IDENTIFICACIÓN DE PATRONES CIRCULARES

María Casasola Calzadilla
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
UVUS: marcascal2@alum.us.es

Contacto: maariacasasola@gmail.com

Emilia Coleto Alcudia
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
UVUS: emicolalc@alum.us.es

Contacto: emiliacoleto@gmail.com

***Resumen*— El trabajo se centra en detectar en una nube de puntos, aquellos que forman circunferencias, indicando de una manera óptima el radio de la circunferencia, su centro y los puntos que la delimitan. El algoritmo en el que se basa el proyecto es el algoritmo de k-medias estudiado en teoría. Este algoritmo permite actualizar los *clústeres* en cada iteración, devolviendo un resultado que, o bien cumpla con el criterio de parada, o bien llegue a un número máximo de iteraciones.**

Conforme el trabajo avanzaba, los métodos heurísticos y las conclusiones iban mejorando. Desde un principio, se trabajó sobre un ejemplo sencillo, en concreto sobre el ejemplo 1 proporcionado por el profesorado, y una vez comprobada la poca eficacia del algoritmo inicial (con iteraciones limitadas), se decidió optar por no fijar el número de iteraciones desde el principio, sino implementar una condición de parada. Gracias a este criterio se optimizó el algoritmo bastante en cuanto a las pruebas con el primer ejemplo. Sin embargo, las pruebas con el ejemplo 2 proporcionado, el cual cuenta con circunferencias concéntricas, no dieron un buen resultado. Se concluyó, tras esto, que la condición de parada no estaba completa aún.

Para optimizar aún más el método, se pensó en incluir, en el criterio de parada, una cota para los grados de pertenencia de cada punto a la circunferencia del *clúster* con el que se encuentra asociado. Lo que mejoraba la representación final del resultado para una mejor percepción de las circunferencias, pero las circunferencias concéntricas seguían sin estar bien localizadas.

Tras muchas actualizaciones del algoritmo inicial, intentando optimizar aún más en la salida, cambiando la manera de enfocar el problema inicialmente y modificando significativamente la condición de parada, se concluye con la idea de condicionar al método desde el principio, realizando ciertas actualizaciones para guiar mejor al algoritmo en la primera iteración. Gracias a estos cambios, el algoritmo funciona en los 5 ejemplos utilizados para las pruebas, siendo el ejemplo con peor resultado el de las circunferencias concéntricas. Por ello, se optó finalmente por contar con una condición de parada que optimizase el resultado, pero también con un número fijo de iteraciones para evitar bucles infinitos en el algoritmo.

Concluyendo, el algoritmo desarrollado tiene una buena eficiencia, sin embargo, el hecho de seguir contando con un número fijo de iteraciones asegura que no siempre saldrá un resultado óptimo.

I. VOCABULARIO

Algoritmo: Conjunto ordenado de operaciones sistemáticas que permite hacer un cálculo y hallar la solución de un tipo de problemas.

Clúster: Conjunto, grupo o cúmulo

Clustering con incertidumbre: El clustering consiste en la agrupación automática de datos. Al ser un aprendizaje no-supervisado, no hay una respuesta correcta. Esto hace que la evaluación de los grupos identificados sea un poco subjetiva.

Cota de pertenencia: Indica el mínimo grado de pertenencia que deben tener todos los puntos de los clústeres respecto a su circunferencia.

Eficiencia: Capacidad para realizar o cumplir adecuadamente una función.

Grado de pertenencia: Marca el grado en que un punto forma parte de una circunferencia.

Heurística: Método basado en la experiencia que puede utilizarse como ayuda para resolver problemas de diseño.

Inteligencia Artificial: Es la inteligencia llevada a cabo por máquinas. Sistemas que piensan como humanos, automatizan actividades como la toma de decisiones, la resolución de problemas y el aprendizaje.

Iteraciones: Repetir varias veces un proceso con la intención de alcanzar una meta deseada, objetivo o resultado. Cada repetición del proceso también se le denomina una "iteración", y los resultados de una iteración se utilizan como punto de partida para la siguiente iteración.

K-medias: Método de agrupamiento, que tiene como objetivo la partición de un conjunto de n observaciones en k grupos en el que cada observación pertenece al grupo cuyo valor medio es más cercano. Es un método utilizado en minería de datos.

Resultado óptimo: Si en él no es posible mejorar la situación de alguien sin empeorar simultáneamente la de otros.

II. Introducción

Este proyecto, para la asignatura de Inteligencia Artificial del curso 19/20, trata sobre la detección de grupos de puntos que formen circunferencias haciendo uso de un algoritmo de clustering con incertidumbre adaptado al objetivo.

Se decidió interpretar el problema bajo la hipótesis de que los puntos están repartidos de forma más o menos uniforme a lo largo de toda trayectoria de la circunferencia, por lo que la estimación de centro y radio de los clústeres que simulan las circunferencias a las que pertenecen los puntos, se actualizan en cada iteración usando los puntos que hay asociados a cada clúster, haciendo uso del baricentro para la actualización del centro y la media aritmética para la del radio.

Por otra parte, la pertenencia o no de un punto a un clúster se estima con el uso de un conjunto de métodos entre los cuáles el primero calcula la distancia del punto a cada clúster para indicar inversamente que mientras menor sea la distancia, mayor será el grado de pertenencia, el cual es calculado en otro método. Y finalmente, los valores son normalizados, para lo que decidimos hacer uso de un reparto inverso.

Una vez abarcados los aspectos iniciales del algoritmo, se estudiaron los detalles para refinarlo. Es decir, no solo vale con diferenciar un número de clústeres (k) con sus puntos correspondientes, sino que, además, estos puntos deben formar una circunferencia, como se explica anteriormente. Para ello, una vez alcanzado el final del algoritmo k-medias empleado, antes de devolver el resultado, se realiza una comprobación que asegura si los puntos están repartidos de forma más o menos equitativa a lo largo de una circunferencia y si todos los clústeres cuentan con, al menos, 2 puntos. Aunque, la comprobación con más peso es la de los grados de pertenencia de cada punto del clúster con la cota estimada, que asegura un resultado óptimo en caso de parada antes del fin de iteraciones.

Para finalizar, se completó el algoritmo con la inicialización de las circunferencias de forma aleatoria seguido de una optimización para situarlas de acuerdo con los grados de pertenencia máximos de cada una, mejorando la eficiencia desde el inicio de nuestro método.

III. Preliminares

A. *Métodos empleados*

Se ha empleado un algoritmo de clustering para agrupar los puntos en clústeres en forma de anillos. Por tanto, se está haciendo uso del aprendizaje automático no supervisado, pues no conocemos a priori a qué clúster pertenece cada dato. Este tipo de programas a partir de una serie de datos mejoran sus elecciones a través de la experiencia ganada al realizar iteraciones y así poder resolver problemas de una manera óptima.

En concreto se ha utilizado el algoritmo de K-medias. Esta técnica se utiliza para dividir variables en grupos. Se pretende relacionar cada variable con su grupo afín a través de cálculos con los datos de entrada intentando en cada iteración minimizar el error cometido hasta cumplir la condición de un determinado criterio de parada.

En esta práctica, en concreto las variables son puntos repartidos en forma circular y los grupos a los que van a pertenecer son clústeres con forma circular con puntos asociados a ellos. A través del algoritmo, se pretende que los puntos vayan aprendiendo en cada iteración cual es la circunferencia a la que pertenece calculando el grado de pertenencia de cada uno de ellos con cada clúster. Cuanto mayor sea el grado de pertenencia menos error se comete, por tanto, se pretende alcanzar el máximo grado de pertenencia asumiendo el ruido de los puntos que no están perfectamente contenidos en el perímetro de la circunferencia. El algoritmo acaba cuando se ha obtenido el mayor grado de pertenencia, los clústeres no varían y tienen al menos dos puntos asociados.

Los datos iniciales están contenidos en un archivo *CSV* que es interpretado por *pandas*. Como de las circunferencias resultantes solo se sabe cuántas son, se generan las iniciales de manera aleatoria. El centro de cada una de ellas se elige de forma aleatoria entre el máximo y el mínimo valor de la coordenada x de los puntos de entrada para asegurarse de que no se va a alejar demasiado de los grupos y el radio se genera con un número entero aleatorio del 1 al 3.

En cuanto a la asociación de puntos con circunferencias se ha utilizado un método para calcular los grados de pertenencia. Esto se ha conseguido calculando la distancia inversa del punto a cada uno de los k clústeres y posteriormente normalizarlos. Entonces los puntos se asocian obteniendo los grados de pertenencia mayores en cada uno de los puntos y estos se añaden al clúster de mayor preferencia, eliminando así los puntos que tenía anteriormente.

Una vez añadidos los puntos, se actualiza la circunferencia porque como los puntos han variado, su posición y tamaño también lo han hecho. Se calcula el nuevo baricentro con todos los puntos pertenecientes a dicho clúster y con una media de todas las distancias del nuevo centro a cada uno de estos puntos de obtiene el nuevo radio.

El último método empleado se utiliza al final de cada iteración. Si las circunferencias obtenidas en la anterior iteración y la actual son iguales, se comprueba que los grados de pertenencia sean máximos, si no se han conseguido, entonces hace falta seguir iterando.

IV. Metodología

La estructura del algoritmo principal, el k-medias, sigue el Pseudocódigo 1:

Algoritmo k-medias
Entrada:
- Un conjunto de puntos
- Un conjunto de circunferencias
- Un número de iteraciones
- Umbral de pertenencia

Salidas:
- Un conjunto de circunferencias con puntos asociados

Algoritmo:
1. Inicializar las circunferencias
2. Mientras iteración < número de iteraciones
    a. Calcular el grado de pertenencia
    b. Actualizar las circunferencias
    c. Si se cumple el criterio de parada
        i. Cortar el bucle
    d. Incrementar iteración en 1
3. Devolver circunferencias

Pseudocódigo 1. Algoritmo k-medias

El algoritmo recibe como parámetro de entrada un conjunto de puntos, otro de circunferencias y un número máximo de iteraciones que se realizarán antes de devolver la solución en caso de que no se cumpla el criterio de parada. El parámetro *umbral de pertenencia*, indica la cota mínima de pertenencia que deben tener todos los puntos de un clúster con su clúster correspondiente.

Al comenzar la primera iteración, se calcula el grado de pertenencia de las circunferencias generadas aleatoriamente y se inicializan. Las nuevas circunferencias entran en el bucle donde se calcula el grado de pertenencia de cada punto con cada circunferencia creada y se actualizan sus centros y radios en función de los puntos escogidos para cada una de ellas.

En cada iteración se comprueba si se cumple el criterio de parada o hay que seguir iterando en busca de una mejor solución incrementando en uno el contador.

Una vez se ha cumplido el criterio de parada o se ha llegado a la iteración final, se devuelve la lista con las circunferencias resultantes y los puntos agrupados.

A continuación, la explicación de cada uno de los conjuntos de métodos auxiliares utilizados para la resolución del problema:

1) Lectura de datos de entrada

Los parámetros de entrada son un conjunto de puntos en un archivo *CSV* y un número k de circunferencias. En este método se crean los datos de tipo Punto y Circunferencia.

Con ayuda de *pandas* se lee el archivo *CSV* y se añaden todos los puntos a una lista.

Para las circunferencias se obtiene el valor de x máximo y mínimo de todos los puntos y en el área resultante (área que contiene todos los puntos del problema) se escogen k puntos para ser los centros de los clústeres. El radio se obtiene de manera aleatoria desde 1 hasta 3.

Una vez creadas las listas se devuelve el resultado, que se utilizará como parámetros de entrada para el algoritmo principal.

2) Generar gráfico

Al acabar el algoritmo principal se genera un gráfico que muestra la solución al problema. Se representan en un diagrama todos los puntos y las circunferencias resultantes de aplicar k-medias. Cada una de las circunferencias está representada de un color diferente.

3) Calcular grado de pertenencia

Toma como parámetros de entrada la lista de puntos y la de circunferencias representativas de cada clúster y genera un diccionario que cuenta con todos los puntos como claves, asociados cada uno a una lista que contiene el índice de la circunferencia con la que tiene un grado de pertenencia máximo y dicho grado.

Para ello, se hace referencia a los métodos que calculan la distancia de cada punto a cada una de las k circunferencias, sus inversas y los valores normalizados de estas distancias, es decir, todos los grados de pertenencia de un mismo punto tienen que sumar 1. Lo que aprovechamos para indicar que, si un punto tenía distancia 0 con una circunferencia, su grado de pertenencia con esa circunferencia sería 1 y con las demás 0, normalizando y maximizando equitativamente los valores.

Como ya se ha mencionado en apartados anteriores, la distancia inversa ayuda a reflejar el propósito de los grados de pertenencia, indicando así que cuanto más grande sea la distancia entre el punto y la circunferencia, menor será el grado de dicho punto a esa circunferencia. La normalización se consigue gracias a un reparto inverso de las cantidades invertidas obtenidas.

Una vez obtenidas las listas con los k valores de pertenencia de cada punto, se guarda con cada uno su máximo valor y el índice indicando a que circunferencia se refiere.

En el pseudocódigo 2, se muestra el pseudocódigo que se sigue para realizar el cálculo del grado de pertenencia.

Cálculo del grado de pertenencia
Entrada:
- Un conjunto de puntos
- Un conjunto de k circunferencias

Salidas:
- Un mapa con un punto de clave relacionado con su máximo grado de pertenencia

Algoritmo:
1. Para cada punto:
   a. Calcular la distancia inversa (punto, circunferencias)
   b. Normalizar los valores
   c. Agregar el máximo valor al mapa junto con su punto
2. Devolver el mapa

Pseudocódigo 2. Cálculo del grado de pertenencia

El cálculo de la distancia inversa y su normalización se detallan en el pseudocódigo 3 y pseudocódigo 4 respectivamente.

Cálculo de la distancia inversa
Entrada:
- Un punto
- Un conjunto de k circunferencias

Salidas:
- Una lista con k valores

Algoritmo:
1. Para cada circunferencia:
   a. Calcular la distancia del punto al centro – radio
   b. Almacenar la distancia en una lista
2. Para cada distancia:
   a. Si la distancia == 0 entonces devuelve una lista de ceros con un 1 en la posición de dicha distancia
   b. Sino entonces devuelve una lista con distancia = 1/distancia
3. Devolver la lista de distancias

Pseudocódigo 3. Cálculo de las distancias inversas

Normalización de las distancias
Entrada:
- Una lista de distancias

Salidas:
- Una lista con k valores normalizados

Algoritmo:
1. Suma de todas las distancias
2. Si suma == 1 entonces devuelve la lista
3. Sino entonces para cada distancia:
   a. Si distancia no == 0:
      i. n = 1/(suma*1/d)
      ii. Almacenar n en la lista
   b. Sino entonces devolver distancias
   c. Devolver la lista

Pseudocódigo 4. Normalización de las distancias inversas

4) Crear clústeres

En cada iteración se llama al método crear clústeres, el cual crea los clústeres siguiendo el mapa de grados de pertenencia máximos y la lista de las circunferencias.

Así pues, por cada circunferencia se crea un nuevo clúster y se hace uso del método obtener máxima pertenencia, que devuelve la lista de clústeres que se le pasa como parámetro, con las listas de puntos vacías, con todos los puntos que pertenecen a cada clúster ya incluidos en la lista de puntos de cada uno. Para lo cual se ayuda del mapa de máximos grados de pertenencia.

Tras cada iteración, las circunferencias son modificadas, actualizando su centro y su radio, por lo que cada vez que se llame a este método, el mapa de pertenencia máxima y las circunferencias de los parámetros de entrada habrán variado.

5) Actualizar clúster

Al crear los clústeres en cada iteración, las circunferencias deben ser actualizadas dependiendo de los puntos con los que estén relacionadas en cada clúster.

Para ello, se recorre la lista de clústeres creada y actualizamos cada uno de forma que primero se recalcula el centro y luego el radio progresivamente.

La actualización del centro se realiza con la medida del baricentro que se calcula con todos los puntos incluidos en el clúster.

Una vez realizada esta actualización, se vuelve a calcular el mapa de pertenencia máxima con el nuevo centro y se crean los clústeres una vez más ya con una nueva circunferencia y el nuevo mapa.

Gracias a la actualización del centro y la lista de clústeres realizada antes de la modificación del radio, se asegura el avance del algoritmo, ya que, para calcular dicha actualización, no solo necesitamos el nuevo centro, sino también la nueva lista de puntos que corresponden a ese centro. El cálculo del radio se realiza con una media aritmética entre las distancias de todos los puntos del clúster al centro, menos en caso de tener una lista de puntos vacía o con un solo puntos, que se genera aleatoriamente un radio entre 1 y 3 unidades.

Por último, se actualiza el radio y se devuelve el clúster actualizado.

6) Criterio de parada

Al final de cada iteración se debe comprobar el buen avance del algoritmo y si ha llegado a un resultado óptimo para devolver. Esto se consigue con el método que comprueba si se cumplen los criterios de parada.

El algoritmo k-medias cuenta con dos variables que almacenan en cada vuelta la lista de clústeres anterior y la lista actualizada en dicha vuelta. Así se asegura poder comprobar uno de los criterios de parada principales, si las circunferencias de los conjuntos de puntos han sido modificadas. Además, la solución no puede contar con un clúster que contenga una lista con menos de dos puntos.

Finalmente, una vez que las circunferencias comienzan a estabilizarse, para comprobar que se trata de un buen resultado, se compara cada grado de pertenencia de cada punto con su circunferencia correspondiente. Por tanto, para poder devolver la solución, todos los puntos deben superar un valor de cota de pertenencia (fijado en 0.95 que asegura una buena solución).

V. RESULTADOS

El algoritmo ha sido testeado con diferentes ejemplos. Cada uno de los ejemplos recibe diferentes datos con el objetivo de probar cada una de las posibilidades de solución.

Con el objetivo de sacar una mejor conclusión de la fiabilidad del algoritmo se han establecido tres categorías de testeo diferenciadas por una diferente cota mínima de pertenencia. En cada todas las categorías se contemplan los mismos ejemplos, pero con diferentes cotas.

Cada uno de los bloques cuenta con 4 ejemplos de diferentes resultados.

1) La primera de las pruebas tiene como parámetros de entrada 8 puntos y una k igual a 2 con un número máximo de 200 iteraciones. La solución óptima sería dos circunferencias concéntricas y cada una de ellas con 4 puntos.

   Como se observa en la imagen 1 si se establece una cota de grado de pertenencia mínimo de 0.95 se obtiene la solución es óptima antes de realizar todas iteraciones ya que se cumple el criterio de parada y todos los puntos pertenecen a las circunferencias resultantes.

Cota de pertenencia para condición de parada: 0.95

```
cls = k_medias(puntos,circunferencias,200)
generar_grafico(puntos,cls)
```

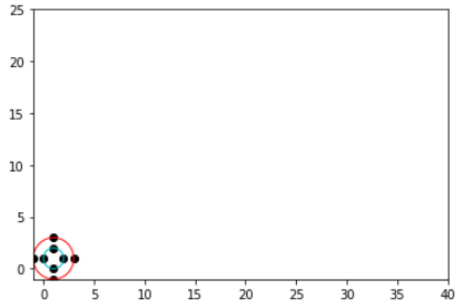

Figura 1. Ejemplo 3, cota alta

En la imagen 2 se muestran los resultados obtenidos variando la cota a 0.75. La solución óptima también se devuelve antes de llegar al máximo de iteraciones, devolviendo el resultado exacto del problema.

Cota de pertenencia para condición de parada: 0.75

```
[56]: cls = k_medias(puntos,circunferencias,200)
      generar_grafico(puntos,cls)
```

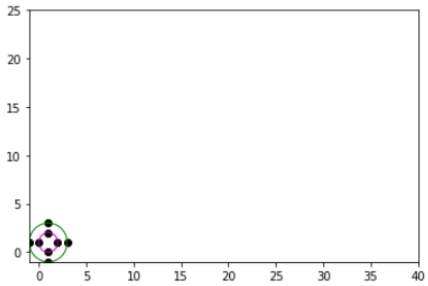

Figura 2. Ejemplo 3, cota media

Por último, se vuelve a modificar la cota a un mínimo de 0.5. Como muestra la imagen 3, no se obtiene el mismo resultado que en los casos anteriores. El programa para antes de las 200 iteraciones, pero el resultado no es el más exacto debido a la cota establecida.

Cota de pertenencia para condición de parada: 0.5

```
cls = k_medias(puntos,circunferencias,200)
generar_grafico(puntos,cls)
```

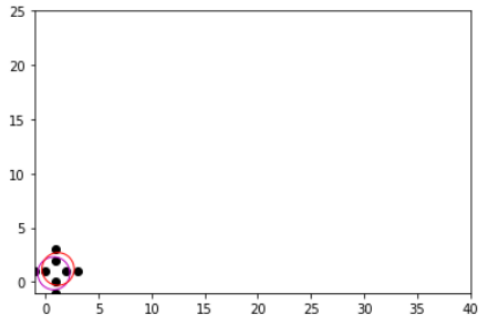

Figura 3. Ejemplo 3, cota baja

2) Se testea un ejemplo con k igual a 3 y puntos repartidos de manera no uniforme en forma de circunferencia conteniendo determinados arcos más puntos que otros y con 200 iteraciones máximo. La solución óptima debería devolver las 3 circunferencias con todos los puntos situados justamente en ellas

En la imagen 4 se muestra la solución con una cota de 0.95. El algoritmo finaliza antes de alcanzar el máximo de iteraciones porque se cumple el criterio de parada. Las circunferencias son muy similares a la solución óptima, pero aún así no ha conseguido alcanzar el máximo.

Cota de pertenencia para condición de parada: 0.95

```
cls = k_medias(puntos,circunferencias,200)
generar_grafico(puntos,cls)
```

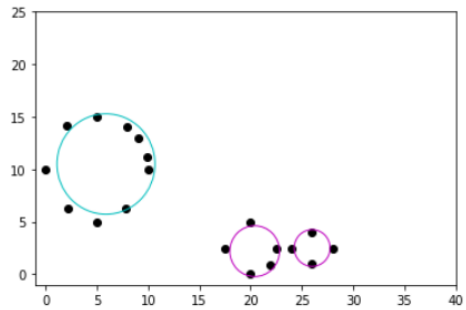

Figura 4. Ejemplo 4, cota alta

En la imagen 5 se representa el resultado con una cota de 0.75. Se ha obtenido una solución bastante similar a la solución anterior. El algoritmo ha finalizado puesto que se ha cumplido el criterio de parada, las circunferencias no han variado con respecto a la iteración anterior.

Cota de pertenencia para condición de parada: 0.75

```
cls = k_medias(puntos,circunferencias,200)
generar_grafico(puntos,cls)
```

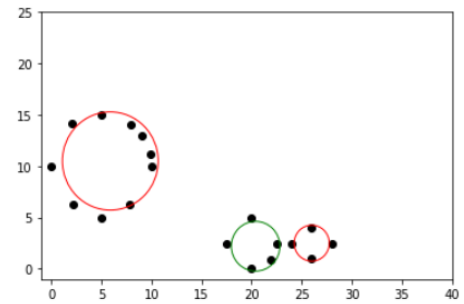

Figura 5. Ejemplo 4, cota media

Por último, se genera el mismo problema en la imagen 6 con una cota de 0.5. Otra vez se obtiene la misma solución que con las otras cotas antes de llegar a las 200 iteraciones. Es una solución válida, aunque no sea la óptima es muy aproximada a ella conteniendo el resultado el mismo agrupamiento de puntos y variando solo en la posición de las circunferencias.

Cota de pertenencia para condición de parada: 0.5

```
cls = k_medias(puntos,circunferencias,200)
generar_grafico(puntos,cls)
```

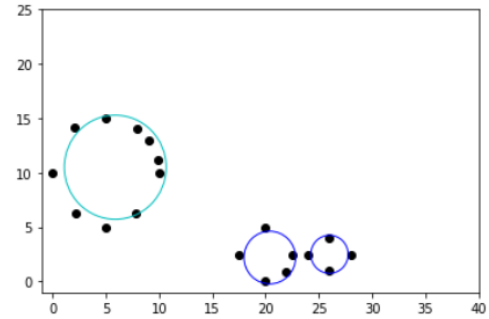

Figura 6. Ejemplo 4, cota baja

3) Los puntos de la siguiente prueba forman 5 circunferencias de distinto tamaño y posición, dos de ellas intersectan. El resultado más exacto debería contener las 5 circunferencias posicionadas exactamente encima de cada uno de los puntos.

   Con la cota establecida a 0.95, se consigue el resultado mostrado en la imagen 7. El algoritmo para antes de completar todas las iteraciones cumpliendo entonces el criterio de parada. Como se puede observar, no es el resultado óptimo pues las circunferencias que intersectan no clasifican correctamente sus puntos. Aún así es un resultado muy aproximado al exacto.

Cota de pertenencia para condición de parada: 0.95

```
cls = k_medias(puntos,circunferencias,200)
generar_grafico(puntos,cls)
```

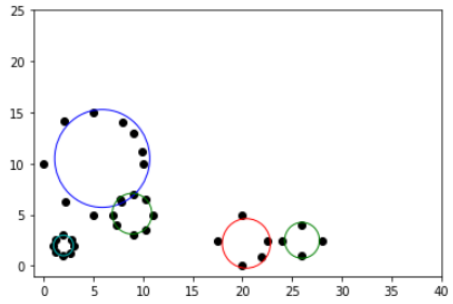

Figura 7. Ejemplo 5, cota alta

En la imagen 8 se muestra la solución obtenida con la cota en 0.75. Se obtiene una solución peor que en la imagen 7 puesto que la cota es menor. El máximo problema se encuentra en las dos circunferencias que intersectan.

Cota de pertenencia para condición de parada: 0.75

```
cls = k_medias(puntos,circunferencias,200)
generar_grafico(puntos,cls)
```

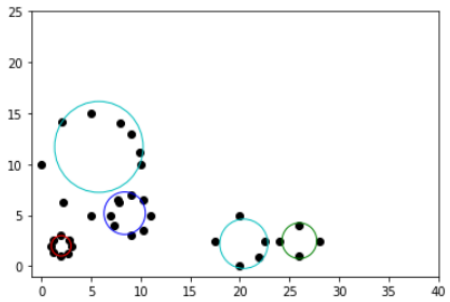

Figura 8. Ejemplo 5, cota media

Por último, se hace una prueba con la cota situada a 0.5. El resultado que se obtiene al cumplirse el criterio de parada, representado por la imagen 9, es prácticamente igual al anterior.

Cota de pertenencia para condición de parada: 0.5

```
cls = k_medias(puntos,circunferencias,200)
generar_grafico(puntos,cls)
```

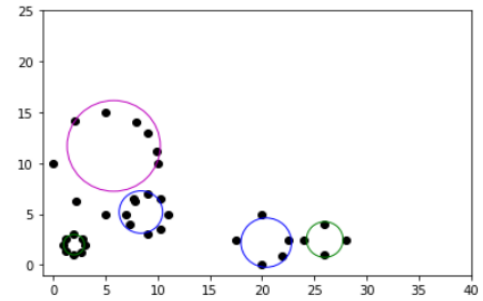

Figura 9. Ejemplo 5, cota baja

4) El último caso debe devolver 3 circunferencias donde dos de ellas son concéntricas con un número de 200 iteraciones máximas.

Se establece la cota mínima de grado de pertenencia a 0.95. Esta vez el algoritmo se detiene cuando se han realizado las 200 iteraciones y no porque se haya cumplido el criterio de parada. Como se muestra en la imagen 10, la circunferencia que se encuentra sola encuentra su mejor grado de pertenencia, pero las concéntricas al tener entre ellas una distancia tan pequeña, el algoritmo las divide en dos circunferencias que se tocan.

Cota de pertenencia para condición de parada: 0.95

```
cls = k_medias(puntos,circunferencias,200)
generar_grafico(puntos,cls)
```

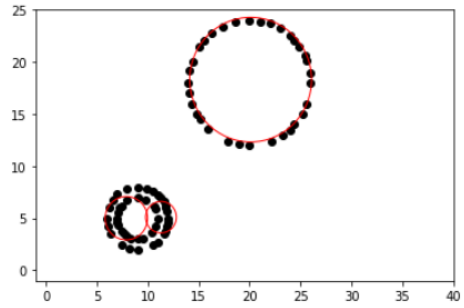

Figura 10. Ejemplo 2, cota alta

En la imagen 11 se puede observar el resultado obtenido cuando el algoritmo ha alcanzado las 200 iteraciones, pues no ha encontrado la solución óptima. Se consigue aproximadamente la misma solución que estableciendo la cota a 0.95.

Cota de pertenencia para condición de parada: 0.75

```
cls = k_medias(puntos,circunferencias,200)
generar_grafico(puntos,cls)
```

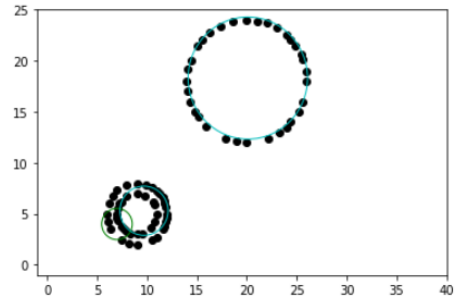

Figura 11. Ejemplo 2, cota media

.

Finalmente, se establece para el mismo ejemplo una cota de 0.5. En este caso el algoritmo finaliza debido al cumplimiento del criterio de parada, pero el resultado es mucho peor que en los dos anteriores casos ya que la cota mínima es muy baja como se puede observar en la imagen 12.

Cota de pertenencia para condición de parada: 0.5

```
cls = k_medias(puntos,circunferencias,200)
generar_grafico(puntos,cls)
```

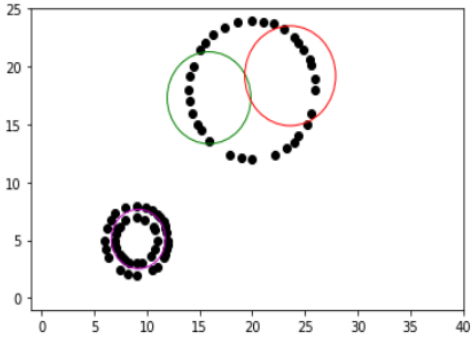

Figura 12. Ejemplo 2, cota baja

VI. Conclusiones

Tras realizar las pruebas pertinentes para el proyecto, se puede concluir que el algoritmo tiene una buena efectividad, aunque no la máxima, pero abarca en gran medida el problema abordado.

Gracias a los resultados obtenidos con los distintos ejemplos mostrados, se observa que cuanto mayor es la cota mínima de pertenencia a superar, mayor es el tiempo de ejecución, pero mejor es la efectividad del algoritmo.

Cabe destacar que, de no ser por el uso de un número de iteraciones máximo, para ciertos ejemplos, el algoritmo no hubiese acabado nunca, ya que establecer un grado de pertenencia alto a superar por todos los puntos de un clúster puede provocar un bucle infinito. Por ello, la estimación de un número de iteraciones máximo aun habiendo implementado un método como criterio de parada, nos asegura que, independientemente de que sea una solución óptima o no, el algoritmo siempre devolverá un resultado de la manera más actualizada posible.

Disminuir la cota mínima de pertenencia implica una mejora en tiempo de ejecución, pero las soluciones devueltas distan bastante de las soluciones que se esperan, aunque en algunos casos los resultados obtenidos son los resultados esperados. Pero para las circunferencias concéntricas o que intersectan, este mínimo empeoraría la resolución del algoritmo, ya que en estos casos la precisión para diferenciar puntos cercanos, pero de diferentes clústeres, es crucial.

Por tanto, aunque el método cuenta con un número de vueltas finito, si en algún ejemplo, el algoritmo cumple las condiciones finales antes de acabar el bucle de iteraciones finitas, este puede devolver la solución obtenida sin necesidad de esperar a que acabe el número de vueltas, y para asegurar que la solución sea efectiva, la cota mínima para el criterio de parada es alta (0.95).

VII. ANEXO. MARCO TEÓRICO

La idea básica de los algoritmos de partición es dividir la base de datos en k grupos que son representado por la gravedad del grupo (k-medias) o por un objeto representativo del grupo (k-medoid).

Para llevar a cabo este proyecto, como se ha comentado en todos los apartados anteriores, se ha realizado clustering con el algoritmo k-medias que divide puntos en grupos de forma circular tratando de conseguir grupos con el grado de pertenencia máximo.

En este apartado se documenta otra de las formas en la que el problema podría haber sido abordado: mediante clustering basado en densidad.

La herramienta Clustering basado en densidad se trata de la detección de en qué áreas existen concentraciones de puntos y dónde están separados por áreas vacías o con escasos puntos. Los puntos que no forman parte de un clúster se etiquetan como ruido. El criterio de agrupamiento local, por tanto, es la densidad de puntos, regiones densas de puntos separadas de otras regiones densas por regiones poco densas.

Habitualmente, estos tipos de algoritmos identifican clústeres de formas arbitrarias, aunque también podría estudiarse enfocado a problemas como el que se plantea para este proyecto. Aunque este tipo de clustering es más recomendado para puntos que no siguen ninguna forma geométrica, se podría haber utilizado puesto que es más preciso que k-medias.

Los métodos basados en la densidad son interesantes de estudiar ya que, como indica el nombre, los clústeres agrupan los puntos por densidad y la facilidad para identificar masas de puntos aumenta, pudiendo considerar dicha opción para la inicialización de clústeres en el proyecto actual, por ejemplo. Determinando desde el principio la modificación de las posiciones de las circunferencias iniciales, generadas aleatoriamente, según las densidades de puntos encontradas en los datos de entrada, se podría conducir al algoritmo a la solución de manera más rápida.

Existen distintos algoritmos basados en densidad, entre ellos se van a estudiar de manera poco profunda tres tipos, DBSCAN, OPTICS y DENCLUE.

1) DBSCAN

El algoritmo DBSCAN (Density Based Spatial Clustering of Applications with Noise) es un algoritmo de densidad simple que implementa la noción de densidad por medio de un procedimiento basado en centro. Es uno de los métodos más rápidos existentes. El primer paso consiste en definir los parámetros de entrada para que el resultado sea lo más exacto posible:

- Épsilon define la distancia máxima para que dos puntos sean vecinos (radio del clúster).
- El número mínimo de puntos para que el área se considere densa.

Algoritmo DBSCAN
Entrada:
- El número mínimo de puntos
- Épsilon
- Conjunto de puntos

Salidas:
- Un conjunto de clústeres con puntos asociados

Algoritmo:
1. Para cada punto:
   a. Encontrar sus vecinos
   b. Si número de vecinos < mínimo de puntos:
      i. Marcar el punto como ruido
   c. Si no:
      i. Crea clúster con punto
      ii. Añade sus vecinos al clúster
   d. Incrementar iteración en 1
2. Devolver clústeres

Pseudocódigo 5. Algoritmo DBSCAN

Una vez definidos los parámetros de entrada se comienza a iterar con el algoritmo definido en el pseudocódigo 5. Se elige un punto que no haya sido tratado todavía y si no es un punto de ruido y se crea un clúster a partir de este punto añadiendo todos sus vecinos al clúster también. Iterando así hasta que se consigue un clúster densamente conectado

Identifica la densidad según el número de puntos en un radio específico, marcado por el parámetro de entrada épsilon, que indica la distancia máxima entre dos puntos 'vecinos', es decir, del mismo clúster. La densidad de cada punto se estima contando el número de puntos, incluido él, que se encuentran a una distancia no mayor que épsilon del punto, dependiendo de dicho valor, cada punto es clasificado como central (core), frontera (border) o ruido. Al final de cada iteración se elimina el ruido del conjunto de datos y se agrupan los datos restantes.

La eficiencia de este algoritmo es de orden: O(n log n). El comportamiento de DBSCAN depende de la elección de sus parámetros (épsilon y MinPts), si estos parámetros no se fijan adecuadamente se obtendrán clústeres poco útiles. Un procedimiento ampliamente empleado para fijar el valor de estos parámetros es usar el concepto de k-distancia. La k-distancia de un punto se define como la distancia al k-ésimo punto más cercano. Tras calcular la k-distancia de cada punto, se ordenan estas de menor a mayor y se muestran en un gráfico, el valor en el que se produce un cambio drástico de la curva es un valor apropiado para épsilon.

La diferencia entre este algoritmo y k-medias se basa principalmente en que en DBSCAN no hay que definir un número de clústeres finales, el propio algoritmo lo consigue a través de los parámetros iniciales, por eso es tan importante

elegir épsilon y el mínimo de puntos de manera correcta. Si épsilon es muy pequeña, entonces cada punto será un clúster diferente, y, sin embargo, si es muy grande, todos los puntos pertenecerán a un solo clúster.

2) OPTICS

OPTICS (Ordering Points To Identify the Clustering Structure) ordena los datos de acuerdo a la estructura de sus clústeres. La eficiencia de este método es también de orden O(n log n) aunque es mejor que DBSCAN con clústeres de distinta densidad.

También se agregan un par de definiciones:

- Distancia del núcleo: mínimo épsilon para convertir un punto distinto en un punto central, dado un parámetro MinPts finito.
- Distancia de accesibilidad: la distancia de accesibilidad de un objeto P con respecto a otro objeto O es la distancia más pequeña desde O si es un objeto central. Tampoco puede ser menor que la distancia del núcleo de O.

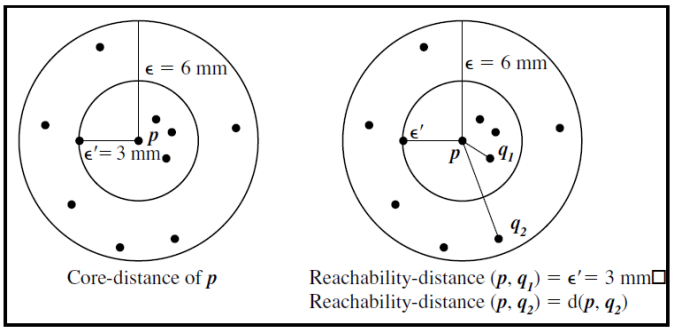

Figura 13. Parámetros

Como podemos observar, la distancia de p a q1 es menor que la distancia central de p ($\mathcal{E}'$), por tanto en el método accesibilidad-distancia(), que comprueba si ésto ocurre, sustituye el valor de la distancia por el valor de la distancia central del puntos, ya que la distancia de accesibilidad nunca puede ser menor que la distancia central.

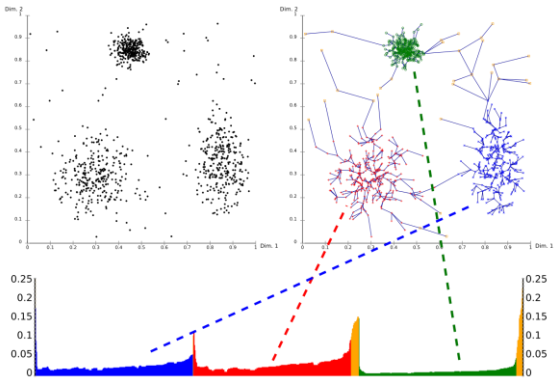

Figura 14. Ejemplo 1 de OPTICS

Aquí se muestra un ejemplo resuelto sencillo del algoritmo.

Primero, nuestro algoritmo, comienza calculando las distancias centrales en todos los puntos de datos en el conjunto. Luego se recorre todo el conjunto de datos y se actualizan las distancias de alcance, procesando cada punto solo una vez. Solo se actualizarán las distancias de alcance de los puntos que se mejorarán y que aún no se han procesado. Esto se debe a que cuando procesamos un punto, se almacena su orden y su distancia de accesibilidad y el siguiente punto de datos elegido para procesar será el que tenga la distancia de accesibilidad más cercana. Así es como el algoritmo mantiene los clústeres cerca uno del otro en el orden de salida.

El siguiente paso será extraer las etiquetas de clúster reales del gráfico. La forma más común de hacerlo es buscando "valles" en la trama, utilizando mínimos y máximos locales. Aquí podrían entrar en juego algunos parámetros más, dependiendo del método utilizado.

Vea a continuación una comparación de algunos datos de muestra generados y las etiquetas ópticas resultantes y el gráfico de accesibilidad. Los puntos coloreados son aquellos identificados como grupos, mientras que los grises representan ruido.

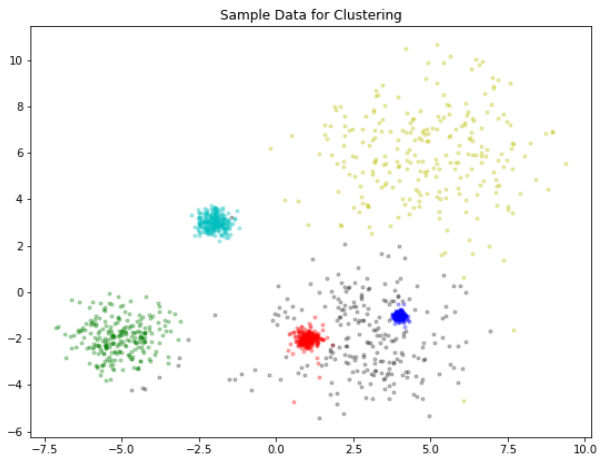

Figura 15. Ejemplo 2 de OPTICS

Observe que hay una buena cantidad de puntos identificados como puntos de ruido en este ejemplo generado. Tienen densidades similares a las del grupo amarillo, pero no se reconocen en esta extracción porque se enfoca en separar las regiones más densas. Otros métodos de extracción de grupos también podrían utilizarse en este caso.

Aunque no es un nuevo algoritmo de agrupación, OPTICS es una técnica muy interesante sobre la que no se ha encontrado una gran cantidad de discusión. Sus ventajas incluyen encontrar densidades variables, así como muy pocos ajustes de parámetros.

3) DENCLUE

Por último, DENCLUE (DENsity-based CLUstEring), cuenta con una base matemática sólida y funciona bien en conjuntos de datos con ruido. Permite una descripción compacta de clústeres de formas arbitrarias en conjuntos de datos con muchas dimensiones. Su principal ventaja es que es más rápido que otros algoritmos (p.ej. DBSCAN), pero una de sus peores características es que necesita un número elevado de parámetros, y como sabemos, una de las principales características de los algoritmos de clustering es el uso del menor número de parámetros posibles, ya que un número elevado de parámetros afectaría considerablemente a la dimensionalidad del problema.

El algoritmo DENCLUE emplea un modelo de clúster basado en la estimación de densidad del núcleo. Un grupo se define por un máximo local de la función de densidad estimada. Los puntos de datos se asignan a los grupos de manera que los puntos que van al mismo máximo local se colocan en el mismo grupo. Una desventaja de DENCLUE es que la escalada en pendientes utilizada puede hacer pequeños pasos innecesarios al principio y nunca converge exactamente al máximo, simplemente se acerca.

Principalmente, DENCLUE opera a través de dos etapas, el paso de pre-agrupamiento y el paso de agrupamiento como se ilustra en la figura 14. El primer paso es construir un mapa (un hiper-rectángulo) de los datos. Este mapa se usa para acelerar el cálculo de la función de densidad. En cuanto al segundo paso, permite identificar grupos de cubos altamente poblados y sus cubos poblados vecinos.

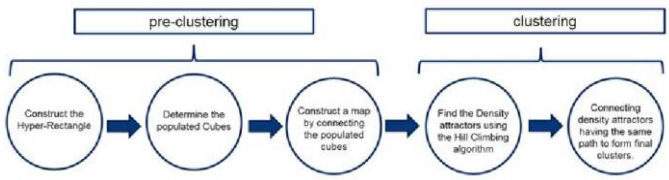

Figura 16. Estructura del algoritmo DENCLUE

DENCLUE se basa en el cálculo de la influencia de los puntos entre ellos. La suma total de estas funciones de influencia representa la función de densidad. Existen muchas funciones de influencia, basadas en la distancia entre dos puntos x e y.

Más eficientes son los algoritmos de agrupamiento basados en la localidad, ya que generalmente agrupan elementos de datos vecinos en grupos basados en condiciones locales y, por lo tanto, permiten que el agrupamiento se realice en un escaneo de los datos. DBSCAN, como se ha visto, utiliza una noción de grupos basada en la densidad y permite el descubrimiento de grupos de formas arbitrarias. Por tanto, aunque DENCLUE sea más rápido que DBSCAN, éste es más eficiente que DENCLUE.

Un problema de los enfoques existentes en el contexto de la agrupación de datos es que la mayoría de los algoritmos no están diseñados para agrupar vectores de alta dimensión y, por lo tanto, el rendimiento de los algoritmos existentes se degenera rápidamente con el aumento de la dimensión. Además, pocos algoritmos pueden manejar datos que contienen grandes cantidades de ruido.

La idea básica sobre los algoritmos de agrupación, o clustering, es que la clave está en marcar un umbral para guiar al algoritmo al final. En Clustering basado en densidad, para cada punto de un clúster la densidad de los puntos de datos en el *vecindario* debe superar algún umbral (p.e. épsilon). Por otra parte, en el proyecto realizado, "Identificación de patrones circulares", el umbral marca el grado de pertenencia mínimo a alcanzar por todos los puntos respecto a su clúster correspondiente.

REFERENCIAS

[1] Wikipedia. https://es.wikipedia.org/

[2] Documento de la web de la UGR sobre la introducción al Análisis Cluster https://www.ugr.es/~gallardo/pdf/cluster-g.pdf. Consultado el 17/05/2020

[3] Página web del curso IA de Ingeniería del Software en la universidad de Sevilla. https://www.cs.us.es/cursos/iais. Consultada el 22/04/2020.

[4] Página sobre clustering en la web de IArtificial.net https://iartificial.net/clustering-agrupamiento-kmeans-ejemplos-en-python/

[5] Presentación informativa sobre distintos tipos de algoritmos de Clustering basados en densidad, documento de la Universidad de Granada. https://elvex.ugr.es/idbis/dm/slides/43%20Clustering%20-%20Density.pdf

[6] Artículo sobre el algoritmo de clustering OPTICS. https://medium.com/@xzz201920/optics-d80b41fd042a

[7] Documento científico sobre Clustering con OPTICS. Institute for Computer Science, University of Munich. https://www.dbs.ifi.lmu.de/Publikationen/Papers/OPTICS.pdf

[8] Documento científico sobre el algoritmo DENCLUE para clustering. Institute of Computer Science, University of Halle, Germany. https://bib.dbvis.de/uploadedFiles/176.pdf

[9] Documento de un estudio sobre el algoritmo DENCLUE. Institute of Computer Science Martin-Luther-University Halle-Wittenberg, Germany. https://link.springer.com/chapter/10.1007/978-3-540-74825-0_7