Métodos no supervisados

Jordi Casas Roma

1 crédito
xxx/xxxxx/xxx

Índice

Introducción

El aprendizaje automático no supervisado se enfoca en el análisis y modelado de datos sin utilizar etiquetas de clase o salidas conocidas. A diferencia del aprendizaje supervisado, donde se requiere un conjunto de datos con ejemplos etiquetados para entrenar al modelo, el aprendizaje no supervisado se basa en la identificación de patrones, estructuras y relaciones inherentes en los datos sin una supervisión explícita. Esta característica lo convierte en un enfoque especialmente útil para abordar problemas en los que las etiquetas o categorías son desconocidas o difíciles de obtener.

El objetivo principal del aprendizaje automático no supervisado es descubrir información oculta y significativa dentro de los datos, a través de técnicas basadas en el agrupamiento (*clustering*). En el agrupamiento, se busca clasificar automáticamente los datos en grupos o clústeres similares según sus características compartidas, lo que facilita la segmentación y comprensión de la estructura subyacente en los datos.

El aprendizaje no supervisado tiene importantes aplicaciones en diversas áreas, como análisis de datos, procesamiento de imágenes y señales, minería de texto, entre otros. Al permitir la exploración y el descubrimiento de conocimiento sin la necesidad de etiquetas previas, este enfoque se convierte en una herramienta para problemas de clasificación no trivial de datos, así la segmentación de datos complejos y la generación de modelos descriptivos y predictivos en contextos donde la supervisión es limitada o costosa de obtener.

Objetivos

En los materiales didácticos de este módulo encontraremos las herramientas indispensables para asimilar los siguientes objetivos:

1. Comprender las características y principales aplicaciones del aprendizaje no supervisado, así como sus limitaciones.
2. Conocer los modelos de aprendizaje no supervisado basados en agrupamiento jerárquico.
3. Conocer el método k-means y sus derivados.
4. Saber cuando y cómo aplicar el algoritmo de agrupamiento Canopy.
5. Conocer el método HDBScan y sus principales características.

1. Contextualización

El aprendizaje no supervisado se basa en la identificación de patrones, estructuras y relaciones inherentes en los datos sin una supervisión explícita. Esta característica lo convierte en un enfoque especialmente útil para abordar problemas en los que las etiquetas o categorías son desconocidas o difíciles de obtener.

El objetivo principal del aprendizaje no supervisado es descubrir información oculta y significativa dentro de los datos, a través de técnicas basadas en el agrupamiento (*clustering*), donde el objetivo es clasificar automáticamente los datos en grupos o clústeres similares según sus características compartidas.

En relación con la notación de este texto, emplearemos la siguiente notación para referirnos al conjunto de entrada de datos:

$$D_{n,m} = \begin{pmatrix} d_1 = & a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ d_2 = & a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_n = & a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{pmatrix}$$

donde:

- n es número de elementos o instancias,
- m el número de atributos del conjunto de datos o también su dimensionalidad,
- d_i indica cada una de las instancias del juego de datos,
- $a_{i,j}$ indica cada uno de los atributos descriptivos.

2. Agrupamiento jerárquico

El agrupamiento jerárquico (*hierarchical clustering* o HC, en inglés) es un método de clasificación no supervisada que busca construir una jerarquía de particiones o clusters en el conjunto de datos.

Hay dos tipos de algoritmos de agrupamiento jerárquico, que aunque pueden proporcionar resultados similares, tienen enfoques inversos:

- El algoritmo aglomerativo (*Agglomerative Hirerchical Clustering, AHC*) parte de una fragmentación completa de los datos, es decir, cada dato tiene su propio grupo, y fusiona los grupos progresivamente hasta que todos los datos pertenecen a un único grupo. Aplica una estrategia *bottom-up*. En este caso hablaremos de clustering o agrupamiento.

- El algoritmo divisivo (*Divisive Hirerchical Clustering, DHC*) parte de un único grupo que contiene todos los datos y lo va dividiendo de forma recursiva hasta obtener un grupo para cada dato. Es decir, aplica una estrategia *top-down*, procediendo de forma inversa a los algoritmos aglomerativos. En este caso hablaremos de segmentación.

Conceptualmente ambos tipos de algoritmos, los aglomerativos y los divisivos, son equivalentes. Sin embargo, los algoritmos aglomerativos son de más fácil construcción, dado que sólo existe un modo de unir dos conjuntos, mientras que existen múltiples formas de separar un conjunto de más de dos elementos.

A través de los algoritmos aglomerativos, es posible construir recomendadores basados en modelo, de modo que para producir una recomendación, sólo es necesario asignar una instancia nueva a uno de los grupos generados por el modelo.

Merece la pena observar que para llevar a cabo la operación de recomendación, tan solo será necesario almacenar la descripción de los grupos generados en el modelo, y no en todo el juego de datos entero, suponiendo así un ahorro notable en recursos.

El principal inconveniente de este tipo de algoritmos es que no son capaces de determinar, por sí mismos, el número idóneo de grupos a generar, sino que es necesario fijarlos de antemano, o bien definir algún criterio de parada en el proceso de construcción jerárquica.

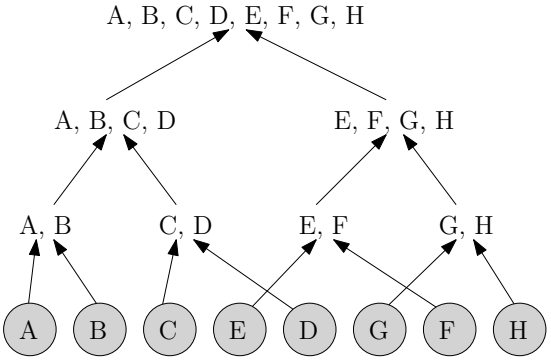
2.1. Dendrogramas

El dendrograma es un diagrama que muestra las agrupaciones (divisiones) sucesivas que genera un algoritmo jerárquico.

Sin duda, es una forma muy intuitiva de representar el proceso de construcción de los grupos. Sin embargo, un punto débil es no ofrecer información sobre la distancia entre los distintos objetos, aunque se puede utilizar el tamaño de las flechas o trabajar con tonalidades de un mismo color para añadir este tipo de información.

En la figura 1 vemos un ejemplo de dendrograma, en el que tenemos una colección de 8 letras y podemos ver como se agrupan progresivamente (algoritmo aglomerativo, estrategia *bottom-up*) o como se dividen recursivamente (algoritmo divisivo, estrategia *top-down*).

Figura 1. Dendrograma aglomerativo. Si las flechas fueran en sentido contrario, tendríamos un dendograma divisivo



El principal interés de los algoritmos jerárquicos es que generan diferentes niveles de agrupamiento, lo que proporciona una información igual o más útil, en algunos casos, que la obtención del agrupamiento definitivo. Es importante destacar que los algoritmos jerárquicos son **voraces** (*greedy*), lo que significa que buscan la mejor decisión local en cada momento, sin asegurar una solución global óptima.

2.2. Algoritmos aglomerativos

En el Algoritmo 1 podemos ver el detalle del pseudo-código del algoritmo aglomerativo. Como se puede observar, el principal punto de discusión es la medida de la distancia entre dos grupos. Hay dos criterios que afectan ésta medida:

1) ¿Cómo se calcula la distancia entre dos puntos? Para éste cálculo podemos utilizar diferentes **métricas de distancia**, como por ejemplo la distancia euclidiana, distancia de Gauss o distancia de Mahalanobis.

2) ¿Como determinamos la distancia mínima entre dos grupos que contienen más de un punto? Cuando tenemos grupos formados por varios puntos, podemos determinar

la distancia entre los dos grupos según diferentes expresiones, que reciben el nombre de **criterios de enlace**.

Para construir un dendograma aglomerativo deberemos, inicialmente establecer con qué métrica de distancia desearemos trabajar y qué criterio de enlace de grupos o segmentos utilizaremos.

El siguiente paso será considerar cada instancia del juego de datos como un grupo o segmento en sí mismo y a partir de aquí empezaremos a calcular distancias entre grupos. En este punto entraremos en un proceso iterativo en el que en cada repetición fusionaremos los grupos más cercanos.

Algoritmo 1 Pseudocódigo del agrupamiento jerárquico aglomerativo (AHC)

Require: D (conjunto de datos individuales)

1: Inicializar $\ell = 0$
2: Inicializar el conjunto de clusters $C_\ell = D$
3: **while** $|C_\ell| > 1$ **do**
4: Encontrar $c_i, c_j \in C_\ell$ tal que la distancia $d(c_i, c_j)$ sea mínima
5: Crear $c_* = c_i \cup c_j$ como padre de los clusters c_i i c_j
6: Actualizar $C_{\ell+1} = C_\ell - \{c_i, c_j\} \cup c_*$
7: $\ell = \ell + 1$
8: **end while**

Algunos de los criterios de enlace más utilizados para medir la distancia entre dos grupos A y B son los siguientes:

1) Enlace simple (*simple linkage*): Tomaremos como criterio la distancia mínima entre elementos de los grupos:

$$d(A, B) = min\{d(a_i, b_j) \,|\, a_i \in A,\, b_j \in B\} \tag{1}$$

Puede ser apropiado para encontrar grupos de forma no elíptica. Sin embargo, es muy sensible al ruido en los datos y puede llegar a provocar el efecto cadena. Éste consiste en el hecho de que puede llegar a forzar la unión de dos grupos que a priori deberían permanecer bien diferenciados, por el hecho de que estos compartan algún elemento muy próximo.

2) Enlace completo (*Complete linkage*): Tomaremos como criterio la distancia máxima entre elementos de los grupos:

$$d(A, B) = max\{d(a_i, b_j) \,|\, a_i \in A,\, b_j \in B\} \tag{2}$$

No produce el efecto cadena, pero es sensible a los valores *outliers*. Sin embargo, suele dar mejores resultados que el criterio simple.

3) Enlace medio (*average linkage*): Tomaremos como criterio la distancia media entre elementos de los grupos:

$$d(A,B) = \frac{1}{|A||B|} \sum_{a_i \in A} \sum_{b_j \in B} d(a_i, b_j) \tag{3}$$

Se trata de un criterio que trata de mitigar los inconvenientes de los dos anteriores, sin acabar de resolverlos por completo.

4) Enlace centroide (*centroid linkage*): La distancia entre dos grupos será la distancia entre sus dos centroides. Presenta la ventaja de que su coste computacional es muy inferior al de los criterios anteriores, de modo que está indicado para juegos de datos de gran volumen.

El enlace simple asume, de forma implícita, que la medida de similitud debería de ser transitiva, lo cual es sólo aplicable en algunos dominios. En general, es preferible escoger el enlace completo, que promueve la formación de clusters compactos y homogéneos, aunque pierde la propiedad de monotonicidad. El enlace medio puede representar un buen compromiso entre ambos.

2.3. Algoritmos divisivos

Aunque el agrupamiento aglomerativo representa la forma más natural de construir un dendograma, su coste computacional es alto. El agrupamiento divisivo suele ser más eficiente a nivel computacional, principalmente porque no se suele llegar al nivel mínimo del dendograma (es decir, una clase por dato), dado que no aporta información útil. Se define un criterio de parada que determinará el nivel o profundidad máximo del árbol.

Algoritmo 2 Pseudocódigo del agrupamiento jerárquico divisivo (DHC)

Require: D (conjunto de datos individuales)
  Crear cluster inicial $C = \{D\}$ y marcarlo como "abierto"
  while (existe un cluster abierto) **do**
    Encontrar el cluster abiero $c^d \in C$
    if (c^d no cumple la condición de parada) **then**
      Dividir $c^d = c_1^d \cup c_2^d \cup \ldots \cup c_m^d$
      for $i = 1, 2, \ldots, m$ **do**
        Crear el cluster c_i^d como descendiente de c^d
      end for
      Marcar c^d como "cerrado"
    else
      Marcar c^d como "cerrado"
    end if
  end while

El psuedocódigo del algoritmo divisivo se puede ver en el Algoritmo 2. Es importante

destacar los dos criterios principales que afectan el diseño de este tipo de algoritmos: el criterio de parada y el criterio de división de particiones.

El criterio último de parada es cuando una partición tiene un único elemento. En este caso, el algoritmo descendería hasta el nivel de granularidad máximo. Pero generalmente este nivel de detalle no es necesario y añade mucha complejidad de cálculo al algoritmo. Existen tres condiciones de parada básicas, basadas en definir:

- La profundidad máxima del árbol resultante.
- El número mínimo de instancias que deberán de tener las particiones. Las particiones con cardinalidad inferior ya no serán procesadas otra vez.
- Un valor máximo para una medida de distancia o disimilitud. Las particiones que no superen este umbral, no serán divididas otra vez, ya que presentan un nivel de cohesión suficiente.

Referente a los criterios de división, es habitual utilizar algoritmos particionales (no jerárquicos) para crear las divisiones de cada partición. En algunos casos, como por ejemplo en el *k-means*, en el que debemos indicar el número de particiones a realizar, puede ser incluso una ventaja, ya que habilita la construcción de árboles del tamaño deseado (por ejemplo, árboles binarios).

2.4. Ejemplo de aplicación

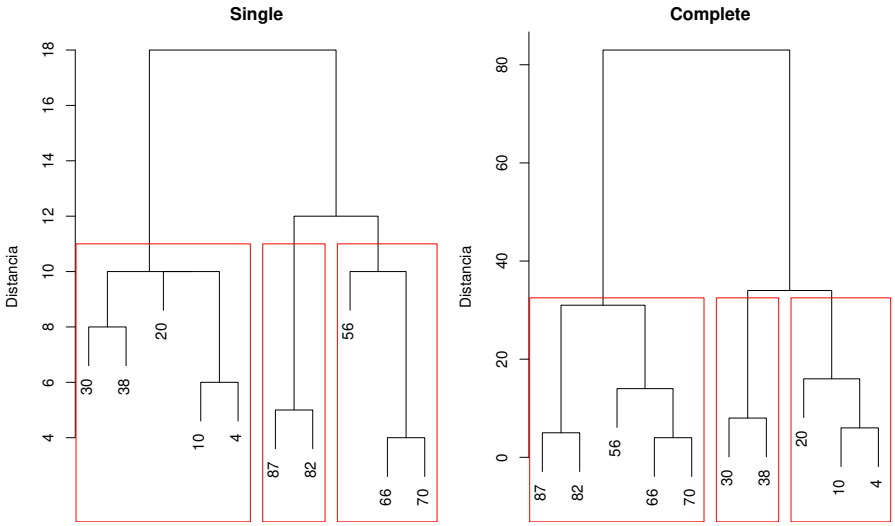
Supongamos que deseamos clasificar un conjunto de 10 números aleatorios entre el 0 y el 100, por ejemplo el conjunto $\{10, 4, 20, 30, 38, 87, 82, 56, 66, 70\}$. La Figura 2 muestra los dendogramas generados por el algoritmo aglomerativo utilizando la distancia euclídea y los criterios de enlace simple y completo. El eje vertical nos muestra la distancia entre los distintos números, dando información sobre el grado de similitud entre ellos. Como ya hemos comentado, una de las principales ventajas de los algoritmos jerárquicos es que permiten diferentes niveles de agrupación o granularidad. Por ejemplo, en la Figura 2 podemos escoger un conjunto de tres clusters, lo que genera los grupos marcados (marco rojo).

2.5. Resumen

Los métodos de agrupamiento jerárquico permiten, no sólo descubrir patrones o grupos de similitud en los datos, sino también organizar los datos para un mejor entendimiento. Además, permiten posponer la decisión del nivel de granularidad deseado, permitiendo modificarlo *a posteriori* según los resultados obtenidos.

El agrupamiento jerárquico se suele utilizar como una forma de modelado descriptivo

Figura 2. Ejemplo de dendrogramas aglomerativos utilizando la distancia euclídea y los criterios de enlace simple y completo



más que predictivo. Aunque esta visión es adecuada para la mayoría de aplicaciones, es importante remarcar que el agrupamiento jerárquico también puede ser empleado como modelo predictivo, asignando las nuevas instancias de datos a las hojas o nodos del árbol de agrupación según la métrica de distancia empleada en la construcción del modelo.

Uno de sus mayores problemas es el coste computacional, aunque éste puede ser reducido fijando una profundidad máxima del árbol (en los algoritmos divisivos) o partiendo de una agrupación inicial de instancias (en el caso de los algoritmos aglomerativos).

3. El método *k-means* y derivados

Iniciaremos este capítulo revisando uno de los métodos de clasificación no supervisada más conocidos y empleados, el algoritmo *k-means*. Continuaremos, en la Sección 3.2, presentando algunos de los métodos derivados del algoritmo *k-means* más importantes. Y finalizaremos este capítulo con la versión difusa del algoritmo *k-means*, conocido como *fuzzy C-means*, en la Sección 3.3.

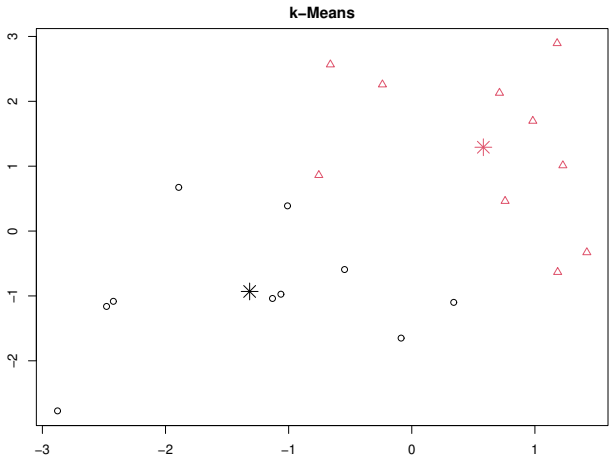
3.1. *k-means*

En esta sección veremos un algoritmo de clasificación no supervisada o segmentación de tipo particional, es decir, que genera una estructura plana. El algoritmo está basado en la partición de un conjunto de n observaciones en k grupos en el que cada observación pertenece al grupo cuya distancia es menor.

Algunas consideraciones importantes respecto al algoritmo *k-means* (o k-medias) son:

- El número de grupos o clusters, denotado como k, debe definirse antes de ejecutar el algoritmo.
- Cada grupo o cluster está definido por un punto, generalmente identificado como el centro y llamado semilla o *centroide* del cluster. Aunque puede recibir otros nombres en implementaciones concretas del algoritmo.

Figura 3. Ejemplo de visualización de los resultados de *k-Means*



A grandes rasgos, el algoritmo funciona en dos fases principales, tal y como podemos ver en el pseudocódigo del Algoritmo 3, que corresponden a las dos tareas siguientes:

1) En primer lugar, la fase de inicialización, identifica k puntos como *centroides* iniciales. Aunque no es necesario que sean puntos del conjunto de datos, si es importante que sean puntos dentro del mismo intervalo.

2) La segunda fase es iterativa, y consiste en

a) asignar a cada *centroide* los puntos del conjunto de datos más próximos, formando k grupos disjuntos,

b) y a continuación, recalcular los *centroides* en base a los puntos que forman parte de su grupo o partición.

Estos dos pasos se repiten hasta que se satisface la condición de parada.

La Figura 3 muestra el resultado de ejecutar el algoritmo sobre un conjunto de veinte puntos en un espacio bidimensional. Las cruces indican los *centroides* de cada partición.

Algoritmo 3 Pseudocódigo del algoritmo *k-means*

Require: D (conjunto de datos) y k (número de clusters)
  Inicializar los clusters vacíos $P = \{p_1, p_2, \ldots, p_k\}$
  Seleccionar los *centroides* iniciales $\zeta_1, \zeta_2, \ldots, \zeta_k$
  while (Condición de parada no satisfecha) **do**
    for all $d_i \in D$ **do**
      Asignar la instancia d_i al cluster $p_j \mid d(d_i, \zeta_j)$ es mínima
    end for
    for all $p_i \in P$ **do**
      Recalcular ζ_i según los valores $d_i \in p_i$
    end for
  end while
    return El conjunto de clusters P

De los pasos anteriores se desprenden cuatro criterios importantes para definir el comportamiento del algoritmo:

- ¿Cómo podemos inicializar los *centroides*?
- ¿Cómo podemos calcular la distancia entre cada punto d_i y los *centroides*?
- ¿Cómo podemos recalcular los *centroides* a partir de las instancias que forman su grupo o partición?
- ¿Cómo podemos establecer la condición de parada del algoritmo?

3.1.1. Inicialiación de los *centroides*

Un primer enfoque, simple y ámpliamente usado, consiste en inicializar los *centroides* con k puntos aleatorios del conjunto de datos. Es decir, $\zeta_i = rand(d_j) \mid d_j \in D, i = \{1, \ldots, k\}$.

Una de las principales consecuencias derivadas de este tipo de inicialización de *centroides* es que implica que el proceso de clustering será **no determinista**. Es decir, diferentes ejecuciones del algoritmo pueden conducir a diferentes modelos y, lógicamente, a diferentes niveles de calidad del resultado. Esta característica, que en principio podría parecer un inconveniente importante, puede ser superada gracias a la simplicidad y efectividad del método, que permite ejecutar el algoritmo múltiples veces con distintos *centroides* y escoger el mejor resultado.

El algoritmo *k-means* utiliza la inicialización de *centroides* aleatoria, pero existen otras aproximaciones más complejas que son utilizadas por métodos derivados del *k-means*. Por ejemplo, se pueden seleccionar los *centroides* iniciales a partir de la distribución de probabilidades de las instancias de D, intentando cubrir todo el rango de valores de los datos, especialmente las zonas con mayor concentración de instancias.

3.1.2. Cálculo de la distancia

El algoritmo *k-means*, generalmente, utiliza la distancia euclídea. Aún así, es posible utilizar algún otro tipo de métrica de similitud.

3.1.3. Recálculo de los *centroides*

El valor de cada *centroide* es calculado como la media (*mean*) de todos los puntos que pertenecen a este segmento (de aquí el nombre del algoritmo, *k-means*). Por lo tanto, este algoritmo sólo es aplicable a atributos continuos. En caso de tener atributos no continuos en el conjunto de datos, deberemos aplicar una transformación previa.

3.1.4. Criterio de parada

La **convergencia** del algoritmo es la condición de parada natural y última. Ésta se alcanza cuando no hay cambios en el recálculo de los *centroides* durante una iteración completa, provocando que no haya alteraciones en la distribución de las instancias en las distintas particiones o clusters. Es decir, se llega a una situación de estabilidad en la distribución de las instancias. Esta condición se puede garantizar después de un número finito de iteraciones, dependiendo de la métrica empleada en el cálculo de la distancia y el recálculo de los *centroides*.

En la práctica, no es necesario que el método converja, y generalmente es suficiente cuando se tiene pequeños cambios en la pertenencia de las instancias en los distintos grupos o particiones, es decir, cuando estamos próximos a una situación de convergencia.

3.1.5. Criterios para seleccionar un valor k

Una de las principales características de *k-means* es que se le deben indicar el número de particiones o clusters a identificar. Aunque esto le permite mantener un nivel de simplicidad y eficiencia elevados, puede ser considerado un inconveniente importante. Identificar el número correcto de particiones (k) no es una tarea elemental, e incluso en muchas ocasiones no se puede extraer directamente del conocimiento del dominio. Aún así, en muchos casos el rango de valores de k a considerar no es muy grande, con lo cual se pueden probar diferentes valores y seleccionar el que proporcione mejores resultados.

Un criterio es minimizar la suma de residuos cuadrados (*residual sum of squares*, RSS), es decir, la suma de las distancias de cualquier vector o instancia a su *centroide* más cercano. Este criterio busca la creación de segmentos lo más compactos posibles.

Otro criterio podría ser el de maximizar la suma de distancias entre segmentos, por ejemplo entre sus centros. En este caso estaríamos priorizando tener segmentos los más alejados posible entre sí, es decir, tener segmentos lo más diferenciados posible.

Como sucede en muchos problemas de maximización o minimización, es fácil intuir que el algoritmo no siempre alcanzará un óptimo global, puede darse el caso de que antes encuentre un óptimo local. Además, también es posible que el algoritmo no sea capaz de encontrar ningún óptimo, bien porque simplemente el juego de datos no presente ninguna estructura de segmentos, bien porque no se haya escogido correctamente el número k de segmentos a construir.

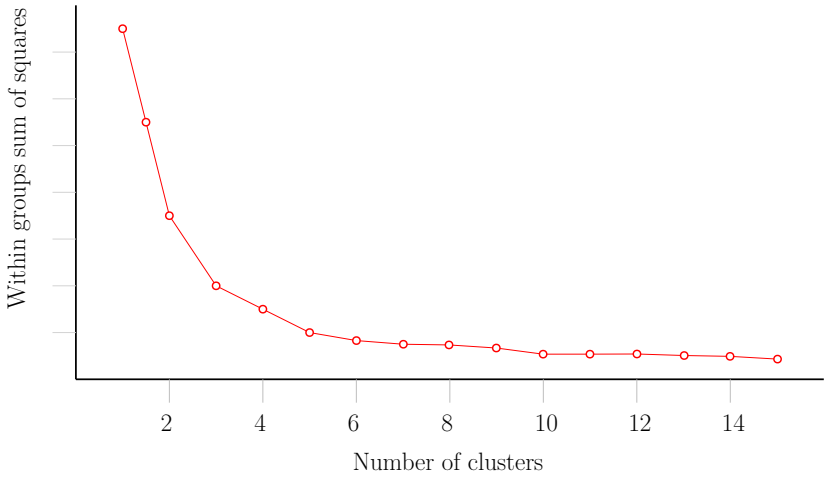
Los criterios anteriores (minimización de distancias intra grupo o maximización de distancias inter grupo) pueden usarse para establecer un valor adecuado para el parámetro k. Valores k para los que ya no se consiguen mejoras significativas en la homogeneidad interna de los segmentos o la heterogeneidad entre segmentos distintos, deberían descartarse.

Por ejemplo, en la Figura 4 podemos ver como evoluciona la métrica de calidad respecto al número de particiones creadas. En concreto, observamos cómo a partir de 5 segmentos, la mejora que se produce en la distancia interna de los segmentos ya es insignificante. Este hecho debería ser indicativo de que 5 segmentos es un valor adecuado para k.

Otra aproximación para solucionar este problema, consiste en permitir que el número k se modifique durante la ejecución del algoritmo. En este caso, se inicia el algoritmo con un valor proporcionado de k, pero éste se puede modificar (incrementar o disminuir) según ciertos criterios. Por ejemplo:

- Si existen dos particiones con los centros muy juntos, quizás es mejor unir ambas particiones y reducir el número de particiones.

Figura 4. Criterio para seleccionar un valor k



- Si existe una partición con un nivel de diversidad muy elevado, se puede dividir ésta en dos nuevas particiones, incrementando por tanto el valor de k.

3.1.6. Resumen

El algoritmo *k-means* es, en esencia, un algoritmo de clasificación o segmentación, lo que permite identificar estructuras dentro de un conjunto de datos sin etiquetas o clases. Aún así, también es ámpliamente usado para tareas de predicción, donde se utilizan los *centroides* para clasificar nuevas instancias que no estaban en el conjunto de datos original. Una vez construido el modelo, debe preservarse el valor de los *centroides*, que serán imprescindibles para poder realizar el análisis predictivo.

El algoritmo *k-means* es muy usado como primer intento de clasificación en conjuntos de datos continuos. Por el contrario, no es una buena opción en conjuntos de datos categóricos.

3.2. Métodos derivados de *k-means*

El uso de la distancia media para el cálculo de los *centroides* proporciona un modelo simple y eficiente, y además proporciona unos resultados muy fácilmente interpretables. Aún así, tiene sus limitaciones. Por ejemplo, cuando los datos están distribuidos de forma asimétrica y contienen valores extremos (*outliers*) en algunos atributos.

3.2.1. *k-medians*

El algoritmo k-mediana (*k-medians*) es una variación del método anterior, donde se sustituye el cálculo basado en el valor medio, por el valor de la mediana. Aunque la obtención del valor de la mediana requiere mayor complejidad computacional, es preferible en determinados contextos.

Al igual en el caso del *k-means*, éste método puede implementarse mediante distintas métricas de similaridad, aunque existe el riesgo de perder la garantía de convergencia. Una de las métricas de distancia más empleadas en este algoritmo es la distancia de Manhattan.

3.2.2. *k-medoids*

El método *k-medians* presenta mejores resultados que el *k-means* cuando las distribuciones son asimétricas o existen valores *outliers*. Aún así, *k-medians* no garantiza que los centros sean similares a alguna instancia del conjunto de datos. Esto se debe a que, a menos que los atributos sean independientes, los vectores de atributos medianos pueden no parecerse a ninguna instancia existente del conjunto de datos ni del dominio completo.

El algoritmo *k-medoids* propone el recáculco de los centros a partir de instancias que presentan un valor de disimilitud mínimo respecto a las demás instancias de la partición o cluster. La identificación de estos puntos, conocidos como *medoids*, es computacionalmente más costosa que las aproximaciones vistas anteriormente.

Aunque esta aproximación sea considerablemente más costosa, a nivel de cálculo, que *k-means* o *k-medians*, es preferible en algunos dominios donde puede ser interesante que los puntos de centro sean puntos del conjunto de datos. Además, este método es particularmente robusto al ruido y a los valores *outliers*.

3.3. *Fuzzy C-Means*

El algoritmo C-Medios difuso (*Fuzzy C-Means*) puede ser interpretado como una generalización de los algoritmos vistos anteriormente. La diferencia fundamental, desde el punto de vista operacional, es que este algoritmo construye una partición difusa, mientras que los algoritmos vistos anteriormente construyen una partición nítida o disjunta del conjunto de datos.

La idea que subyace en los métodos difusos se basa en la idea de que no todas las instancias tienen la misma relación de pertenencia con el grupo al que pertenecen. Existen puntos que se encuentran muy próximos al centro de una partición, de los cuales podemos decir que están fuertemente asociados con la partición en cuestión. Otros, por el contrario, se encuentran más próximos al borde que separa dos o más particiones, y por lo tanto, podemos decir que su grado de asociación es más débil que en el caso anterior. Resultaría lógico poder decir que estos segundos mantienen cierta relación con múltiples particiones, aunque pueda haber una partición principal. Esta es la idea principal subyacente en los métodos de segmentación o clustering difuso.

3.3.1. Funcionamiento del método

En este caso, el algoritmo es prácticamente idéntico al Algoritmo 3 visto anteriormente, pero con algunas diferencias que detallamos a continuación:

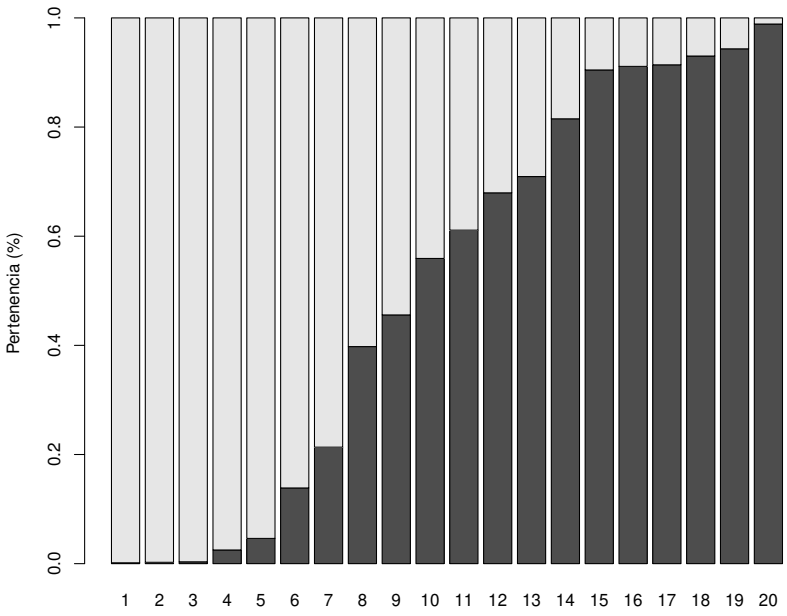
- Cada instancia d_i tiene asociado un vector de k valores reales que indican el grado de pertenencia de esta instancia a cada uno de los k grupos o particiones. El grado de pertenencia dependerá de la distancia que lo separa con el *centroide* de dicha partición. En general, se normaliza dicho valor para que la suma de todos los valores sea igual a 1. De esta forma, se puede ver el valor como la probabilidad de pertenencia a cada una de las k particiones.

- Al inicio del algoritmo, se asigna el grado de pertenencia de cada instancia a cada grupo aleatoriamente, y después se calculan los *centroides* a partir de estos datos.

- El recálculo de los centroides está ponderado por el grado de pertenencia de cada punto a la partición correspondiente.

La Figura 5 muestra una posible visualización de los resultados de aplicar el algoritmo *Fuzzy C-Means* sobre un conjunto de veinte puntos en el espacio bidimensional. Para cada punto (eje horizontal) se muestra la probabilidad de pertenencia a cada una de las particiones. Se han ordenado los datos según la probabilidad de pertenencia al primer grupo. Es interesante ver que, aunque se pueden diferenciar las dos clases claramente, algunos datos están fuertemente asociados con una única partición, mientras que otros mantienen relaciones de pertinencia un poco más débiles con ambas clases.

Figura 5. Ejemplo de visualización de los resultados de *Fuzzy C-Means*



3.3.2. Resumen

Esta variante, o generalización, presenta las mismas ventajas e inconvenientes que el algoritmo *k-Means*. Por un lado, la simplicidad y eficiencia son sus principales puntos fuertes. Por el otro, el no determinismo y la excesiva dependencia de la métrica utilizada para calcular las distancias son sus posibles limitaciones o inconvenientes.

4. Algoritmo de agrupamiento Canopy

El algoritmo de agrupamiento Canopy (*Canopy clustering algorithm*) (McCallum et al., 2000) es un método no-supervisado de pre-clustering. El pre-clustering se puede entender como el conjunto de tareas de procesamiento previo a un proceso de agrupamiento o clustering, con el objetivo de acelerar el proceso de clustering, especialmente con grandes conjuntos de datos.

Lectura complementaria

A. McCallum, K. Nigam, L. H. Ungar (2000). "Efficient clustering of high-dimensional data sets with application to reference matching". In Proc. of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, New York, NY, USA, pp. 169-178. https://doi.org/10.1145/347090.347123

4.1. El concepto de pre-clustering

Este algoritmo es utilizado, generalmente, como pre-procesamiento previo a algoritmos de clustering, como por ejemplo el clustering jerárquico aglomerativo (AHC) o el *k-Means*. En conjuntos grandes de datos puede resultar impracticable aplicar directamente los algoritmos de clustering, especialmente en el caso de agrupamiento jerárquico aglomerativo.

La idea que subyace a esta técnica es que podemos reducir drásticamente el número de cálculos que requeridos en el proceso de clustering posterior, introduciendo un proceso previo de generación de grupos superpuestos (*canopies*) a partir de una métrica más sencilla de calcular (*cheapest metric*). De esta forma, sólo calcularemos distancias con la métrica inicial, más estricta y pesada en cálculos, para los puntos que pertenecen al mismo *canopy*.

Podríamos resumirlo diciendo que previamente, mediante una métrica simple, decidimos qué puntos están definitivamente lejos y en consecuencia, para estos puntos alejados ya no valdrá la pena malgastar más cálculos con una métrica más exigente.

4.2. Funcionamiento del método

El método de agrupamiento Canopy divide el proceso de segmentación en dos etapas:

- En una primera etapa, usaremos una métrica sencilla en cálculos, con el objetivo de generar los *canopies* o subgrupos superpuestos de instancias. Cada instancia pueda pertenecer a más de un *canopy* y, a su vez, todas las instancias tienen que pertenecer, al menos, a un *canopy*.

- En una segunda etapa, utilizaremos un método de segmentación tradicional, como por ejemplo el agrupamiento jerárquico aglomerativo (AHC) o el método *k-means*, pero lo haremos con la restricción de no calcular la distancia entre los puntos que

no pertenecen al mismo *canopy*.

El método, descrito en el Algoritmo 4, parte del conjunto de datos y dos valores límite (*threshold*): T_1 que indica la distancia máxima de la periferia (*the loose distance*) y T_2 que indica la distancia máxima del núcleo (*the tight distance*), donde $T_1 > T_2$. A partir del conjunto inicial de instancias o puntos, se escoge un punto (aleatoriamente) para formar un nuevo *canopy*. A continuación, se calcula la distancia entre este punto y los demás puntos del conjunto de datos, utilizando una métrica más sencilla de calcular – que hemos llamado *cheapest metric*. Se incluyen en el mismo *canopy* todos los puntos que tengan una distancia inferior al *threshold* T_1. Además, los puntos que tengan una distancia inferior a T_2, son eliminados del conjunto de datos. De esta forma, se excluye a estos puntos para formar un nuevo *canopy* y ser el centro de éste. Este proceso se repite de forma iterativa hasta que no quedan puntos en el conjunto de datos.

Algoritmo 4 Pseudocódigo del algoritmo Canopy

Require: D (conjunto de datos), $T_1, T_2 \mid T_1 > T_2$ (distancias)
  while ($D <> \emptyset$) **do**
    Elegir un d_i aleatoriamente e inicializar nuevo *canopy* $\omega_v = \{d_i\}$
    Eliminar d_i del conjunto de datos $D = D - \{d_i\}$
    for all $d_j \in D$ **do**
      Calcular la distancia $\Delta_{i,j} = d(d_i, d_j)$ con la métrica sencilla
      if ($\Delta_{i,j} < T_1$) **then**
        Añadir d_j al *canopy*, i.e. $\omega_v = \omega_v \cup \{d_j\}$
      end if
      if ($\Delta_{i,j} < T_2$) **then**
        Eliminar d_j del conjunto de datos $D = D - \{d_j\}$
      end if
    end for
  end while

Para facilitar la comprensión, vamos a situarnos en los dos casos extremos:

1) Supongamos, como consecuencia de la primera etapa, que nuestro universo de puntos cae por completo en un solo *canopy*. Entonces el método de segmentación por *canopies* sería exactamente igual al del método de segmentación tradicional seleccionado, es decir, AHC o *k-means*.

2) Supongamos que como resultado de la primera etapa, generamos *canopies* relativamente pequeños y con muy poca superposición. En este caso, al aplicar la técnica tradicional sólo dentro de cada *canopy*, habremos ahorrado un gran número de cálculos.

4.3. Ejemplo de aplicación

Veamos de una forma gráfica cómo funciona el algoritmo:

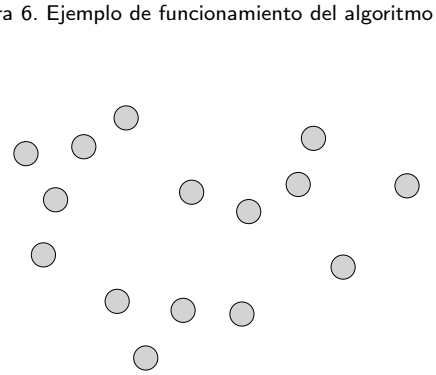
1) En primer lugar, debemos obtener el conjunto de datos D y establecer una valores límite (*threshold*) T_1 y T_2 adecuados para el conjunto de datos con el que estamos trabajado. La Figura 6 (a) muestra un pequeño conjunto de datos, que utilizaremos para ejemplificar.

2) A continuación escogemos un punto (aletoriamente) $d_i \in D$ y lo definimos como el centro del primer *canopy*. Calcularemos la distancia a todos los demás puntos del conjunto de datos D, utilizando la métrica más sencilla de calcular (*cheapest metric*). Por lo tanto, podremos fácilmente definir los puntos que están a distancia menor que T_2, los que están entre T_2 y T_1 y los que están más allá de T_1, tal y como muestra la Figura 6 (b).

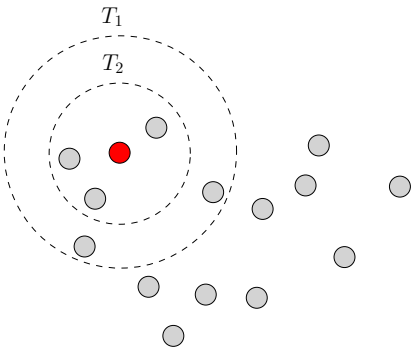
3) Eliminaremos del conjunto de datos D los puntos que están a distancia menor que T_2, ya que éstos no podrán formar ningún *canopy* nuevo ni formar parte del centro (Figura 6 (c)).

4) Repetiremos este proceso forma iterativa hasta que no queden puntos en el conjunto de datos. Es decir, seleccionamos otro punto para forma un nuevo *canopy* y volvemos a aplicar los mismos pasos, manteniendo fijo el valor de T_1 y T_2 (Figura 6 (d)).

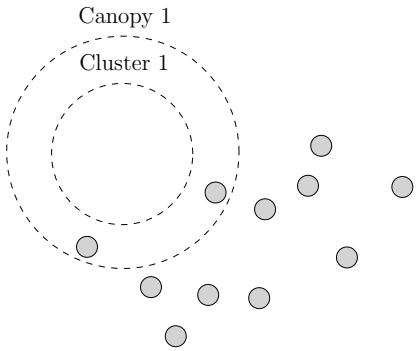
Figura 6. Ejemplo de funcionamiento del algoritmo Canopy



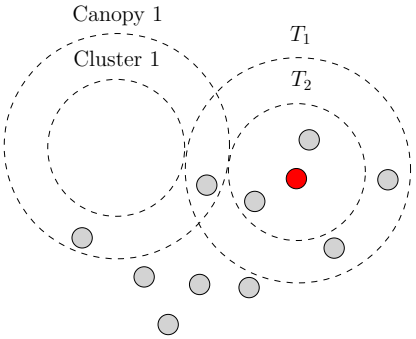
(a) Conjunto de datos inicial



(b) Seleccionar del punto inicial y las distancias T_1 y T_2



(c) Eliminar los puntos con distancia inferior a T_2



(d) Seleccionar otro punto inicial y repetir el proceso

4.4. Resumen

El cuarto bloque de este libro se centra en los métodos de aprendizaje no supervisado, es decir, aquellos métodos que utilizan conjuntos de datos no etiquetados. En concreto,

se verán algoritmos jerárquicos, el método *k-means* y sus derivados y el concepto y proceso de pre-clustering empleando el algoritmo Canopy.

El algoritmo de agrupamiento Canopy, o en general los métodos de pre-clustering, presentan la principal ventaja de reducir el número de casos de datos de entrenamiento que se deben comparar en cada paso del método de agrupamiento que se utilice, ya sea *k-means*, agrupamiento jerárquico u otros métodos de agrupamiento no supervisado.

Estos métodos son especialmente indicados en el caso de lidiar con grandes conjuntos de datos, incluyendo conjuntos con millares o millones de instancias y con un número elevado de atributos a considerar.

Aunque se pueda pensar que el uso de este tipo de estrategias de pre-clustering puedan reducir el rendimiento posterior de los métodos de agrupamiento, no se observa un deterioro en los clusters resultantes e incluso, en algunos casos, los resultados mejoran con el proceso de pre-clustering.

5. HDBScan

Lectura complementaria

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu (1996). "A density-based algorithm for discovering clusters in large spatial databases with noise". In Proc. of the Int. Conf. on Knowledge Discovery and Data Mining (KDD'96), pp 226-231. AAAI Press.

El HDBScan (*Hierarchical Density-Based Spatial Clustering of Applications with Noise*) es un algoritmo de agrupamiento no supervisado que se basa en la densidad de los datos para identificar clústeres de diferentes formas y tamaños en un conjunto de datos. A diferencia de algunos algoritmos de agrupamiento, como por ejemplo el $k-means$, el algoritmo HDBScan no requiere el número de clústeres de antemano. Esta característica lo hace especialmente útil en conjuntos de datos donde el número de clústeres es desconocido o puede variar.

El algoritmo HDBScan utiliza dos conceptos clave para identificar los clústeres:

- Densidad: Mide la cantidad de puntos cercanos a un punto dado. Cuanto mayor sea la densidad de un punto, más cercanos están sus vecinos.

- Distancia de alcance (*Reachability Distance*): Es la distancia mínima necesaria para alcanzar un punto desde otro, pasando por una secuencia de puntos vecinos con una densidad mayor o igual. La distancia de alcance combina la información de densidad y distancia para determinar si dos puntos pertenecen al mismo clúster.

5.1. Funcionamiento del método

En esta sección veremos el funcionamiento del algoritmo HDBScan. En primer lugar, veremos los pasos que sigue el algoritmo para determinar los puntos núcleo y los clústeres.

Dado un conjunto de datos D con n instancias y una distancia de vecindad ϵ, el algoritmo HDBScan realiza los siguientes pasos:

1) Calcular la matriz de distancia entre todos los puntos del conjunto de datos D.

2) Para cada punto d_i en D, encontrar todos sus vecinos que estén a una distancia menor o igual a ϵ.

3) Calcular la densidad de cada punto, es decir, el número de vecinos que tiene.

4) Encontrar los puntos núcleo, que son aquellos puntos cuya densidad supera un umbral predefinido mínimo.

5) Construir el grafo de conectividad entre los puntos núcleo, donde dos puntos núcleo están conectados si su distancia de alcance es menor o igual a ϵ.

6) Utilizar el algoritmo de la jerarquía de enlace para identificar los clústeres de manera jerárquica, considerando las conexiones en el grafo de conectividad.

7) Aplicar un proceso de unión y división para obtener los clústeres finales, agrupando los puntos que comparten conexiones en el grafo de conectividad.

A continuación, para formalizar el funcionamiento del algoritmo, presentamos el pseudocódigo de HDBScan, en el Algoritmo 5.

La función del procedimiento "BuscarVecinos" es encontrar los puntos vecinos de un punto dado dentro de un conjunto de datos. Esta función es crucial para determinar la densidad local alrededor de cada punto y, en última instancia, para identificar los puntos núcleo y construir los clústeres. Esta función toma como entrada un punto p del conjunto de datos D y un valor de distancia ϵ. Luego, recorre todos los puntos del conjunto de datos D y verifica si la distancia entre el punto p y cada punto q es menor o igual a ϵ. Si la distancia cumple con esta condición, se agrega el punto q al conjunto de vecinos de p, representado por la variable $neighbors$.

La función del procedimiento "ExpandirClúster" es expandir un clúster a partir de un punto núcleo identificado previamente. Esta función es crucial para agregar todos los puntos vecinos cercanos al clúster y, de esta manera, construir los clústeres completos. Esta función toma como entrada un punto p que es un punto núcleo identificado previamente, el conjunto de puntos vecinos cercanos $neighbors$ al punto p, el clúster $cluster$ en construcción, el conjunto de datos D, el valor de distancia ϵ, el número mínimo de puntos para formar un clúster minPts y el conjunto de puntos no visitados $unvisited$.

Finalmente, el procedimiento principal ("HDBScan") implementa el algoritmo de agrupamiento HDBScan en sí mismo. Este procedimiento es el punto de entrada del algoritmo y coordina todas las etapas principales del proceso de agrupamiento. El procedimiento toma como entrada el conjunto de datos D, el valor de distancia de vecindad ϵ y el número mínimo de puntos para formar un clúster minPts.

5.2. Características principales

Las principales ventajas o puntos fuertes del algoritmo HDBScan son las siguientes:

- Identificación de clústeres de formas y tamaños variables: HDBScan es capaz de encontrar clústeres de diferentes formas y tamaños en el conjunto de datos, lo que lo hace más flexible y versátil en comparación con algunos algoritmos tradicionales que asumen clústeres con formas y tamaños predefinidos.

- Tolerancia al ruido y puntos aislados: HDBScan puede identificar puntos que no pertenecen a ningún clúster (ruido) y puntos aislados que tienen densidades muy bajas.

Algoritmo 5 Pseudocódigo del algoritmo HDBScan

```
procedure HDBSCAN(D, ε, minPts)
    clusters ← ∅                                  ▷ Conjunto de clústeres vacío
    unvisited ← D                                 ▷ Conjunto de puntos no visitados
    while unvisited no está vacío do
        Escoge un punto p de unvisited
        unvisited ← unvisited \ {p}
        neighbors ← BuscarVecinos(p, D, ε)
        if |neighbors| ≥ minPts then
            cluster ← NuevoClúster()
            ExpandirClúster(p, neighbors, cluster, D, ε, minPts, unvisited)
            clusters ← clusters ∪ {cluster}
        end if
    end while
    return clusters
end procedure
procedure EXPANDIRCLÚSTER(p, neighbors, cluster, D, ε, minPts, unvisited)
    Agregar p al clúster cluster
    for cada punto q en neighbors do
        if q no ha sido visitado then
            unvisited ← unvisited \ {q}
            neighbors_q ← BuscarVecinos(q, D, ε)
            if |neighbors_q| ≥ minPts then
                neighbors ← neighbors ∪ neighbors_q
            end if
        end if
        if q no está en ningún clúster then
            Agregar q al clúster cluster
        end if
    end for
end procedure
procedure BUSCARVECINOS(p, D, ε)
    neighbors ← ∅
    for cada punto q en D do
        if distancia(p, q) ≤ ε then
            neighbors ← neighbors ∪ {q}
        end if
    end for
    return neighbors
end procedure
```

- No requiere especificar el número de clústeres: A diferencia de otros algoritmos de agrupamiento que requieren el número de clústeres como parámetro de entrada, HDBScan determina automáticamente el número de clústeres en función de la densidad y la conectividad de los datos.

- Eficiente y escalable: El algoritmo utiliza estructuras de datos eficientes y técnicas de indexación espacial para acelerar el proceso de búsqueda de vecinos, lo que lo hace adecuado para conjuntos de datos grandes.

Por el contrario, sus principales debilidades o puntos mejorables son los siguientes:

- No es completamente determinista: los puntos borde que son alcanzables desde más de un clúster pueden etiquetarse en cualquiera de estos. Afortunadamente, esta situación no es usual, y tiene un impacto relativamente pequeño sobre los resultados.

- La calidad de los resultados de DBSCAN depende, en gran medida, de la noción de distancia. La distancia más usada es la distancia euclidiana.

5.3. Resumen

En resumen, el algoritmo HDBScan es una poderosa herramienta de agrupamiento no supervisado que se basa en la densidad de los datos para identificar clústeres de diferentes formas y tamaños sin la necesidad de especificar previamente el número de clústeres. Su capacidad para manejar ruido y puntos aislados, junto con su eficiencia en conjuntos de datos grandes, lo convierten en una opción atractiva para diversas aplicaciones de análisis y exploración de datos.

Resumen

En este módulo didáctico hemos presentado las principales características del aprendizaje automático no supervisado, así como algunos de los principales modelos no supervisados.

En la primera sección de este texto hemos presentado los algoritmos de agrupamiento jerárquico, haciendo especial émfasis en los algoritmos aglomerativos y divisivos.

A continuación, en la segunda sección del módulo, hemos revisado uno de los algoritmos más conocidos para tareas de agrupamiento no supervisado, como es el $k-means$ y sus derivados. También hemos introducido una variante de este modelo para entornos difusos, como es el *fuzzy C-Means*.

En la sección cuarta hemos introducido el concepto de pre-clustering, así como el algoritmo Canopy, que permite realizar dicha tarea.

Finalmente, hemos introducido el algoritmo HDBScan en la sección quinta y última de este módulo didáctico.

Glosario

Algoritmo voraz (*greedy*) Algoritmo que en cada paso toma la decisión óptima en ese momento, sin tener en cuenta lo que dicha decisión implicará en la futura toma de decisiones. Los algoritmos voraces no suelen proporcionar la solución óptima a un problema, aunque pueden proporcionar soluciones razonables a un coste reducido.

Centroide Punto central de un conjunto de puntos que conforman una región. Diferentes algoritmos (p.e. k-means) usan el centroide como representante de los elementos que caen en dicha región.

Dendrograma Representación gráfica de las decisiones tomadas para construir un modelo desde la perspectiva de los datos, mostrando las agrupaciones existentes en cada paso.

Bibliografía

Géron, Aurélien (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Editio*n. O'Reilly Media, Inc.

Cichosz, Pawel (2015). *Data mining algorithms. Explained using R.* Wiley.

J. Gironés Roig, J. Casas Roma, J. Minguillón Alfonso, R. Caihuelas Quiles (2017). *Minería de datos: Modelos y algoritmos.* Barcelona: Editorial UOC.