Análisis de clustering aplicado a datos de la misión espacial GAIA

Rafael López

2 de junio de 2019

1. Introducción

Las estrellas se forman conjuntamente a partir de grandes nubes de gas y polvo en objetos denominados cúmulos estelares o cúmulos de estrellas [1]. Los cúmulos de estrellas son, por tanto, aglomeraciones de estrellas que suelen verse en el cielo como una sobredensidad de estrellas que comparten un movimiento común. Estos cúmulos son sumamente importantes en astronomía, ya que al ser estrellas que se formaron al mismo tiempo y en el mismo lugar, pueden proporcionar información sobre velocidad, posición y estructura de la Vía Láctea. Existen tres tipos principales de agrupaciones de estrellas:

- Las asociaciones.
- Los cúmulos globulares.
- Los cúmulos abiertos.

Las asociaciones son grupos de varias decenas de estrellas muy jóvenes que ocupan una región del cielo de gran tamaño angular. Esta gran extensión hace difícil su identificación como parte de un mismo grupo con entidad propia. Una asociación de estrellas no está ligada gravitacionalmente y contiene pocas estrellas. Sus componentes se forman en una región relativamente pequeña a partir de una nube molecular gigante. Una vez el gas y el polvo desaparecen, la asociación es visible y comienza a disgregarse, alcanzando grandes tamaños y bajas densidades. Se concentran mayoritariamente en los brazos principales de nuestra galaxia y, debido a sus bajas densidades, son muy difíciles de detectar. Normalmente se identifican por su movimiento común.

Los cúmulos globulares son el extremo opuesto a las asociaciones. Se trata de aglomeraciones del orden de 10^5 estrellas ligadas gravitacionalmente

y distribuidas de forma esferoidal a lo largo de lo que se denomina el halo galáctico. La densidad de estrellas es muy alta en la parte interna. Además, se trata de estrellas muy viejas, formadas en el origen de la Vía Láctea, cuando la abundancia de metales en la galaxia era muy baja.

Los cúmulos abiertos son grupos de varios cientos de estrellas que con el tiempo tienden a disgregarse debido al movimiento de las mismas, la rotación diferencial de la Vía Láctea y perturbaciones gravitatorias externas [1]. Estos cúmulos se localizan en el plano de la Galaxia, en donde se encuentran el gas y el polvo del que se formaron. Los cúmulos abiertos tienen gran importancia en astronomía, ya que proporcionan una herramienta muy valiosa para medir distancias en el universo, bien sea a través del método del paralaje cinemático o del método de ajuste a la secuencia principal [2]. Además, son muy buen instrumento para el estudio de la evolución estelar, ya que, al tratarse de estrellas con aproximadamente la misma edad y con una composición inicial muy similar, permiten estudiar los distintos estados evolutivos en función de su masa. A día de hoy se conocen entre dos y tres mil cúmulos abiertos, pero se cree que debe de haber muchos más que aún no han sido detectados [3]. En este trabajo nos centraremos en la detección automatizada de este tipo de cúmulos.

Los cúmulos abiertos suelen detectarse como una sobredensidad de estrellas en el cielo. El primer registro sistemático de cúmulos abiertos se hizo como parte del catálogo de Messier en el año 1771 [4]. Ya en tiempos modernos, el catálogo de Lynga en 1987 [5] fue una referencia básica en el campo, y posteriormente el de Mermilliod [6], que estaba disponible a través de la base de datos WEBDA, la cual permitía acceder vía internet a toda la información recopilada de los cúmulos conocidos. En el año 2002 Dias [7] publica el catálogo que se ha usado como referente en la comunidad científica, tras recopilar todas las propiedades e información existentes de más de dos mil cúmulos abiertos conocidos hasta la fecha. Existen, obviamente, otros catálogos de cúmulos estelares, por ejemplo el de Kharchenko [8], pero es el de Dias, que se ha ido actualizando hasta el día de hoy, el que continúa siendo el catálogo por antonomasia en la comunidad astronómica.

Muchos de los cúmulos del catálogo de Dias han sido identificados, en primer lugar, a simple vista como sobredensidades de estrellas. Por ello, la mayoría de los cúmulos en este catálogo suelen ser los más grandes y cercanos. Con la llegada de miles de millones de datos de estrellas que están arrojando los telescopios en tierra y en el espacio, hoy en día se está tendiendo a la búsqueda sistemática y automática de cúmulos estelares [9]. La dificultad en la búsqueda de cúmulos abiertos radica en que estos no necesariamente muestran un fuerte gradiente de densidad en dirección radial con respecto a los objetos de fondo. Es decir, no son cúmulos que resalten fuertemente a simple

vista. Schmeja [10] hace una comparativa de cuatro algoritmos que se han usado principalmente en la detección de cúmulos abiertos. Estos algoritmos son:

- *Star Counts* (SC): se divide el trozo de cielo en estudio en diferentes regiones y se hace un conteo de las estrellas en cada región. Aquellas regiones que superen un umbral determinado se consideran como posibles candidatas a albergar un cúmulo abierto. El principal inconveniente de este algoritmo es la división del trozo de cielo en regiones y la determinación del umbral.

- *Nearest Neighbour density* (NN): basado en el típico algoritmo KNN(*K-Nearest Neighbours*), este algoritmo calcula una medida de densidad local para cada estrella a partir de las distancias de cada una con sus N vecinas más cercanas. También permite una estimación del radio y el centro de cada cúmulo detectado. Sin embargo, requiere de mucho tiempo de procesamiento.

- *Voronoi Tesselation* (VT): se hace un diagrama de Voronoi de manera que, en las zonas más densas, el área de las teselas es menor que en zonas menos densas. La densidad de cada región es, por tanto, inversamente proporcional al área de las teselas y se puede estimar de esta forma. El principal inconveniente de este algoritmo estriba en la delimitación de la región de separación de las teselas.

- *Minimum Spanning Tree separation* (MST): está basado en grafos en donde cada estrella representa un vértice y se unen a través de ejes. El grafo no puede formar bucles cerrados. Mediante este algoritmo podemos detectar zonas más densas como aquellas en donde la longitud de los ejes es menor. De hecho, eliminando aquellos ejes con una longitud mayor que un umbral determinado, obtenemos una serie de subgrafos que representarían los cúmulos detectados. Este método, sin embargo, es sumamente sensible al valor de longitud del eje umbral seleccionado y necesario para localizar cúmulos.

Estos algoritmos han sido evaluados con cúmulos creados artificialmente y, por tanto, con datos ficticios. Se demuestra en [10] que, dado que cada algoritmo tiene sus ventajas y sus inconvenientes, la elección del método depende fuertemente del conjunto de datos y, por ende, de la región específica de cielo y el propósito del estudio. En general, para analizar grandes cantidades de datos, los algoritmos NN y MST no son factibles debido a su complejidad computacional. Por otro lado, los métodos SC y MST requieren

de la elección de parámetros a priori, lo que puede dificultar especialmente el proceso. Finalmente, el método VT está bastante desaconsejado, ya que en la práctica no da buenos resultados [10].

Otros trabajos han usado el algoritmo DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*) para la detección de aglomeraciones estelares pero no con una aproximación tan pragmática como la que se pretende en este trabajo. Así en el artículo publicado por Caballero [11], se usa DBSCAN junto con la base de datos Hipparcos [12] para estudiar la estructura interna de asociaciones y cúmulos de estrellas conocidos. La selección de parámetros para DBSCAN se realiza, en este caso, asumiendo un número mínimo de estrellas que deben de contener estas aglomeraciones y, a partir de esta premisa, se calculan de forma heurística dichos parámetros. En este trabajo pretendemos estimar dichos parámetros sin tener en cuenta ninguna asunción.

En un trabajo reciente, Castro-Ginard [3] realiza un proceso en el cual primero se usa DBSCAN para detectar sobredensidades y luego se usa una red neuronal para confirmar que esas sobredensidades representan cúmulos abiertos. Para detectar sobredensidades, en el primer paso, se usan las variables de posición (ascensión recta y declinación) junto con los paralajes y movimientos propios en el algoritmo DBSCAN. Después, se clasifican estas sobredensidades como cúmulos abiertos o no usando diagramas color-magnitud en una red neuronal. Para encontrar los parámetros óptimos de estos algoritmos se usa un conjunto de datos ficticios junto con los datos provenientes de TGAS (*Tycho-Gaia Astrometric Solution*). Una vez optimizados los parámetros se confirma su fiabilidad aplicando dicho proceso a la base de datos GAIA. Como en el trabajo de Caballero [11], en este trabajo también se lleva a cabo un método para determinar los parámetros de DBSCAN usando el algoritmo KNN. Esto hace que, aunque los resultados son bastante buenos, el proceso sea muy lento y tedioso.

La base de datos de estrellas más completa generada hasta la fecha es la de la misión espacial GAIA, lanzada por la Agencia Espacial Europea en el año 2013 [13]. Desde entonces, la misión GAIA ha estado registrando y procesando con una precisión sin precedente las posiciones, movimientos propios, velocidades radiales y otros datos sobre mil millones de estrellas de nuestra galaxia. De forma periódica se hacen públicos los datos generados por la misión en lo que se conoce como liberación o publicación de los datos (DR,*Data Release*). La versión más completa y actualizada de los datos de la misión GAIA es la DR2 [14], que se hizo pública el 25 de abril de 2018, y que será la que utilizaremos aquí.

En este trabajo queremos evaluar la eficacia y fiabilidad del algoritmo DBSCAN para detectar cúmulos abiertos de estrellas a partir de los datos

generados por la misión GAIA. A diferencia de trabajos previos en los que se suele analizar el funcionamiento de un método con objetos de referencia o simulaciones y posteriormente se aplica el método optimizado a la base de datos, aquí planteamos un enfoque empírico en el que los resultados del método se comparan con la lista de cúmulos abiertos ya conocidos. Para esto último usaremos como referencia el catálogo de Dias [21], aunque la estrategia usada en este trabajo puede aplicarse con otros catálogos diferentes o incluso con algún meta-catálogo que contenga todos los catálogos publicados. Este trabajo consiste en un estudio detallado de la viabilidad de la estrategia propuesta: preprocesamiento de los datos, implementación y optimización del algoritmo, aplicación y análisis a una región del cielo. En la sección (2) explicamos la metodología propuesta incluyendo el catálogo de referencia de cúmulos abiertos (2.1), el contenido que utilizaremos de la base de datos GAIA (2.2) y el algoritmo DBSCAN (2.3). En la sección (3) presentaremos y discutiremos todos los resultados encontrados. Finalmente, en la sección (4) resumiremos las principales conclusiones y mencionaremos futuros trabajos que pueden abordarse.

2. Metodología

En esta sección se explica la metodología empleada en el presente trabajo. Por un lado, se describen las fuentes de datos y, por otro, el algoritmo usado para la detección de cúmulos estelares abiertos.

2.1. Catálogo de cúmulos abiertos

Un catálogo de cúmulos abiertos es básicamente una base de datos en la que se tienen registradas las medidas de posición y otras variables físicas de posibles cúmulos estelares abiertos. Se necesita un catálogo de referencia, en donde se encuentren los posibles cúmulos abiertos detectados hasta la fecha, a fin de poder tener una medida sobre la fiabilidad de nuestro algoritmo, aunque es posible que algunos de los cúmulos presentes en estos catálogos no sean realmente cúmulos estelares y que, por supuesto, puedan existir más cúmulos estelares que no estén aún representados.

El catálogo de cúmulos abiertos elegido es el catálogo de Dias [21]. Este catálogo es una fuente de datos muy utilizada, puesto que recopila la información actualizada de todos los cúmulos que se van descubriendo. La última versión del mismo contiene 2167 cúmulos abiertos distribuidos en todo el cielo, lo que representa un aumento en 986 objetos con respecto a catálogos de cúmulos abiertos más antiguos. Toda la información de este catálogo está

Full	P	Cluster	RAJ2000 "h:m:s"	DEJ2000 "d:m:s"	Class	Diam arcmin	Dist pc	E(B-V) mag	Age [yr]	pmRA mas/yr	pmDE mas/yr	Nc	K14	RV km/s	o_	[Fe/H] Sun	o_	TrType	WEBDA
1	P	Berkeley 58	00 00 12	+60 58 00		11.0	2700	0.720	8.470	0.56	1.56	88	K14	-87.00	1				WEBDA
2		NGC 7801	00 00 21	+50 44 30	r	8.0	1275	0.170	9.230	-3.56	-2.60	85	K14					4-3-p-*	WEBDA
3		FSR 0459	00 00 39	+59 14 20	IR	2.0	3800	1.103	8.950	-1.66	-0.01	8	K14						WEBDA
4		Stock 18	00 01 37	+64 37 30		6.0	1242	0.710	8.100	-2.78	-0.15	109	K14					4-2-p-*	WEBDA
5		Berkeley 59	00 02 14	+67 25 00		10.0	1000	1.241	6.100	-2.11	-1.20	2	K14	-5.17	3			3-3-m-n	WEBDA
6		Berkeley 104	00 03 30	+63 35 00		3.0	4365	0.450	8.890	-2.35	-0.30	76	K14			0.070		4-2-p-*	WEBDA
7	P	Blanco 1	00 04 07	-29 50 00		70.0	269	0.010	7.796	20.17	3.00	27	K14	5.53	49	0.040	8		WEBDA
8		Stock 19	00 04 41	+56 05 00		4.0				1.43	-2.04	42	K14					2-2-p-*	WEBDA
9		NGC 7826	00 05 17	-20 41 30	r	20.0	620	0.030	9.340	6.48	-6.20	39	K14	-4.62	1			4-2-p-*	WEBDA
10		Majaess 1	00 07 22	+64 58 22	IR	5.0				4.63	1.72	20	K14						WEBDA
11		Czernik 1	00 07 38	+61 28 30		2.5	2530	1.230	6.700	-3.83	1.66	5	K14					4-2-m-*	WEBDA
12		SAI 1	00 08 20	+51 43 15		4.0	2170	0.340	9.100	0.31	-0.88	35	K14						WEBDA
13	P	Berkeley 1	00 09 36	+60 28 30		5.0	2420	0.780	8.600	-1.83	-0.75	135	K14					3-1-p-*	WEBDA
14		ASCC 1	00 09 36	+62 40 48		24.0	4000	0.160	8.250	-3.27	1.19	1189	K14	-74.10	3				WEBDA
15	P	King 13	00 10 06	+61 10 00		5.0	3100	0.820	8.500	-4.15	-2.46	127	K14					2-2-m-*	WEBDA
16		Alessi 20	00 10 34	+58 45 36	e	40.0	600	0.220	6.480	8.73	-3.11	17	K14	-11.50	1				WEBDA
17		FSR 0474	00 10 53	+59 44 50	IR	4.6	4638	1.145	8.900	-0.41	-0.81	53	K14						WEBDA
18		FSR 0480	00 14 55	+61 28 23	IR	1.5	5492	0.937	9.300	-1.66	-1.81	24	K14						WEBDA
19		Juchert Saloran 1	00 16 20	+59 57 43		5.0	3859	0.830	9.050	-0.59	0.58	60	K14						WEBDA
20		Majaess 8	00 16 43	+64 30 21	IR	3.0				-2.38	2.06	24	K14						WEBDA
21	P	Berkeley 60	00 17 42	+60 56 00		3.0	4365	0.860	8.200	-3.64	0.20	40	K14			0.070		3-1-p-*	WEBDA
22		ASCC 2	00 19 52	+55 42 36		36.0	1200	0.100	8.830	-1.56	-1.34	20	K14						WEBDA
23		FSR 0486	00 20 21	+59 19 05	IR	2.5	1850	0.550	8.900	-2.10	-2.11	5	K14						WEBDA
24		Majaess 9	00 21 13	+63 19 28	IR	2.0							K14						WEBDA
25		Mayer 1	00 21 54	+61 45 00		7.0	1430	0.500	7.740	-1.68	-1.18	122	K14	-20.90	1			4-2-p-n	WEBDA
26		King 1	00 22 04	+64 22 50	d	24.6	1080	0.760	9.600	-2.43	1.02	700	K14	-38.40	10	-0.010	10	2-2-r-*	WEBDA

Figura 1: Detalle del catálogo de Dias en donde se muestran las variables de las que haremos uso RAJ2000, DEJ2000 y Diam.

presente en una única tabla, lo cual facilita su uso. Además, el 99,7 % de los objetos contienen su diámetro aparente y el 74,5 % contienen distancias. Finalmente, respecto a sus propiedades cinemáticas, el 54,7 % de los registros contienen la media de sus movimientos propios, un 25 % sus velocidades medias radiales, y un 24,2 % tienen ambas características simultáneamente. Esto hace de este catálogo el mejor candidato para tomar como referencia. La Figura 1 muestra a manera de ejemplo las primeras entradas del catálogo de Dias. Para nuestro estudio los datos relevantes son la posición del centro del cúmulo (columnas RAJ2000 y DEJ2000) y el diámetro del cúmulo (columna Diam), puesto que debemos comparar nuestros resultados con las posiciones de los cúmulos existentes.

2.2. La base de datos GAIA

La base de datos GAIA es una base de datos pública que contiene los datos generados por la misión espacial GAIA [13]. Proporciona medidas de posición, movimiento propio y velocidad radial con una precisión sin precedente sobre mil millones de estrellas de nuestra galaxia.

La información proporcionada por la segunda liberación de datos (DR2) contiene, entre otros:

- Cinco parámetros astrométricos como son la posición en el cielo (as-

	# sources in Gaia DR2	# sources in Gaia DR1
Total number of sources	**1,692,919,135**	**1,142,679,769**
Number of 5-parameter sources	1,331,909,727	2,057,050
Number of 2-parameter sources	361,009,408	1,140,622,719
Sources with mean G magnitude	1,692,919,135	1,142,679,769
Sources with mean G_{BP}-band photometry	1,381,964,755	-
Sources with mean G_{RP}-band photometry	1,383,551,713	-
Sources with radial velocities	7,224,631	-
Variable sources	550,737	3,194
Known asteroids with epoch data	14,099	-
Gaia-CRF sources	556,869	2,191
Effective temperatures (T_{eff})	161,497,595	-
Extinction (A_G) and reddening ($E(G_{BP}-G_{RP})$)	87,733,672	-
Sources with radius and luminosity	76,956,778	-

Figura 2: Tabla comparativa entre la primera y la segunda liberación de datos de la misión GAIA (DR1 y DR2) mostrando el número total de datos de las variables más relevantes.

censión recta y declinación), paralajes y movimientos propios de más de 1300 millones de fuentes. Junto a estos se presentan los correspondientes errores de medición.

- Velocidades radiales con sus respectivos errores. Esto conduce a un conjunto total de seis parámetros físicos.
- Las medidas anteriores están, además, combinadas con medidas de magnitud en la banda G. La base de datos proporciona la magnitud en la banda G para más de 1690 millones de fuentes con su error correspondiente.
- Las magnitudes G_{BP} y G_{RP} para más de 1800 millones de fuentes junto a sus temperaturas efectivas T_{eff} y otras variables físicas.

En la Figura 2 se representa, en números, una visión general de la segunda liberación de la base de datos GAIA.

2.3. El algoritmo DBSCAN

DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*) es un algoritmo basado en densidad, propuesto por Martin Ester, Hans Peter Kriegel, Jorg Sander y Xiaowei Xu en 1996 [15]. En este algoritmo, dado un conjunto de puntos, estos se agrupan en *clusters* haciendo uso no solo de la distancia entre los mismos, sino de su densidad local.

Para conocer cómo funcionan, se define una bola de radio ϵ alrededor de un punto $x \in R$, llamado vecindario de x como sigue:

$$N_\epsilon(x) = B_d(x, \epsilon) = \{y | \delta(x, y) \leq \epsilon\} \tag{1}$$

Aquí, $\delta(x, y)$ representa la distancia entre los puntos x e y. Normalmente esta distancia se asume euclídea: $||x - y||_2$.

Para cualquier punto x, decimos que x es un punto *core* si hay, al menos, $Nmin$ puntos en su vecindario, definido por ϵ. En otras palabras, x es un punto *core* si $|N_\epsilon(x)| \geq Nmin$, donde $Nmin$ es una densidad local de puntos o umbral definido por el usuario.

Un punto se dice que es *border* si en su vecindario no llega a tener $Nmin$ puntos, es decir, $|N_\epsilon(x)| < Nmin$, pero pertenece al vecindario de algún punto *core* z, es decir, $x \in N_\epsilon(z)$.

Finalmente, si un punto no es ni *core* ni *border*, se dice que es un punto ruidoso (*noisy*).

Decimos que un punto x es alcanzable directamente por densidad desde otro punto y si $x \in N_\epsilon(y)$ e y es un punto *core*. Decimos que un punto x es alcanzable por densidad desde otro punto y si existe una cadena de puntos $x_0, x_1, ..., x_l$, de manera que $x = x_0$ e $y = x_l$ y x_i es directamente alcanzable por densidad desde x_{i-1}, para todo $i = 0, 1, 2, 3, ...l$. En otras palabras, existe un conjunto de puntos *core* que conducen desde y hasta x.

Definidos dos puntos x e y que están conectados por densidad, si existe un punto *core* z de manera que ambos x e y son alcanzables mediante densidad desde z, se define un *cluster* basado en densidad como el conjunto máximo de puntos conectados por densidad. En la Figura 3 se puede apreciar cómo los puntos B y C son alcanzables mediante densidad por el punto core A y, por tanto, forman parte del mismo *cluster*.

El algoritmo DBSCAN funciona, entonces, de la siguiente forma:

- Primero calcula el vecindario $N_\epsilon(x)$ para cada punto x en el conjunto de datos y comprueba si es un punto *core*.

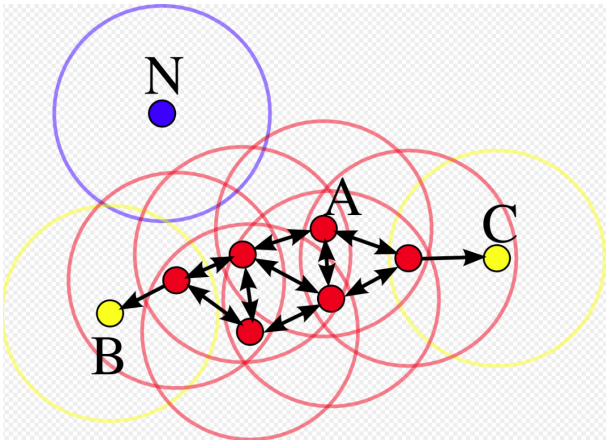

Figura 3: En esta figura, los puntos rojos representan puntos *core*, los puntos amarillos puntos *border* y el punto azul es un punto *noisy*. Los puntos B y C son alcanzables mediante densidad por el punto A y, por tanto, forman parte del mismo *cluster*.

- A continuación, asigna un identificador $id(x_i) = \emptyset$ para todos los puntos que no están asignados a un *cluster*.
- Seguidamente, empezando por cada punto que no es *core* y, de forma recursiva, el algoritmo encuentra todos los puntos que son alcanzables mediante densidad, asignándolos al *cluster* correspondiente.
- Algunos puntos *border* pueden ser alcanzados mediante puntos que pertenecen a *clusters* diferentes. En este caso, normalmente, son asignados de forma arbitraria a alguno de los *clusters*.
- Finalmente, aquellos puntos que no pertenecen a ningún *cluster* se consideran puntos ruidosos.

Una de las mayores desventajas de este algoritmo, y que ha sido puesta de manifiesto en el presente trabajo, es que es muy sensible a los cambios del parámetro ϵ. Para valores muy pequeños de este parámetro podemos encontrar grandes *clusters* categorizados como ruido, y al contrario, para valores grandes de ϵ, *clusters* densos pueden ser considerados como uno solo. En otras palabras, si hay *clusters* con densidades locales diferentes, un único valor de ϵ podría no ser suficiente para localizarlos a todos.

Otra desventaja es su coste computacional. Su principal cuello de botella se encuentra en el primer paso, cuando DBSCAN calcula el vecindario de cada uno de los puntos del conjunto de datos. Cuando la dimensionalidad

del conjunto de datos es alta, este algoritmo puede tener una complejidad del orden de $O(n^2)$.

2.3.1. ¿Por qué DBSCAN?

Dentro de los algoritmos de *clustering* más representativos, encontramos los algoritmos *K-means* o *Expectation-Maximization*. Estos algoritmos, sin embargo, no proporcionan muy buenos resultados cuando los *clusters* no presentan una forma convexa y, dado que los cúmulos que pretendemos detectar son más disgregados que los globulares, no siempre tienen una forma perfectamente convexa. Por tanto, algoritmos como *K-means* o *Expectation-Maximization* no darían buenos resultados. Los algoritmos basados en densidad, por el contrario, son capaces de descubrir *clusters* que no tienen una forma total o parcialmente convexa. Además, con DBSCAN, no necesitamos saber, a priori, el número de *clusters* presentes. Si el objetivo final es poder descubrir cúmulos estelares en zonas donde no sabemos si existen, no tiene mucho sentido el hecho de que tengamos que conocer, por adelantado, el número de cúmulos en esa zona del cielo. Estas son las razones por las que, en el presente trabajo, hemos optado por elegir el algoritmo basado en densidad DBSCAN y evaluar su rendimiento para descubrir posibles cúmulos estelares en un futuro.

2.4. ¿Por qué Python?

Python es un lenguaje de programación con el que se pueden escribir programas muy complejos de manera sencilla. Su facilidad para crear prototipos rápidamente y su amplio uso en el ámbito científico, hacen que sea el lenguaje preferido en muy diversos campos de la ciencia.

Por un lado, en el caso de la astronomía de posición, Python proporciona librerías con una enorme funcionalidad para trabajar con datos astronómicos, como es *astropy* [19], y otras que permiten conectarse con bases de datos astronómicas y realizar consultas sobre ellas, como es *astroquery*. Por otro lado, la librería *scikit learn* [20], proporciona una amplia funcionalidad para trabajar con algoritmos de *machine learning*.

3. Resultados y discusión

En esta sección se describe cómo se ha implementado el algoritmo DBSCAN en este trabajo, así como los resultados obtenidos y las dificultades encontradas. Al mismo tiempo, se discute cómo determinados parámetros influyen en esta metodología.

3.1. Implementación

A continuación se describe la implementación completa de los diferentes módulos necesarios para llevar a cabo este trabajo.

3.1.1. Uso del catálogo de cúmulos abiertos

Para representar esta base de datos se ha implementado la clase *DiasCatalog.py*. En esta clase se hace una conexión al catálogo de Dias que se almacena en una tabla interna de la librería *astropy*. La conexión y descarga de los datos se realiza mediante la clase *Vizier* del paquete *astroquery.vizier*. Esta clase contiene, además de métodos para obtener el diámetro de un cúmulo concreto, métodos para obtener coordenadas de ascensión recta y declinación de algún cúmulo estelar, entre otros.

El principal método, sin embargo, es el que nos permite obtener una medida de la efectividad de nuestro algoritmo. Para ello, dadas las coordenadas de centro y radio de una región específica de cielo y las coordenadas de los cúmulos detectados por el algoritmo, la medida de la fiabilidad, denominada M, la definimos de la siguiente forma:

$$M = \frac{N_{matches}}{N_{dias} + N_{extra}} \tag{2}$$

en donde $N_{matches}$ indica el número de cúmulos detectados por el algoritmo que además coinciden con los registrados en el catálogo de Dias; N_{dias} es el número de cúmulos registrados en el catálogo de Dias para la región específica del cielo que se esté tratando, y N_{extra} es el número de cúmulos que detecta el algoritmo pero que no tienen correspondencia en la base de datos de Dias.

El cálculo de $N_{matches}$ se realiza de la siguiente forma: se considera que hay un acierto cuando las coordenadas de un cúmulo detectado por DBSCAN están dentro de la zona delimitada por las coordenadas de un cúmulo registrado en Dias más menos su radio. Esto se implementa en la clase *DiasCatalog.py* mediante dos métodos: uno para obtener el número de cúmulos estelares en una región de cielo específica, *get_clusters*, y otro para obtener el número de cúmulos acertados por nuestro algoritmo, *get_match*.

```
def get_clusters(self, center, r):
        clusters = []
        ra_center = Longitude(center[0], unit=u.deg)
        dec_center = Latitude(center[1], unit=u.deg)
        center = SkyCoord(ra_center, dec_center)
        for i in range(self.table['Cluster'].shape[0]):
            ra = Angle(self.table[i]['RAJ2000'], unit=u.hourangle)
```

```
            ra = ra.deg
            ra = Longitude(ra, unit=u.deg)
            dec = Latitude(self.table[i]['DEJ2000'], unit=u.deg)
            cluster = SkyCoord(ra, dec)
            diam = float(self.table[i]['Diam'])*u.arcmin
            if(np.isnan(diam)):
                diam = 0.05
            else:
                diam = diam.to('deg').value
            if(cluster.separation(center).value <= r):
                clusters.append((self.table['Cluster'][i], cluster, diam))
        return(clusters)
```

```
def get_match(self, center, r, clusters_detected):
        existing_clusters = self.get_clusters(center, r)
        tot_matches = []
        for cluster in clusters_detected:
            ra_cluster_detected = Longitude(cluster[0], unit=u.deg)
            dec_cluster_detected = Latitude(cluster[1], unit=u.deg)
            cluster_detected = SkyCoord(ra_cluster_detected,
            dec_cluster_detected)
            l_tmp = [match for match in existing_clusters if
            match[1].separation(cluster_detected).value <= match[2]/2.0]
            tot_matches = list(set(tot_matches + l_tmp))
        return tot_matches
```

De esta forma, con el método *get_clusters* obtenemos la variable N_{dias}, mientras que con el método *get_match* obtenemos la variable $N_{matches}$. La última variable, N_{extra}, se obtiene restando al número total de cúmulos localizados por el algoritmo, el número de ellos que existen en el catálogo de Dias.

3.1.2. Uso de la base de datos GAIA

Para representar los datos extraídos de la base de datos GAIA se ha implementado la clase *GaiaData.py*, en donde se han definido funciones básicas para conectarse a dicha base de datos y descargar la información necesaria. Aunque a esta clase se le puede introducir más funcionalidad, solo tiene completamente implementado un tipo de consulta. Esta consulta selecciona un trozo circular de cielo a partir de cuatro parámetros:

- Centro del trozo de cielo a seleccionar.
- Radio del trozo circular de cielo a seleccionar.
- Error mínimo tolerable para ascensión recta, declinación y paralaje.
- Mínimo valor en magnitud G.

Debido a la cantidad de datos que puede contener una pequeña porción de cielo y a fin de hacer estos datos manejables, se han introducido los parámetros de error mínimo tolerable y valor mínimo en magnitud G. Estos parámetros nos sirven como filtrado de la consulta para que el número de datos que obtenemos con ella sea más manejable y para poder analizar el efecto de dichos parámetros en la ejecución del algoritmo. Por otro lado, y por simplicidad, se ha incluido un único parámetro de error que se puede usar también en la consulta para filtrar los datos, tanto en paralaje como en ascensión recta y en declinación. En un trabajo futuro esto se puede modificar fácilmente y hacer que la consulta sea más específica. El valor en magnitud G nos proporciona una medida del brillo de las estrellas. Filtrando por un valor mínimo nos aseguramos de que obtenemos solo las estrellas más brillantes, dejando las más alejadas fuera de los datos.

Otro punto a tener en cuenta es el tratamiento de los valores nulos. En este trabajo, debido de nuevo a la cantidad ingente de datos que se manejan, se ha optado por prescindir de ellos, así como de las magnitudes de paralaje negativas que, desde un punto de vista físico, no tienen sentido y son datos erróneos.

El método implementado para realizar una consulta de un trozo circular de cielo es el siguiente:

```
def astrometric_query_circle(self, point, radius, err_pos, min_g_mag):
        """Construct a query of a sky circle centered in point
           specified when creating this object

        Parameters:
        -----------
        radius: Radius of the circle to query

        Returns
        --------
        ADQL Query
        data retrieved from the query
        """
        query = "SELECT %s"%', '.join(self.attributes)
```

```
        query += " FROM gaiadr2.gaia_source"
        query += " WHERE CONTAINS(POINT('%s', ra, dec),
                  CIRCLE('%s', %s, %s))=1"%(self.coordsys,
                                self.coordsys,0
                                 ', '.join([str(n) for n in point]),
                                str(radius))
        query +=" AND parallax IS NOT NULL"
        query +=" AND parallax >= 0.0"
        query +=" AND ra IS NOT NULL"
        query +=" AND dec IS NOT NULL"
        query +=" AND pmra IS NOT NULL"
        query +=" AND pmdec IS NOT NULL"
        query +=" AND ABS(ra_error) < %s"%str(err_pos)
        query +=" AND ABS(dec_error) < %s"%str(err_pos)
        query +=" AND ABS(parallax_error) < %s"%str(err_pos)
        query +=" AND ABS(phot_g_mean_mag) > %s"%str(min_g_mag)
        print(query)
        data = self.get_results(query)
        return(data)
```

Como ejemplo, si suponemos que queremos filtrar datos con un error en posición menor de 0,1 y un valor mínimo en magnitud G de 15,5, la consulta sería la siguiente:

```
SELECT source_id, ra, ra_error, dec, dec_error, parallax,
       parallax_error, pmra, pmdec, phot_g_mean_mag
FROM gaiadr2.gaia_source
WHERE CONTAINS(POINT('ICRS', ra, dec),
CIRCLE('ICRS', 93.6084, 14.6927, 1.0))=1
AND parallax IS NOT NULL AND parallax >= 0.0
AND ra IS NOT NULL
AND dec IS NOT NULL
AND pmra IS NOT NULL
AND pmdec IS NOT NULL
AND ABS(ra_error) < 0.1
AND ABS(dec_error) < 0.1
AND ABS(parallax_error) < 0.1
AND ABS(phot_g_mean_mag) > 15.5
```

Como puede observarse, obtenemos la clave primaria de cada registro junto a las medidas de posición (ascensión recta, declinación y paralaje) y sus respectivos errores, movimientos propios en declinación y ascensión recta, y la magnitud en la banda G.

Dicha consulta se realiza de manera asíncrona, ya que cuando hacemos uso de muchos datos, este tipo de consulta permite no acaparar todo el procesamiento en el servidor.

```
def get_results(self, query):
        """Runs a job in GAIA archive to retrieve data

        Parameters.
        -----------
        query: Query for GAIA database

        Returns
        --------
        data: Data retrieved from ADQL query
        """
        try:
            job = Gaia.launch_job_async(query)
        except:
            print('Query has failed')
        else:
            data = job.get_results()
        return(data)
```

3.1.3. Preprocesado

Como se ha descrito en la sección anterior, ya en la propia consulta se hace un preprocesado de datos al filtrar aquellos cuyos valores posicionales no sean nulos, errores posicionales menores de un valor determinado, y estrellas de cierto brillo. Con esto, ya estamos preseleccionando datos de calidad y descartando aquellos erróneos.

Además de este preprocesado se ha implementado un método para, por un lado, poder realizar un muestreo aleatorio que reduzca aún más el conjunto de datos y, por otro, adaptar las coordenadas con sus correspondientes unidades y crear tres conjuntos de datos diferentes:

- Un primer conjunto con los datos posicionales en coordenadas esféricas en grados.
- Un segundo conjunto con los datos posicionales en coordenadas cartesianas en parsecs.
- Un tercer conjunto con los datos posicionales en coordenadas cartesianas junto con los movimientos propios.

Cada uno de los conjuntos anteriores se puede usar en el algoritmo para buscar cúmulos estelares. Para seleccionar qué conjunto usar cuando se lanza el programa, se debe especificar en la línea de comandos *sphe* si se quiere usar el primer conjunto de datos con las posiciones de las estrellas en coordenadas esféricas, *cart* para usar el segundo conjunto de datos con coordenadas cartesianas, y *pm* para usar el tercer conjunto de datos en donde se tienen las coordenadas cartesianas junto a los movimientos propios en ascensión recta y declinación. Un ejemplo de ejecución del programa sería *python3 main.py fichero.csv sphe*.

Con respecto al muestreo, este se especifica en el fichero *.csv* de entrada al programa. Un valor de 1,0 indica que no se realiza un muestreo aleatorio sobre el conjunto de datos consultado, un valor mayor que 1,0 dará error, y uno menor obtendrá un conjunto de datos reducido al valor especificado. Así, un factor de 0,5 indica que muestreamos de forma aleatoria y nos quedamos con el 50 % de los datos, es decir, la mitad.

El código correspondiente es el siguiente:

```
def preprocessing(data, sample_factor = None):
    #Preprocessing
    data_metrics = data
    data_metrics['parallax'] = data_metrics['parallax'].
    to(u.parsec, equivalencies=u.parallax())
    #Adapt data metrics to a numpy array
    np_data_metrics =
    np.transpose(np.array([data_metrics['ra'],
    data_metrics['dec'],
    data_metrics['parallax'],
    data_metrics['pmra'], data_metrics['pmdec']]))
    #Sample data
    if(sample_factor != None):
        idx = np.random.choice(np_data_metrics.shape[0],
        size=int(sample_factor*np_data_metrics.shape[0]),
                                replace= False)
        np_data_metrics = np_data_metrics[idx,:]
    #Change coordinates from Spherical to Cartesian coordinate system
    ra = Longitude(np_data_metrics[:,0], unit=u.deg)
    ra.wrap_angle = 180 * u.deg
    dec = Latitude(np_data_metrics[:,1], unit=u.deg)
    dist = Distance(np_data_metrics[:,2], unit=u.parsec)
    sphe_coordinates = SkyCoord(ra, dec, distance = dist, frame='icrs',
    representation_type='spherical')
    cart_coordinates = sphe_coordinates.cartesian
    #Adapt data to normalize it correctly
```

```
data_sphe_adapted = np.transpose(np.array([sphe_coordinates.ra,
sphe_coordinates.dec,
sphe_coordinates.distance]))
data_cart_adapted = np.transpose(np.array([cart_coordinates.x,
cart_coordinates.y,
cart_coordinates.z]))
data_pm_adapted = np_data_metrics[:,3:5]
data_all_adapted = np.append(data_cart_adapted, data_pm_adapted, axis=1)
return(data_sphe_adapted, data_cart_adapted, data_all_adapted)
```

3.1.4. Matriz de distancias

Debido al coste computacional que supondría que el algoritmo DBSCAN recalculase la matriz de distancias para cada combinación de valores $\epsilon - Nmin$, se ha implementado un método que, por anticipado, calcula la matriz de distancias entre cada uno de los puntos del conjunto de datos. Esta matriz de distancias la usará posteriormente el algoritmo a modo de solo lectura, mejorando de esta forma la eficiencia de nuestro programa.

La matriz de distancias se puede obtener de varias formas, dependiendo de cómo se le especifique al programa el parámetro correspondiente en el fichero de entrada. Las posibles formas de calcular la matriz de distancias son:

- Calculando la distancia euclídea entre cada par de puntos.
- Calculando las distancias mediante la función *separation* de la librería *astropy* [19]. Esta función permite obtener una medida real de las distancias incluso cuando no se trabaja en coordenadas cartesianas.

La matriz de distancias, por otro lado, se puede calcular a partir de datos normalizados o datos sin normalizar. Este es otro de los parámetros que podemos especificar al método de cálculo de distancias. De esta forma, dependiendo de los datos con los que estemos trabajando, será más conveniente normalizarlos o no. La normalización que se realiza es una normalización estándar para que los datos nos queden entre valores 0 y 1.

Las distancias a calcular se especifican en el parámetro *distance* del fichero *.csv* de entrada. Si dicho parámetro es *euclidean* se calcula una matriz de distancias euclídeas. Para cualquier otro valor, se hace uso de la función *separation* o *separation_3d* de la librería *astropy*. La normalización se especifica en el campo *norm* del fichero de entrada con valor X o dejándolo vacío. Si está a X se normalizan los datos usando la función *MinMaxScaler* de *scikit-learn*.

El método implementado para obtener la matriz de distancias es el siguiente:

```
def get_distance_matrix(data, scale=False, metric='euclidean'):
    if scale:
        scaler = MinMaxScaler()
        data_scaled = scaler.fit_transform(data)
    else:
        data_scaled = data
    if metric != 'euclidean':
        dist_matrix = pairwise_distances(data_scaled,
        metric=get_distances_for_ML, n_jobs=-1)
    else:
        dist_matrix = pairwise_distances(data_scaled,
        metric='euclidean', n_jobs=-1)
    return(dist_matrix)
```

Como se puede observar, se utiliza la función *pairwise_distances* de la librería *scikit-learn* para el cálculo de la matriz de distancias. Si la distancia especificada no es euclídea se hace uso, además, de la función *get_distances_for_ML*, cuyo código es el que sigue:

```
def get_distances_for_ML(X, Y):
    distance = 0
    ra1 = X[0]*u.deg
    ra2 = Y[0]*u.deg
    dec1 = X[1]*u.deg
    dec2 = Y[1]*u.deg
    if(len(X) == 3):
        dist1 = X[2]*u.parsec
        dist2 = Y[2]*u.parsec
        point1 = SkyCoord(ra1, dec1, dist1)
        point2 = SkyCoord(ra2, dec2, dist2)
        distance = point1.separation_3d(point2)
    else:
        point1 = SkyCoord(ra1, dec1)
        point2 = SkyCoord(ra2, dec2)
        distance = point1.separation(point2)
    return(distance.value)
```

Esta función toma como parámetros dos puntos X e Y y calcula la distancia entre ellos mediante las funciones *separation* o *separation_3d*, dependiendo de la dimensionalidad de los datos (si tienen dos o tres dimensiones). La dimensionalidad es otro parámetro que también se especifica en el fichero de entrada marcando o desmarcando la columna *dim3*.

3.1.5. Algoritmo DBSCAN

En este trabajo se ha usado el algoritmo DBSCAN implementado en la librería *scikit-learn* de *Python*[20]. Para evaluar este algoritmo, se itera en una malla de valores de los parámetros ϵ y $Nmin$ que se especifica en el fichero de entrada. Para cada combinación de valores se calcula la medida M definida en la Ecuación 2.

La malla de valores a iterar se introduce en las columnas *eps_min*, *eps_max* y *eps_num* para el parámetro ϵ, y *min_pts_min*, *min_pts_max* y *min_pts_num* para el parámetro $Nmin$. Con estos valores del fichero de entrada indicamos los valores mínimo y máximo que deben tomar tanto ϵ como $Nmin$, así como el número de puntos en los que se divide ese rango de valores. Cada una de estas combinaciones de valores ϵ y $Nmin$ queda registrada en una estructura interna junto a la medida M. Esta estructura de salida se escribirá, al terminar el algoritmo, en un fichero *.csv* con los resultados obtenidos.

El método que implementa esta funcionalidad es el siguiente:

```
def DBSCAN_eval(data, center, r, sample_factor, ftype,
                dim3, eps_range, min_samples_range, scale=False,
                metric = 'euclidean', pm=False):
    data_sphe, data_cart, data_all = preprocessing(data, sample_factor)
    data_search = data_sphe[:,:2]
    if ftype == 'cart':
        if dim3:
            data = data_cart
        else:
            data = data_cart[:,:2]
    elif ftype == 'sphe':
        if dim3:
            data = data_sphe
        else:
            data = data_sphe[:,:2]
    elif ftype == 'pm':
        data = data_all
    else:
        print('Specify tpye of dataset: cart, sphe, pm')
        sys.exit()

    dist_matrix = get_distance_matrix(data, scale, metric)

    if pm:
        data = data[:,:3]
```

```
    dias_catalog = dc.DiasCatalog()
    num_cum = len(dias_catalog.get_clusters(center, r))
    tmp_sscores = []
    matches = []
    sscores = pd.DataFrame(columns=['epsilon', 'minpts', 'local_score'])
    cluster_centers = []
    for eps in eps_range:
        for min_samples in min_samples_range:
            db = DBSCAN(eps=eps, min_samples=min_samples,
            metric="precomputed",
            n_jobs=-1).fit(dist_matrix)
            labels = db.labels_
            for i in range(len(set(labels))-1):
                cluster = data_search[np.where(labels == i)]
                cluster_center = cluster.mean(axis=0)
                cluster_centers.append(cluster_center)
            matches = dias_catalog.get_match(center, r, cluster_centers)
            tmp_sscores.append(eps)
            tmp_sscores.append(min_samples)
            if(len(matches) > 0 and len(set(labels)) > 1):
                tmp_sscores.append(len(matches)/
                (num_cum + (len(cluster_centers)-len(matches))))
            else:
                tmp_sscores.append(0.0)
            sscores.loc[len(sscores)] = tmp_sscores
            tmp_sscores = []
            cluster_centers = []
    return(sscores, dist_matrix, data_search)
```

Este método se evalúa para cada línea del fichero de entrada y, como se observa, hace uso de los métodos descritos en secciones anteriores para realizar el preprocesado y calcular la matriz de distancias, así como de métodos implementados en la clase *DiasCatalog.py* para calcular M.

Finalmente, también se han implementado métodos para representar gráficamente los cúmulos detectados y las curvas de nivel que representan la dependencia de M respecto a los parámetros ϵ y $Nmin$. Además, estos métodos almacenan los gráficos obtenidos en ficheros *.png*.

```
def plot_clusters(X, labels, size=1.0):
    # Black removed and is used for noise instead.
    f = plt.figure(figsize=(20,20))
    unique_labels = set(labels)
    colors = [plt.cm.Spectral(each)
```

```
            for each in np.linspace(0, 0.5, len(unique_labels))]
    for k, col in zip(unique_labels, colors):
        if k == -1:
            # Black used for noise.
            col = [0, 0, 0, 1]

        class_member_mask = (labels == k)

        xy = X[class_member_mask]
        plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
                 markeredgecolor='k')

    plt.title('Estimated number of clusters: %5d' % int(len(set(labels))-1))
    plt.xlabel("Right Ascension (deg)")
    plt.ylabel("Declination (deg)")
    f.savefig('plot_cluster_execution_%s.png'%(datetime.datetime.now()))



def plot_score(param_scores):
    np_param_scores = param_scores.values
    eps = np.sort(np.array(list(set(param_scores['epsilon']))))
    Nmin = np.sort(np.array(list(set(param_scores['minpts']))))
    Z = np.empty((len(Nmin), len(eps)))
    fig, ax = plt.subplots(constrained_layout = True)
    X, Y = np.meshgrid(eps, Nmin)
    for i, n in enumerate(Nmin):
        for j, e in enumerate(eps):
            Z[i,j] = np_param_scores[np.where
             ((param_scores['epsilon'] == e)
             &
             (param_scores['minpts'] == n)),2]
    extend = "neither"
    cmap = plt.cm.get_cmap('hot')
    CS = ax.contourf(X,Y,Z, cmap=cmap, extend=extend)
    fig.colorbar(CS)
    ax.set_xlabel('Epsilon')
    ax.set_ylabel('Nmin')
    ax.set_title('DBSCAN matchin M')
    fig.savefig('plot_score_execution_%s.png'%(datetime.date.today()))
```

3.2. Elección de la región de cielo y sus parámetros

Para el estudio del algoritmo DBSCAN se ha seleccionado una zona de cielo centrada en las coordenadas $81,8553°$ en ascensión recta y $34,719°$ en declinación, con un radio de $0,5°$. La razón por la que se ha seleccionado esta región es que contiene una cantidad suficientemente grande de estrellas, sin ser demasiadas como para provocar errores de memoria, y porque contiene cinco cúmulos detectados en el catálogo Dias que van a servir para evaluar el algoritmo. Los cinco cúmulos contenidos en esta región de cielo son:

- *FSR 0777.*
- *Stock 8.*
- *FSR 0775.*
- *Kronberg 1.*
- *Majaess 58.*

Hemos realizado una búsqueda previa de los parámetros que necesita el fichero de entrada al programa. Para ello se ha analizado el trozo de cielo en estudio usando la herramienta web que proporciona la Agencia Espacial Europea [16], mediante la cual se han obtenido algunas medidas de interés:

- Número de estrellas: 19695.
- Error máximo en ascensión recta: 3,22 mas.
- Error máximo en declinación: 2,24 mas.
- Error máximo en paralaje: 3,93 mas.
- Media aritmética en la banda G: 18,07 mag.

Las consultas lanzadas en la herramienta para obtener los datos anteriores se pueden observar en las Figuras 4 a 8.

Basándonos en estas medidas, se ha configurado el fichero de entrada para que el algoritmo no tenga en cuenta los errores posicionales. Además, en uno de los casos, se especifica el valor de la magnitud en la banda G a fin de evaluar la dependencia de DBSCAN con respecto a este parámetro. En la Figura 9 podemos ver un detalle del fichero *.csv* de entrada al programa. En él se pueden observar los valores en ascensión recta y declinación obtenidos anteriormente, junto a valores correspondientes a los parámetros de filtrado y muestreo. En nuestro caso, además, hemos utilizado distancias euclídeas sin tener en cuenta el paralaje, únicamente las distancias en dos dimensiones de ascensión recta y declinación.

```
SELECT COUNT(*) AS num_reg
FROM gaiadr2.gaia_source
WHERE CONTAINS(POINT('ICRS', ra, dec), CIRCLE('ICRS', 81.8553, 34.719, 0.5))=1
AND parallax IS NOT NULL AND parallax >= 0
AND dec IS NOT NULL
AND ra IS NOT NULL
AND pmra IS NOT NULL
AND pmdec IS NOT NULL
```

Figura 4: Consulta 1.

```
SELECT MAX (ra_error) AS max_ra_err
FROM gaiadr2.gaia_source
WHERE CONTAINS(POINT('ICRS', ra, dec), CIRCLE('ICRS', 81.8553, 34.719, 0.5))=1
AND parallax IS NOT NULL AND parallax >= 0
AND dec IS NOT NULL
AND ra IS NOT NULL
AND pmra IS NOT NULL
AND pmdec IS NOT NULL
```

Figura 5: Consulta 2.

```
SELECT MAX (dec_error) AS max_dec_err
FROM gaiadr2.gaia_source
WHERE CONTAINS(POINT('ICRS', ra, dec), CIRCLE('ICRS', 81.8553, 34.719, 0.5))=1
AND parallax IS NOT NULL AND parallax >= 0
AND dec IS NOT NULL
AND ra IS NOT NULL
AND pmra IS NOT NULL
AND pmdec IS NOT NULL
```

Figura 6: Consulta 3.

```
SELECT MAX (parallax_error) AS max_parallax_err
FROM gaiadr2.gaia_source
WHERE CONTAINS(POINT('ICRS', ra, dec), CIRCLE('ICRS', 81.8553, 34.719, 0.5))=1
AND parallax IS NOT NULL AND parallax >= 0
AND dec IS NOT NULL
AND ra IS NOT NULL
AND pmra IS NOT NULL
AND pmdec IS NOT NULL
```

Figura 7: Consulta 4.

```
SELECT AVG (phot_g_mean_mag) AS phot_g_mean_mag
FROM gaiadr2.gaia_source
WHERE CONTAINS(POINT('ICRS', ra, dec), CIRCLE('ICRS', 81.8553, 34.719, 0.5))=1
AND parallax IS NOT NULL AND parallax >= 0
AND dec IS NOT NULL
AND ra IS NOT NULL
AND pmra IS NOT NULL
AND pmdec IS NOT NULL
```

Figura 8: Consulta 5.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
ra	dec	r	err_pos	min_mag_g	norm	sample	dim3	distance	eps_min	eps_max	eps_num	min_pts_min	min_pts_max	min_pts_num
81.8553	34.719	0.5	100	17		0.6602521121		euclidean	0.008	0.03	100	10	100	91
81.8553	34.719	0.5	100	0		0.5		euclidean	0.008	0.03	100	10	100	91

Figura 9: Detalle del fichero de entrada al programa.

Es preciso mencionar que el procedimiento concreto a seguir en cuanto a muestreo, normalización, filtrado y elección de variables tiene una fuerte dependencia de la región de cielo escogida. Dependiendo de esta, los parámetros elegidos pueden cambiar. Se necesita, por tanto, un análisis previo de cada zona de cielo en estudio y una selección exhaustiva de los parámetros.

3.3. Dependencia con los parámetros

Para estudiar la dependencia con los parámetros ϵ y $Nmin$ del algoritmo DBSCAN, se ha configurado el fichero de entrada de manera que realice un barrido de 100 puntos del parámetro ϵ entre 0,008 y 0,03[1], y un barrido del parámetro $Nmin$ de 91 puntos entre 10 y 100, tal y como se observa en la Figura 9. Para cada combinación $\epsilon - Nmin$ se evalúa M, representada en la Ecuación 2 y definida como medida de fiabilidad o efectividad del algoritmo, y se genera un fichero con todos los valores de estos parámetros junto a M. En la Figura 10 se puede ver un detalle de uno de los ficheros generados por el programa.

En un primer barrido de parámetros, realizando un filtrado previo en la banda G con una magnitud de corte de 17,0 y un muestreo aleatorio posterior del 66,025 % para quedarnos con el 50 % de los datos, es decir, con 9847 de un total de 19695, obtenemos una efectividad máxima de 0,6. En la Tabla 1 se pueden observar los estadísticos de estos resultados. El máximo en 0,6 nos indica que se han encontrado tres de los cúmulos registrados en Dias y ninguno más, tal y como se observa en la Figura 11, donde mostramos las posiciones encontradas de los cúmulos para la mejor solución ($M = 0,6$). Vemos en esta figura que a simple vista se observan algunas sobredensidades y regiones menos densas que en principio no son más que fluctuaciones estadísticas en la distribución de estrellas. Dependiendo de los parámetros ϵ y $Nmin$ el algoritmo DBSCAN podrá encontrar tantas sobredensidades como estrellas hay o, por el contrario, una sola sobredensidad global para el caso más extremo. Este es precisamente el problema de intentar optimizar DBSCAN con resultados reales: la definición de «sobredensidad» como tal contiene una importante componente subjetiva. De allí el enfoque propuesto en este trabajo que no ha sido usado hasta ahora, y es el de optimizar el uso de DBSCAN usando para ello posiciones ya conocidas de cúmulos reales.

[1]Durante el desarrollo de este trabajo hemos barrido una región mucho más amplia de valores de ϵ y $Nmin$, pero los valores de M siempre eran nulos o casi nulos. Aquí estamos mostrando los resultados solo para los rangos de valores de los parámetros donde se obtuvieron resultados significativos.

	epsilon	minpts	local_score
0	0.008	10	0.1666666667
1	0.008	11	0.3333333333
2	0.008	12	0.4
3	0.008	13	0.2
4	0.008	14	0.2
5	0.008	15	0.2
6	0.008	16	0.2
7	0.008	17	0
8	0.008	18	0
9	0.008	19	0
10	0.008	20	0
11	0.008	21	0

Figura 10: Detalle del fichero de salida generado por el programa.

	epsilon	**Nmin**	**M**
count	9100	9100	9100
mean	0,019	55	0,0784918342
std	0,0064150347	26,2692944806	0,1088574997
min	0,008	10	0
25 %	0,0135	32	0
50 %	0,019	55	0
75 %	0,0245	78	0,2
max	0,03	100	0,6

Cuadro 1: Tabla de estadísticos para los parámetros ϵ, $Nmin$ y la función M en el caso de la ejecución con filtrado en la banda G.

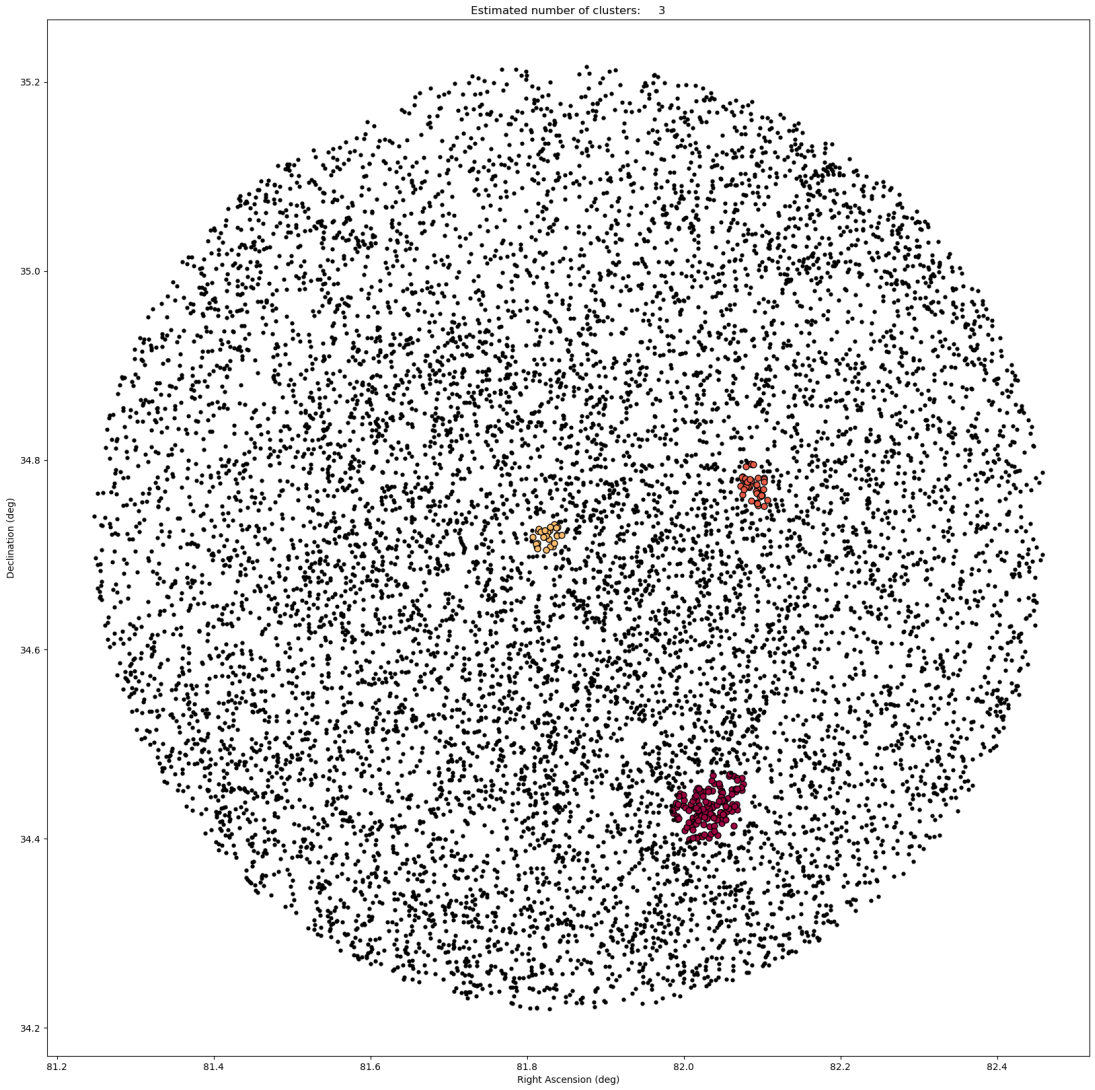

Figura 11: Posiciones de las estrellas en la región del cielo estudiada y resultados para la ejecución con filtrado en la banda G para el caso de la mejor ejecución $M = 0,6$. En círculos de colores se indican los tres cúmulos encontrados.

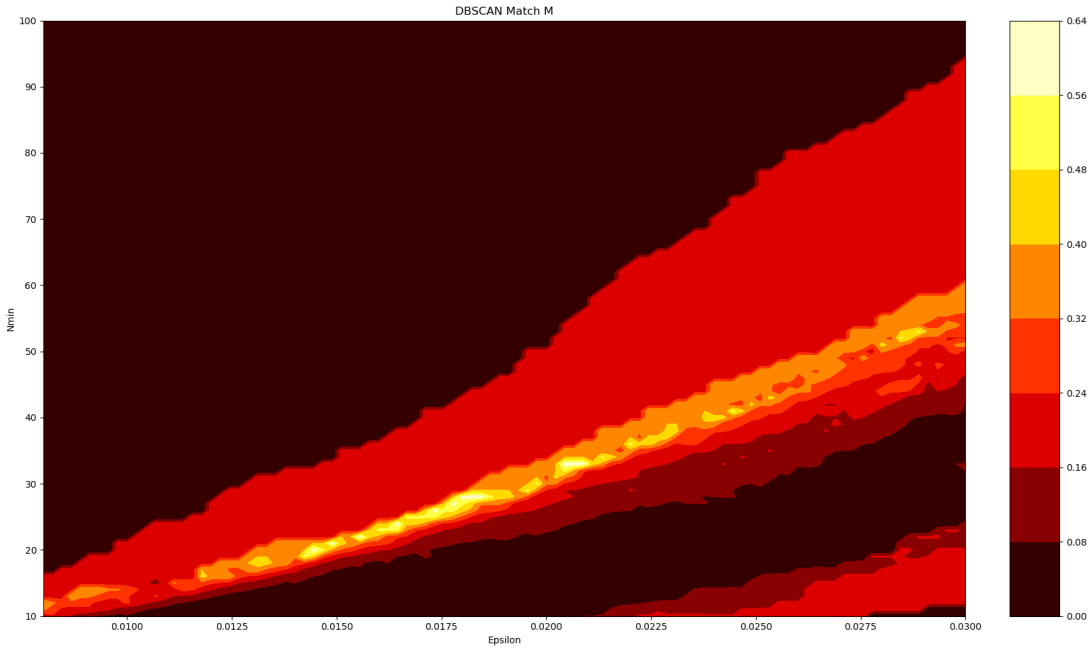

Figura 12: Mapa de líneas de contorno que representa la dependencia de M con respecto de ϵ y $Nmin$ para el caso de filtrado en banda G.

En la Figura 12 podemos ver una representación, mediante líneas de contorno, de la dependencia de la función M con respecto a los parámetros ϵ y $Nmin$. Como se aprecia, se alcanza un máximo de 0,6 cuando el parámetro ϵ se encuentra entre los valores 0,0135 y 0,021, y el parámetro $Nmin$ se encuentra entre los valores 19 y 37 aproximadamente. Estos valores se aprecian mejor en los histogramas representados en las Figuras 13 y 14. Es en este rango cuando el algoritmo encuentra los tres cúmulos estelares registrados en el catálogo de Dias dentro de la región de cielo que se está estudiando. Los cúmulos que encuentra son *FSR 0777*, *Stock 8* y *Kronberg 1*, tal y como se observa en la traza de salida generada por el programa, en la Figura 15.

3.4. Dependencia con el tipo de muestreo

Para comprobar la dependencia con el tipo de muestreo hemos ejecutado un segundo barrido de parámetros, donde no se realiza un filtrado en la banda G, sino que llevamos a cabo un muestreo totalmente aleatorio del 50 % de las estrellas. A diferencia del caso anterior en donde tenemos un filtrado en la banda G previo a un muestreo del 66,025 % aproximadamente. Este segundo barrido se queda con la misma cantidad de estrellas pero obtenidas de forma aleatoria.

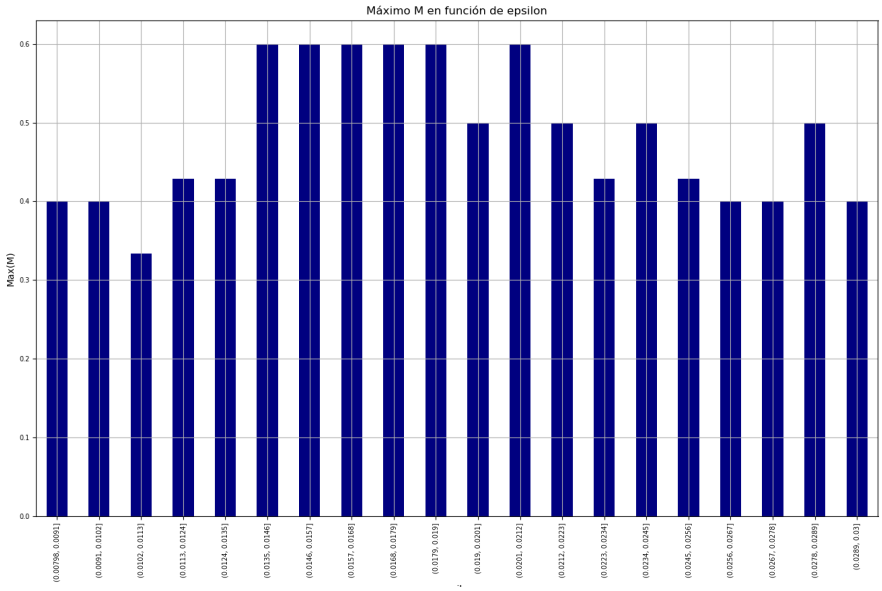

Figura 13: Dependencia de M con respecto de ϵ. En el eje de abscisas se representan distintos intervalos del parámetro ϵ, mientras que en el eje de ordenadas se observa el máximo M obtenido en dicho intervalo (en cada intervalo tenemos diferentes valores de M dependiendo de $Nmin$. Aquí se representa el máximo).

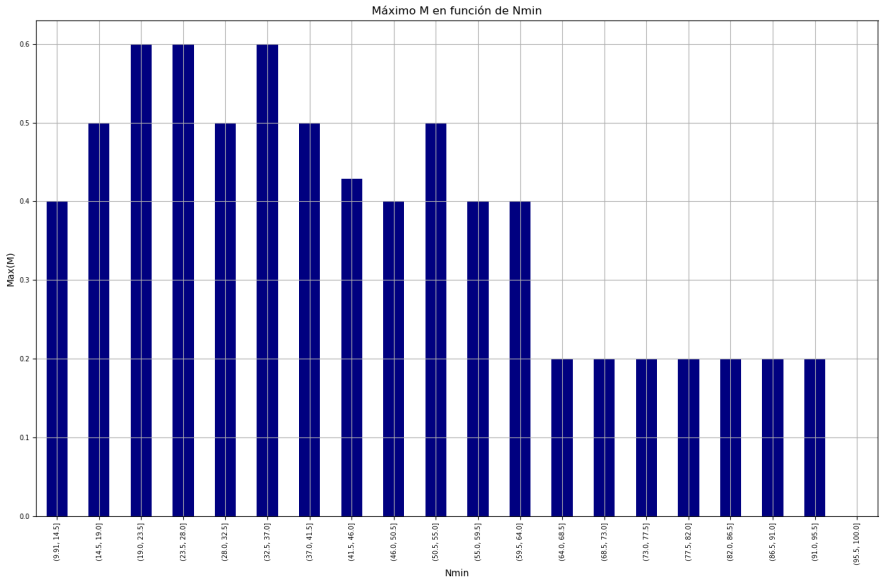

Figura 14: Dependencia de M con respecto de $Nmin$. En el eje de abscisas se representan distintos intervalos del parámetro $Nmin$, mientras que en el eje de ordenadas se observa el máximo M obtenido en dicho intervalo (en cada intervalo tenemos diferentes valores de M dependiendo de ϵ. Aquí se representa el máximo).

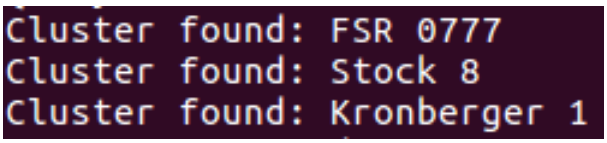

Figura 15: Nombre de cúmulos detectados por el algoritmo para $M = 0,6$ como resultado de la ejecución.

En este caso obtenemos un máximo de 0,5 en la función M, tal y como se observa en la Tabla 2. Aquí, como puede verse en la Figura 16, el algoritmo consigue encontrar hasta cuatro cúmulos, uno de ellos no registrado en el catálogo Dias. Cabe decir que, como se mencionó en secciones anteriores, el catálogo de Dias no tiene necesariamente que estar completo y pueden existir cúmulos no registrados en él. Por lo tanto, aunque la medida de efectividad M en este caso sea menor, no significa que el algoritmo no esté funcionando de manera adecuada. Aquí se necesitaría un estudio en mayor profundidad del nuevo cúmulo detectado para poder realmente descartarlo o no como tal. Los cúmulos detectados por el algoritmo y que están registrados en el catálogo Dias son los mismos que los detectados en el primer barrido: *FSR 0777*, *Stock 8* y *Kronberg 1*, tal y como se observa en la Figura 17.

Como se ha comprobado, cuando se realiza primero un filtrado en la banda G, se obtiene un valor más alto en M que cuando se realiza un muestreo de las estrellas totalmente aleatorio. De hecho, como se puede observar en las Figuras 12 y 20, el rango de parámetros para los cuales se consiguen valores máximos se desplaza ligeramente hacia la derecha y hacia arriba cuando el muestreo es totalmente aleatorio. Esto es debido a que, al realizar un muestreo totalmente aleatorio, estamos eliminando estrellas que pertenecen a cúmulos y, por tanto, haciéndolos menos densos. Al hacerlos menos densos, necesitamos valores más altos de los parámetros ϵ y $Nmin$ para poder detectarlos. Además, filtrando de forma precisa en la banda G, se elimina la mayor parte de las estrellas lejanas de fondo que introducen mucho ruido al conjunto de datos. Eliminando este ruido de forma correcta podemos obtener mejores resultados, por lo que el algoritmo detecta los cúmulos abiertos con mayor exactitud ya que, mediante este filtrado, eliminamos menos estrellas de los cúmulos de manera que su densidad permanece casi intacta y, por tanto, son más fáciles de localizar. Por otro lado, también se observa que la franja de parámetros en donde se presentan valores máximos es más estrecha y plana con el muestreo aleatorio.

La dependencia en este segundo caso con la medida de efectividad M respecto de los parámetros ϵ y $Nmin$ se puede observar tanto en los histogramas de las Figuras 18 y 19, que representan su dependencia de forma aislada, como en la Figura 20, donde se observan las líneas de contorno correspondientes. En este caso, el rango de parámetros en donde se consigue obtener el valor máximo se encuentra entre los valores 0,021 y 0,026 para ϵ, y entre 33 y 50 para $Nmin$ aproximadamente.

Finalmente cabe mencionar que, como se comentó cuando se hablaba del algoritmo DBSCAN, este es sumamente sensible a variaciones en sus parámetros. Por lo tanto, aunque en los casos anteriores solo se han detectado tres de los cinco posibles cúmulos existentes, esto no quiere decir que los otros dos

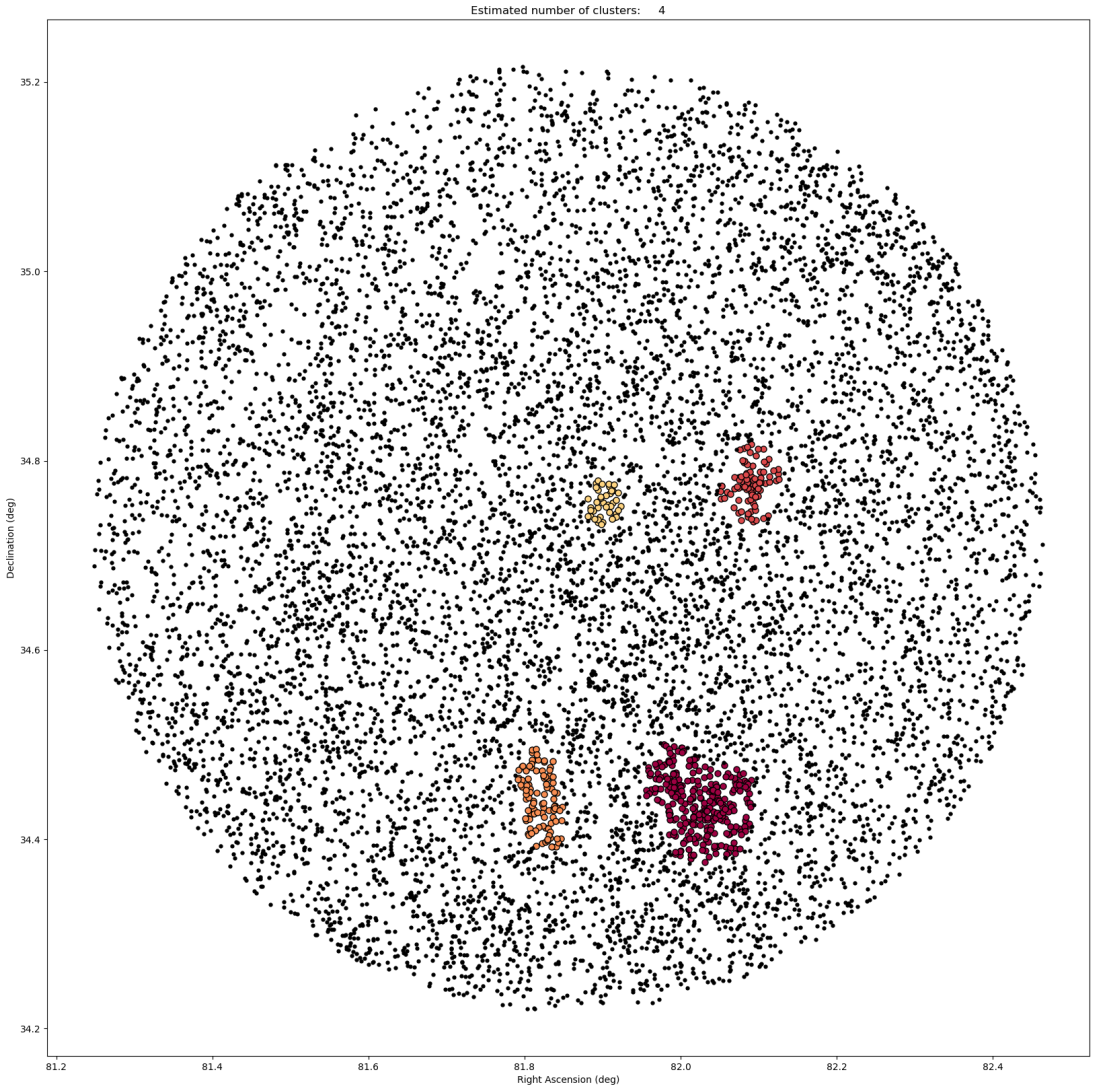

Figura 16: Posiciones de las estrellas en la región del cielo estudiada y resultados para la ejecución sin filtrado en la banda G para el caso de la mejor ejecución $M = 0,5$. En círculos de colores se indican los tres cúmulos encontrados.

```
Cluster found: FSR 0777
Cluster found: Kronberger 1
Cluster found: Stock 8
```

Figura 17: Nombre de cúmulos detectados por el algoritmo para $M = 0,5$ como resultado de la ejecución.

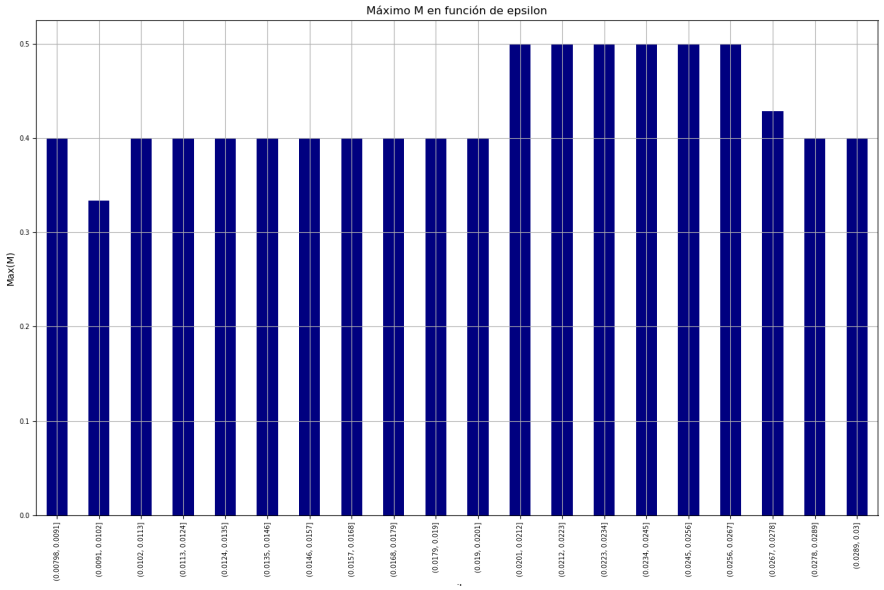

Figura 18: Dependencia de M con respecto de ϵ. En el eje de abscisas se representan distintos intervalos del parámetro ϵ, mientras que en el eje de ordenadas se observa el máximo M obtenido en dicho intervalo (en cada intervalo tenemos diferentes valores de M dependiendo de $Nmin$. Aquí se representa el máximo).

	epsilon	Nmin	M
count	9100	9100	9100
mean	0,019	55	0,1023756682
std	0,0064150347	26,2692944806	0,1310511493
min	0,008	10	0
25 %	0,0135	32	0
50 %	0,019	55	0
75 %	0,0245	78	0,2
max	0,03	100	0,5

Cuadro 2: Tabla de estadísticos para los parámetros ϵ, $Nmin$ y la función M en el caso de la ejecución sin filtrado en la banda G.

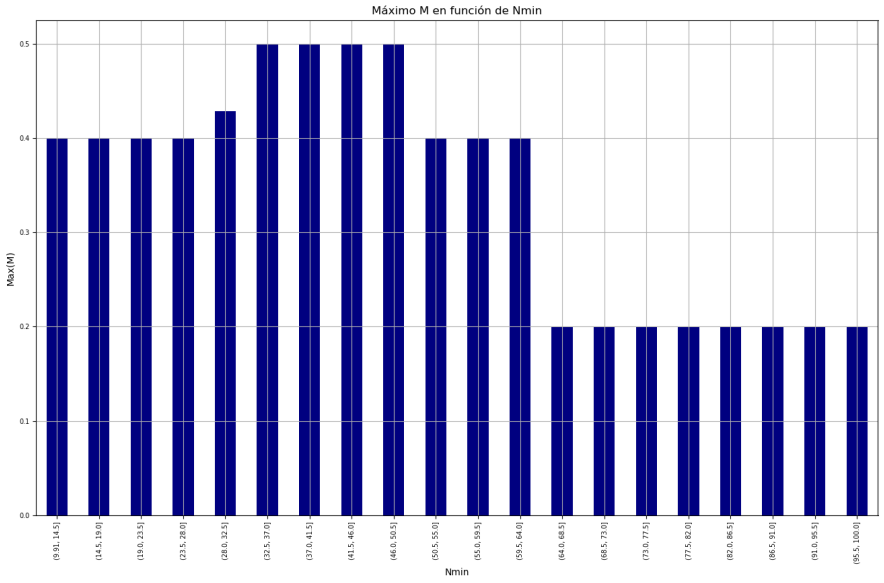

Figura 19: Dependencia de M con respecto de $Nmin$. En el eje de abscisas se representan distintos intervalos del parámetro $Nmin$, mientras que en el eje de ordenadas se observa el máximo M obtenido en dicho intervalo (en cada intervalo tenemos diferentes valores de M dependiendo de ϵ. Aquí se representa el máximo).

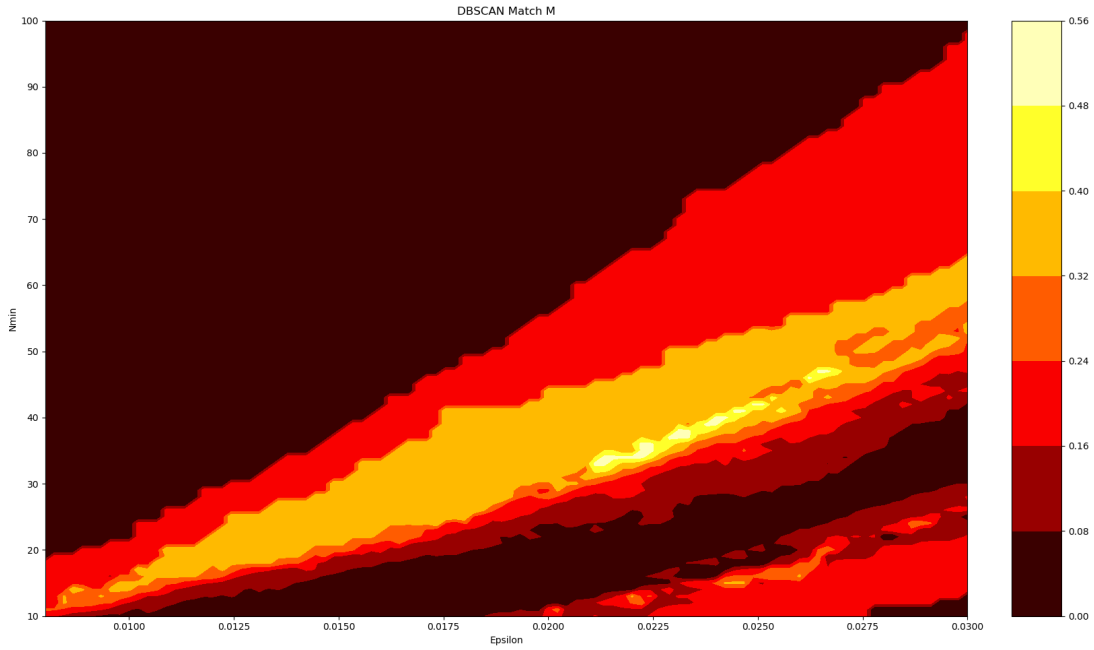

Figura 20: Mapa de líneas de contorno que representa la dependencia de M con respecto de ϵ y $Nmin$ para el caso de no filtrado en banda G.

cúmulos no hayan sido detectados. De hecho, también se consiguen detectar pero con valores de ϵ y $Nmin$ diferentes. Valores para los cuales se detectan otros posibles falsos cúmulos, lo que hace que la efectividad M sea menor. Por tanto, se puede concluir diciendo que no basta con usar únicamente las variables de posición para que DBSCAN localice estos cúmulos abiertos, sino que se necesitarían otras variables físicas, como paralajes y movimientos propios.

4. Conclusiones y trabajo futuro

Se ha evaluado el algoritmo basado en densidad DBSCAN para detectar cúmulos estelares a partir de los datos generados por la misión espacial GAIA. Para ello se ha usado una medida de efectividad o acierto M definida en la Ecuación 2 y se ha realizado un barrido en una malla de valores ϵ-$Nmin$. Se ha demostrado que, eliminando ruido de fondo provocado por estrellas lejanas, se consiguen mejores resultados, ya que se mantiene intacta la densidad de los cúmulos y se reduce la cantidad de datos en memoria. Además se ha comprobado que el principal cuello de botella se produce al crear la matriz de distancias entre cada par de puntos, pero una vez creada, el algoritmo es bastante rápido.

Hemos visto que, para la región de cielo bajo estudio, los valores de parámetros que mejores resultados dan están en el rango 0,0135 a 0,021 para ϵ y 19 a 37 para $Nmin$ cuando hacemos un filtrado en la banda G, y en el rango 0,021 a 0,026 para ϵ y entre 33 a 50 para $Nmin$ en el caso de que hagamos un muestreo totalmente aleatorio. Haría falta un estudio sistemático para analizar la dependencia de estos valores óptimos con otras regiones del cielo. Sin embargo, debemos destacar que siempre es posible implementar una estrategia automatizada que previamente haga un barrido de parámetros y luego, con valores óptimos, produzca un listado de candidatos a cúmulos (aquellos que no están catalogados previamente).

Nuestros resultados indican que, en caso de ser necesario hacer un muestreo para optimizar la ejecución del algoritmo, es el filtrado por magnitud en la banda G el que tiende a arrojar mejores resultados, al eliminar las estrellas más débiles de fondo que causan ruido al algoritmo.

En cualquier caso, se debe tener en cuenta que todos los algoritmos estudiados hasta la fecha detectan sobredensidades, pero no proporcionan información para saber si los objetos identificados están físicamente relacionados con cúmulos estelares. Se necesitaría más información proveniente de otras variables físicas como los movimientos propios, paralajes o luminosidad. Además, como se menciona en [3], no existe un parámetro ϵ universal que nos permita detectar todas las aglomeraciones presentes en una zona de cielo y, por lo tanto, se podría estudiar la posibilidad de usar algoritmos y variables adicionales que confirmen esos cúmulos.

```
SELECT [ ALL | DISTINCT ]
  [ TOP unsigned_integer ]
  { * | { value_expression [ [AS] column_name ] }, ... }
FROM {
  { table_name [ [AS] identifier ] |
  ( SELECT ....) [ [AS] identifier ] |
  table_name [NATURAL] [ INNER | { LEFT | RIGHT | FULL [OUTER] } ] JOIN table_name
  [ON search_condition | USING ( column_name,...) ] }
  , ...}
[ WHERE search_condition ]
[ GROUP BY column_name, ... ]
[ HAVING search_condition ]
[ ORDER BY { column_name | unsigned_integer } [ ASC | DESC
  ],...]
```

Figura 21: Esquema de construcción de una consulta usando el lenguaje ADQL sobre la base de datos de GAIA.

Anexo

A. El lenguaje de consulta ADQL

El lenguaje de consulta astronómico (ADQL, *Astronomical Data Query Language*) es un lenguaje de consulta basado en el estándar SQL. A diferencia de SQL, sin embargo, proporciona una serie de funcionalidades extra que permiten realizar consultas de forma sencilla en bases de datos astronómicas. La sintaxis básica es la misma que para SQL. Una consulta está compuesta por las columnas de la tabla que queremos consultar (SELECT), la tabla o tablas almacenadas (FROM) y las condiciones de consulta (WHERE). Además soporta toda la sintaxis de SQL tal y como se puede ver en la Figura 21.

GAIA proporciona un entorno que nos permite realizar consultas de este tipo además de poder visualizar de forma esquemática las diferentes tablas de las que se compone, tal y como se puede ver en la Figura 22. Esto último nos va a permitir cruzar tablas para comparar y obtener consultas más precisas.

Entre las principales características del lenguaje de consulta ADQL se encuentran las siguientes:

- Funciones trigonométricas
  - SIN(X), COS(X), TAN(X): funciones trigonométricas estándar cuyo argumento debe ser pasado en radianes.
  - ASIN(X), ACOS(X), ATAN(X): funciones trigonométricas inversas estándar.
  - ATAN2(X,Y): proporciona el arco cuya tangente viene dada por y/x.

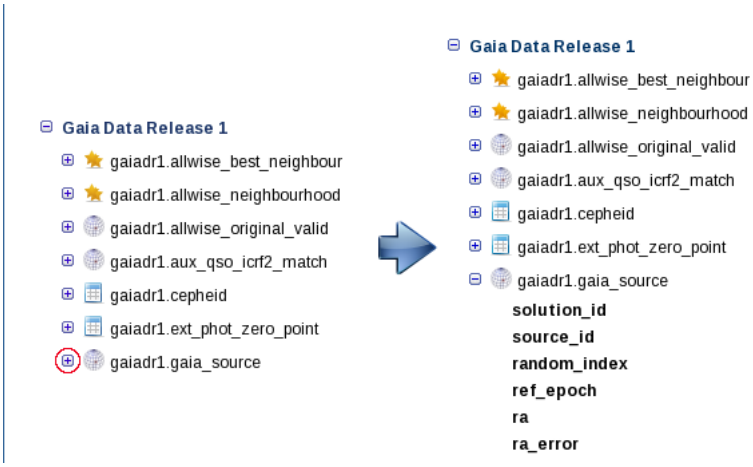

Figura 22: Detalle visual de la estructura de la primera liberación DR1 de datos de la misión GAIA.

  - DEGREES(X): convierte un ángulo de radianes a grados.
  - RADIANS(X): convierte un ángulo de grados a radianes.

- Funciones exponenciales y logarítmicas
  - EXP(X): número de Euler e elevado a la potencia de X.
  - LOG(X): logaritmo en base natural.
  - LOG10(X): logaritmo en base 10.
  - POWER(X,Y): devuelve X elevado a la potencia Y.
  - SQRT(X): raíz cuadrada de X.
- Funciones de truncado y redondeo
  - ROUND(X,n): redondea un número en coma flotante a un número de decimales concretos.
  - FLOOR(X): redondea un número a su entero menor.
  - CEILING(X): redondea un número a su entero mayor.
  - TRUNCATE(X,n): trunca un número X a n decimales.
- Otras funciones
  - ABS(X): valor absoluto de un número.
  - RAND(n): devuelve un número aleatorio entre 0 y 1 usando una semilla n.

    - MOD(X,Y): devuelve el resto de la división de x/y.
    - PI(): devuelve el número PI.

Sin embargo, la principal diferencia de ADQL con respecto a SQL se encuentra en sus funciones geométricas. A fin de construir consultas geométricas que requieren coordenadas esféricas, ADQL define una extensión de SQL con una serie de funciones que nos permiten seleccionar regiones concretas de cielo. Se puede encontrar una lista completa de estas funciones en [17].

En la base de datos GAIA, en concreto, estas funciones se pueden clasificar de la siguiente forma:

- Funciones de tipos de datos. Estas funciones especifican una geometría dada que se inserta a una función de predicado.
    - BOX.
    - CIRCLE.
    - POINT.
    - POLYGON.
- Funciones de predicado. Estas son funciones que se usan en la sentencia WHERE de la consulta y toman como argumento una función de tipo de dato. Estas funciones devuelven un valor 1 (verdadero) o 0 (falso).
    - CONTAINS: comprueba si una geometría (función de tipo de dato) contiene totalmente a otra.
    - INTERSECTS: comprueba si una geometría intersecta con otra.
- Funciones de utilidad. Todas estas son funciones que toman como argumento cualquier geometría y devuelven un valor numérico o de tipo carácter.
    - AREA.
    - COORD1.
    - COORD2.
    - COORDSYS.
    - DISTANCE.

Combinando estas funciones podemos, por ejemplo, realizar una consulta a la base de datos de GAIA de forma que obtengamos las coordenadas de ascensión recta y declinación de todos los puntos contenidos en un círculo de centro en las coordenadas 56,25 grados en ascensión recta, 24 grados en declinación y radio 1 grado de la siguiente forma:

```
SELECT ra, dec
FROM gaiadr2.gaia_source
WHERE CONTAINS(POINT('ICRS',ra,dec), CIRCLE('ICRS',56.25,24.0,1.0))=1
```

En donde *ICRS* define el sistema de coordenadas usado. Para obtener más información consultar [18].

Referencias

[1] Lara Garrido, L.; Introducción a la Física del Cosmos. Departamento de Física Teórica y del Cosmos, Editorial Universidad de Granada

[2] Martin, V. Z.; Handbook of Space Astronomy and Astrophysics. Third Edition. Cambridge University Press, 1990

[3] Castro-Ginard, A.; Jordi, C.; Luri, X.; Julbe, F.; Morvan, M.; Balaguer-Núñez, L.; Cantat-Gaudin, T. A new method for unveiling open clusters in Gaia. New nearby open clusters confirmed by DR2. Astronomy Astrophysics, Volume 618, id.A59. (2018)

[4] https://en.wikipedia.org/wiki/Messier_object

[5] https://heasarc.gsfc.nasa.gov/W3Browse/star-catalog/lyngaclust.html

[6] Mermilliod, Jean-Claude. The Database for Galactic Open Clusters (BDA). Information On-Line Data in Astronomy, edited by Daniel Egret and Miguel A. Albrecht. ISBN 0-7923-3659-3, 1995. Astrophysics and Space Science Library, Vol. 203, p.127.

[7] Dias, W. S.; Alessi, B. S.; Moitinho, A.; Lépine, J. R. D. New catalogue of optically visible open clusters and candidates. Astronomy and Astrophysics, v.389, p.871-873 (2002).

[8] Kharchenko, N. V.; Piskunov, A. E.; Schilbach, E.; Röser, S.; Scholz, R. -D. Global survey of star clusters in the Milky Way. II. The catalogue of basic parameters. Astronomy Astrophysics, Volume 558, id.A53, 8 pp. (2013).

[9] Vicente, B.; Sánchez, N.; Alfaro, E. J. NGC 2548: clumpy spatial and kinematic structure in an intermediate-age Galactic cluster. Monthly Notices of the Royal Astronomical Society, Volume 461, Issue 3, p.2519-2526 (2016).

[10] Schmeja, S. Identifying star clusters in a field: A comparison of different algorithms. Astronomische Nachrichten, Vol.332, Issue 2, p.172 (2011).

[11] Caballero, J. A.; Dinis, L. A revisit to agglomerates of early-type Hipparcos stars. Astronomische Nachrichten, Vol.329, Issue 8, p.801-834. (2008).

[12] http://vizier.cfa.harvard.edu/viz-bin/VizieR-3?-source=l/239/hip$_main$

[13] Prusti, T.; de Bruijne, J. H. J.; Brown, A. G. A.; Vallenari, A.; Babusiaux, C.; Bailer-Jones, C. A. L.; Bastian, U.; Biermann, M.; Evans, D. W.; Eyer, L.; Jansen, F.; Jordi, C.; Klioner, S. A.; Lammers, U.; Lindegren, L.; Luri, X.; Mignard, F.; Milligan, D. J...: The Gaia Mission. Astronomy Astrophysics, Volume 595, id.A1, 36 pp. November 2016

[14] A. G. A. Brown1,*, A. Vallenari2, T. Prusti3, J. H. J. de Bruijne3, C. Babusiaux4,5, C. A. L. Bailer-Jones6, M. Biermann7, D. W. Evans8, L. Eyer9, F. Jansen10, C. Jordi11, S. A. Klioner12...: Gaia Data Release 2: Summary of the contents and survey properties. AA Volume 616, August 2018.

[15] Ester, M.; Kriegel, H. P.; Sander, J.; Xu, X. (1996). Simoudis, E.; Han, J.; Fayyad, U. M. (eds.). A density-based algorithm for discovering clusters in large spatial databases with noise. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96). AAAI Press. pp. 226–231. CiteSeerX 10.1.1.121.9220. ISBN 1-57735-004-9.

[16] https://www.cosmos.esa.int/web/gaia/dr2

[17] http://www.ivoa.net/documents/ADQL/2.0

[18] http://www.ivoa.net/documents/REC/DM/

[19] http://docs.astropy.org/en/stable/coordinates/index.htmlmodule-astropy.coordinates

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

[21] http://vizier.u-strasbg.fr/viz-bin/VizieR-3?-source=B/ocl/clusters