



(a) Gráfica del codo. (b) Clasificación de los clúster.

FIGURA 8.16: Conjunto de datos de prueba 2.



(a) Gráfica del codo. (b) Clasificación de los clúster.

FIGURA 8.17: Conjunto de datos de prueba 3.

Como se puede apreciar en los resultados obtenidos, el método del codo devuelve unos valores muy coherentes y se ajusta a los resultados esperados. Es posible que al aplicar este método para otro conjunto de datos no se aprecie "el codo" o incluso se observen dos o más codos (o cambios bruscos en la evolución de la inercia). En ese caso habría que estudiar más en detalle o con otras técnicas el número óptimo de Clusters a seleccionar. Dada la finalidad didáctica de estos ejemplos, se aprecia muy bien "el codo" en la evolución de la inercia, pero en la realidad no siempre se observa este comportamiento tan claro.

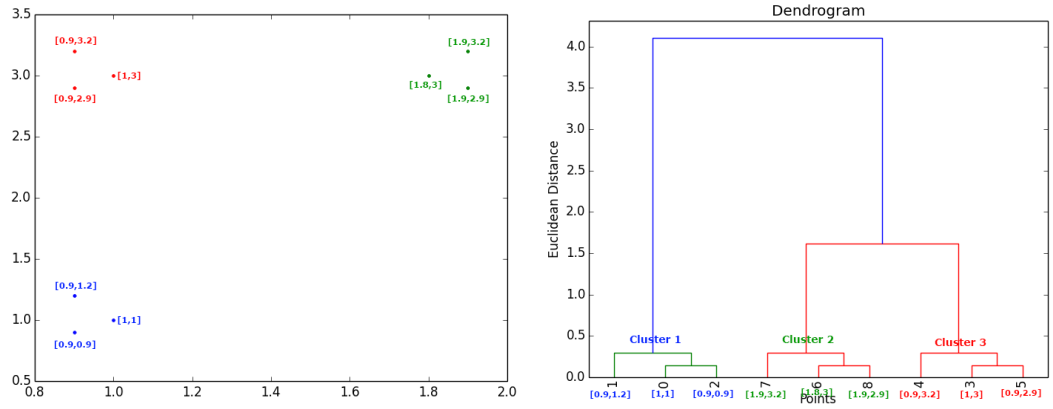
8.3.2. Dendograma

Un dendrograma es un tipo de representación gráfica en forma de árbol que organiza y agrupa los datos en subcategorías según su similitud; dada por alguna medida de distancia. Los objetos similares se representan en el dendrograma por medio de un enlace cuya posición está determinada por el nivel de similitud entre los objetos o grupos de objetos. Dadas estas características, hace que los dendrogramas sean un tipo de diagrama muy útil para estudiar las agrupaciones de objetos; es decir, para estudiar los Clusters que pueden darse en un dataset.

Veamos a continuación un sencillo ejemplo, en el que tenemos 9 objetos representados en un plano, ver Figura 8.18(a).

Se puede observar claramente que estos 9 objetos los podemos agrupar en 3 Clusters, pero comencemos estudiando las similitudes y distancias entre objetos. En primer lugar fijémonos en los objetos de color azul y veamos las distancias (euclideas en este caso) entre esos objetos. Vemos que los dos puntos más cercanos entre sí son el punto $1, 1$ y el punto $0.9, 0.9$, por lo que los uniríamos en el dendrograma con un enlace. Si agrupamos estos dos puntos y calculamos su centroide, obtenemos el punto $0.95, 0.95$ y este nuevo punto tiene como punto más cercano de entre todos los que hay en el plano al punto $0.9, 1.2$; por tanto, uniríamos con otro enlace al primer grupo formado, con el punto $0.9, 1.2$. En resumen lo que se hace es calcular la distancia entre todos los puntos, cogemos la menor distancia, agrupamos esos dos puntos, y volvemos a calcular la distancia entre todos los puntos o entre todos los grupos ya formados y puntos.

Siguiendo estos pasos, el dendrograma resultante para estudiar la relación entre los 9 puntos antes mostrados sería el siguiente, dando lugar a interpretar que el número óptimo de Clusters a seleccionar serian 3, ver Figura 8.18(b).



(a) Representación de 9 puntos.

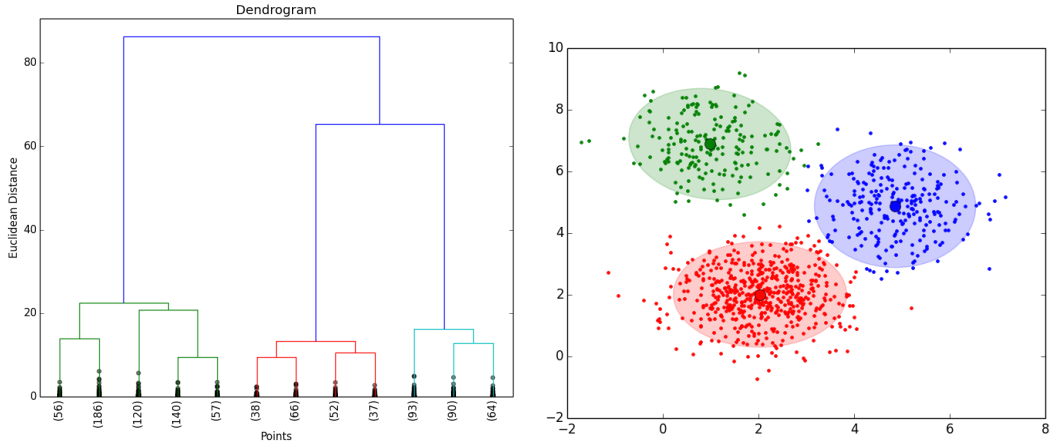
(b) Dendograma de los 9 puntos.

FIGURA 8.18: Representación de puntos en dos dimensiones en un dendograma.

En Python utilizaremos la función `linkage()` correspondiente a la clase `scipy.cluster.hierarchy.linkage`[20].

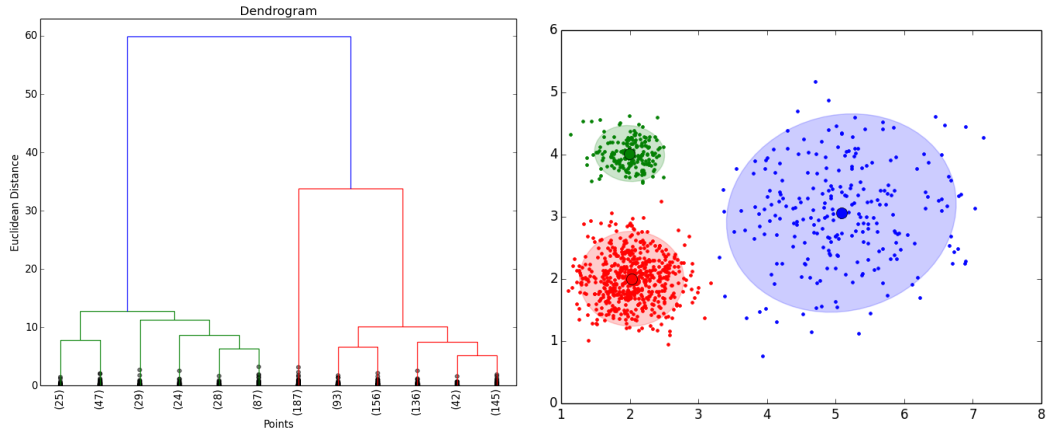
[20] https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.cluster.hierarchy.linkage.html

A continuación se muestran los resultados obtenidos para cada uno de los tres datasets. Como puede observarse en los resultados obtenidos, los dendrogramas muestran las agrupaciones (Clusters) de objetos esperados. Al igual que en la aplicación del método del codo, es posible que no se puedan apreciar claramente las agrupaciones de objetos, por lo que habría que estudiar con otras técnicas el número óptimo de Clusters a seleccionar, ver pares de Figuras 8.19, 8.20 y 8.21.



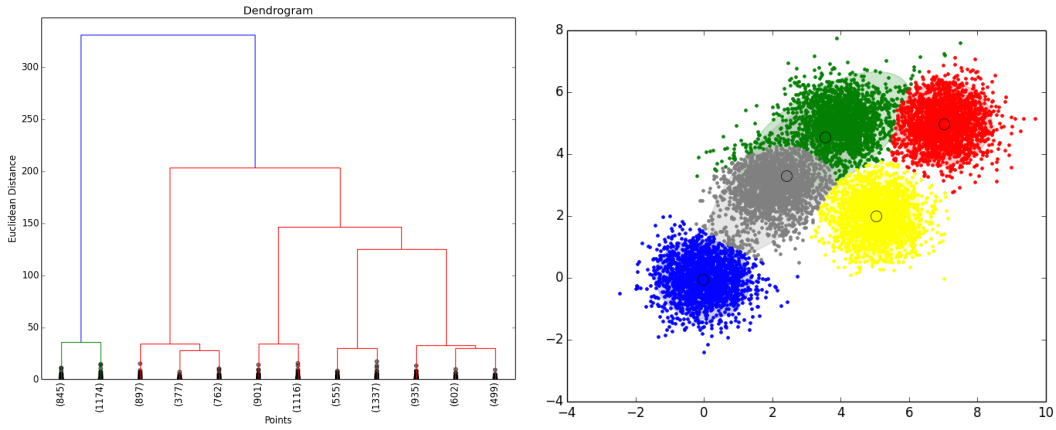
(a) Dendograma. (b) Clasificación de los clúster.

FIGURA 8.19: Conjunto de datos de prueba 1.



(a) Dendograma. (b) Clasificación de los clúster.

FIGURA 8.20: Conjunto de datos de prueba 2.



(a) Dendograma. (b) Clasificación de los clúster.

FIGURA 8.21: Conjunto de datos de prueba 3.

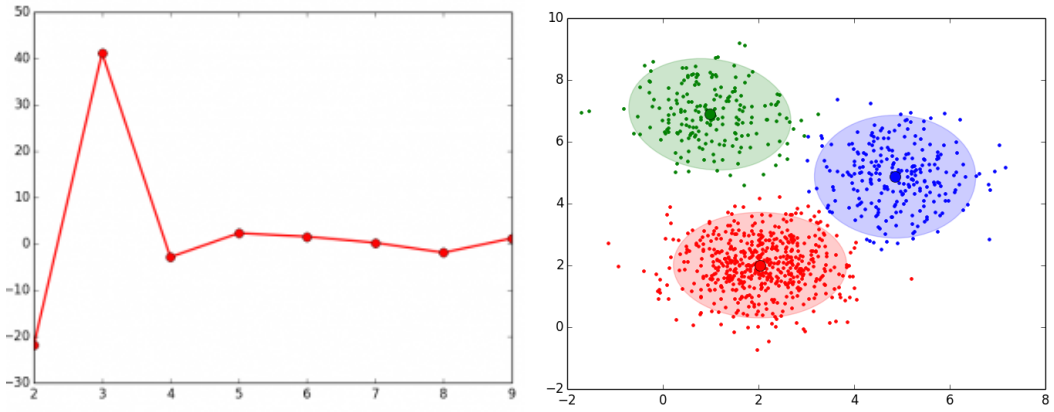
8.3.3. GAP

GAP (brecha) es similar al método del codo, cuya finalidad es la de encontrar la mayor diferencia o distancia que hay entre los diferentes grupos de objetos que vamos formando para representarlos en un dendrograma. Para ello vamos cogiendo las distancias que hay de cada uno de los enlaces que forman el dendrograma y vemos cual es la mayor diferencia que hay entre cada uno de estos enlaces.

Como puede observarse en los resultados obtenidos estudiando el GAP, devuelve el número de agrupaciones (Clusters) esperados. Al igual que en el resto de métodos mostrados anteriormente, es posible que no se puedan apreciar claramente las agrupaciones de objetos, por lo que habría que estudiar con otras técnicas el número óptimo de Clusters a seleccionar, ver pares de Figuras 8.22, 8.23 y 8.24.

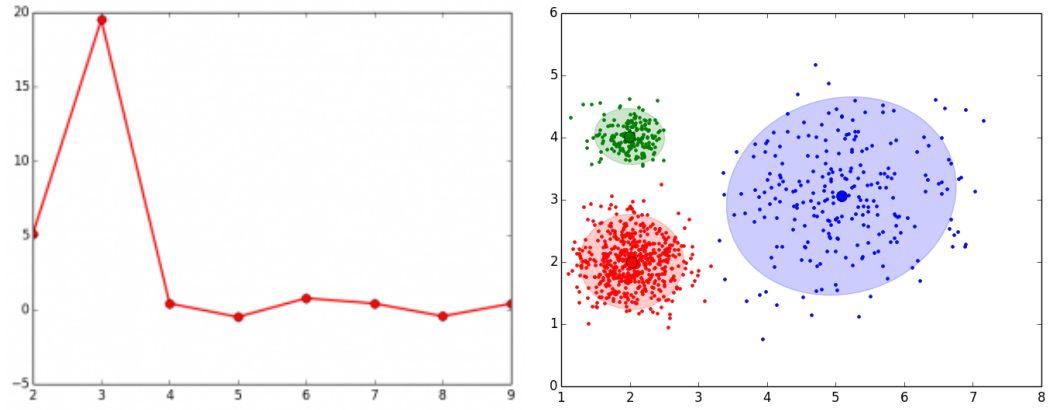
8.3.4. Silhouette

El análisis Silhouette se puede utilizar para estudiar la distancia de separación entre los grupos resultantes. El diagrama de Silhouette muestra una medida de cuán cerca está cada punto de un grupo a los puntos en los grupos vecinos y, por lo tanto, proporciona una forma de evaluar visualmente parámetros como el número de grupos. Esta medida tiene un rango de $[-1, 1]$.



(a) Gráfica GAP. (b) Clasificación de los clúster.

FIGURA 8.22: Conjunto de datos de prueba 1.

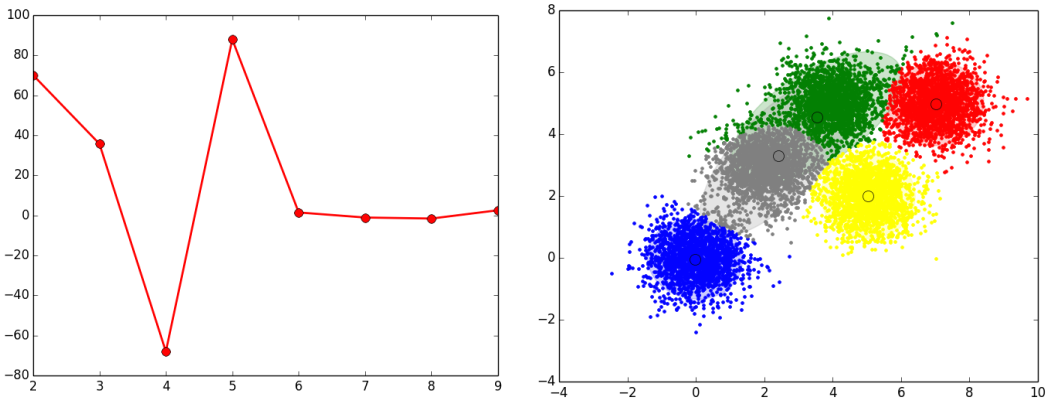


(a) Gráfica GAP. (b) Clasificación de los clúster.

FIGURA 8.23: Conjunto de datos de prueba 2.

Los coeficientes de Silhouette (como se hace referencia a estos valores) cerca de $+1$ indican que la muestra está muy lejos de los grupos vecinos. Un valor de 0 indica que la muestra está dentro o muy cerca del límite de decisión entre dos grupos vecinos. Y los valores negativos indican que esas muestras podrían haberse asignado al grupo incorrecto.

En este ejemplo, el análisis de Silhouette se utiliza para elegir un valor óptimo para un número determinado de clústes (*n_clusters*). El gráfico de Silhouette muestra que el valor *n_clusters* de 3 y 5 es una mala elección para los datos dados debido a la presencia de grupos **con puntajes de Silhouette por debajo del promedio** y también debido **a amplias fluctuaciones en el tamaño** de los gráficos de Silhouette. El análisis de Silhouette es más ambivalente al



(a) Gráfica GAP. (b) Clasificación de los clúster.

FIGURA 8.24: Conjunto de datos de prueba 3.

decidir entre 2 y 4.

También desde el grosor de la trama de la Silhouette se puede visualizar el tamaño del grupo. El diagrama de Silhouette para el grupo 0 cuando *n_clusters* es igual a 2, es de mayor tamaño debido a la agrupación de los 3 subgrupos en un gran grupo. Sin embargo, cuando *n_clusters* es igual a 4, todas las gráficas tienen un grosor más o menos similar y, por lo tanto, tienen tamaños similares, como también se puede verificar a partir de la gráfica de dispersión etiquetada a la derecha, ver Figuras 8.25.

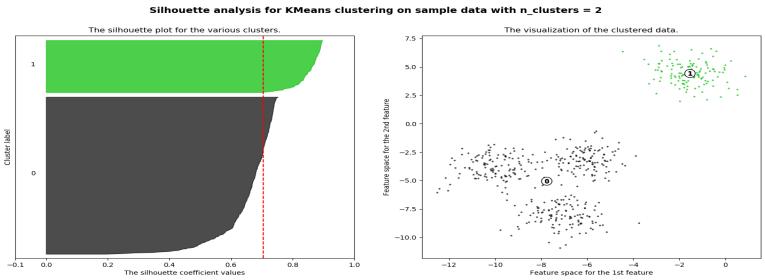
Así mismo, y teniendo en cuenta el ejemplo anterior, Silhouette nos arroja el coeficiente que permite resumir este análisis gráfico como se muestra en el Código. Si nos fijamos 2 y 4 clústers sería una opción óptima, ver Código 8.12.

```
For n_clusters = 2 The average silhouette_score is : 0.7049787496083262
For n_clusters = 3 The average silhouette_score is : 0.5882004012129721
For n_clusters = 4 The average silhouette_score is : 0.6505186632729437
For n_clusters = 5 The average silhouette_score is : 0.5745566973301872
For n_clusters = 6 The average silhouette_score is : 0.43902711183132426
```

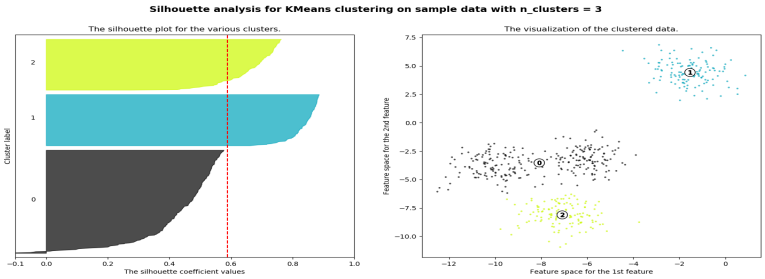
CÓDIGO 8.12: Salida del coeficiente de Silhouette

Ver Vídeo de este apartado

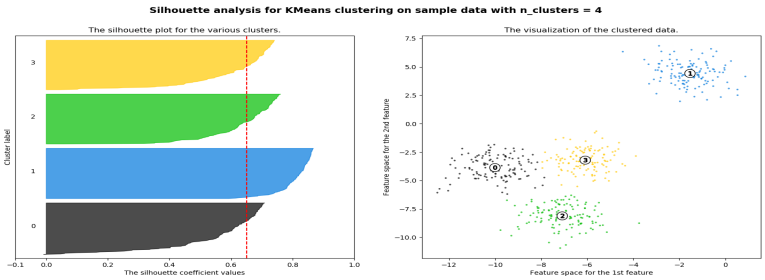
```
    8.9. Determinar clusters.
```



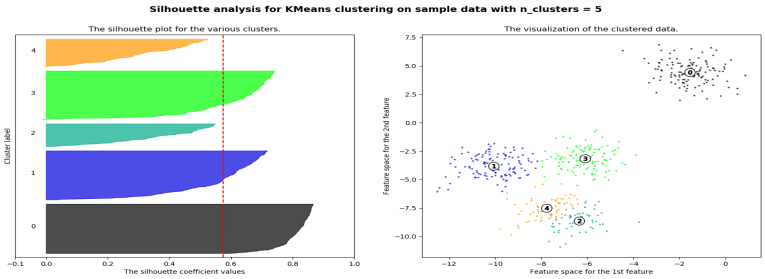
(a) Resultados con 2 clústers.



(b) Resultados con 3 clústers.



(c) Resultados con 4 clústers.



(d) Resultados con 5 clústers.

FIGURA 8.25: Análisis Silhouette.

8.4. Análisis de clustering en un dataset

Veamos un ejemplo de cómo puede trabajarse las técnicas de clustering en un conjunto de datos propio en Python. Para ello, vamos a probar una serie de algoritmos vistos en en la sección 1 y también a analizarlo con algunas técnicas

vistas en la sección 2.

En esta sección se trabajar netamente con el Juputer Notebook correspondiente.

Abrir Jupyter Notebook

```
8.3. Determinar clústers.
```

Ver Vídeo de este apartado

```
8.10. Conjunto de datos.
```

Ver Vídeo de este apartado

```
8.11. Técnicas para determinar clústers.
```

Ver Vídeo de este apartado

```
8.12. Agglomerative.
```

Ver Vídeo de este apartado

```
8.13. k-Means.
```

Ver Vídeo de este apartado

```
8.14. Mean Shift.
```