

Máster en Big Data y Data Science

*Computación distribuida para la detección de
sobre-densidades estelares en datos de la
misión espacial Gaia*

Trabajo Fin de Máster

Alumno: : **Femenía Castellá, Bruno**

Dirección: Tacoronte (Tenerife, España)

Tutores Trabajo Fin de Máster: Dr. Blay Serrano, Pere y Dr. Sánchez Doreste, Nésto

Edición abril 2019 a junio 2020

Índice

Resumen	7
Summary	9
1. Introducción.....	11
2. Objetivos.....	13
2.1. Objetivo General.....	13
2.2. Objetivos específicos.....	13
2.3. Estructura del trabajo.....	13
3. Marco teórico.....	15
3.1. Contexto histórico: misiones Hipparcos y Gaia.....	15
3.2. Sobre-densidades estelares en la Galaxia.....	18
3.3. Interés en el estudio de los cúmulos abiertos galácticos.....	20
3.4. Estado del arte en la búsqueda de cúmulos abiertos galácticos.....	22
4. Desarrollo de la infraestructura de computación distribuida. Resultados.....	23
4.1. Punto de partida.....	23
4.2. Estrategia de búsqueda de candidatos con DBSCAN.....	25
4.3. Evaluación de tiempos de ejecución.....	26
4.3.1. Ejecución de RL19 y optimización BF20.....	27
4.4. Montaje de un <i>cluster</i> de máquinas virtuales basado en contenedores Docker y montaje de Hadoop.....	29
4.5. Justificación de la elección de Hadoop.....	37
4.5.1. Re-formateo del código optimizado como una secuencia MapReduce.....	38
4.5.2. Notas sobre <code>mapperTFM.py</code>	40
4.5.3. Notas sobre <code>reducerTFM.py</code> y ejecución YARN.....	41
4.6. Mapeo de (ϵ , N_{pts}) de DBSCAN sobre el 5% del plano de la Galaxia.....	45
4.6.1. Generación de los ficheros de entrada para el escaneo del catálogo de Dias et al. (2020) con MapReduce.....	45
4.6.2. Resultados de la ejecución. Interpretación de los resultados.....	46
5. Conclusiones y Futuros Trabajos.....	50
6. Referencias	51
7. Anexos: códigos Python.....	53
7.1. <code>generate_ip_dias.py</code>	53
7.2. <code>select_ip_DC_files.py</code>	55
7.3. <code>mapperTFM.py</code>	57

7.4.	reducerTFM.py	57
7.5.	process_MapRed.py.....	69
7.6.	Código completo BF20.	75
7.6.1.	DiasCatalog.py.....	75
7.6.2.	main_BF20.py.....	83

Índice de ilustraciones

Figura 1: impresión artística de la sonda Hipparcos con star trails en el fondo. Fuente: https://www.cosmos.esa.int/web/hipparcos/home	15
Figura 2: impresión artística de la sonda Gaia con el plano de la Galaxia en el fondo. Fuente: https://www.cosmos.esa.int/web/gaia	16
Figura 3: Vista panorámica de 360° de la esfera celeste donde se ve con exquisito detalle la estructura del disco de la Galaxia. Fuente: https://www.eso.org/public/images/eso0932a	19
Figura 4: Mapa de la Galaxia mostrando la ubicación de algunos cúmulos abiertos y globulares. Puede observarse como los cúmulos abiertos tienden a distribuirse en la zona del plano de la Galaxia $ b < 20^\circ$, mientras que los globulares se encuentran en la periferia (halo) de la Galaxia. Créditos:ESA/Gaia/DPAC, A. Moitinho & M. Barros (CENTRA – University of Lisbon). Fuente: https://phys.org/news/2017-11-star-cluster-easy-simply-stars.html	20
Figura 5: Diagrama HR mostrando la distribución de las estrellas en función de su tipo espectral, tamaño, edad, Fuente: https://en.wikipedia.org/wiki/Hertzsprung%E2%80%93Russell_diagram#/media/File:HR_Diagram.png	21
Figura 6: comparación de tiempos en la ejecución del original (izquierda) frente a la optimización (derecha) con un incremento de la velocidad de ejecución por un factor 9.3x. Fuente: Elaboración propia.	28
Figura 7: evolución de la métrica M de bondad del algoritmo en función de (ϵ , Npts). Fuente: Elaboración propia.	29
Figura 8: comparación esquemática de la diferencia entre Máquinas Virtuales (VM acrónimo inglés) y contenedores. Fuente: www.docker.com	30
Figura 9: : comparación entre un proceso conceptual, lo que realmente ocurre con un proceso en linux y lo que ocurre al ejecutar procesos en un container. Fuente: https://sites.google.com/site/mytechnicalcollection/cloud-computing/docker/container-vs-process	31
Figura 10. Arriba : script para generar y arrancar los contenedores con el cluster y Hadoop. Abajo izquierda : script simple para una vez los contenedores han sido creados arrancar el cluster. Abajo derecha : script para detener los contenedores. Fuente: elaboración.....	34
Figura 11: con contenedores para cada uno de los nodos del cluster con Hadoop pero los contenedores parados. Se muestran los volúmenes que se comparten y la red entre contenedores. En este estado basta ejecutar desde línea de comando en la terminal <code>./start_hadoop</code>	35
Figura 12: con <code>./start_hadoop.sh</code> levantamos el cluster y se van notificando los nodos activos. En este caso, al finalizar el script volvemos a la terminal del host donde podemos comprobar que los contenedores están activos y los puertos que se exponen. Fuente: elaboración propia.	35
Figura 13: vista general a través de la interfaz web de un cluster Hadoop con 9 datanodes levantado en la máquina local descrita en la sección 4.3. Fuente: elaboración propia.	36

Figura 14: : ilustración de dos sencillos scripts mapper y reducer para la ejecución distribuida de una tarea más compleja (main_fake) usando YARN con Hadoop. Fuente: elaboración propia.	39
Figura 15: simulación en línea de comando de cómo actuaría el código lanzado desde Hadoop. Además de ilustrativo, el uso de una secuencia como la resaltada es altamente aconsejable para poder depurar de manera eficiente posibles errores en las rutinas mapper y/o reducer. Fuente: elaboración propia.	40
Figura 16: PROBLEMA GRAVE: imposibilidad de ejecutar una consulta a Gaia DR2 desde el contenedor. Fuente: elaboración propia.	42
Figura 17: captura de pantalla del DataFrame en Pandas mostrando los resultados de aproximadamente la mitad de las ejecuciones para muestrear (ϵ , N_{pts}) en función de (l , b) en la región cercana al centro Galáctico. Cada entrada en la tabla corresponde a una secuencia de 9100 iteraciones donde se muestrea el espacio de parámetros de DBSCAN. Fuente: elaboración propia.....	47
Figura 18: parámetro M usado como figura de mérito para determinar las mejores combinaciones (ϵ , N_{pts}) En esta figura se muestra cómo varía M en función de la posición del cielo considerada en base a las ejecuciones mostradas en la Figura 17.	48
Figura 19: mapa del parámetro ϵ como función de su posición en el plano de la Galaxia en base a las ejecuciones mostradas en la Figura 17.....	48
Figura 20: mapa del parámetro M como función de su posición en el plano de la Galaxia en base a las ejecuciones mostradas en la Figura 17.....	49

Índice de tablas

Tabla 1: Visión general de GDR2 vs GDR1 Fuente:
<https://www.cosmos.esa.int/web/gaia/dr2> 17

Tabla 2: distribución acumulada del porcentaje de fuentes en Gaia DR2 en función de su magnitud en banda G. Fuente: <https://www.cosmos.esa.int/web/gaia/dr2> 18

Resumen

El objetivo del TFM en López (2019), la presente tesis, y posibles futuros trabajos es identificar candidatos a cúmulos estelares abiertos no catalogados.

El alcance del presente trabajo es habilitar una infraestructura que permita el procesado masivo de datos de la misión espacial Gaia (*Gaia Collaboration*, 2016) para identificar candidatos a sobre-densidades. Un estudio posterior usaría como punto de partida dichos candidatos y procesarlos para dirimir con técnicas de IA (aprendizaje supervisado) el tipo de sobre-densidad al que corresponden: cúmulos abiertos, asociaciones estelares, cúmulos globulares o falsos positivos.

El presente Trabajo de Fin de Máster (TFM) se basa en el TFM presentado por D. Rafael López (2019) en la edición de abril 2018 a abril 2019 bajo la supervisión de los mismo tutores que en este trabajo, a saber, Dr. Pere Blay Serrano y Dr. Néstor Sánchez Doreste. Sin embargo, la implementación en López (2019) (referida como **RL19** a partir de ahora) no resulta viable dado el elevado tiempo de computación que requiere.

El objetivo fundamental del presente TFM es el de resolver dicho problema y ser capaz de proporcionar una versión del código desplegada sobre una infraestructura que sea capaz de hacer uso práctico del algoritmo de búsqueda. Para ello, en este trabajo se ha procedido de tres maneras paralelas:

1. Estudio detallado del código original localizando cuellos de botella en el código; se ha proporcionado una versión optimizada que incrementa la velocidad de ejecución del código por **un factor ~10.2**.
2. Implementación y despliegue de un **cluster**¹ de máquinas usando Docker donde se ha implementado Hadoop. Debido a las restricciones de recursos del *hardware* disponible, el *cluster* está formado hasta 11 nodos: 9 son nodos esclavos donde se ejecuta el cálculo distribuido.
3. Despliegue sobre Hadoop de una nueva versión optimizada y modificada para ser usada en procesos por lotes usando MapReduce.

En definitiva, sobre la misma máquina de uso personal se ha ganado un **factor ~70-90** en la ejecución implementando **optimización** y **cálculo distribuido** con respecto a la ejecución original en RL19.

Dada la característica inherente de escasa labilidad horizontal de nuestra solución basada en despliegue de un *cluster* con Docker y Hadoop y el hecho de la posibilidad de exportar nuestra plataforma a soluciones comerciales de *clusters* Hadoop tales como, entre

¹A día de hoy la Real Academia Española no incorpora (al menos en su versión en línea) el término *clúster* ni *cluster* y uno debe considerarse un barbarismo y el otro un extranjerismo. Sin embargo, se trata de una grafía habitualmente en tanto en ámbitos genéricos como en especializados y se ha dado uso usado desde hace varias décadas. De acuerdo a la Fundación Fundeu (www.fundeu.es), el término *clúster* ya ha sido recogido en Diccionarios en español y su uso resulta legítimo. Lo mismo ocurre con muchas palabras técnicas para las cuales encontrar la contrapartida en español resulta no factible o, cuando menos, extraño y/o engorroso. En este trabajo se opta por dar el término inglés y se evidencia marcándolo en cursiva.

otros, Cloudera-Hortonworks², AWS EMR³, Microsoft Azure HDInsight⁴ ... como Infraestructuras como Servicios (**IaaS**, acrónimo del inglés “Infrastructure as a Service”) la estrategia aquí planteada abre la posibilidad a desplegar en un *cluster* comercial con cientos de nodos y poder ejecutar el algoritmo con el objetivo de rastrear grandes zonas del cielo para la búsqueda de sobre-densidades.

Como aplicación demostrativa se han lanzado un conjunto de ejecuciones masivas para generar un mapa de parámetros pseudo-óptimos de DBSCAN en ~5 % del plano de la Galaxia. Eventualmente, este mapeo de parámetros se plantea en un trabajo futuro (y no parte de este TFM) para la búsqueda de sobre-densidades en una fracción importante del plano de la Galaxia.

²<https://www.cloudera.com/products/hdp.html>

³<https://aws.amazon.com/es/emr/>

⁴<https://azure.microsoft.com/en-us/services/hdinsight/>

Summary

The ultimate goal of this work, the previous Master project this work starts from (López, 2019) and possible upcoming projects is the identification of new candidates of open clusters.

The scope in this project is the massive processing of data from the ESA spacial satellite Gaia mission (Gaia Collaboration, 2016) to identify stellar over densities in the Galaxy. A later work(s) should use those candidates as starting points and deploy AI techniques (e.g. supervised learning) to determine whether the identified candidates are indeed open cluster, stellar associations, globular clusters or false positives.

This MSc thesis takes as the starting point the thesis defended in June 2019 by Mr. Rafael López under the supervision of the same mentors in this project, namely, Dr. Pere Blay Serrano y Dr. Néstor Sánchez Doreste. However, the implementation in López (2019) (hereafter **RL19**) is not feasible in terms of computational requirements.

Therefore the main goal of the this project was to solve the situation by providing a version of the and deploy it into a framework and/or infrastructure allowing a practical use of the algorithm. This has been achieved by working on three parallel strategies:

1. The original code was studied in detailed to identify bottlenecks causing excessive delays in CPU time. This was successfully achieved **and a performance gain factor of 10.2 has been accomplished.**
2. A Docker-based cluster was deployed. Due to the hardware restrictions during the execution of this project (personal laptop) the deployed cluster takes up 11 nodes where 9 of them are the slaving computing nodes.
3. Over this Docker-based virtual machine cluster, a Hadoop system was mounted and the optimized code underwent an additional optimization and modification to be used in a MapReduce scheme with batch processing.

In summary, over the same personal laptop on which this Master thesis has been conducted, a final **performance gain of ~90** has been achieved by **optimizing** the original code in RL19 and implementing a **distributed computing** infrastructure

Given the inherent characteristics of horizontal scalability of Dockers and Hadoop-based approaches and the straightforward export of the solution in this work into commercially available IaaS such as Cloudera-Hortonworks⁵, AWS EMR⁶, Microsoft Azure HDInsight⁷ ... it is now feasible with little effort to perform the search for over-densities over clusters with hundreds of slave computing nodes. This opens up the possibility to achieve gain factor substantially larger than the ones with my personal laptop.

⁵<https://www.cloudera.com/products/hdp.html>

⁶<https://aws.amazon.com/es/emr/>

⁷<https://azure.microsoft.com/en-us/services/hdinsight/>

The original case of interest is the identification of candidates of open clusters which happen mostly on the Galactic plane. With the solution implemented in this TFM and to show its capabilities a set of massive with the purpose of creating creating a map of pseudo-optimal parameters for the DBSCAN clustering over a 5% of the plane of the Galaxy. Eventually, in a future work, we plan to make use of such maps to search for stellar over-densities over a mosaic covering a significant fraction of the Galactic Plane (this not in this project).

1. Introducción.

El concepto de *Big Data* en Astronomía no es nada nuevo: ya son años desde que la comunidad astronómica ha acumulado ingentes de ingentes cantidades de datos cuyo análisis e interpretación requiere de técnicas de *Big Data*.. Posiblemente, una de las primeras inclusiones del concepto de *Big Data* en astronomía sea el desarrollo del Observatorio Virtual (VO, acrónimo de *Virtual Observatory*) por Szalay y Gray (2001).

En 2015 Zhang y Zhao ya ponían de manifiesto el tremendo aumento en la cantidad de datos y ponían el contexto de nueva situación comparando la situación a mitad de los 2000s cuando, por ejemplo, el *Sloan Digital Sky Survey* (SDSS) con unos 40 TB de datos se consideraba uno de los mayores y más completos *surveys* con la nueva situación en cuanto a cantidad de datos que se esperaba de *surveys* que se esperan operativos durante la década 2020-2030 tales como el *Large Synoptic Survey Telescope* (LSST) con el que se estima unos ~200 PB de datos estimados o, aún más significativo en cuanto a la cantidad de datos que se prevé con el *Square Kilometer Array* (SKA): 4.6 EB.

Cabe resaltar que el trabajo de Zhang y Zhao (2015) fue publicado poco después de la plena entrada en funcionamiento a mitad de 2013 de ALMA y poco después de la primera luz científica de Gaia, la nueva misión astrométrica de la *European Space Agency* (ESA). Si bien ninguna de las dos misiones ALMA y Gaia se encuadran en estrategias de *surveys*, las cantidades de datos que están proporcionando resultan descomunales.

En Szalay y Gray (2001), en una especie de ley de Moore aplicada a astronomía, ya se constataba la tendencia que mientras la superficie colectora de los telescopios terrestres y espaciales se dobla cada 25 años, el número de píxeles en detectores y sensores (tales como CCDs o CMOS) se dobla cada 2 años.

Gaia no solo ha confirmado dicha tendencia sino que la ha sobrepasado con su mosaico de 106 CCDs que proporcionan imágenes de ~ 940 megapíxeles de manera constante 24/7. Para poner en contexto el volumen masivo de datos que proporciona Gaia basta pensar que en su segunda distribución de datos (Gaia DR2, de Data Release #2) solo los valores astrométricos de los objetos identificados en las imágenes que adquiere y envía a Tierra suponen ya cerca de 500 GB de datos y dado que DR2 corresponde a los dos primeros años de operación 2014-2016 y la misión se planea operativa hasta al menos 2022, es de esperar que dicha cifra se incremente considerablemente.

En los últimos años, y tras la publicación del Gaia DR2, se está observando un incremento notable actividad en programas tales como la identificación y caracterización de cúmulos abiertos. Esta tendencia actual persistirá en tanto en cuanto se sigan proporcionado nuevos DRs y los astrónomos sean capaces de desarrollar algoritmos más veloces y eficientes en la detección de objetos que, hasta Gaia, se contaban en apenas un par de miles.

Es en este contexto en el que se encuadra esta tesis de máster. En RL19 se desarrolla y proporciona un algoritmo basado en la detección de sobre-densidades en campos estelares con el objetivo de ser aplicado de manera masiva al catálogo de fuentes generada por la misión espacial Gaia. Sin embargo, la posibilidad de usar dicho algoritmo no es viable dado el excesivo tiempo de computación necesario que se requeriría.

El objetivo de esta tesis es el de desarrollar una infraestructura que, partiendo de dicho algoritmo, sea competitiva en la búsqueda de posibles candidatos y permita explotar científicamente el uso de dicho algoritmo en los datos de Gaia DR2.

2. Objetivos.

2.1. Objetivo General.

Partiendo de la implementación en Python en RL19 y dado los excesivos tiempos de ejecución de dicho código, se plantea montar una infraestructura típica de *Big Data* que permita la explotación científica del código.

2.2. Objetivos específicos.

Partimos del código RL19 que, pese a su adecuada estructura en clases y paquetes, no resulta de utilidad práctica por los excesivos tiempos de ejecución requeridos incluso para el tratamiento de pequeñas regiones de cielo. Sin embargo, el objetivo último que se persigue es analizar grandes secciones de la esfera celeste para la búsqueda de cúmulos abiertos. Los objetivos consisten en:

- Re-escritura del código original con el fin de acelerar la ejecución en una máquina *stand-alone*.
- Implementación de un *cluster* de varias máquinas donde poder ejecutar computación distribuida.
- Despliegue de una estructura que permita la computación distribuida sobre el *cluster* previamente habilitado.
- Proporcionar un ejemplo de las capacidades de la nueva implementación en un entorno de computación distribuida con una ejecución masiva que en su versión original RL19 no hubiese podido abordar de manera eficiente.

2.3. Estructura del trabajo.

Siguiendo las pautas establecidas en la Guía Didáctica se ha estructurado este Trabajo de Fin de Máster (TFM) de la siguiente manera:

- Sección 1: breve introducción al problema a atacar, con énfasis en lo apropiado de metodologías de *Big Data* y, en concreto, de computación distribuida para abordar el problema de ETL en una base específica de datos (misión Gaia, de plena actualidad) y un código competitivo que permita extraer conocimiento de dichos datos.
- Sección 2 (presente sección): se delimitan el objetivo general e hitos (objetivos específicos) a conseguir con este TFM.
- Sección 3: se presenta el caso científico a abordar, primero desde un aspecto histórico en lo relativo a misiones astrométricas espaciales Europeas para seguir con una justificación del porqué del interés en el tipo de objetos astronómicos

que se desean poder detectar. Se concluye esta sección proporcionando un breve repaso del “estado de arte” en este campo científico.

- Sección 4: se presentan las herramientas, se discuten los retos y problemas y las soluciones seguidas para conseguir los objetivos y se documenta el trabajo realizado en este TFM. Se empieza proporcionando una descripción detallada del *software* de partida y desde el punto de vista del usuario: los aspectos de programación. Se pone de manifiesto la inviabilidad de usar RL19 por motivos de tiempos de computación requerida y se presenta una nueva versión optimizada del código: **BF20**. Continuamos presentando las alternativas escogidas para montar una infraestructura de cálculo distribuida en la que desplegar de manera efectiva BF20, así como los cambios requeridos en BF20 y los parámetros a optimizar en la configuración de la solución de infraestructura distribuida escogida (Hadoop). Como demostración de la nueva herramienta desplegada en un sistema Hadoop con computación distribuida por lotes, se ejecuta un procesamiento masivo sobre un ~5% del plano de la Galaxia donde disponemos del catálogo de Dias et al. (2002) como conjunto de datos para realizar catálogos sobre los que realizar un pseudo-entrenamiento del código BF20 sobre una infraestructura de computación distribuida.
- Sección 5: conclusiones, futura mejoras. Futuros trabajos y colaboraciones.
- Anexo: donde se proporcionan los códigos utilizados en este TFM. Se proporciona también el *link* a un repositorio donde se han depositado el código, esta Tesis y una versión en inglés de la misma.

3. Marco teórico.

3.1. Contexto histórico: misiones Hipparcos y Gaia.

Gaia⁸ es una misión espacial de la *European Space Agency* (ESA) lanzada en 2013, con una vida operativa estimada hasta 2022 y que fue concebida como una misión espacial sucesora de la misión de la ESA Hipparcos⁹ (Perryman et al., 1986; Høg, 2018), lanzada en 1989 y que con solo 3.5 años de operación revolucionó nuestro conocimiento del Universo y, más concretamente, de la Galaxia y vecindad solar. En la Figura 1 se muestra una impresión artística de Hipparcos.

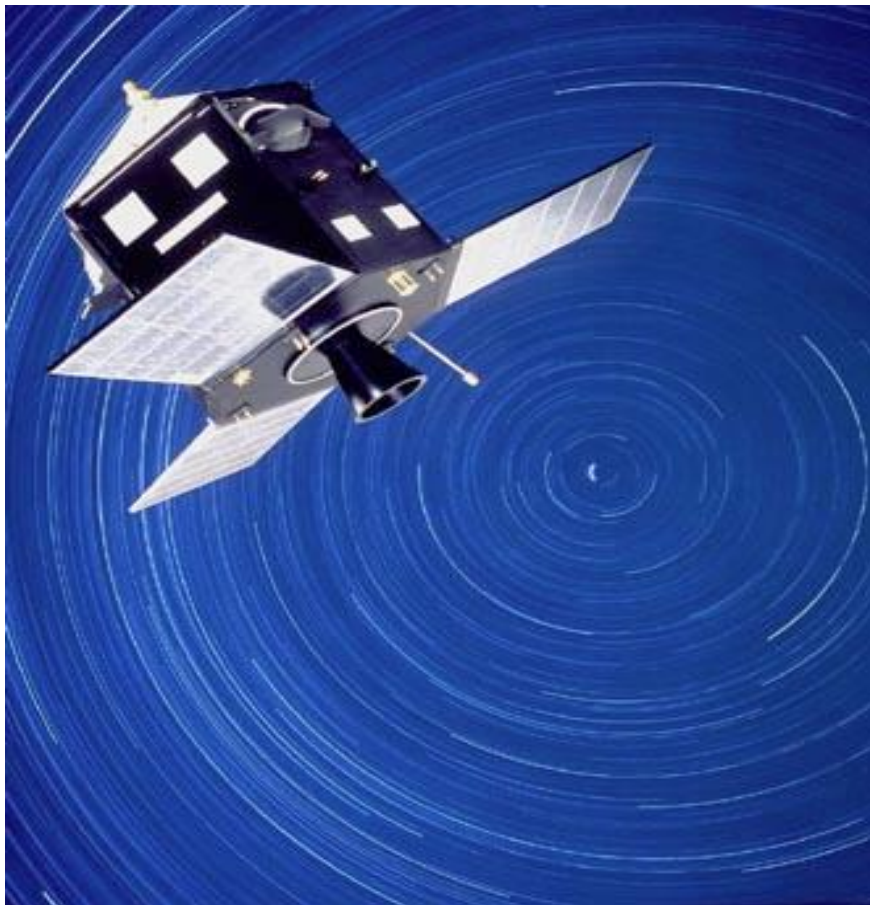


Figura 1: impresión artística de la sonda Hipparcos con star trails en el fondo. Fuente: <https://www.cosmos.esa.int/web/hipparcos/home>

El éxito e importancia de Hipparcos en la astronomía moderna puede ponerse en contexto si consideramos lo limitados en cuanto al número de objetos catalogados y, sobre todo, faltos de precisión astrométrica por culpa del efector perturbador de la turbulencia atmosférica de los catálogos estelares disponibles hasta dicha misión y por

⁸<https://www.cosmos.esa.int/web/gaia>

⁹<https://www.cosmos.esa.int/web/hipparcos>

otro lado, el impacto que en la astronomía observacional tiene el conocimiento preciso de la distribución y movimientos precisos de los objetos estelares.

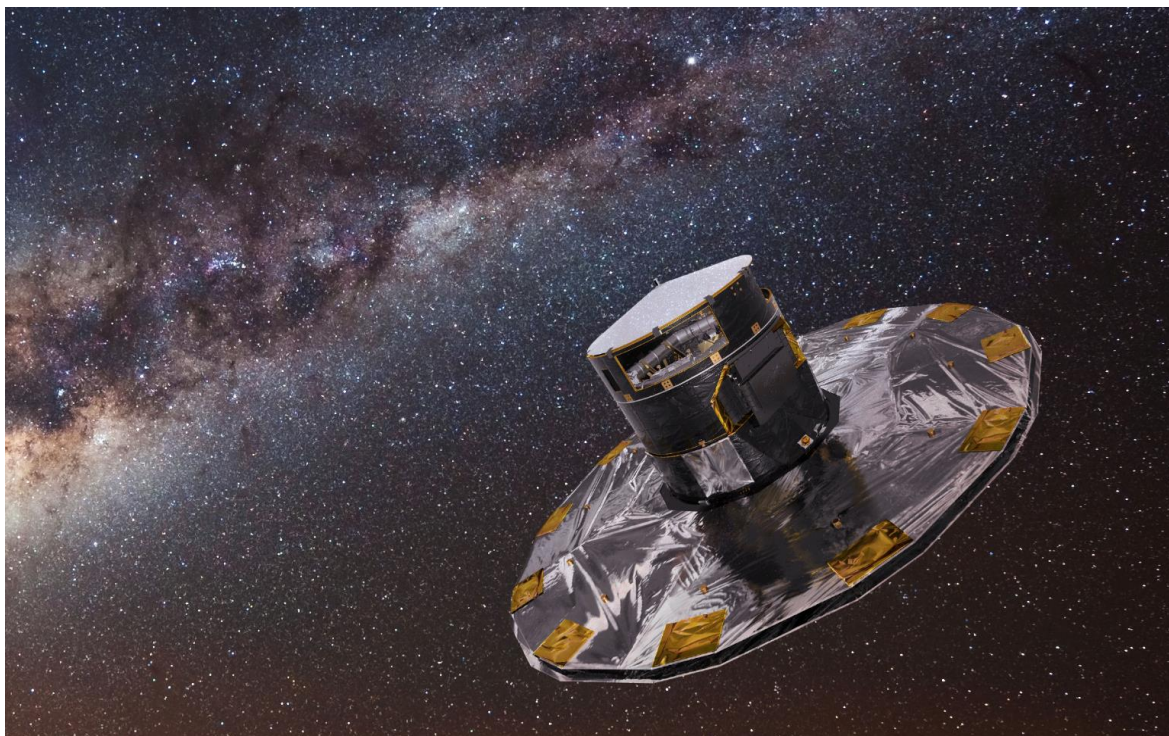


Figura 2: impresión artística de la sonda Gaia con el plano de la Galaxia en en el fondo. Fuente: <https://www.cosmos.esa.int/web/gaia>

Hipparcos proporcionó a la comunidad astronómica un catálogo estelar de la Galaxia con unos 120000 objetos estelares con una precisión astrométrica de 1 mas¹⁰ y con una magnitud¹¹ límite de ~13. No obstante el éxito de Hipparcos¹² debemos considerar que las 120000 estrellas catalogadas representan una fracción, estadísticamente hablando, insignificante de los aproximadamente $(2.5 \pm 1.5) \times 10^{11}$ objetos estelares que conforman la Galaxia, es decir, Hipparcos catalogó algo menos del 0.00005% de los objetos en nuestra Galaxia. Es en este contexto cuando surge el concepto de Gaia¹³, ver en Figura 2 una impresión artística. Gaia fue concebida como una nueva misión astronómica de la ESA con mayor resolución y mayor profundidad en magnitudes que las alcanzadas por Hipparcos. En la actualidad, Gaia que sigue en activo, ya ha

¹⁰La precisión astrométrica la definimos a partir del ángulo más pequeño que se es capaz de resolver. El término **mas** es la abreviación de mili-arcosegundo (*milli-arsecond* en inglés) y corresponde a una milésima parte de un segundo de arco; dicho de otra manera, una zona de cielo que subtiende un ángulo de 1 grado contiene $60 \times 60 \times 1000 = 3600000$ mas (en un grado: 60 minutos, cada minuto 60 segundos, cada segundo 1000 mas)

¹¹Para el lector no familiarizado con el término de magnitud en astronomía, indicar que la magnitud es una medida logarítmica del brillo de un objeto. A mayor valor de magnitud más débil es el objeto. Ver: [https://en.wikipedia.org/wiki/Magnitude_\(astronomy\)](https://en.wikipedia.org/wiki/Magnitude_(astronomy))

¹²El objetivo al planificar Hipparcos era de 100000 estrellas con una resolución de 2 mas.

¹³Además de la referencia bibliográfica, véase también: <https://sci.esa.int/web/gaia>

proporcionado en su segunda y más reciente entrega Gaia DR2¹⁴ (Gaia Collaboration et al., 2018) datos de casi 1.7×10^9 estrellas (aproximadamente un 0.68% del total de la Galaxia) tal y como se muestra en las siguientes tablas al comparar la evolución entre los “Data Releases” DR2 y DR1 .

Tabla 1: Visión general de GDR2 vs GDR1 Fuente: <https://www.cosmos.esa.int/web/gaia/dr2>

	# sources in DR2	# sources in 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 GBP-band photometry	1,381,964,755	-
Sources with mean GRP-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 (Teff)	161,497,595	-
Extinction (AG) and reddening (E(GBP-GRP))	87,733,672	-
Sources with radius and luminosity	76,956,778	-

Ya se ha mencionado que tanto Hipparcos como Gaia son misiones astrométricas, es decir, miden con extrema precisión las posiciones, paralajes y movimientos propios de los objetos catalogados. El número de casos científicos que se benefician de estos datos de gran calidad astrométrica es enorme y abarca muchas disciplinas en astronomía y casos científicos entre los que podemos mencionar:

- Censos estelares.
- Estudios cinemáticos y dinámicos de la Galaxia.
- Estudios de la estructura de la Galaxia.
- Detección de exo-planetas y sistemas planetarios.

¹⁴Gaia DR2 representa el total de datos recogidos entre el 25 de Julio de 2014 y el 23 de Mayo de 2016.

- Estudios de sobre-densidades estelares: asociaciones estelares, **cúmulos abiertos**, cúmulos globulares.
- Evolución y formación estelar.
- Fuentes extra-galácticas.
- Identificación de NEOs (*Near Earth Objects*).

Tabla 2: distribución acumulada del porcentaje de fuentes en Gaia DR2 en función de su magnitud en banda G. Fuente: <https://www.cosmos.esa.int/web/gaia/dr2>

Percentile	All	5-parameter	2-parameter
0.135%	11.6	11.4	15.3
2.275%	15.0	14.7	18.5
15.866%	17.8	17.4	19.8
50%	19.6	19.3	20.6
84.134%	20.6	20.3	21.0
97.725%	21.0	20.8	21.2
99.865%	21.3	20.9	21.4

3.2. Sobre-densidades estelares en la Galaxia.

Basta observar el cielo en una noche cualquiera para percatarse inmediatamente que el cielo presenta una distribución no uniforme de objetos estelares y la pregunta inmediata es por qué no uniforme o incluso uniformemente constante. Es lo que se conoce por paradoja de Olber¹⁵, que también puede re-formularse de como *¿por qué es oscura la noche?* La paradoja de Olber ha jugado un papel importante en el desarrollo de la astronomía (más concretamente en cosmología a la hora de descartar un modelo infinito y uniforme de universo) y filosofía. Sabemos (porque lo vemos a simple vista) que en escalas planetarias, galáctica e intergalácticas el Universo tal como lo observamos en el presente es muy inhomogéneo y con grandes cambios de densidad y estructura; a escalas cosmológicas y en especial a edades tempranas del universo (según el modelo estándar cosmológico) el universo era enormemente uniforme con cambios de densidad/energía de apenas el 0.001%.

¹⁵Para el lector interesado en la paradoja de Olber se recomiendan, ente muchas otras, en Wikipedia: https://en.wikipedia.org/wiki/Olbers%27_paradox o en la entrada del *American Museum of Natural History*: <https://www.amnh.org/exhibitions/journey-to-the-stars/educator-resources/stars/olbers-paradox>

Como se evidencia en la Figura 3 en nuestra época hay gran estructura nuestra Galaxia; vemos multitud de detalles. Nuestra Galaxia está constituida por grandes cantidades de gas y centenares de miles de millones de estrellas muchas de las cuales se agrupan en sobre-densidades; asociaciones estelares, **cúmulos abiertos** y cúmulos globulares.

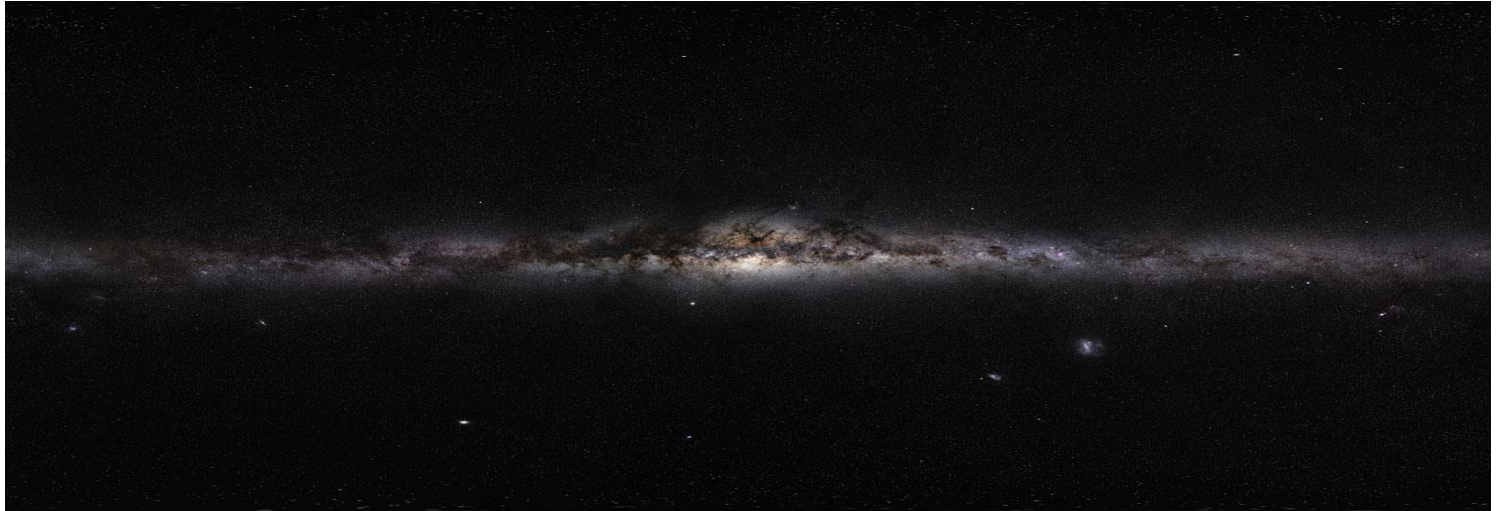


Figura 3: Vista panorámica de 360° de la esfera celeste donde se ve con exquisito detalle la estructura del disco de la Galaxia. Fuente: <https://www.eso.org/public/images/eso0932a>.

Nota: se exceden los márgenes laterales para una correcta visualización de la imagen.

Las **asociaciones de estrellas** son conjuntos con pocas decenas de estrellas muy jóvenes, nacidas de una misma (pequeña) nube molecular distribuidas sobre grandes regiones angulares del cielo (y por lo tanto cercanas a nuestra ubicación en la Galaxia). Su distribución angular extensa y su relativo número de miembros implican valores muy bajos de sobre-densidad y resultan difíciles de detectar y caracterizar por lo generalmente a través del hecho de que todas las estrellas de una misma asociación presentan movimientos propios muy similares. Las asociaciones estelares tienden a disolverse ya que la atracción gravitatoria de las estrellas que forman la asociación no es suficientemente intensa para mantener el grupo durante escalas temporales significativas en el contexto de la evolución de la Galaxia. Por último resulta relevante mencionar que las asociaciones estelares se ubican principalmente en el plano de la Galaxia (ver Figura 4).

Los **cúmulos abiertos** son agrupaciones de hasta unos pocos miles de estrellas; también formados a partir de una misma nube molecular por lo que todas las estrellas muestran edades similares si bien, dada la presencia de miles de estrellas, existe una gran variedad en los distintos tipos de estrellas (lo que en el argot astronómico conocemos por tipo espectral de la estrella). Los cúmulos abiertos presentan fuerzas gravitatorias más intensas que en el caso de las asociaciones estelares pero tienden también a disolverse aunque a escalas temporales mucho mayores que en el caso de las asociaciones estelares y, al igual que éstas, se ubican en el plano de la Galaxia (ver Figura 4).

1. Como se ha mencionado son conjuntos con números suficientemente elevados de miembros que comparten origen y edad.
2. Se encuentran en el plano de la Galaxia, algunos de ellos suficientemente cercanos para poder resolver individualmente y con gran precisión a gran parte de sus miembros.
3. El origen común y al mismo tiempo la diversidad de sus tipos espectrales convierten a los cúmulos abiertos en laboratorios idóneos donde poder estudiar cómo evolucionan las estrellas en función de su tipo espectral; es decir son laboratorios perfectos donde testear modelos de evolución estelar a través, por ejemplo, del estudio y ajuste de modelos a las isócronas en diagramas Hertzsprung–Russell como el que se muestra en la Figura 5.

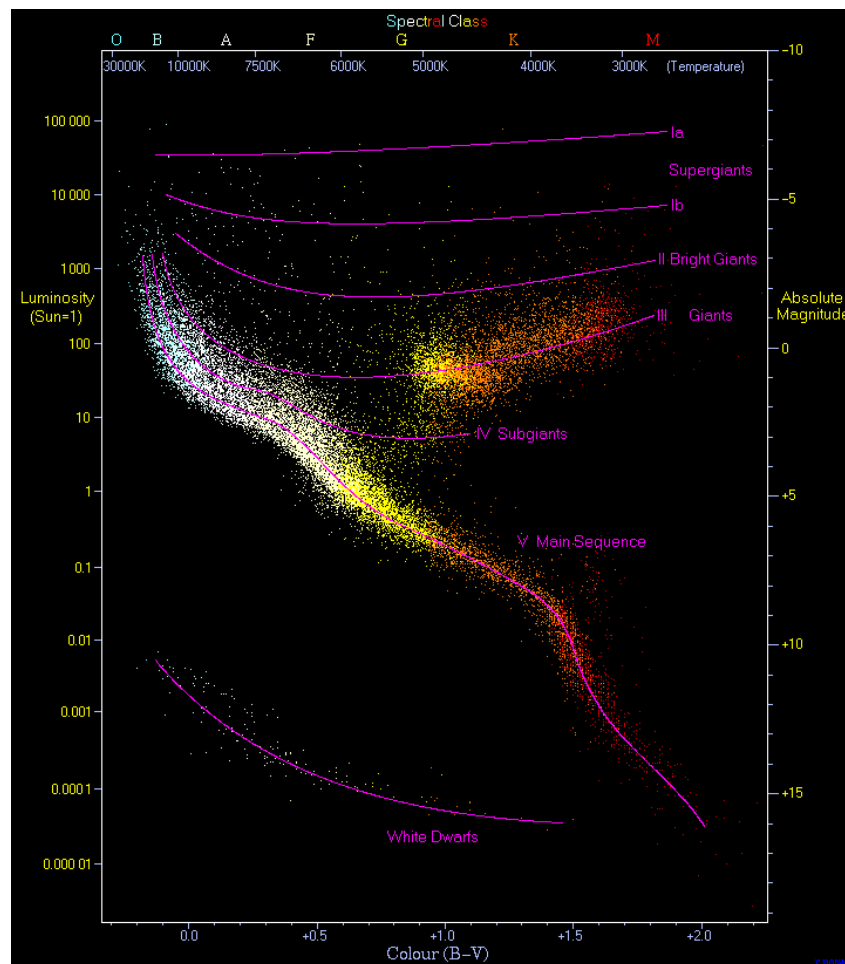


Figura 5: Diagrama HR mostrando la distribución de las estrellas en función de su tipo espectral, tamaño, edad, Fuente:

https://en.wikipedia.org/wiki/Hertzsprung%E2%80%93Russell_diagram#/media/File:HRDiagram.png

4. En el caso de cúmulos estelares jóvenes, que suelen localizarse en los brazos espirales donde hay mayores posibilidades de encontrar todavía nubes de gas y polvo donde formar los cúmulos, proporcionan información sobre la dinámica y formación de los brazos espirales además de constituir laboratorios idóneos para el estudio de la función de masa inicial estelar que proporciona información sobre los procesos evolutivos iniciales estelares.
5. En el caso de cúmulos abiertos de edad intermedia/alta se localizan principalmente en el disco de la Galaxia sus edades corresponden a las de la formación del disco de la Galaxia por lo que resultan esenciales para la comprensión de los procesos evolutivos, dinámicos y cinemáticos de formación del disco Galáctico además de seguir proporcionando información sobre formación estelar.

3.4. Estado del arte en la búsqueda de cúmulos abiertos galácticos.

Como se ha indicado en la Sección 1, el análisis de los datos de Gaia para la identificación de cúmulos abiertos es un tema de plena actualidad con varios equipos produciendo resultados con un gran incremento en el número de nuevos cúmulos identificados.

En este sentido, el caso más reciente y llamativo por el número muy elevado de cúmulos identificados es el de Castro-Ginard et al. (2020) donde se comunica la identificación de 582 nuevos cúmulos en el disco Galáctico usando técnicas también basadas en DBSCAN para la identificación de sobre-densidades y técnicas de clasificación mediante redes neuronales para identificar si dichos candidatos son cúmulos abiertos. Esta última parte no se aborda en este trabajo de tesis.

En lo relativo a la identificación de sobre-densidades es interesante mencionar el enfoque de cómo evaluar los parámetros que definen DBSCAN lo diferencian del trabajo en esta tesis (ver sección 4.2). En el caso de Castro-Ginard et al. (2020) se recurre a la estrategia propuesta en otro artículo del mismo grupo (Castro-Ginard et al. (2018)) donde se realizan simulaciones masivas de poblaciones estelares basadas en el conocimiento/modelado de la distribución de estrellas y cúmulos en la Galaxia. Sobre realizaciones sintéticas de campos estelares se ejecutan búsquedas con DBSCAN y se seleccionan los parámetros (ϵ , N_{pts}). Para ello, los autores tienen a su disposición el super-computador MareNostrum4 donde implementan técnicas de computación paralelizada basadas en COMPSs (Tejedor et al. (2017)).

Una conclusión muy relevante a la que llegan los autores es el hecho que hasta la fecha ninguna de las técnicas de aprendizaje automático es capaz de identificar y caracterizar más del 75% de los cúmulos que se inyectan en los campos estelares sintéticos.

4. Desarrollo de la infraestructura de computación distribuida. Resultados.

4.1. Punto de partida.

El punto de partida de este TFM es el código RL19. En esta sub-sección se pretende dar una visión general y conceptual de trabajo en dicho TFM; para los detalles más técnicos/programación acudir directamente a RL19.

El código escrito en Python responde a las necesidades de un proceso clásico ETL donde:

- Se realiza una consulta a la base de datos VizieR¹⁶ (VizieR, 2019) usando el módulo `DiasCatalog.py`, desarrollado entorno al paquete `astropy` para la realización de consultas astronómicas a un gran numero de catálogos astronómicos. En concreto, en el módulo `DiasCatalog.py` se hace uso del sub-paquete `astroquery.vizier`. De esta base de datos se recupera el catálogo de cúmulos abiertos en Dias et al. (2002) (~ 2200 objetos) que tradicionalmente ha sido considerado el más completo, si bien dado el impacto que los datos de Gaia están proporcionando (p.e. Castro-Ginard et al. (2020) presentan 582 nuevos cúmulos abiertos previamente no catalogados en Dias et al. (2020) es muy posible que pronto sea superado por nuevos catálogos.
- Se realiza una consulta a la base de datos Gaia DR2¹⁷, de nuevo usando el sub-paquete `astroquery.gaia` se gana en conveniencia al disponer de objetos ya serializados por Python. Esta constituye realmente el proceso ETL del algoritmo al descargar todas las estrellas que satisfacen los criterios de búsqueda establecidos en el módulo `GaiaData.py`. Conviene resaltar que la consulta se hace en base a un dialecto de SQL denominado ADQL¹⁸ (*Astronomical Data Query Language*). En la versión de `GaiaData.py` en RL19 se implementa un método para realizar búsqueda en regiones circulares; esto ha sido modificado en la nueva versión en BF20 por búsquedas en cajas rectangulares ya que eventualmente el código se usará para muestrear zonas extensas en base a mosaicos.
- Las dos fases anteriores corresponden al pase **E**(*extraction*) ejecutados por los módulos `DiasCatalog.py` y `GaiaData.py`; el proceso **T**(*transformation*) tiene lugar en el módulo `main.py` que además resulta ser el módulo principal

¹⁶ La base de datos es accesible en <https://vizier.u-strasbg.fr/viz-bin/VizieR> si bien el sub-paquete `astroquery` dentro del paquete de Python `astropy` permite hacer consultas directas a VizieR y generar objetos con características convenientes para el tratamiento directo de los dtos desde Python.

¹⁷ <https://www.cosmos.esa.int/web/gaia/dr2>

¹⁸ <http://www.ivoa.net/documents/ADQL/2.0>

y que aglutina todas las funciones de interés para la detección de candidatos y también el post-procesado de los resultados de la búsqueda de sobre-densidades. De hecho desde `main.py` se realizan las consulta a Gaia y Vizier y las etapas **T** y **L**(load) implican:

- Transformación de coordenadas 2-D angulares en cielo a 3-D en un espacio cartesiano (x,y,z) al considerar los valores de paralaje proporcionados en Gaia DR2.
- Ídem a un espacio 5-D consistente en el anterior 3-D (x,y,z) pero añadiendo dos nuevas dimensiones correspondientes a los movimientos propios en unidades de mas/yr.
- Muestreo del número de estrellas a usar en el proceso de *clustering*.
- Normalización de los datos tal y como resulta imperativo en proceso de aprendizaje no supervisado.

El módulo `main.py` carga los distintos parámetros que definen una ejecución a partir de un fichero .csv que a su vez es cargado en un *DataFrame* de Pandas con los siguientes campos:

1. **ra** y **dec**: coordenadas angulares que definen el campo a tratar. En el algoritmo original en (RA, Dec), en el algoritmo final se implementa también con las coordenadas galácticas (l,b).
2. **r**: radio de la zona a tratar.
3. **err_pos**: Error máximo (en mas) tolerable en RA, Dec o paralaje. Aquellos registros que en alguno de estos valores presenten errores mayores a **err_pos** son descartados.
4. **g_lim**: magnitud límite en banda G a considerar.
5. **norm**: aplicar normalización de los datos.
6. **sample**: factor de muestreo aleatorio. Valor comprendido entre [0, 1], donde 1 representa aceptar todos los objetos que Gaia proporciona para la zona de cielo consultada con centro en (**ra**, **dec**).
7. **dim3**: determina si el algoritmo usará la información de paralaje.
8. **distance**: especifica la forma de calcular las distancias en la generación de la matriz de distancias en DBSCAN. Puede optarse por el valor “*euclidean*” donde las distancias son calculadas simplemente como distancias angulares proyectados sobre la bóveda celeste (no considere paralaje) o las distancias son 3D (con paralaje) o una alternativa “*pm*” en la que que considere el espacio de valores en 5D al incorporar en el análisis de búsqueda los valores de movimientos propios. Esta opción, dese el punto de vista computacional implica una mayor complejidad pero añade valor al proceso de búsqueda pues hay que considerar que los

objetos pertenecientes a una misma sobre-densidad (asociaciones y cúmulos abiertos) exhiben valores de movimiento propio muy similares al haberse formado todos sus miembros a partir de la misma región molecular.

9. **eps_min, eps_max y eps_num**: definen el rango de ϵ (ver abajo) como el mínimo valor, el máximo y el número de muestras equi-espaciadas entre ambos.
10. **min_pts_min, min_pts_max y min_pts_num**: definen el rango de N_{pts} (ver abajo) como el mínimo valor, el máximo y el número de muestras equi-espaciadas entre ambos.

Una vez provistos los valores de los parámetros en el fichero de entrada .csv el software puede usarse de dos maneras:

- Como un algoritmo de búsqueda usando como núcleo central del aprendizaje no supervisado (*clustering*) la implementación de DBSCAN (Ester et al., 1996; Romero, 2019) en el paquete `scikit-learn` de Python¹⁹ (Pedregosa et al., 2011). Como es bien sabido, el resultado del agrupamiento con DBSCAN es altamente sensible a la elección de los parámetros que determinan el algoritmo, a saber, el radio de la hiperesfera en el espacio multi-dimensional de parámetros y el número mínimo de puntos para considerar un grupo de estrellas como un *cluster* denotados habitualmente por ϵ y N_{pts} , respectivamente. Este es el modo de búsqueda de **nuevos candidatos** en una determinada zona del cielo.
- Uso del propio código para estimar un conjunto de valores (ϵ , N_{pts}) pseudo-óptimo en zonas del cielo usando conocimiento a priori a partir del catálogo de Vizier y usando dicho conocimiento como un pseudo-entrenamiento del código DBSCAN.

4.2. Estrategia de búsqueda de candidatos con DBSCAN.

Es el último apartado mencionado en la sección anterior lo que caracterizará nuestra estrategia a la hora de realizar búsquedas y que tiene profundas connotaciones en el desempeño del código a la hora de explorar zonas del cielo donde se quiere acelerar el cálculo y no “perder” tiempo en encontrar el rango de valores de (ϵ , N_{pts}).

Tal y como se ha mencionado en la sección 3.4, el enfoque de otros grupos (p.e. Castro-Ginard et al., 2018, 2019, 2020) que también usan DBSCAN para el aprendizaje no supervisado es el de obtener “valores óptimos” a partir de simulaciones numérica donde se recrean poblaciones estelares y se ejecuta el código DBSCAN.

¹⁹ <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html>

En el enfoque que se considera en nuestra implementación ahora se trata de usar el catálogo de Dias et al. (2002) y entrenar a DBSCAN buscando aquéllos valores de (ϵ, N_{pts}) que proporcionan mayor éxito en la identificación de los cúmulos abiertos conocidos. De hecho, cuantos más catálogos podamos incorporar en nuestro proceso de aprendizaje tanto mejor: Este es un aspecto que no resulta posible en RL19 pero es inmediato de implementar en la versión BF20 (ver más adelante).

Una vez se logran determinar mapas de ϵ y N_{pts} , para el análisis de una determinada zona del cielo podremos restringir el valor de combinaciones (ϵ, N_{pts}) durante la ejecución de DBSCAN, partiendo del valor resultante de la interpolación de los mapas de (ϵ, N_{pts}) generados del análisis de los catálogos de cúmulos abiertos conocidos.

Se hace notar que este enfoque presenta la bondad de ser un proceso, por definición, iterativo. Se ha mencionado que Gaia DR2 está incrementando el número de cúmulos abiertos identificados e incluso poniendo en cuestión si los que venían considerándose en el catálogo de Dias et al. (2002) son todos ellos aciertos o hay una fracción no despreciable de objetos hasta la fecha considerados cúmulos abiertos pero que en realidad tienen altas posibilidades de ser falsos positivos. La implementación BF20 permite ir incorporando tantos catálogos y nuevos candidatos como se vayan descubriendo.

Por último, la naturaleza de selección de (ϵ, N_{pts}) en base a cúmulos abiertos puede tener el beneficio de favorecer que el conjunto de parámetros de partida en DBSCAN sea cercano al que tenga mayor probabilidad a identificar cúmulos abiertos frente a posibles asociaciones estelares o cúmulos globulares. En definitiva, la selección de (ϵ, N_{pts}) introduce un sesgo positivo a nuestros intereses.

4.3. Evaluación de tiempos de ejecución.

Como se ha dicho desde el principio de esta memoria el código RL19 no resulta viable dado el excesivo tiempo de computación necesario. En esta sección analizamos el problema usando como “*benchmark*” los tiempos de ejecución en un ordenador portátil de alta gama cuyas características fundamentales²⁰ (al ejecutar en modo local) son:

Hardware: Dell Precision 5540

- CPU: Intel Core i9-9980HK: basado en arquitectura Coffee Lake. El reloj de procesador funciona a velocidades nominales de 2.4 y hasta 5 GHz con *turbo-boost*, seleccionando ésta última de manera por defecto en la configuración final del portátil. El procesador puede ejecutar simultáneamente hasta 16 procesos ya que cuenta con 8 cores con *Hyper-Threading*.

²⁰<https://www.notebookcheck.net/Intel-Core-i9-9980HK-Processor-Benchmarks-and-Specs.416730.0.html#:~:text=The%20Intel%20Core%20i9%2D9980HK,simultaneously%20thanks%20to%20Hyper%2DThreading.>

- *Level 1 cache*: 512 KB.
- *Level 2 cache*: 2 MB.
- *Level 3 cache*: 16 MB.
- 32 GB de RAM a 2667 MHz.

Sistema Operativo: Ubuntu 18.04

Versión Python: 3.7.7

4.3.1. Ejecución de RL19 y optimización BF20.

Para el *benchmarking* de RL19 ejecutamos el programa en la modalidad de identificar la combinación óptima de partida mediante un rastreo del espacio dimensional (\mathcal{C} , N_{pts}) en el mismo campo estelar de pruebas que el considerado RL19:

- **(RA, Dec)** = (81.8553°, 34.719°)
- **g_lim**=19,
- **sample**= 0.66 (el 66% de los objetos)
- **distance**= *euclidean*²¹.

El código original tarda 37672 segundos (~10.5 horas) para realizar el muestreo en una parrilla de $\mathcal{C} \times N_{pts}$, donde \mathcal{C} se muestrea con 100 puntos cubriendo el rango $\mathcal{C}=[0.008, 0.03]$ y $N_{pts}=[10, 100]$ con 91 muestras; en total 9100 combinaciones.

Se realiza un desglose detallado en una simple iteración para averiguar donde está el cuello de botella. Una vez calculada la matriz de distancias y realizada la consulta a la base de datos (es decir, proceso ETL finalizado), una iteración para una tupla (\mathcal{C} , N_{pts}) tarda **4.40 segundos** en ejecutarse, de los cuales solo 0.46 segundos corresponden a la ejecución de DBSCAN y 3.94 segundos se invierten en identificar si alguno de los candidatos localizados puede asociarse a cualquiera de los cúmulos en el catálogo de entrenamiento (Dias et al., 2002) con 2167 cúmulos abiertos catalogados.

Esta ejecución es tremendamente costosa e ineficiente, pues de antemano podemos determinar y seleccionar un pequeño grupo de cúmulos que potencialmente estén a la distancia r (radio) y descartar de antemano a la gran mayoría que sabemos por sus coordenadas que no se encuentran en la región a estudiar. Por ejemplo, en el caso del campo nominal de ejecución sabemos que en Dias et al. (2002) solo hay 5 cúmulos

²¹ La prueba de “*benchmarking*” usa la métrica *euclidean* sobre los datos considerando únicamente RA y Dec. RA y Dec son magnitudes angulares en rangos absolutos comparables por lo que resulta lícito no habilitar la opción de normalización de los datos.

abiertos catalogados, por lo que es totalmente innecesario intentar realizar el *cross-match* sobre los 2167 cúmulos.

La optimización implementada en BF20 restringe la comparación con el catálogo a aquellos miembros en un radio $1.5 \times r$. Este factor (1.5) es configurable y su razón de ser es por motivos de seguridad para no descartar a un cúmulo que quede “a mitad” entre dos zonas consecutivas de radio r .

Al ejecutar el código con esta simple estrategia, el proceso de *cross-matching* de candidatos con cúmulos catalogados se acelera un factor 245 y una iteración completa DBSCAN + identificación de candidatos en catálogo requiere de **0.445 segundos**.

En la Figura 6 se muestran los tiempos de ejecución total con 9100 iteraciones incluyendo también los *overheads* para ejecutar consulta al catálogo de Gaia y al catálogo de Dias et al. (2002) en Vizier.

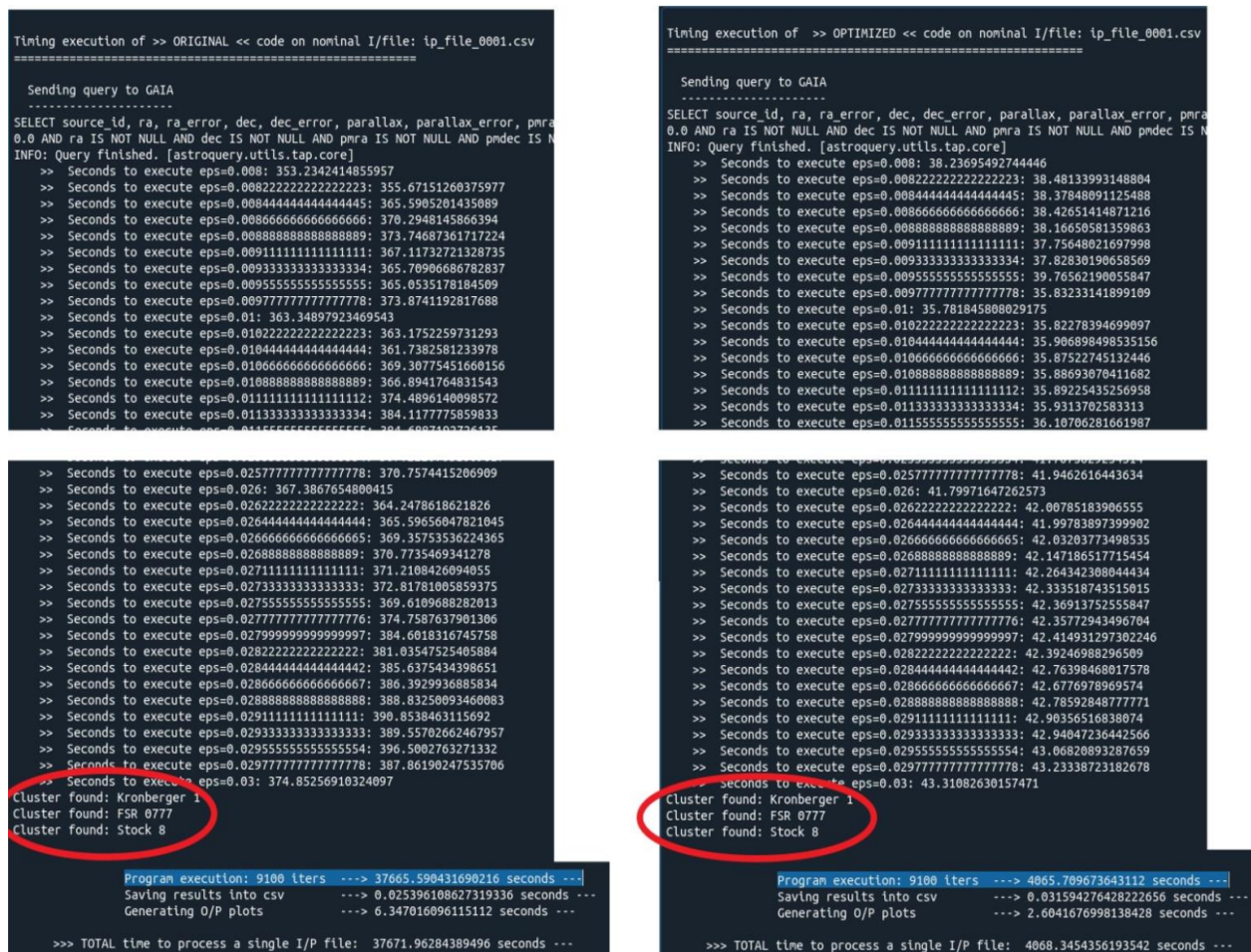


Figura 6: comparación de tiempos en la ejecución del original (izquierda) frente a la optimización (derecha) con un incremento de la velocidad de ejecución por un factor 9.3x. Fuente: Elaboración propia.

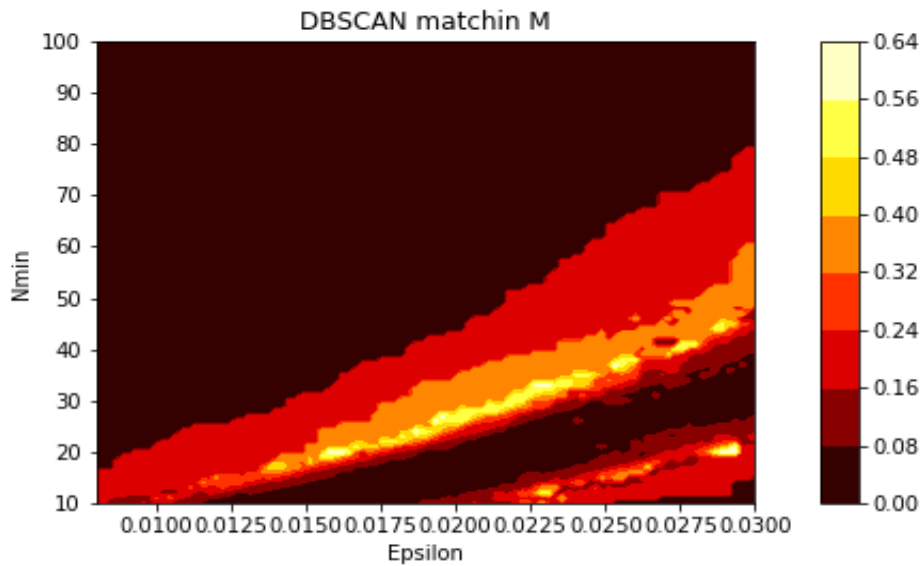


Figura 7: evolución de la métrica M de bondad del algoritmo en función de (ϵ , N_{pts}). Fuente: Elaboración propia.

La interpretación de la Figura 7 es inmediata al recordar de la definición en RL19 del parámetro de bondad M:

$$M = \frac{N_{matches}}{N_{Dias} + N_{extra}}$$

done $N_{matches}$, N_{dias} y N_{extra} representan el número de detecciones con una correspondencia en Dias et al. (2002), el número de cúmulos catalogados para esa región de cielo en Dias et al.(2002) y el número de candidatos detectados y no presentes en Dias et al. (2002) que pueden ser o falsos positivos o nuevas detecciones.

Para cada campo con un número significativo de cúmulos en Dias et al. (2020) podemos ejecutar el código para encontrar el rango de (ϵ , N_{pts}) que proporciona los valores más altos de $M \sim 0.6$.

4.4. Montaje de un *cluster* de máquinas virtuales basado en contenedores Docker y montaje de Hadoop.

Ya se ha mencionado que para la ejecución de este TFM se dispone de un ordenador *octa-core* con *hyper-threading*, por lo que es viable instalar un *cluster* de máquinas en nuestro *host* y sobre dicho *cluster* ejecutar de manera distribuida varios procesos simultáneos.

La elección de Docker²² en este trabajo para generar el *cluster* de máquinas sigue las pautas y recomendaciones estudiadas durante el Máster. A modo de sumario las razones que justifican esta elección son:

- Por pragmatismo: en este Máster nos hemos familiarizado con Docker principalmente en Fernández Pena (2019a). Además, a día de hoy Docker resulta si no la tecnología más extendida una de la más populares; el repositorio de imágenes de Docker es amplio y cubre todas nuestras necesidades. Es una solución con alta fiabilidad.
- El contenedor Docker realmente **NO** genera una Máquina Virtual (MV) sino que se apoya directamente sobre el OS de la máquina *host*, lo que evita desperdiciar recursos frente a otras implementaciones que consideran MVs: no se requiere de un *hypervisor* y/o simulación del SO huésped por parte del SO anfitrión (ahorro de CPU) y el tamaño de RAM que un contenedor necesita es menor y, en el caso de Docker, no está restringido de ninguna manera sino que usa tanto como necesita en función de la RAM disponible en el *host*. Esta comparación entre VMs tradicionales y contenedores se ilustra en la Figura 8.
- Tal y como ilustra la Figura 9, la ejecución de un proceso en un contenedor es un proceso aislado; desde el punto de vista de la máquina anfitriona (*host*) es un proceso más gestionado por el *kernel* del *host* como otro proceso cualquiera. Esto resulta ideal para nuestras aplicaciones en cálculo distribuido.

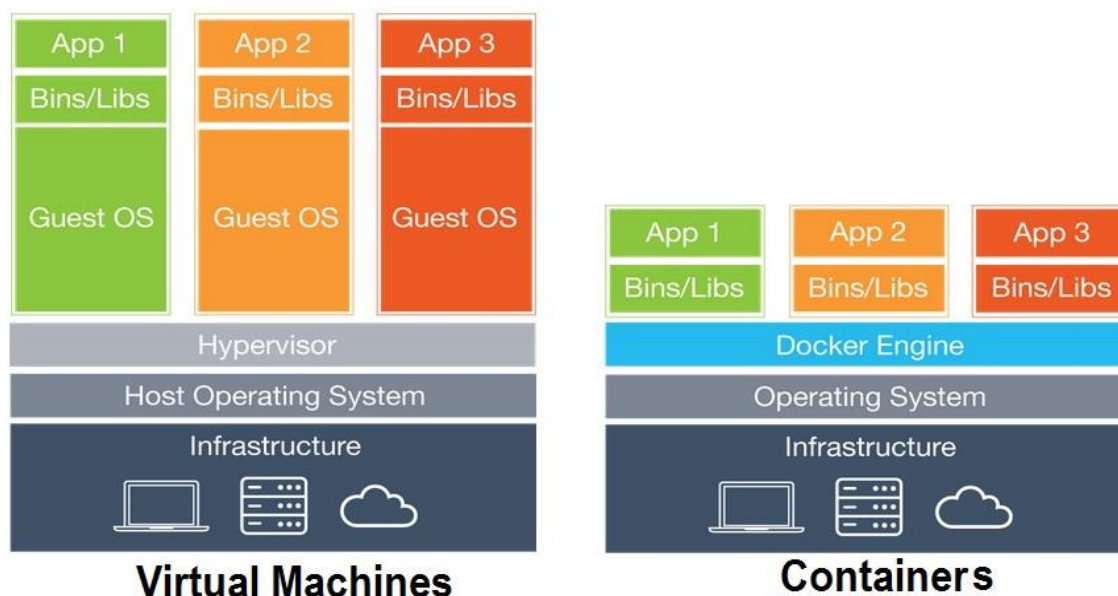


Figura 8: comparación esquemática de la diferencia entre Máquinas Virtuales (VM acrónimo inglés) y contenedores. Fuente: www.docker.com

²²<https://www.docker.com/>

Containers vs. Processes

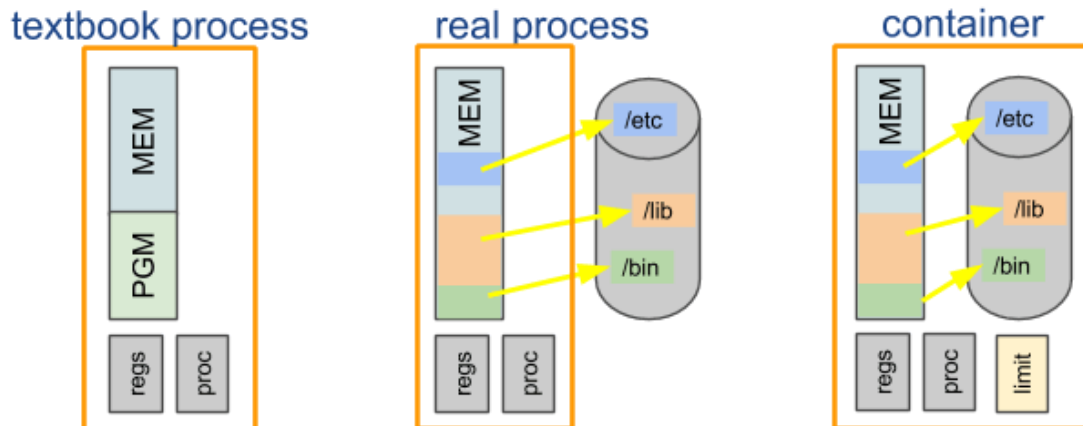


Figura 9: : comparación entre un proceso conceptual, lo que realmente ocurre con un proceso en linux y lo que ocurre al ejecutar procesos en un container. Fuente: <https://sites.google.com/site/mytechnicalcollection/cloud-computing/docker/container-vs-process>

- Lanzar y parar contenedores es extremadamente fácil y **rápido**. Esto es importante cuando sobre el *cluster* montemos Hadoop y sus sistema de ficheros distribuidos HDFS: es necesario parar los contenedores antes de apagar la máquina *host* o se corre el riesgo de corromper el sistema de ficheros distribuidos HFS usado en Hadoop.
- Una vez creada las imágenes Docker para los distintos elementos del nodo Hadoop (*namenode*, *datanode*, *checkpointnode*) generar clones de los nodos de trabajo resulta inmediato. Además, la generación de las imágenes se va haciendo por capas dotando exactamente a cada tipo de imagen de los recursos y dependencias estrictamente necesarios. En resumen, aprovechamos de manera eficiente la “**encapsulación**” proporcionada por Docker.
- Aunque no se ha podido llegar a este punto del proceso, una vez generada la imagen que cumple nuestros objetivos en el ordenador portátil dicha imagen puede ser exportada a máquinas/*clusters* con mayor potencia de cálculo y/o mayor número de elementos (nodos) en el *cluster* (incremento de la escalabilidad vertical y horizontal, respectivamente)

Se procede a montar un sistema Hadoop pseudo-distribuido por ser residente en una sola máquina física, aunque con varios nodos. El *hardware* disponible consta de 8 cores con *hyper-threading*. Por un lado parece sensato reservar al menos un *thread* para el SO nativo. Por otro lado el objetivo es implementar Hadoop que además de los nodos que actúan como esclavos y donde se ejecutarán las tareas, requiere de un *namenode* y un *checkpointnode*. Así pues, se decide en un primer momento generar un *cluster* con 14 nodos, 12 de ellos esclavos. A posteriori, por restricciones de memoria RAM en la ejecución de las tareas se deberá, en función de la ejecución, eliminar entre 3 y 5 nodos esclavos pero las taras de “de-comisionado” de nodos en un *cluster* Hadoop pueden hacerse en caliente sin necesidad de echar abajo todo el sistema.

A pesar de existir soluciones comerciales como Cloudera-Hortonworks, AWS EMR, Microsoft Azure HDInsight se opta por un proceso de instalación manual como el ejecutado durante la asignatura 02-MBIG (Fernández Pena, 2019b). El proceso aparece también descrito en varios tutoriales en línea , p.e. Sidhu (2020).

1. Instalación sobre el *host* Linux (Ubuntu 18.04) de la versión de Docker CE más reciente: <https://docs.docker.com/get-docker/>
2. Descarga de una imagen Docker desde el repositorio oficial con Ubuntu 18.04. Arrancamos dicha imagen en un contenedor al que llamamos `hadoop-install` corriendo en la máquina anfitriona.
3. El objetivo es instalar Hadoop por lo que instalamos el software necesario:
 - 3.1. Java OpenJDK 8.
 - 3.2. Openssh
 - 3.3. libssl1-dev
 - 3.4. Python 3. **NOTA:** eventualmente sugirieron problemas de compatibilidad de los módulos de consulta desde Python a Vizier y Gaia. La forma de solucionarlo fue forzar la descarga de una versión de Python que no es la más reciente (ver más adelante).
 - 3.5. Utilidades variadas: `nano` (editor en terminal desde el propio contenedor), `less`, `localess`.
 - 3.6. Módulos específicos para la ejecución del código python: `numpy`, `pandas`, `astropy`, `requests`, `keyring`, `beautifulsoap4`, `html5lib`, `siz`, `curl`, `pytest-astropy`, `astroquery`, `matplotlib`, `sklearn`, `mgzip`.
 - 3.7. Instalación de Hadoop-3.2.1:
 - a) Descarga del fichero `tgz` del repositorio oficial.
 - b) Creación del grupo `hadoop`. Creación de un usuario administrador de `hadoop` (`hadmin`) y un usuario local (`luser`) desde cuya cuenta se ejecutarán los procesos MapReduce. Habilitar permisos y *ownership* de los directorios de manera apropiada.
 - c) Configuración de los demonios de Hadoop: modificaciones necesarias a los ficheros de configuración de Hadoop: `core-default.xml`, `hdfs-default.xml`, `yarn-default.xml` y `mapred-default.xml`.
4. El contenedor `hadoop-install` ya está configurado para generar la primera imagen sobre la que construir el *cluster* con Hadoop. Paramos el contenedor y realizamos una consolidación (*commit*) generando una nueva imagen: `hadoop-base` usando el comando:

```
docker container commit hadoop-install hadoop-base
```

5. Una vez creada la imagen podemos eliminar el contenedor `hadoop-install`.
6. Se crea una red para permitir que los contenedores puedan hablar entre sí:

```
docker network create hadoop-cluster
```


7. A partir de la imagen `hadoop-base` se lanzan tres nuevos contenedores sobre los que ir montando elementos para eventualmente tener una imagen por tipo de contenedor: una imagen para el namenode, una imagen para el checkpointnode y una imagen que se usará en cada uno de los esclavos datanodes. Al lanzar estos tres contenedores declaramos los puertos que se exponen y los puertos de acceso a las interfaces de Hadoop via *web browser*. Tal y como se describe en Fernández Pena (2019b), los comandos a ejecutar para cada tipo de contenedor son:

Contenedor del que se obtendrá la imagen para generar el **namenode**:

```
docker container run -ti --name namenode --network=hadoop-cluster --hostname namenode --net-alias resourcemanager --expose 8000-10000 -p 9870:9870 -p 8088:8088 hadoop-base /bin/bash
```

Contenedor del que se obtendrá la imagen para crear el **checkpointnode**:

```
docker container run -ti -name checkpointnode --network=hadoop-cluster -hostname checkpointnode --net-alias timelineserver --expose=8188-10200 -p 9868:9868 -p 8188:8188 hadoop-base /bin/bash
```

Contenedor para generar la imagen desde la que crear cada **datanode**:

```
docker container run -ti --name datanode --network=hadoop-cluster --hostname datanode --expose 8000-10000 --expose 50000-50200 hadoop-base /bin/bash
```

8. Iniciamos Hadoop en tres fases:
- 8.1. Preparación del sistema de ficheros HDFS.
 - 8.2. Inicio de los demonios:
 - a) En **namenode**: `namenode` y `resourcemanager`.
 - b) En **datanode**: `datanode` y `nodemanager`.
 - c) En **checkpointnode**: `secondarynamenode` y `timelineserver`.
 - 8.3. Incorporamos a los contenedores en sus respectivos directorios raíz *scripts* de inicio en los que:
 - a) Cada tipo de contenedor lanza sus demonios como en el punto 8.2.
 - b) En cada contenedor se inicia el servicio `ssh`.
 - c) El *script* de inicio finaliza con un loop infinito de carga de CPU insignificante: `while true; do sleep 10000; done` y que sirve para mantener activo los contenedores:
9. Llegados a este punto ya se dispone un sistema Hadoop básico con 1-namenode, 1-checkpointnode, 1-datanode y cada uno de ellos ejecutándose en un contenedor específico. Basta con parar los contenedores y consolidar imágenes para cada tipo de contenedor. Estas son las **últimas imágenes** que se crean y los contenedores finales se arrancan usando dichas imágenes. La

captura de pantalla en la Figura 10 muestra los *shell scripts* donde se crean y lanzan por primera vez los contenedores, se lanzan (si ya están creados) y se paran los contenedores. Resaltar que en la creación de los contenedores declaramos los volúmenes que comparten los contenedores con la estructura de ficheros en la máquina anfitriona así como los volúmenes para intercambiar ficheros entre contenedores.

```

#!/bin/bash

# Step 1: Creating volume in namenode with Python packages
#=====
export TFM_PATH=/home/bfemenia/TFM

#Step 2: creating containers for namenode / datanode / checkpointnode
#===== NOTE volumes being shared among containers and with HOST. -v option!!

docker container run -d --name namenode -v $TFM_PATH:/home/TFM/ \
--network=hadoop-cluster --hostname namenode --net-alias resourcemanager \
--expose 8000-10000 -p 9870:9870 -p 8088:8088 namenode-image /inicio.sh

for i in {1..7}; do docker container run -d --name datanode$i --volumes-from namenode \
-v $TFM_PATH:/home/TFM/ --network=hadoop-cluster --hostname datanode$i \
--expose 8000-10000 --expose 50000-50200 datanode-image /inicio.sh; done

docker container run -d --name checkpointnode --volumes-from namenode -v $TFM_PATH:/home/TFM/ \
--network=hadoop-cluster --hostname checkpointnode --net-alias timelineserver \
--expose=8188-10200 -p 9868:9868 -p 8188:8188 checkpointnode-image /inicio.sh

-:--- script_hadoop_create_containers.sh All (16,0) (Shell-script[bash]) jue jun 18 17:50 1.83

#!/bin/bash
docker start namenode datanode[1..9] checkpointnode

docker container exec -ti namenode /bin/bash

-:--- start_hadoop.sh All (5,0) (Shell-script[bash]) -:--- stop_hadoop.sh All (4,0) (Shell-script[bash])
  
```

Figura 10. **Arriba:** script para generar y arrancar los contenedores con el cluster y Hadoop. **Abajo izquierda:** script simple para una vez los contenedores han sido creados arrancar el cluster. **Abajo derecha:** script para detener los contenedores. Fuente: elaboración

Para ilustrar lo sencillo que resulta poner en marcha el *cluster* desde un estado de parada se adjuntan las siguientes capturas de pantalla, asumiendo que el primer *script* (*script_hadoop_create_containers.sh*) ha sido previamente ejecutado para crear un *cluster* con siete nodos esclavos y los contenedores ya existen. En la Figura 11 vemos el *cluster* está echado abajo; la figura muestra que en nuestro repositorio las imágenes de los distintos tipos de nodo ya existen y que los contenedores ya han sido creados y están inactivos. Mostramos los volúmenes que se comparten y la red entre contenedores. Tras ejecutar desde línea de comando `./start_hadoop` pasamos al estado mostrado en la Figura 12. En esta secuencia de dos imágenes se ha decidido tener solo 7 datanodes; si quisiéramos incrementar el número de datanodes basta con hacer un sencillo cambio en el lazo en *script_hadoop_create_containers.sh*. Por completitud en esta discusión, al querer lanzar un bucle con 9 nodos podemos verificar su estado a través de la interfaz web, tal y como se muestra en la Figura 13.

```
(base) [10] bfemmlag@kisp:~/JFH > echo '';echo '';docker images;echo '';echo '';docker ps -a;echo '';echo '';docker volume ls;echo ''; echo '';docker network ls

REPOSITORY          TAG                 IMAGE ID            CREATED            SIZE
checkpointnode-image latest             94eade63758b       10 days ago       2.33GB
datanode-image       latest             dd21c218064a       10 days ago       2.33GB
namenode-image       latest             a1e6baa1a9d1       10 days ago       2.33GB
hadoop-base          latest             fe6a9b35d41        10 days ago       2.33GB
ubuntu               18.04             c3c384c4f22        7 weeks ago       64.2MB
hello-world          latest             bf756fbae55        5 months ago      13.3kB

CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS             PORTS              NAMES
ee3db08ea75b       datanode-image     "/initclo.sh"       12 hours ago      Exited (137) 7 minutes ago              datanode9
ba115f9ebfd        datanode-image     "/initclo.sh"       12 hours ago      Exited (137) 7 minutes ago              datanode8
76a7d958d3fe       checkpointnode-image "/initclo.sh"       18 hours ago      Exited (137) 7 minutes ago              checkpointnode
7fc925d4e6da       datanode-image     "/initclo.sh"       18 hours ago      Exited (137) 7 minutes ago              datanode7
c9c26ec3a67a       datanode-image     "/initclo.sh"       18 hours ago      Exited (137) 7 minutes ago              datanode6
7cee9f64a604       datanode-image     "/initclo.sh"       18 hours ago      Exited (137) 7 minutes ago              datanode5
52ec2d0f7f77       datanode-image     "/initclo.sh"       18 hours ago      Exited (137) 7 minutes ago              datanode4
b2483d95cf33       datanode-image     "/initclo.sh"       18 hours ago      Exited (137) 7 minutes ago              datanode3
3cc1942ea355       datanode-image     "/initclo.sh"       18 hours ago      Exited (137) 7 minutes ago              datanode2
4f8094762ec1       datanode-image     "/initclo.sh"       18 hours ago      Exited (137) 7 minutes ago              datanode1
c192bdf85d65       namenode-image     "/initclo.sh"       18 hours ago      Exited (137) 7 minutes ago              namenode

DRIVER              VOLUME NAME
local               7b27e92b393a58ae27a1f50180e9bcb8a4923eab580198a16eeb254cfd3cd3
local               63aa012e02aedc24cfbd4f2f17a940701d51ca70e17db2bc8d6b633616d7f72

NETWORK ID          NAME                DRIVER              SCOPE
e4fb41e199da        bridge             bridge              local
39ed638dd165        hadoop-cluster     bridge              local
73d72a3c2e06        host               host                local
10035d6ff7a4        none               null                local
(base) [11] bfemmlag@kisp:~/JFH > []
```

Figura 11: con contenedores para cada uno de los nodos del cluster con Hadoop pero los contenedores parados. Se muestran los volúmenes que se comparten y la red entre contenedores. En este estado basta ejecutar desde línea de comando en la terminal `./start_hadoop`

```
(base) [12] bfemmlag@kisp:~/JFH > echo '';echo '';echo ''

(base) [13] bfemmlag@kisp:~/JFH > ./start_hadoop.sh
namenode
datanode1
datanode2
datanode3
datanode4
datanode5
datanode6
datanode7
datanode8
datanode9
checkpointnode
root@namenode:/# exit
(base) [14] bfemmlag@kisp:~/JFH > echo '';echo '';docker images;echo '';echo '';docker ps -a;echo '';echo '';docker volume ls;echo ''; echo '';docker network ls

REPOSITORY          TAG                 IMAGE ID            CREATED            SIZE
checkpointnode-image latest             94eade63758b       10 days ago       2.33GB
datanode-image       latest             dd21c218064a       10 days ago       2.33GB
namenode-image       latest             a1e6baa1a9d1       10 days ago       2.33GB
hadoop-base          latest             fe6a9b35d41        10 days ago       2.33GB
ubuntu               18.04             c3c384c4f22        7 weeks ago       64.2MB
hello-world          latest             bf756fbae55        5 months ago      13.3kB

CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS              NAMES
ee3db08ea75b       datanode-image     "/initclo.sh"       12 hours ago      Up About a minute   8088-10088/tcp, 50000-50200/tcp              datanode9
ba115f9ebfd        datanode-image     "/initclo.sh"       12 hours ago      Up About a minute   8088-10088/tcp, 50000-50200/tcp              datanode8
76a7d958d3fe       checkpointnode-image "/initclo.sh"       18 hours ago      Up About a minute   0.0.0.0:3188->3188/tcp, 8189-9867/tcp, 0.0.0.0:9868->9868/tcp, 9869-10200/tcp              checkpointnode
7fc925d4e6da       datanode-image     "/initclo.sh"       18 hours ago      Up About a minute   8088-10088/tcp, 50000-50200/tcp              datanode7
c9c26ec3a67a       datanode-image     "/initclo.sh"       18 hours ago      Up About a minute   8088-10088/tcp, 50000-50200/tcp              datanode6
7cee9f64a604       datanode-image     "/initclo.sh"       18 hours ago      Up About a minute   8088-10088/tcp, 50000-50200/tcp              datanode5
52ec2d0f7f77       datanode-image     "/initclo.sh"       18 hours ago      Up About a minute   8088-10088/tcp, 50000-50200/tcp              datanode4
b2483d95cf33       datanode-image     "/initclo.sh"       18 hours ago      Up About a minute   8088-10088/tcp, 50000-50200/tcp              datanode3
3cc1942ea355       datanode-image     "/initclo.sh"       18 hours ago      Up About a minute   8088-10088/tcp, 50000-50200/tcp              datanode2
4f8094762ec1       datanode-image     "/initclo.sh"       18 hours ago      Up About a minute   8088-10088/tcp, 50000-50200/tcp              datanode1
c192bdf85d65       namenode-image     "/initclo.sh"       18 hours ago      Up About a minute   8088-8088/tcp, 0.0.0.0:8088->8088/tcp, 8089-9869/tcp, 9871-10080/tcp, 0.0.0.0:9870->9870/tcp              namenode

DRIVER              VOLUME NAME
local               7b27e92b393a58ae27a1f50180e9bcb8a4923eab580198a16eeb254cfd3cd3
local               63aa012e02aedc24cfbd4f2f17a940701d51ca70e17db2bc8d6b633616d7f72

NETWORK ID          NAME                DRIVER              SCOPE
e4fb41e199da        bridge             bridge              local
39ed638dd165        hadoop-cluster     bridge              local
73d72a3c2e06        host               host                local
10035d6ff7a4        none               null                local
(base) [15] bfemmlag@kisp:~/JFH > []
```

Figura 12: con `./start_hadoop.sh` levantamos el cluster y se van notificando los nodos activos. En este caso, al finalizar el script volvemos a la terminal del host donde podemos comprobar que los contenedores están activos y los puertos que se exponen. Fuente: elaboración propia.

Overview 'namenode:9000' (active)

Started:	Tue Jun 16 10:19:35 +0100 2020
Version:	3.2.1, rb3cbbb467e22ea829b3808f4b7b01d07e0bf3842
Compiled:	Tue Sep 10 16:56:00 +0100 2019 by rohitsharmaks from branch-3.2.1
Cluster ID:	CID-16d336f8-dba8-42f9-8057-8fe4d1f47225
Block Pool ID:	BP-1297593088-172.18.0.2-1591399107517

Summary

Security is off.

Safemode is off.

45 files and directories, 28 blocks (28 replicated blocks, 0 erasure coded block groups) = 73 total filesystem object(s).

Heap Memory used 298.8 MB of 686 MB Heap Memory. Max Heap Memory is 6.88 GB.

Non Heap Memory used 55.67 MB of 57.19 MB Committed Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	4.32 TB
Configured Remote Capacity:	0 B
DFS Used:	172.37 MB (0%)
Non DFS Used:	498.62 GB
DFS Remaining:	3.61 TB (83.62%)
Block Pool Used:	172.37 MB (0%)
DataNodes usages% (Min/Median/Max/stdDev):	0.00% / 0.00% / 0.01% / 0.00%
Live Nodes	9 (Decommissioned: 0, In Maintenance: 0)
Dead Nodes	0 (Decommissioned: 0, In Maintenance: 0)
Decommissioning Nodes	0
Entering Maintenance Nodes	0
Total Datanode Volume Failures	0 (0 B)
Number of Under-Replicated Blocks	0
Number of Blocks Pending Deletion (including replicas)	0
Block Deletion Start Time	Tue Jun 16 10:19:35 +0100 2020
Last Checkpoint Time	Tue Jun 16 10:27:45 +0100 2020
Enabled Erasure Coding Policies	RS-6-3-1024k

NameNode Journal Status

Current transaction ID: 431

Journal Manager	State
FileJournalManager(root=/var/data/hadoop/hdfs/nn)	EditLogFileOutputStream(/var/data/hadoop/hdfs/nn/current/edits_inprogress_000000000000000431)

NameNode Storage

Storage Directory	Type	State
/var/data/hadoop/hdfs/nn	IMAGE_AND_EDITS	Active

DFS Storage Types

Storage Type	Configured Capacity	Capacity Used	Capacity Remaining	Block Pool Used	Nodes In Service
DISK	4.32 TB	172.37 MB (0%)	3.61 TB (83.62%)	172.37 MB	9

Hadoop, 2019.

Figura 13: vista general a través de la interfaz web de un cluster Hadoop con 9 datanodes levantado en la máquina local descrita en la sección 4.3. Fuente: elaboración propia.

4.5. Justificación de la elección de Hadoop.

El plan original era desplegar alguna de las variedades de marcos de trabajo y/o *wrappers* que permiten ejecutar Python de manera distribuida y/o paralela. Inspirados en el trabajo de Castro-Ginard et al. (2020) en el que despliegan PyCOMPSs (Tejedor et al. 2017) en el super-ordenador MareNostrum4, nuestra estrategia inicial era instalar PyCOPSs en el *cluster* con Docker.

Sin embargo, tras varios intentos infructuosos por lograr una instalación de PyCOMPSs propongo la alternativa de Hadoop. A primera vista no parecería que Hadoop fuese la opción más aconsejable en un proceso de computación como el previsto en nuestro trabajo en base a 4 argumentos de peso usualmente descritos como situaciones en las que Hadoop puede no resultar la opción más aconsejable:

Hadoop está pensado para ejecución masiva por lotes de pequeños trabajos, cuya entrada/salida es por *stdin* y *stdout*, respectivamente. El resultado de cualquier proceso resulta en la modificación de un fichero en HDFS. Si durante la ejecución de dicho proceso hay un fallo de Hardware, Hadoop es capaz de redirigir la tarea a otro nodo. Es por ello que está especialmente indicado para procesos cortos (según ciertas fuentes, se aconseja procesos de menos de 10 minutos y de hecho veremos que hay que modificar los *timeouts* establecidos por defecto). Ni LR19 ni BF20 resultan en procesos cortos ni originalmente están escritos para realizar procesos de entrada/salida exclusivamente por *stdin* / *stdout*.

Hadoop favorece en su HFS nativo el uso de grandes ficheros (*Big Data*) y resulta muy ineficiente al almacenar ficheros de pequeño tamaño como los que posteriormente veremos que estamos forzados a trabajar. Además, para tolerancia de fallos propone replicación de los datos en distintos nodos en *racks* distintos para aumentar la robustez a fallos. En nuestro caso, con un *laptop*, no estamos haciendo uso de esta característica.

Hadoop usa Java de manera nativa; el código en RL19 usa Python y librerías muy específicas que no hacen factible reescribir todo en Java;

Sin embargo Hadoop presenta características interesantes para el eficaz desarrollo (como ha sido el caso) del trabajo propuesto en este TFM:

El aspecto programático en Java puede evitarse porque Hadoop proporciona un API (Hadoop Streaming²³) que permite programación con cualquier tipo de lenguaje que sea compatible con *streams* de Unix con entrada/salida a través de *stdin* / *stdout*. Además, existen multitud de recursos en línea sobre como programar Python en Hadoop (p.e. Ratdka y Miner, 2015).

Como en el caso de contenedores Docker, durante el desarrollo del Máster hemos adquirido cierta familiaridad con el despliegue de *cluster* Hadoop (lo descrito en la sección anterior fue realizado como una de las Actividades Guiadas en la asignatura 02-MBIG).

²³Más información en <http://hadoop.apache.org/docs/stable3/hadoop-streaming/HadoopStreaming.html>

Hadoop y su *resourcemanager* YARN están fuertemente orientados al desempeño de tareas *MapReduce* para el procesamiento por lotes. Una inspección BF20 muestra que conceptualmente es relativamente sencillo adoptarlo como una tarea *MapReduce*.

Por último es esencial percatarse del hecho que para nuestra aplicación en particular realmente no necesitamos computación paralela sino distribuida. Las tareas realizadas durante la ejecución para el proceso de una región de cielo no se solapan con las mismas tareas para la ejecución en otra zona de cielo. Podría argumentarse que solo en el caso de regiones de cielo con un solape significativo una estrategia paralela pudiese implementar en el cálculo de la matriz de distancias de manera más efectiva. Sin embargo, ya se ha indicado que en el caso de operación con el modo “*sphe*” (caso 2D, solo usando RA y Dec) la obtención de la matriz de distancias apenas supone cómputo operacional.

En el resto de esta sección se describen los distintos retos y sus soluciones hasta haber modificado e implementado el código optimizado para ser ejecutado en un marco MapReduce con Hadoop.

4.5.1. Re-formateo del código optimizado como una secuencia MapReduce.

El primer objetivo a abordar con una infraestructura de computación distribuida es la del cálculo sobre una fracción (idealmente la totalidad) del plano de la Galaxia con el fin de recabar mapas de los parámetros (ϵ , N_{pts}) de DBSCAN. Las zonas sobre las que lanzar los procesos son discretos pues usamos aquellas regiones de cielo que contengan un número apreciable de cúmulos ya detectados y registrados en Dias et al. (2002). Se trata, por tanto de **procesos en lotes**.

Por otro lado, el código acepta su entrada como un fichero csv. Una alternativa es proporcionar un conjunto de rutinas *mapper/reducer* donde el *mapper* lee un fichero cuyas líneas son los nombres de los ficheros de entrada al código RL19 y el *reducer* invoca una ligera²⁴ modificación del código ya optimizado.

En este caso las funciones *mapper/reducer* son simplemente funciones identidad y una estructura como la que se muestra en la donde se simula un procedimiento MapReduce. El *mapper* simplemente va leyendo un fichero con los nombres de los ficheros de parámetros a procesar; el proceso de lectura es línea a línea (un nombre de fichero por línea) y la salida la hace por *stdout* de manera que es el *shuffler* de YARN el que se encargará de redirigir cada una de estas salidas hacia un *mapper* que, a su vez, recoge el nombre de un fichero y llama a la verdadera tarea compleja a ejecutar.

²⁴A medida que se fue implementando el procedimiento observaremos como lo de “ligera modificación” es un eufemismo.

Es decir, el enfoque es tan sencillo como usar YARN (el gestor de recursos de Hadoop) para que actúe como coordinador de la ejecución de uno a uno de los ficheros de parámetros, siendo YARN quien determina en cada momento y en función de la disponibilidad del sistema qué contenedor *datanode* se encuentra disponible en cada momento para efectuar el trabajo requerido. ¡Posiblemente éste sea el ejemplo más sencillo e inmediato de generar un cálculo distribuido!

El proceso se ilustra en la Figura 14 con los programas `pruebaMapper.py` y `pruebaReducer.py`, con el listado de ficheros a “procesar” listados línea a línea en `pruebaBatch.txt`. La simulación de paso de `<key1,value1>` de un proceso *Mapper* al *Reducer* y la generación del par `<key2,value2>` puede simularse en línea de comando redireccionando salidas de *stdout* con *pipelines* tal y como se ilustra en la Figura 15. De hecho, esto es una práctica muy recomendable para depurar posibles errores de código sin necesidad de ejecutar YARN y su engorroso sistema de mensajes de error.

```

emacs@ username: bfemenia
File Edit Options Buffers Tools Python Help

#!/usr/bin/env python3

# very simple MAPPER function: identify function...
# no key and as a value the IP filename for the
# reduce routine

import sys

for line in sys.stdin:
    print(line, end='')

U:--- pruebaMapper.py<Memoria_TFM> All (1,0) (Python ElDoc) vie j

#!/usr/bin/env python3

# very simple REDUCER function: function pases value from
# MAPPER as teh I/P to the actual process we want to execute.

import sys

def main_fake(file):
    print('\nPrograma que simula la ejecución de una tarea compleja')
    print(f'\t\t >> Simulando procesado de fichero I/P {file}')

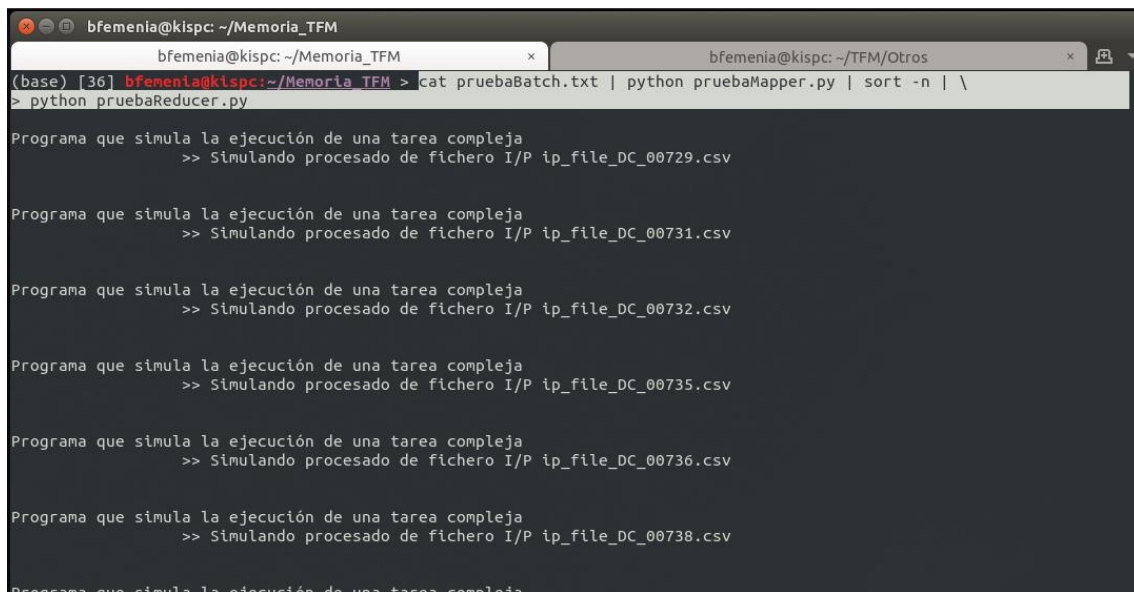
for value in sys.stdin:
    main_fake(value)

U:--- pruebaReducer.py<Memoria_TFM> All (9,67) (Python ElDoc) vie U:--- pruebaBatch.txt Top (24,20) (Te
Wrote /home/bfemenia/Memoria_TFM/pruebaReducer.py

ip_file_DC_00729.csv
ip_file_DC_00731.csv
ip_file_DC_00732.csv
ip_file_DC_00735.csv
ip_file_DC_00736.csv
ip_file_DC_00738.csv
ip_file_DC_00739.csv
ip_file_DC_00741.csv
ip_file_DC_00742.csv
ip_file_DC_00744.csv
ip_file_DC_00745.csv
ip_file_DC_00746.csv
ip_file_DC_00747.csv
ip_file_DC_00748.csv
ip_file_DC_00749.csv
ip_file_DC_00750.csv
ip_file_DC_00751.csv
ip_file_DC_00752.csv
ip_file_DC_00753.csv
ip_file_DC_00754.csv
ip_file_DC_00755.csv
ip_file_DC_00756.csv
ip_file_DC_00757.csv
ip_file_DC_00758.csv
ip_file_DC_00759.csv
ip_file_DC_00760.csv
ip_file_DC_00761.csv
ip_file_DC_00762.csv
ip_file_DC_00763.csv
ip_file_DC_00764.csv
ip_file_DC_00765.csv

```

Figura 14: : ilustración de dos sencillos scripts mapper y reducer para la ejecución distribuida de una tarea más compleja (*main_fake*) usando YARN con Hadoop. Fuente: elaboración propia.



```
bfemenia@kiscpc: ~/Memoria_TFM
bfemenia@kiscpc: ~/Memoria_TFM
(base) [36] bfemenia@kiscpc:~/Memoria_TFM > cat pruebaBatch.txt | python pruebaMapper.py | sort -n | \
> python pruebaReducer.py

Programa que simula la ejecución de una tarea compleja
>> Simulando procesado de fichero I/P ip_file_DC_00729.csv

Programa que simula la ejecución de una tarea compleja
>> Simulando procesado de fichero I/P ip_file_DC_00731.csv

Programa que simula la ejecución de una tarea compleja
>> Simulando procesado de fichero I/P ip_file_DC_00732.csv

Programa que simula la ejecución de una tarea compleja
>> Simulando procesado de fichero I/P ip_file_DC_00735.csv

Programa que simula la ejecución de una tarea compleja
>> Simulando procesado de fichero I/P ip_file_DC_00736.csv

Programa que simula la ejecución de una tarea compleja
>> Simulando procesado de fichero I/P ip_file_DC_00738.csv

Programa que simula la ejecución de una tarea compleja
```

Figura 15: simulación en línea de comando de cómo actuaría el código lanzado desde Hadoop. Además de ilustrativo, el uso de una secuencia como la resaltada es altamente aconsejable para poder depurar de manera eficiente posibles errores en las rutinas mapper y/o reducer. Fuente: elaboración propia.

¿Estamos ya en condiciones de iniciar el proceso de ejecución distribuida del código optimizado? De acuerdo a lo expuesto hasta ahora la estrategia parecería bastante simple:

1. Generar el fichero `batch.txt` (el equivalente de `pruebaBatch.txt` en el ejemplo de arriba) con los nombres de los ficheros a procesar. (ver más abajo cómo se han generado los ficheros de parámetros).
2. Ligera re-escritura del código en Sección 27; generar una función principal (p.e. `main_tfm.py`) y en el *reducer* hacer la llamada a dicha función en lugar de a `main_fake`.

En el Anexo se proporcionan los ficheros `mapperTFM.py` y `reducerTFM.py` que realizan todo esto de manera orquestada con Hadoop YARN, pero es importante mencionar brevemente ahora los distintos problemas y soluciones que se han encontrado hasta que este esquema ha funcionado.

4.5.2. Notas sobre `mapperTFM.py`

No hay mucho que añadir. El *mapper* finalmente implementado es casi idéntico al proporcionado en el ejemplo de juguete descrito en la sección anterior.

La única reseña con respecto al *mapper* es indicar que en un proceso *MapReduce* usando YARN, el usuario final **NO** determina el número de *mappers* que se ejecutan. El número de *mappers* viene dictado por el número de *splits* con el que el fichero de entrada a YARN (el `batch.txt`) se divide en bloques “mínimos” en HFS. Por defecto, el bloque mínimo en HFS es de 128 MB, nuestros ficheros de salida sobre HFS caben perfectamente en bloques de 10 MB y el fichero de entrada a *MapReduce* es apenas de unos KB. Modificamos el parámetro de configuración de Hadoop a bloques más pequeños pero en cualquier caso siempre mucho mayores que `batch.txt`. En

definitiva, todo el fichero `batch.txt` cabe en un solo bloque lo que implica que solo tenemos un proceso *mapper*. A efectos prácticos esta apreciación no tiene ninguna implicación, pero es bueno hacerlo constar por motivos conceptuales.

4.5.3. Notas sobre `reducerTFM.py` y ejecución YARN.

Aquí es donde han surgido gran cantidad de obstáculos En esta sección se detallan brevemente los más relevantes y la solución empleada:

- **Problema LEVE:** El código optimizado usa módulos `GaiaData.py` y `DiasCatalog.py` que no lograban pasarse como ficheros al comando YARN.

Solución: tras infructuosos intentos de generar un paquete con `GaiaData.py` y `DiasCatalog.py`, ponerlo en el repositorio de pip y descargarlo y habilitarlo en el directorio accesible por la instalación local de Python en cada nodo, se procede de una forma muy rudimentaria: los mencionados módulos son declaraciones de clases y se incorporan directamente al código `reducerTFM.py`.

- **Problema GRAVE:** en la máquina local las consultas a Vizier y a Gaia DR2 usando `astroquery` funcionan sin problemas. Al ejecutar el mismo código, desde terminal en un contenedor (no desde YARN) no hay manera de ejecutar una consulta a Gaia DR2. El problema se ilustra en la Figura 16 con la ejecución básica de una consulta `astroquery.gaia` correspondiente a un ejemplo básico (líneas resaltadas; primer ejemplo en la propia documentación del subpaquete `astroquery.gaia`). El terminal de arriba en la Figura 16 corresponde a la ejecución en Python desde el terminal del contenedor que actúa como *namenode*; el panel inferior en la misma figura corresponde a exactamente la misma búsqueda desde un terminal en la máquina anfitriona. Resolver este problema ha sido uno de los grandes obstáculos para poder llevara cabo el proyecto.

Solución: tras infructuosos intentos de resolver el problema accediendo a numerosos recursos on-line y realizar consultas al equipo desarrollador de `astropy` (sin respuesta), se opta por una solución radical, reinstalando todo el *cluster* partiendo de cero pero desde una imagen Ubuntu no etiquetada como *latest* sino explícitamente 18.04 (la misma que en máquina local) y exigiendo una versión de Python3 no la última sino 3.7. (como en máquina local).

```

user@namenode:~$ python
Python 3.7.7 (default, Jun 3 2020, 14:15:59)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import astropy.units as u
>>> from astropy.coordinates import SkyCoord
>>> from astroquery.gaia import Gaia
Created TAP+ (v1.2.1) - Connection:
  Host: gea.esac.esa.int
  Use HTTPS: True
  Port: 443
  SSL Port: 443
Created TAP+ (v1.2.1) - Connection:
  Host: geodata.esac.esa.int
  Use HTTPS: True
  Port: 443
  SSL Port: 443
>>>
>>> coord = SkyCoord(ra=280, dec=-60, unit=(u.degree, u.degree), frame='lcrs')
>>> width = u.Quantity(0.1, u.deg)
>>> height = u.Quantity(0.1, u.deg)
>>> r = Gaia.query_object_async(coordinate=coord, width=width, height=height)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/luser/.pyenv/versions/3.7.7/lib/python3.7/site-packages/astroquery/gaia/core.py", line 274, in query_object_async
    async_job=True, verbose=verbose, columns=columns)
  File "/home/luser/.pyenv/versions/3.7.7/lib/python3.7/site-packages/astroquery/gaia/core.py", line 216, in __query_object
    job = self.launch_job_async(query, verbose=verbose)
  File "/home/luser/.pyenv/versions/3.7.7/lib/python3.7/site-packages/astroquery/utills/tap/core.py", line 402, in launch_job_async
    autorun)
  File "/home/luser/.pyenv/versions/3.7.7/lib/python3.7/site-packages/astroquery/utills/tap/core.py", line 611, in __launchJob
    verbose=verbose)
  File "/home/luser/.pyenv/versions/3.7.7/lib/python3.7/site-packages/astroquery/utills/tap/conn/tapconn.py", line 273, in execute_tappost
    return self.__execute_post(context, data, content_type, verbose)
  File "/home/luser/.pyenv/versions/3.7.7/lib/python3.7/site-packages/astroquery/utills/tap/conn/tapconn.py", line 403, in __execute_post
    conn.request("POST", context, data, self.__postHeaders)
  File "/home/luser/.pyenv/versions/3.7.7/lib/python3.7/http/client.py", line 1252, in request
    self._send_request(method, url, body, headers, encode_chunked)
  File "/home/luser/.pyenv/versions/3.7.7/lib/python3.7/http/client.py", line 1298, in _send_request
    self._endheaders(body, encode_chunked=encode_chunked)
  File "/home/luser/.pyenv/versions/3.7.7/lib/python3.7/http/client.py", line 1247, in _endheaders
    self._send_output(message_body, encode_chunked=encode_chunked)
  File "/home/luser/.pyenv/versions/3.7.7/lib/python3.7/http/client.py", line 1026, in _send_output
    self.send(msg)
  File "/home/luser/.pyenv/versions/3.7.7/lib/python3.7/http/client.py", line 966, in send
    self.connect()
  File "/home/luser/.pyenv/versions/3.7.7/lib/python3.7/http/client.py", line 1422, in connect
    server_hostname=server_hostname)
  File "/home/luser/.pyenv/versions/3.7.7/lib/python3.7/ssl.py", line 423, in wrap_socket
    session=session
  File "/home/luser/.pyenv/versions/3.7.7/lib/python3.7/ssl.py", line 870, in _create
    self.do_handshake()
  File "/home/luser/.pyenv/versions/3.7.7/lib/python3.7/ssl.py", line 1139, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLError: [SSL: DH_KEY_TOO_SMALL] dh key too small (_ssl.c:1076)
>>>

```

```

bfemenia@kisp: ~/TFM
(base) [12] bfemenia@kisp:~/TFM > python
Python 3.7.7 (default, Mar 26 2020, 15:48:22)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import astropy.units as u
>>> from astropy.coordinates import SkyCoord
>>> from astroquery.gaia import Gaia
Created TAP+ (v1.2.1) - Connection:
  Host: gea.esac.esa.int
  Use HTTPS: True
  Port: 443
  SSL Port: 443
Created TAP+ (v1.2.1) - Connection:
  Host: geodata.esac.esa.int
  Use HTTPS: True
  Port: 443
  SSL Port: 443
>>>
>>> coord = SkyCoord(ra=280, dec=-60, unit=(u.degree, u.degree), frame='lcrs')
>>> width = u.Quantity(0.1, u.deg)
>>> height = u.Quantity(0.1, u.deg)
>>> r = Gaia.query_object_async(coordinate=coord, width=width, height=height)
INFO: Query finished. [astroquery.utills.tap.core]
>>>

```

Figura 16: PROBLEMA GRAVE: imposibilidad de ejecutar una consulta a Gaia DR2 desde el contenedor. Fuente: elaboración propia.

- **Problema MODERADO:** desconexiones esporádicas al servicio Gaia al realizar consultas con cierta frecuencia.

Solución: este problema solo aparece en caso de consultas repetidas con cierta frecuencia durante la fase de pruebas. Para evitar un problema similar pudiese arruinar una ejecución muy larga se modifica en BF20 la secuencia de consulta forzando que encaso de un error de conexión el sistema intente de nuevo la búsqueda pasados 30 segundos.

- **Problema MODERADO:** por motivos desconocidas es imposible importar el sub-paquete `astroquery.vizier` necesario para las consultas al catálogo Dias et al. (2002) al ejecutar el código desde YARN. A diferencia de lo descrito con las consultas al catálogo de Gaia, al realizar la consulta desde terminales tanto en contenedor `namenode` como en terminal en la máquina anfitriona, la consulta se ejecutaba sin problemas, sin embargo el solo hecho de importar el paquete `astroquery.vizier` generaba un error en YARN.

Solución: Se etiqueta como problema moderado porque estas consultas son solo necesarias durante la fase de determinación de los parámetros de DBSCAN y, además, el catálogo contiene un número muy limitado por lo que se puede descargar localmente y realizar las búsquedas de manera local cuando sea necesario. Para ello se modifica sustancialmente la parte correspondiente en BF20 que originalmente estaba en el módulo `DiasCatalog.py`, se descarga todo el catálogo Dias et al. (2020) y se salva en un objeto `pickle` (~256 KB). Incidentalmente, al eliminar la consulta *on-line* al catálogo Vizier se incrementa la velocidad de ejecución alcanzado un **factor de ganancia de 10.2**.

- **Problema GRAVE:** el programa optimizado y modificado para *MapReduce* recibe correctamente sus entradas por *stdin* pero realiza salidas en forma de escritura a ficheros csv y gráficos en etapas que corresponden a post-procesado y no deberían ser considerados en el proceso *MapReduce*.

Solución: Se reescriben partes del código `reducerTFM.py` haciendo que todos los resultados se hagan por *stdout* (es decir, imprimiendo en pantalla con la salvedad que durante la ejecución de *MapReduce* no se verán dichos resultados en pantalla). Aprovechando estas modificaciones, ya directamente todo incorporado en `reducerTFM.py`, se procede a la limpieza de código redundante o no usado y se eliminan de `reducerTFM.py` las partes que no van a ser ejecutadas durante el proceso *MapReduce* sino que son de post-procesado.

- **Problema GRAVE:** cuando ya todo funciona, YARN no da problemas al compilar y se lanza el proceso se observa que el gestor YARN está ejecutando un solo *Mapper* (esto se ha mencionado previamente y era esperable) y **un solo Reducer**, es decir, no hay procesos paralelos ejecutándose sino que YARN ejecuta secuencialmente *Mapper-Reducer* sin proporcionar computación distribuida.

Solución: esto resultó totalmente inesperado y a día de hoy se interpreta a que en configuraciones por defecto se consideran un número menor de *reducers* que de *mappers*. Tras numerosas consultas a recursos *on-line* se encuentra la solución: opción de ejecución **numReduceTasks** en llamada a YARN que permite especificar el número de *reducees* a ejecutar.

Llegados a este punto, ya todo funciona. Antes de iniciar la ejecución del programa de detección de sobre-densidades, se decide probar con un proceso *Map-Reduce* sobre YARN usando el *Hadoop Streaming* sobre la versión Python del clásico ejemplo de *word count* sobre un único fichero que cabe en un solo bloque de HFS para forzar solo un *mapper*. En una ejecución donde no se usa **numReduceTasks** en la llamada a YARN efectivamente solo se lanza un *mapper* y un solo *reducer* y la tarea se completa en **2.128 segundos**. Al tener habilitado un *cluster* con 12 *datanodes* se realiza la misma prueba con **-numReduceTasks 12** en la llamada a YARN y el mismo proceso se completa en **0.2256 segundos**.

Para concluir esta sección, el comando a lanzar desde el terminal en *namenode* cuando vayamos a ejecutar `mapperTFM.py` y `reducerTFM.py` adopta la siguiente sintaxis (asumiendo que se quieren 7 *reducers*):

```
yarn jar \
$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar \
-files mapperTFM.py, reducerTFM.py \
-mapper mapperTFM.py -reducer reducerTFM.py \
-numReduceTasks 7 \
-input batch_files/batch.txt -output result
```

donde:

- `yarn jar` es el comando par invocar ejecución del a tarea Java que lanza YARN.
- `$HADOOP_HOME/... streaming-*.jar` es el API para realizar el *Hadoop-streaming* y poder ejecutar tareas *MapReduce* sobre código con Python.
- `-files` es necesario para distribuir entre el *cluster* los ficheros a ejecutar. En caso de un solo nodo no sería necesario, pero en nuestro caso es **mandatario**.
- `-mapper` y `-reducer` especifican los scripts que actúan como *mapper* y *reducer*, respectivamente.
- `-numReduceTasks 7` especifica que se desean 7 tareas *reducer*.
- `-input` es la dirección en la estructura de ficheros HFS donde se encuentra la entrada de los datos, y `-output` es la dirección en HFS donde se salvarán los resultados de *MapReduce*. NOTA: el directorio especificado de salida en NO PUEDE existir (esto causa un error en YARN).

4.6. Mapeo de $(\epsilon, N_{\text{pts}})$ de DBSCAN sobre el 5% del plano de la Galaxia.

En esta sección se describe la aplicación del código ya desplegado en una plataforma de computación distribuida y optimizado para ejecutarse según un proceso MapReduce en el *cluster* Hadoop descrito en las secciones anteriores.

Antes de iniciar el trabajo se ejecutan las siguientes acciones basadas en el conocimiento básico del dominio del problema específico y de los datos a tratar:

1. La inmensa mayoría de cúmulos abiertos están distribuidos en zonas del plano de la Galaxia, y más concretamente a latitudes galácticas comprendidas en el rango $-20^\circ < b < 20^\circ$.
2. Por este hecho resulta más conveniente realizar búsquedas usando coordenadas galácticas (l, b) en lugar de (RA, Dec) . Esto conlleva una ulterior modificación incorporada en BF20.
3. El catálogo Gaia DR2 es completo²⁵ en el rango de magnitudes $G=[12, 17]$. En regiones muy densas la magnitud límite es $G=18$. Por esta razón, más allá de las pruebas de verificación de los códigos en las anteriores secciones, a día de hoy no tiene mucho sentido realizar búsquedas con $G > 18$. Por esta razón nuestro análisis se restringe a $G < 18$.

4.6.1. Generación de los ficheros de entrada para el escaneo del catálogo de Dias et al. (2020) con MapReduce.

Para ello se escribe el script `generate_ip_dias.py` que muestrea todo el catálogo de Dias et al. (2020) generando ficheros de entrada al código.

Los nombres de estos ficheros son los que introduciremos en el fichero `batch.txt` en HFS cuando vayamos a ejecutar *MapReduce* en Hadoop. La rutina `generate_ip_dias.py` genera todos los ficheros correspondientes al catálogo de Dias et al. (2020) completo. Nótese que como se ha indicado previamente, podemos incluir tantos catálogos como sea posible.

Posteriormente con la rutina `select_ip_DC_files.py` se seleccionan los ficheros comprendidos en el rango $-20^\circ < b < 20^\circ$, $l < 93^\circ$. Los ficheros en dicho zona de la Galaxia y que satisfacen los requerimientos para su selección suman un total de 192.

²⁵<https://www.cosmos.esa.int/web/gaia/dr2#:~:text=In%20dense%20areas%20on%20the,as%20bright%20as%20G%3D18.>

Lanzamos el código sobre el `batch.txt` con estos ficheros para realizar mapas espaciales de parámetros pseudo-óptimos de partida (ϵ , N_{pts}).

Los requerimientos impuestos en `generate_ip_dias.py` para generar ficheros en este trabajo son:

- Cada fichero debe corresponder a una zona de cielo con un diámetro menor o igual a 3° .
- Cada fichero debe contener al menos 3 cúmulos catalogados en Dias et al. (2002).
- Para evitar redundancias de ficheros que muestran la misma zona del cielo, se calcular el baricentro de las posiciones de los cúmulos catalogados en dicha zona. No pueden haber dos ficheros que tengan el mismo valor de baricentro: de esta manera evitamos ficheros que siendo ligeramente distintos en posición contienen exactamente los mismo cúmulos de catálogo.

4.6.2. Resultados de la ejecución. Interpretación de los resultados.

En la Figura 17 se muestra el resumen del procesado de 35 campos estelares donde es posible satisfacer los requisitos impuestos para generar un fichero con varios cúmulos abiertos del catálogo de Dias et al. (2002). Para cada entrada en la tabla en la Figura 17 se ha generado el equivalente de lo representado en la Figura 7 y el punto más brillante es el que aparece listado en la tabla.

En este caso concreto se ha buscado cubrir la zona comprendida en longitudes galácticas $l=[0^\circ, 28^\circ]$ y latitudes galácticas $b=[-20^\circ, 20^\circ]$ (aunque todos los cúmulos localizados en el rango del se distribuyen en una banda centrada en el disco de anchura 5°). Puede observarse que en la zona del centro de la Galaxia todos los campos seleccionados están a un par de grados en latitud del disco de la Galaxia.

Se observa una gran variabilidad de la función de mérito con la que determina el desempeño del algoritmo en la identificación de cúmulos conocidos y catalogados. Así, somos capaces de obtener aciertos de hasta el 100% en algunos campos mientras que en otros no es capaz de obtener ningún positivo. Las ejecuciones hicieron uso del rango predeterminado en RL19, a saber, $(\epsilon, N_{pts}) = [0.008, 0.03] \times [10, 100]$ que en la Figura 7 parecía indicar que muestreaba correctamente el espacio 2D de (ϵ, N_{pts}) . Ya en RL19 se apuntaba a la extrema variabilidad en la elección de los parámetros de DBSCAN y, sin embargo, las Figuras 17, 18, 19 y 20 muestran que tal variabilidad es mucho mayor de lo esperado y el rango posible de valores debe expandirse. Así pues, deberemos retomar este trabajo ejecutando primera una exploración del espacio 2D de (ϵ, N_{pts}) con un tamaño de paso de iteración mayor y lanzar una segunda iteración de ejecuciones con un paso más fino..

	id	ra	dec	l	b	eps	n_pts	M
0	729.0	266.940313	-28.759341	0.394551	-0.308970	0.008000	10.0	0.000000
1	731.0	267.447137	-28.828128	0.564521	-0.725049	0.020444	98.0	0.222222
2	732.0	267.472397	-29.010600	0.419279	-0.837714	0.024000	75.0	0.133333
3	735.0	267.555556	-28.881113	0.567776	-0.833741	0.016889	71.0	0.333333
4	736.0	267.580555	-28.880556	0.579477	-0.852249	0.012889	43.0	0.333333
5	738.0	268.073662	-27.862846	1.676098	-0.705210	0.008000	10.0	0.000000
6	739.0	268.227891	-28.482816	1.211445	-1.137646	0.020889	99.0	0.428571
7	742.0	268.116129	-27.312325	2.169093	-0.457276	0.017333	29.0	0.500000
8	745.0	269.850455	-25.369919	4.629928	-0.822316	0.024667	23.0	0.333333
9	746.0	269.860062	-28.705603	1.738982	-2.488706	0.016000	11.0	0.027027
10	747.0	269.883345	-24.716498	5.211493	-0.522854	0.015556	57.0	0.666667
11	748.0	270.234245	-28.203941	2.338308	-2.525947	0.027556	36.0	0.062500
12	749.0	269.963758	-24.892347	5.095263	-0.673663	0.027333	70.0	0.333333
13	753.0	270.818468	-27.839832	2.909770	-2.795557	0.022444	12.0	0.058824
14	741.0	268.936448	-22.781769	6.452345	1.196071	0.014889	19.0	0.333333
15	744.0	269.590941	-23.201311	6.391547	0.464411	0.015111	28.0	0.500000
16	750.0	270.479429	-23.396201	6.627942	-0.340580	0.015111	51.0	1.000000
17	751.0	270.827447	-22.178647	7.845333	-0.019088	0.015333	40.0	1.000000
18	752.0	270.664900	-22.998653	7.057831	-0.292547	0.025556	12.0	1.000000
19	754.0	270.866890	-22.530814	7.556723	-0.224167	0.026000	93.0	0.333333
20	755.0	271.485940	-20.494427	9.613904	0.272368	0.026000	11.0	0.030303
21	756.0	270.537890	-24.242088	5.918882	-0.804985	0.019333	22.0	0.200000
22	757.0	271.951146	-21.622855	8.840072	-0.656620	0.026444	50.0	1.000000
23	758.0	272.118713	-23.128594	7.598543	-1.522906	0.026889	48.0	1.000000
24	759.0	272.095914	-21.917416	8.647828	-0.917178	0.029333	29.0	0.333333
25	760.0	272.678979	-24.241208	6.871411	-2.509358	0.029556	13.0	0.333333
26	761.0	272.218606	-23.885284	6.980546	-1.969455	0.027333	65.0	0.333333
27	762.0	273.519011	-21.953654	9.251463	-2.091818	0.028667	18.0	0.166667
28	764.0	273.532000	-16.900254	13.696959	0.312811	0.030000	15.0	0.029412
29	765.0	274.133477	-18.841509	12.263452	-1.115965	0.022222	33.0	0.250000
30	766.0	274.344438	-18.472874	12.682865	-1.116851	0.014000	49.0	0.333333
31	767.0	274.167512	-19.511865	11.688696	-1.462715	0.029556	12.0	0.333333
32	768.0	274.323595	-18.796025	12.388950	-1.252747	0.015556	57.0	0.666667
33	769.0	274.858489	-16.641386	14.528381	-0.681593	0.022222	38.0	0.333333
34	770.0	274.358374	-19.466319	11.814194	-1.599497	0.029333	36.0	0.500000

Figura 17: captura de pantalla del DataFrame en Pandas mostrando los resultados de aproximadamente la mitad de las ejecuciones para muestrear (ϵ , N_{pts}) en función de (l , b) en la región cercana al centro Galáctico. Cada entrada en la tabla corresponde a una secuencia de 9100 iteraciones donde se muestrea el espacio de parámetros de DBSCAN. Fuente: elaboración propia.

Esta estrategia de doble ejecución con distintos tamaños de paso refuerza aún más el valor de tener disponible esta herramienta BF20 más veloz, distribuida y que puede beneficiarse de escalabilidad vertical y horizontal para acortar los tiempos de ejecución.

Con los datos en la tabla de la Figura 17 ya estamos en disposición de generar mapas de M , ϵ y N_{pts} y usarlos cuando lancemos el código en zonas del cielo donde no se tiene a disposición ningún cúmulo globular catalogado que pueda servir de entrenamiento para el algoritmo DBSCAN. A modo de ilustración, en las Figuras 18, 19 y 20 se muestran los mapas generados en esta demostración del código distribuido.

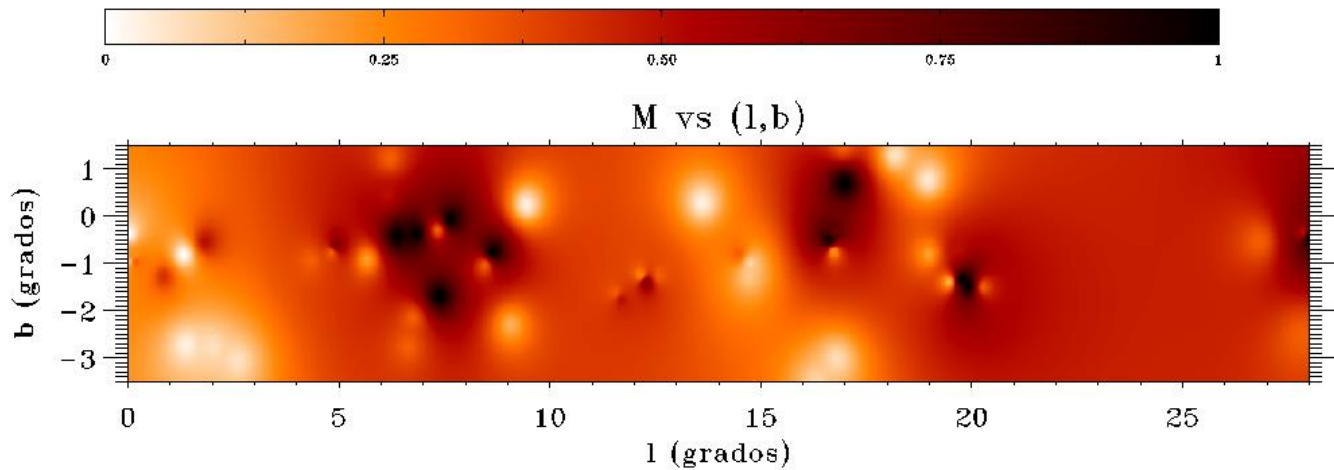


Figura 18: parámetro M usado como figura de mérito para determinar las mejores combinaciones (ϵ , N_{pts}) En esta figura se muestra cómo varía M en función de la posición del cielo considerada en base a las ejecuciones mostradas en la Figura 17.

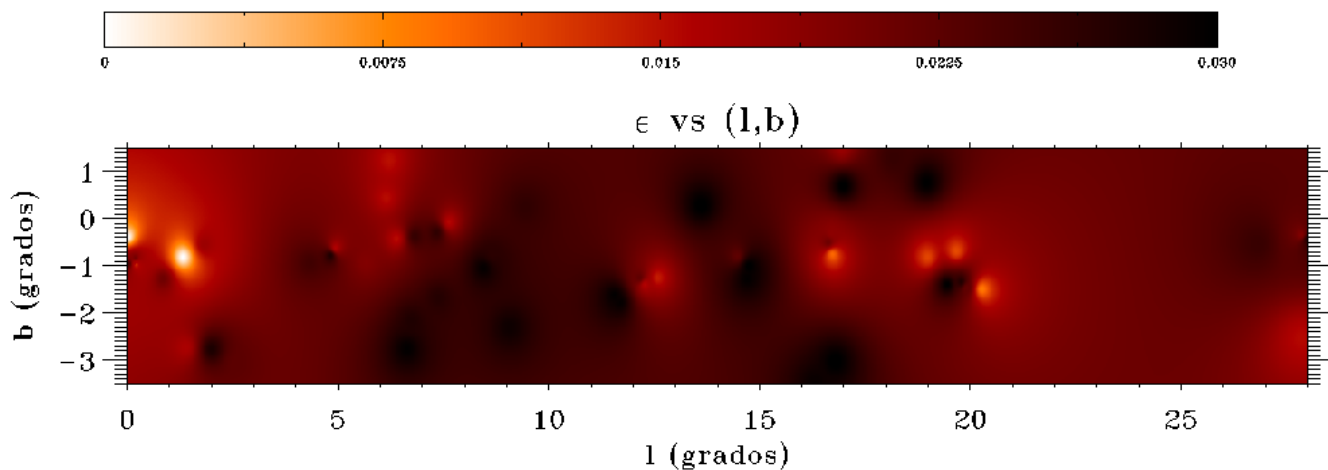


Figura 19: mapa del parámetro ϵ como función de su posición en el plano de la Galaxia en base a las ejecuciones mostradas en la Figura 17.

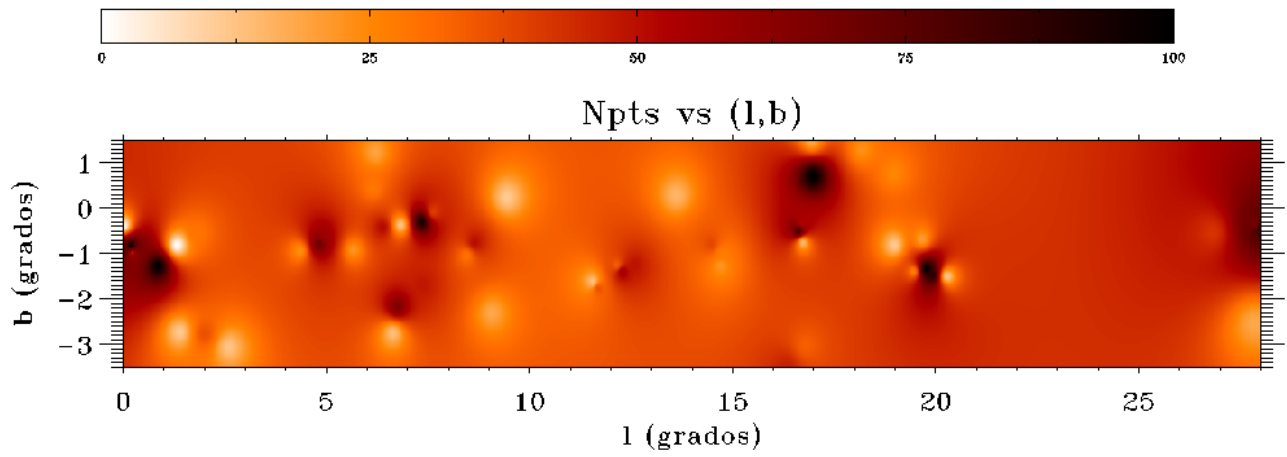


Figura 20: mapa del parámetro M como función de su posición en el plano de la Galaxia en base a las ejecuciones mostradas en la Figura 17.

5. Conclusiones y Futuros Trabajos.

A modo de sumario las principales conclusiones e hitos alcanzados en este trabajo son:

1. Se ha optimizado código para ser ejecutado en una máquina individual hasta por un factor 10.2.
2. Se ha implementado un *cluster* en el que se ha montado Hadoop.
3. Se ha modificado el código para ser ejecutado mediante un procedimiento *MapReduce* en el anterior *cluster*.
4. Se ha podido ejecutar de manera efectiva el código usando en su modalidad de tratamiento de los datos en 2-D.

La ganancia combinada en los puntos 1 a 3 punto permiten velocidades de ejecución de hasta un factor 90 con respecto al código en RL19 lanzado sobre la misma máquina.

Respecto a trabajos futuros que impliquen el uso de la optimización de BF20 y su implementación en un *cluster* con Hadoop como a la aquí presentada pero en un *cluster* con mayor número de máquinas y recursos RAM y CPU (escalabilidad horizontal y vertical):

- Re-ordenación del código en un paquete con módulos bien diferenciados en un repositorio `pip`.
- Re-escritura de la salida en *stdout* de `reducerTFM.py`. Es aconsejable simplificar las tareas y, como se aconseja para Hadoop, crear secuencias de trabajo que no sean más largas de unos ~10 minutos.
- Investigar posibles cuellos de botella en el código cuando se hace *clustering* sobre los datos 5-D.
- Incorporación de nuevos catálogos como *clusters* abiertos detectados para la mejora del pseudo-entrenamiento del algoritmo de búsqueda BF20 (p.e. los 582 cúmulos abiertos detectados y publicados recientemente por Castro-Ginard et al. (2020)).
- Aplicación en modalidad mosaico para la detección de sobre-densidades en los datos Gaia sobre todo el plano de la Galaxia.

6. Referencias

- Castro-Ginard, A. et al. (2018), *A new method for unveiling open clusters in Gaia. New nearby open clusters confirmed by DR2*, Astronomy & Astrophysics, 618, A59, doi: 10.1051/0004-6361/201833390
- Castro-Ginard, A. et al. (2019), *Hunting for open clusters in Gaia DR2: the Galactic anticentre*, Astronomy & Astrophysics, 627, A35, doi: 10.1051/0004-6361/201935531
- Castro-Ginard, A. et al. (2020), *Hunting for open clusters in Gaia DR2: 582 new open clusters in the Galactic disc*, Astronomy & Astrophysics, 635, A45. Doi: 10.1051/0004-6361/201937386
- Dias, W. S., Alessi, B. S., Moitinho, A., Lépine, J. R. D. (2002), *New catalogue of optically visible open clusters and candidates*. Astronomy & Astrophysics, 389, p.871-873 (2002).
- 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.
- Fernández Pena, T. (2019a), *Tema 0: Introducción a Docker*. Material no publicado. Recuperado el 30 de Mayo de 2019 de: https://campus.viu.es/bbcswebdav/pid-2032255-dt-content-rid-24584451_1/courses/2019_04_A_10297/Tema_0-IntroDocker.pdf (la mayor parte del material presentado aquí corresponde a Lorenzo del Castillo, 2019).
- Fernández Pena, T. (2019b), *Tema 4: Procesamiento distribuido del Big Data con Hadoop*. Universidad Internacional de Valencia. Material no publicado. Recuperado el 12 de Junio de 2019 de: https://campus.viu.es/bbcswebdav/pid-2042099-dt-content-rid-24842743_1/courses/2019_04_A_10297/Tema4-2x1%281%29.pdf
- Gaia Collaboration, Prusti, T. et al. (2016), *The Gaia Mission*. Astronomy & Astrophysics, 595, A1-A36. doi: 10.1051/0004-6361/201629272
- Gaia Collaboration, Brown, A.G.A. et al. (2018), *Gaia Data Release 2. Summary of the contents and survey properties*. Astronomy & Astrophysics, 616, 1G. Doi: 10.1051/0004-6361/201833051
- Høg, E. (2018), *Astrometry history: Hipparcos from 1964 to 1980*. Material no publicado en revista. Recuperado el 12 de Junio de 2020 de: <https://arxiv.org/abs/1804.10881>
- López, R. (2019), *Análisis de clustering aplicado a datos de la misión espacial GAIA*. (Tesis de Maestría). Universidad Internacional de Valencia. Recuperado de https://github.com/RAFAELLOPE/Star_Cluster_Detection

- Lorenzo del Castillo, J. A. (2019), *Introducción a los contenedores con Docker*. Material no publicado y distribuido en Fernández Pena (2019a). Recuperado el 30 de Mayo de 2019 de: https://campus.viu.es/bbcswebdav/pid-2032255-dt-content-rid-24584451_1/courses/2019_04_A_10297/Tema_0-IntroDocker.pdf
- Pedregosa , F. et al. (2011), *Scikit-learn: Machine Learning in Python*, Journal of Machine Learning Research, 12, 2825-2830.
- Perryman, M. A. C. (1986) *Hipparcos - the ESA Space Astrometry Mission: overview and status*. En Proceedings of the Nineteenth IAU General Assembly, Delhi, India, November 19-28, 1985.
- Radtka, Z. Y Minner, D. (2015), *Hadoop with Python*, Sebastopol (CA, USA): O'Reilly.
- Romero, F. (2019). Tema 2: Aprendizaje no Supervisado - Clustering. Universidad Internacional de Valencia. Material no publicado. Recuperado el día 1 de Octubre de 2019 de: https://campus.viu.es/bbcswebdav/pid-2218198-dt-content-rid-27486346_1/courses/2019_04_A_10301/S02_VC1_ALGORITMOS%20DE%20CLUSTERING%20I.pdf
- Sidhu, A. (2020, 19 de abril), *Tutorial: Building your Own Big Data infrastructure for Data Science*. Artículo en la web recuperado el 5 de mayo de 2020 de: <https://towardsdatascience.com/tutorial-building-your-own-big-data-infrastructure-for-data-science-579ae46880d8>
- Szalay, A. y Gray, J. (2001) *The Worl-Wide Telescope*, Science, 293, 2037.
- Tejedor, E. et al. (2017), *PyCOMPs: Parallel Computational Workflows in Python*, International Journal of High Performance Computing Applications, 31, 66.
- Zhang, Y. y Zhao, Y (2015), *Astronomy in the Big Data Era*, Data Science Journal, 14 (11), 1-9.

7. Anexos: códigos Python.

Todos los códigos en esta sección están disponibles en:
<https://github.com/bfemenia/VIU-TFM>

7.1. generate_ip_dias.py

Realiza la generación de los ficheros de entrada del código para la generación de mapa de parámetros.

```

1.  #!/usr/bin/env python3
2.  # -*- coding: utf-8 -*-
3.  """
4.  Created on Thu Jun 11May 14 20:17:23 2020
5.
6.  @author: bfemenia
7.  """
8.
9.  # %% IMPORT SECTION
10. #-----
11. import pandas as pd
12.
13. from astropy import units as u
14. from astropy.coordinates import Angle, Distance, Latitude, Longitude, SkyCoord
15.
16. from reducerTFM import DiasCatalog as DiasCatalog
17.
18. # %% MAIN CODE
19. #-----
20. def generate_ip_dias(l_max=360, r_max=1.5, n_min=3, path='home/bfemenia/TFM/IP
_files_DC'):
21.     """
22.     Bruno Femenia Castella
23.
24.     This routine generates the selection of ip files over the Galactic plane
25.     containing at least 3 clusters in Dias catalog and not beyond r_max degs of
26.     size. Each of these sky patches will be analyzed to derived eps=eps(l,b)
27.     and min_pts=min_pts(l,b) which will guide the search for clusters in a
28.     subsequent exploration of Gaia data.
29.
30.     Based on these files we evaluate the (eps,Nmin) to be used over different
31.     regions of whole Galactic plane to find new candidates with DBSCAN.
32.
33.     Returns
34.     -----
35.     None.
36.
37.     """
38.
39.     full_dc = DiasCatalog('Cluster', 'RAJ2000', 'DEJ2000', 'l', 'b', 'Class',
40.                           'Diam', entire=True)
41.     gal_disk= full_dc.table[ (abs(full_dc.table['b']) < 20) & (full_dc.table['l'
42. ] < l_max) ]
43.     patches=[]

```

```

44.
45.     for i, this_cluster in enumerate(gal_disk):      #Iterating over ALL clusters
46.                                     #in selected region |b|< 20 and l < l_max
47.         c1      = SkyCoord(this_cluster['l']*u.deg,
48.                             this_cluster['b']*u.deg,
49.                             frame='galactic')
50.         clusters = gal_disk[ (abs(gal_disk['b']-this_cluster['b']) <= r_max)]
51.
52.
53.         if len(clusters) < n_min:                    #Requesting a minimum number of clusters in this patch!!
54.             continue
55.
56.         candidates=[{'name':this_cluster['Cluster'], 'ang_sep':0.,
57.                      'l':this_cluster['l'], 'b':this_cluster['b']}]
58.
59.         for cluster in clusters:
60.             c2      = SkyCoord(cluster['l']*u.deg, cluster['b']*u.deg, frame='galactic')
61.             ang_sep = c1.separation(c2).deg
62.
63.             if (ang_sep <= r_max) and (ang_sep > 0):
64.                 new = dict({'name':cluster['Cluster'], 'ang_sep':ang_sep,
65.                             'l':cluster['l'], 'b':cluster['b']})
66.                 candidates.append(new)
67.
68.         if len(candidates) >= n_min:
69.             df = pd.DataFrame.from_records(candidates)
70.             df.sort_values('ang_sep', inplace=True)
71.             patch = df[0:n_min].mean()                #Use only n_min clusters at a time!!
72.             patches.append(patch)
73.
74.
75.         patches_df = pd.DataFrame.from_records(patches)
76.         patches_df.drop_duplicates(subset=['l','b'], inplace=True) #Removing duplicates
77.         patches_df.reset_index(drop=True, inplace=True)
78.
79.         #Adding rest of fields as defined by Rafa in his Thesis. Then save csv.
80.         #-----
81.         op_dict={'ra':[0.],
82.                  'dec':[0.],
83.                  'l':[0.],
84.                  'b':[0.],
85.                  'r':[0.],
86.                  'err_pos':[100],
87.                  'g_lim':[18.0],
88.                  'norm':[None],
89.                  'sample':[1.],
90.                  'dim3':[None],
91.                  'distance':['euclidean'],
92.                  'eps_min':[0.008],
93.                  'eps_max':[0.03],
94.                  'eps_num':[100],

```



```

95.         'min_pts_min':[10],
96.         'min_pts_max':[100],
97.         'min_pts_num':[91]}
98.     op_df=pd.DataFrame(data= op_dict)
99.
100.    for index, row in patches_df.iterrows():
101.        fn = 'ip_file_DC_'+str(index).zfill(5)+'.csv'
102.        c1 = SkyCoord(row['l']*u.deg, row['b']*u.deg, frame='galactic')
103.
104.        op_df['ra']= c1.icrs.ra.deg
105.        op_df['dec']= c1.icrs.dec.deg
106.        op_df['l']= row['l']
107.        op_df['b']= row['b']
108.        op_df['r']= row['ang_sep']
109.
110.        op_df.to_csv(fn, index=False, header=True)
111.
112.    return patches_df

```

7.2. select_ip_DC_files.py

Selecciona los ficheros generados con `generate_ip_dias.py` que se encuentran en una determinada caja del cielo.

```

1.  #!/usr/bin/env python3
2.  # -*- coding: utf-8 -*-
3.  """
4.  Created on Sat May 16 19:21:09 2020
5.
6.  @author: bfemenia
7.  """
8.
9.
10. # %% IMPORT SECTION
11. #-----
12. import pandas as pd
13. import shutil
14.
15.
16. # %% MAIN CODE
17. #-----
18. def select_ip_DC_files(lmin=0, lmax=10, bmax=20, op_name='batch_DC.txt', count
=False,
19.                        ip_path='/home/TFM/IP_files_DC/', op_path='/home/TFM/'):
20.     """
21.     Selects the Dias Catalog files corresponding to clusters with l <= lmax
22.     and |b| <= 20.
23.
24.     The file names are put into file opname and these will be read by the
25.     MapReduce in the Hadoop cluster as a typical batch job.
26.
27.     It also copies the ip_file_DC_? into the folder where MapReduce will be
28.     launched.
29.

```

```

30.     CAUTION: make sure you delete your old batch_DC.txt!!!
31.
32.     NOTICE: although MapRed will only be ran on the files in opname batch file
33.             it is convenient to remove the unneeded ip_file_DC to avoid
34.             confusions.
35.
36.     Parameters
37.     -----
38.     lmax : TYPE, numeric.
39.         Use files with ceter up to lmax. The default is 40.
40.
41.     bmax : TYPE, numeric
42.         Use files whose centers have |b| <= bmax. The default is 20.
43.
44.     ip_path : TYPE, string
45.         Path where ALL ip_file_DC_*.csv files are located.
46.         The default is '/home/TFM/IP_files_DC/'.
47.
48.     op_path : TYPE, string
49.         Path where the batch file and the ip_file_DC_* to be considered will
50.         be saved. ALL ip_file_DC_*.csv files are located.
51.
52.     op_name : TYPE, string
53.         O/P file name including path. The default is 'batch_DC.txt'.
54.
55.     Returns
56.     -----
57.     None. O/P is directly stored in opname file.
58.
59.     ...
60.
61.     df = pd.DataFrame()
62.
63.     for i in range(1021):
64.
65.         ip_file = 'ip_file_DC_'+str(i).zfill(5)+'.csv'
66.         df_tmp = pd.read_csv(ip_path + ip_file)
67.
68.         if ( (df_tmp['l'])[0] >= lmin) and (df_tmp['l'])[0] < lmax) and
69.             (abs(df_tmp['b'])[0]) <= bmax) ):
70.
71.             if count:
72.                 df= df.append(df_tmp, ignore_index=True)
73.
74.             else:
75.                 with open(op_path+ op_name,'a') as f:
76.                     #Writing selected filename into batch file
77.                     f.write(ip_file+'\n')
78.                     #Copy parameter csv file into folder where batch lives
79.                     shutil.copyfile(ip_path + ip_file,
80.                                     op_path + ip_file)
81.
82.             if count:
83.                 print(f'\n\t #Files in DC within l-
range= [{ lmin},{ lmax}]: \t\t {df.shape[0]}')

```

7.3. mapperTFM.py

Código muy sencillo que realiza la tarea de *Mapper*.

```
1.      #!/usr/bin/env python3
2.
3.      #very simple MAPPER function: identify function...
4.      #no key and as a value the IP filename for the reduce routine
5.
6.      #from astroquery.vizier import Vizier #To check whether this
7.      #line is also triggering MAPPER to fail: YES, error occurs
8.
9.      import sys
10.     for line in sys.stdin:
11.         print(line, end='')
```

7.4. reducerTFM.py

Rutina principal que ejecuta el proceso de ETL y posterior análisis de *clustering* con DBSCAN. La autoría original de este código corresponde a RL19 aunque una parte significativa del código original ha sido modificado de manera importante para optimizar el cálculo y adaptarlo a los requerimientos de un proceso MapReduce en Hadoop.

```
1.      #!/usr/bin/env python3
2.      # -*- coding: utf-8 -*-
3.      """
4.      Created on Sun Feb 10 11:23:54 2019
5.
6.      @author: rafa
7.
8.      Heavily modified by Bruno Femenía Castellá (bruno.femenia@gmail.com) to optiim
9.      execution time and arranged to be executed as apReduce process within a Hadoop
10.     cluster
11.     """
12.     # %% ##### IMPORT Section #####
13.     # %%
14.
15.     # Main basic modules: numpy, pandas and then also sys and datetime
16.     import sys
17.     import numpy as np
18.     import pandas as pd
19.     import datetime          # used for O/P filenames
20.     import time              # used to time code execution
21.     import pickle            # used to load Dias catalog
22.     import math
23.
24.     #Sklearn algorithm
25.     from sklearn.metrics.pairwise import pairwise_distances
26.     from sklearn.cluster import DBSCAN
27.     from sklearn.preprocessing import MinMaxScaler
28.     from sklearn.metrics import silhouette_score
29.
30.     #To suppress warnings
31.     import warnings
32.     warnings.filterwarnings("ignore")
```

```

33.
34.     # Third-party dependencies
35.     from astropy import units as u
36.     from astropy.coordinates import Angle, Distance, Latitude, Longitude, SkyCoord

37.     from astropy.table import Column, Table
38.     from astroquery.gaia import Gaia
39.
40.
41.     # %%
42.     def load_full_DiasCatalog(file='/home/TFM/DiasCatalog.pkl', verbose=False):
43.         """
44.         Bruno Femenia Castella:
45.
46.         Simple coad to load a pre-downloaded full Dias Catalog from Vizier
47.         using the DiasCatalog class. The download of the full catalog from
48.         Vizier exected using a lsightly modified version of what was written by
49.         Rafa Lopez in his TFM.
50.
51.         Because of problems with importing astroquery.vizier when running the
52.         code in a MapRed framweork within a Hadoop cluster, this approach was needed
53.         .
54.         As a side effect, I tested that the time to identify the Dias catalog
55.         clusters in an specific part of the sky (region in Rafa's TFM) is now
56.         6.5 times faster than the same specific region query to Vizier. BTW, this
57.         later approac is the one that helped to speed the code a factor 10 wrt
58.         the original Rafa's TFM code
59.         """
60.
61.         table=0
62.
63.         try:
64.             with open(file,'rb') as f:
65.                 table=pickle.load(f)
66.
67.         except IOError as e:
68.             print(f'Error loading file with catalog. Error is{e}')
69.
70.         except:
71.             print(f'Error reading file with catalog')
72.
73.         finally:
74.             if len(table) <= 2:
75.                 print('No catalog loaded!')
76.             else:
77.                 if verbose:
78.                     print(f'\tCatalog in {file} loaded succesfully')
79.
80.         return(table)
81.
82.
83.
84.     # %% ##### CLASSES DEFINITION Section #####
85.     # %%
86.
87.     # %% ***** Dias Catalog class *****

```

```

88.     class DiasCatalog(object):
89.
90.         # %%
91.         def __init__(self, *attributes, l0=0, b0=0, radius=0.5,
92.                       entire=False, DiasFile='/home/TFM/DiasCatalog.pkl'):
93.             """
94.             Bruno Femenia Castella:
95.
96.             in order to speed up calcularions:
97.
98.             1/ it does not make sense to scan the whole catalog but only those clus
99.             ters
100.             found within 1.5*radius [degs] around coordinates [ra_0, dec_0]
101.
102.             2/ It turns out astroquery.Vizier triggers an error in the MapRed in Ha
103.             doop.
104.             Rewritting this module so catalog is in a pickle
105.             """
106.             self.attributes = attributes
107.
108.             table = load_full_DiasCatalog(file=DiasFile, verbose=False)          #
109.             # Full Dias catalog here
110.             if attributes != ():
111.                 table = table[attributes]
112.
113.             if entire:
114.                 self.table = table
115.
116.             else:
117.                 #Computing distance of ALL Dias Catalog clusters to (l0, b0)
118.                 Delta_l = ( (table.as_array(names=['l']).data).astype('float64') - l0)
119.                 *(np.pi/180.) #conversion to rads
120.                 b2      = (table.as_array(names=['b']).data).astype('float64')
121.                 *(np.pi/180.) #conversion to rads
122.                 cb2_cb1 = np.cos(b2)*math.cos(math.radians(b0))
123.                 sb2_sb1 = np.sin(b2)*math.sin(math.radians(b0))
124.                 dist    = np.arccos( np.cos(Delta_l)*cb2_cb1 + sb2_sb1 ) * 180./np.pi
125.                 #conversion to degs
126.
127.                 #Selecting only those with dist <= 1.5 * radius
128.                 self.table = table[dist <= 1.5*radius]
129.
130.             # %%
131.             def get_ra_dec(self, cluster):
132.                 """Method to get right ascension and declination from a cluster
133.
134.                 Parameters
135.                 -----
136.                 cluster: Name of the cluster to get its position
137.
138.                 Returns
139.                 -----
140.                 SkyCoord object containing ra and dec in degrees
141.                 """
142.             try:

```

```

138.         row = self.table[np.where(self.table['Cluster'] == cluster)]
139.         ra = Angle(row['RAJ2000'], unit=u.hourangle)
140.         dec = Angle(row['DEJ2000'], unit=u.deg)
141.         return([float(ra.deg), float(dec.deg)])
142.     except:
143.         print('Cluster does not exist')
144.
145.     # %%
146.     def get_clusters(self, center, r):
147.         clusters = []
148.         ra_center = Longitude(center[0], unit=u.deg)
149.         dec_center = Latitude(center[1], unit=u.deg)
150.         center = SkyCoord(ra_center, dec_center)
151.
152.         for i in range(self.table['Cluster'].shape[0]):
153.             ra = Angle(self.table[i]['RAJ2000'], unit=u.hourangle)
154.             ra = ra.deg
155.             ra = Longitude(ra, unit=u.deg)
156.             dec = Latitude(self.table[i]['DEJ2000'], unit=u.deg)
157.             cluster = SkyCoord(ra, dec, frame='icrs')
158.             diam = float(self.table[i]['Diam'])*u.arcmin
159.
160.             if(np.isnan(diam)):
161.                 diam = 0.05
162.             else:
163.                 diam = diam.to('deg').value
164.
165.             if(cluster.separation(center).value <= r):
166.                 clusters.append((self.table['Cluster'][i], cluster, diam))
167.
168.         return(clusters)
169.
170.     # %%
171.     def get_match(self, center, r, clusters_detected):
172.         existing_clusters = self.get_clusters(center, r)
173.
174.         tot_matches = []
175.         for cluster in clusters_detected:
176.             ra_cluster_detected = Longitude(cluster[0], unit=u.deg)
177.             dec_cluster_detected = Latitude(cluster[1], unit=u.deg)
178.             cluster_detected = SkyCoord(ra_cluster_detected, dec_cluster_detected)
179.
180.             l_tmp = [match for match in existing_clusters if match[1].separation(cluster_detected).value <= match[2]/2.0]
181.             tot_matches = tot_matches + l_tmp
182.
183.             #building a Pandas DataFrame to easily remove duplicates by Name
184.             df = pd.DataFrame.from_records(tot_matches, columns=['Name', 'SkyCoord', 'Diam'])
185.             df.drop_duplicates(subset=['Name'], inplace=True)
186.
187.             #Re-assembling tot_matches as a list of tuples
188.             tot_matches = []
189.             for row in df.iterrows():
190.                 tmp = tuple(row[1])
191.                 tot_matches.append(tmp)

```

```

191.
192.     return tot_matches
193.
194.
195.     # %%          ***** Gaia Data query class *****
196.     class GaiaData(object):
197.
198.     # %%
199.     def __init__(self, attributes, coordsys = 'ICRS'):
200.         """Constructs an object representing an ADQL query
201.
202.         Parameters
203.         -----
204.         attributes: List of fields to query (projection)
205.         point: point in the sky to query
206.         extent: length of the box centered in the point
207.         """
208.         self.attributes = attributes
209.         self.coordsys = coordsys
210.
211.     # %%
212.     def astrometric_query_box(self, point, length, err_pos, g_lim):
213.         """Construct a query of a sky box centered in point specified when creati
ng this object
214.
215.         Parameters:
216.         -----
217.         point: Point of the vertex of the box
218.         extent: lenght of the arms of the bos
219.
220.         Returns
221.         -----
222.         ADQL Query
223.         data retrieve from the query
224.
225.         """
226.         query = "SELECT %s"%', '.join(self.attributes)
227.         query += " FROM gaiadr2.gaia_source"
228.         query += " WHERE CONTAINS(POINT('%s', ra, dec), BOX('%s', %s, %s, %s))=1"
%(self.coordsys,
229.                                self.coordsys, ', '.join([str(n) for n in point]), str(len
gth), str(length))
230.         query += " AND parallax IS NOT NULL"
231.         query += " AND parallax >= 0.0"
232.         query += " AND ra IS NOT NULL"
233.         query += " AND dec IS NOT NULL"
234.         query += " AND pmra IS NOT NULL"
235.         query += " AND pmdec IS NOT NULL"
236.         query += " AND ABS(ra_error) < %s"%str(err_pos)
237.         query += " AND ABS(dec_error) < %s"%str(err_pos)
238.         query += " AND ABS(parallax_error) < %s"%str(err_pos)
239.         query += " AND ABS(phot_g_mean_mag) < %s"%str(g_lim)
240.         #print('\n\t'+query)
241.
242.         data=[[-999]]
243.         while data[0][0] == -999:

```



```

244.         data = self.get_results(query)
245.         return(data)
246.
247.     # %%
248.     def astrometric_query_circle(self, point, radius, err_pos, g_lim):
249.         """Construct a query of a sky circle centered in point specified when creating this object
250.
251.         Parameters:
252.         -----
253.         radius: Radius of the circle to query
254.
255.         Returns
256.         -----
257.         ADQL Query
258.         data retrieved from the query
259.         """
260.         query = "SELECT %s"%', '.join(self.attributes)
261.         query += " FROM gaiadr2.gaia_source"
262.         query += " WHERE CONTAINS(POINT('%s', ra, dec), CIRCLE('%s', %s, %s))=1"%
(self.coordsys,
263.                                     self.coordsys, ', '.join([str(n) for n in point]), str(radius))
264.         query += " AND parallax IS NOT NULL"
265.         query += " AND parallax >= 0.0"
266.         query += " AND ra IS NOT NULL"
267.         query += " AND dec IS NOT NULL"
268.         query += " AND pmra IS NOT NULL"
269.         query += " AND pmdec IS NOT NULL"
270.         query += " AND ABS(ra_error) < %s"%str(err_pos)
271.         query += " AND ABS(dec_error) < %s"%str(err_pos)
272.         query += " AND ABS(parallax_error) < %s"%str(err_pos)
273.         query += " AND ABS(phot_g_mean_mag) < %s"%str(g_lim)
274.         print('\t'+query)
275.         data = self.get_results(query)
276.         return(data)
277.
278.     # %%
279.     def get_results(self, query):
280.         """Runs a job in GAIA archive to retrieve data
281.
282.         Parameters.
283.         -----
284.         query: Query for GAIA database
285.
286.         Returns
287.         -----
288.         data: Data retrieved from ADQL query
289.         """
290.         data = [[-999]]
291.         try:
292.             job = Gaia.launch_job_async(query)
293.         except:
294.             print('Query has failed')
295.             time.sleep(30)             #Waits for Gaia service to restore before running again!!

```

```

296.         else:
297.             data = job.get_results()
298.         return(data)
299.
300.
301.
302.     # %% ##### DATA EXTRACTION Section #####
303.     # %%
304.
305.     def extract_data(r, err_pos, g_lim, cluster=None, coord_icrs=None, coord_gal =
        None):
306.         .....
307.         Bruno Femenía Castellá
308.
309.         Parameters
310.         -----
311.         r : TYPE
312.             DESCRIPTION.
313.         err_pos : TYPE
314.             DESCRIPTION.
315.         g_lim : TYPE
316.             DESCRIPTION.
317.         cluster : TYPE, optional
318.             DESCRIPTION. The default is None.
319.         coord_icrs : TYPE, optional
320.             DESCRIPTION. The default is None.
321.         coord_gal : TYPE, optional
322.             DESCRIPTION. The default is None.
323.
324.         Returns
325.         -----
326.         data      : data retrived from query to Gaia.
327.         center    : ICRS coordinates (ra & dec)
328.         center_gal: Galactirc coordinates corresponding to center
329.         ...
330.
331.         # Define variables needed to make the query
332.         data = None
333.         dias_catalog = DiasCatalog('Cluster', 'RAJ2000', 'DEJ2000', 'l', 'b', 'Class
334.                                     'Diam', 'Dist', 'pmRA', 'pmDE', 'Nc', 'RV', 'o_RV',
335.                                     l0=coord_gal[0], b0=coord_gal[1], radius=r)
336.
337.         if (coord_icrs != None) or (cluster != None):
338.             if coord_icrs == None:
339.                 point = dias_catalog.get_ra_dec(cluster)
340.             else:
341.                 point = coord_icrs
342.             #Create the GAIA query
343.             attributes = ['source_id', 'ra', 'ra_error', 'dec', 'dec_error', 'l', 'b'
344.                           'parallax', 'parallax_error', 'pmra', 'pmdec', 'phot_g_mean_mag']
345.
346.             gaia_query = GaiaData(attributes)
347.             data      = gaia_query.astrometric_query_box( point, 2*r, err_pos, g_l
im)

```

```

347.         #data         = gaia_query.astrometric_query_circle(point,    r, err_pos, g_l
im)
348.
349.     return(data, point)
350.
351.
352.
353.     # %% ##### DATA PREPROCESSing Section #####
354.     # %%
355.
356.     def preprocessing(data, sample_fctr=1, threshold=9500):
357.
358.         global sample_factor
359.         sample_factor = sample_fctr
360.
361.         # 1/ transforms parallax data into distances in parsec
362.         data_metrics = data
363.         data_metrics['parallax'] = data_metrics['parallax'].to(u.parsec, equivalenci
es=u.parallax())
364.
365.         # 2/ Adapt data metrics to a numpy array
366.         np_data_metrics = np.transpose(np.array([data_metrics['ra'], data_metrics['d
ec'], data_metrics['parallax'],
367.                                                    data_metrics['pmra'], data_metrics['pmdec']]))
368.
369.         # 3/ Sample data: if a specific sampling is requested or too many points
370.         n_points= len(np_data_metrics)
371.         if n_points*sample_factor > threshold:
372.             sample_factor=threshold/n_points
373.
374.         if(sample_factor != 1):
375.             np.random.seed(0)
376.             idx = np.random.choice( np_data_metrics.shape[0], replace= False,
377.                                     size=int(sample_factor*np_data_metrics.shape[0]) )
378.             np_data_metrics = np_data_metrics[idx,:]
379.
380.         # 4/ Change coordinates from Spherical to Cartesian coordinate system
381.         ra = Longitude(np_data_metrics[:,0], unit=u.deg)
382.         ra.wrap_angle = 180 * u.deg
383.         dec = Latitude(np_data_metrics[:,1], unit=u.deg)
384.         dist = Distance(np_data_metrics[:,2], unit=u.parsec)
385.         sphe_coordinates = SkyCoord(ra, dec, distance = dist, frame='icrs', represen
tation_type='spherical')
386.         cart_coordinates = sphe_coordinates.cartesian
387.
388.         # 5/ Adapt data to normalize it correctly
389.         data_sphe_adapted = np.transpose(np.array([sphe_coordinates.ra, sphe_coordin
ates.dec, sphe_coordinates.distance]))
390.         data_cart_adapted = np.transpose(np.array([cart_coordinates.x, cart_coordina
tes.y, cart_coordinates.z]))
391.         data_pm_adapted = np_data_metrics[:,3:5]
392.         data_all_adapted = np.append(data_cart_adapted, data_pm_adapted, axis=1)
393.
394.         return(data_sphe_adapted, data_cart_adapted, data_all_adapted)
395.
396.

```

```

397.
398. # %% ##### DBSCAN Section #####
399. # %%
400.
401. def get_distances_for_ML(X, Y):
402.     distance = 0
403.     ra1 = X[0]*u.deg
404.     ra2 = Y[0]*u.deg
405.     dec1 = X[1]*u.deg
406.     dec2 = Y[1]*u.deg
407.     if(len(X) == 3):
408.         dist1 = X[2]*u.parsec
409.         dist2 = Y[2]*u.parsec
410.         point1 = SkyCoord(ra1, dec1, dist1)
411.         point2 = SkyCoord(ra2, dec2, dist2)
412.         distance = point1.separation_3d(point2)
413.     else:
414.         point1 = SkyCoord(ra1, dec1)
415.         point2 = SkyCoord(ra2, dec2)
416.         distance = point1.separation(point2)
417.     return(distance.value)
418.
419.
420. # %%
421. def get_distance_matrix(data, scale=False, metric='euclidean'):
422.     if scale:
423.         scaler = MinMaxScaler()
424.         data_scaled = scaler.fit_transform(data)
425.     else:
426.         data_scaled = data
427.     if metric != 'euclidean':
428.         dist_matrix = pairwise_distances(data_scaled, metric=get_distances_for_ML
, n_jobs=-1)
429.     else:
430.         dist_matrix = pairwise_distances(data_scaled, metric='euclidean', n_jobs=
-1)
431.     return(dist_matrix)
432.
433.
434. # %%
435. def DBSCAN_eval(data, center, center_gal,r,sample_factor,ftype,dim3, eps_range
,
436.                 min_samples_range, scale=False, metric='euclidean', pm=False):
437.
438.     data_sphe, data_cart, data_all = preprocessing(data, sample_factor)
439.     data_search = data_sphe[:,2]
440.     if ftype == 'cart':
441.         if dim3:
442.             data = data_cart
443.         else:
444.             data = data_cart[:,2]
445.     elif ftype == 'sphe':
446.         if dim3:
447.             data = data_sphe
448.         else:
449.             data = data_sphe[:,2]
450.     elif ftype == 'pm':

```

```

451.     data = data_all
452. else:
453.     print('Specify tpye of dataset: cart, sphe, pm')
454.     sys.exit()
455.
456.     dist_matrix = get_distance_matrix(data, scale, metric)
457.
458.     if pm:
459.         data = data[:, :3]
460.
461.     dias_catalog = DiasCatalog('Cluster', 'RAJ2000', 'DEJ2000', 'Class', 'l', 'b',
462.                                'Diam', 'Dist', 'pmRA', 'pmDE', 'Nc', 'RV', 'o_RV',
463.                                l0=center_gal[0], b0=center_gal[1], radius=r)
464.
465.     num_cum      = len(dias_catalog.get_clusters(center, r))
466.     sscores      = {}
467.     tmp_d_sscores = {} # Temp Dict sscor
468.     tmp_l_sscores = [] # Temp List sscor
469.     matches       = []
470.     cluster_centers = []
471.     index         = 0
472.
473.     for eps in eps_range:
474.
475.         for min_samples in min_samples_range:
476.             db = DBSCAN(eps=eps, min_samples=min_samples, metric="precomputed",
477.                          n_jobs=-1).fit(dist_matrix)
478.             labels = db.labels_
479.             for i in range(len(set(labels))-1):
480.                 cluster = data_search[np.where(labels == i)]
481.                 cluster_center = cluster.mean(axis=0)
482.                 cluster_centers.append(cluster_center)
483.
484.                 tmp_l_sscores.append( eps )
485.                 tmp_l_sscores.append( min_samples )
486.
487.                 matches = dias_catalog.get_match(center, r, cluster_centers)
488.                 if(len(matches) > 0 and len(set(labels)) > 1):
489.                     tmp_l_sscores.append(len(matches)/(num_cum + (len(cluster_centers)-
490. len(matches))))
491.                 else:
492.                     tmp_l_sscores.append(0.0)
493.
494.                 tmp_l_sscores.append( cluster_centers)
495.                 tmp_d_sscores = {index:tmp_l_sscores}
496.
497.                 sscores = {**sscores, **tmp_d_sscores} # Merging two dicts
498.                 index += 1 # Increasing index for next iteration
499.                 tmp_d_sscores = {} # Resetting for next iteration

```

```

499.         tmp_l_sscores    = []                                # Resetting         for next ite
ration
500.         cluster_centers = []                                # Resetting         for next ite
ration
501.
502.     return(sscores, dist_matrix, data_search)
503.
504.
505. # %%      MAIN CODE #####
#
506. # %%      This to be uncommeted if wishing to run from terminal.
507.
508. # %%
509. #if __name__ == "__main__":
510. #    main()
511.
512.
513. # %%
514. # %%
515. # %% #####      REDUCER      #####
516.
517. def run_ip_file_test(ip_file):
518.     t0 = time.time()
519.     print(f'Faking parallel execution of file >>{ip_file}<<')
520.     for j in range(10):
521.         dummy = np.random.randn(8192,8192)
522.         dummy[0]=1
523.         print(f'\t\tDone with this fake iteration. Seconds: {time.time()-t0}\n\n')
524.
525.
526.
527. def run_ip_file(ip_file='ip_file_000001.csv', ftype='sphe'):
528.
529.     #Parsing filename
530.     #-----
531.     ip_file=ip_file.split(sep='\n')[0]
532.     ip_file=ip_file.split(sep='\t')[0]
533.
534.     # From ip_file loading I/P parameters into PAndas dataframe: params
535.     #-----
536.     params = pd.read_csv('/home/TFM/'+ip_file)
537.
538.     #These lines directly copied from deprectaeed functions: process_line & run_
ip_file
539.     #-----
-----
540.     ra            = params['ra'][0]
541.     dec           = params['dec'][0]
542.     l             = params['l'][0]
543.     b             = params['b'][0]
544.     radius        = params['r'][0]
545.     err_pos       = params['err_pos'][0]
546.     g_lim         = params['g_lim'][0]
547.     sample_factor = params['sample'][0]
548.     scale         = params['norm'][0] == 'X'
549.     metric        = params['distance'][0]
550.     dim3          = params['dim3'][0] == 'X'

```

```

551.     eps_min      = params['eps_min'][0]
552.     eps_max      = params['eps_max'][0]
553.     eps_num       = params['eps_num'][0]
554.     min_pts_min   = params['min_pts_min'][0]
555.     min_pts_max   = params['min_pts_max'][0]
556.     min_pts_num   = params['min_pts_num'][0]
557.     eps_range     = np.linspace(eps_min, eps_max, eps_num)
558.     min_pts_range = np.linspace(min_pts_min, min_pts_max, min_pts_num)
559.     center_icrs   = [ra, dec]
560.     center_gal    = [l, b]
561.
562.     if sample_factor == None:
563.         sample_factor = 1
564.
565.
566.     # print(f'\n\nReading file {ip_file}      #eps: {eps_num}      #Npts: {min_pts_n
um}') #For TESTING purposes
567.
568.     # Running ETL process: query(Extraction), Loading and Transformation
569.     #-----
570.     data, center = extract_data(radius, err_pos, g_lim, coord_icrs=center_icrs,
coord_gal=center_gal)
571.     # print('\t Done with extract_data')
572.
573.     # Running DBSCAN process: -> Main O/P is param_scores
574.     #-----
575.     param_scores, dist_matrix, data_search = DBSCAN_eval(data, center, center_ga
l, radius, sample_factor,
576.                                                         ftype, dim3, eps_range, min_pts_range, scal
e, metric)
577.     # print('\t Done with DBSCAN_eval')
578.
579.     # Reporting execution times: This to be commented out when running within Ma
pRed in HAdoop
580.     #-----
581.     # print("\n\n\t\t\t\t\t Program execution: 9100 iters   ---> %s seconds ---
" % (t1 - t0))
582.     # print("\t\t\t\t\t Saving results into csv           ---> %s seconds ---
" % (t2 - t1))
583.     # print("\n\t >>> TOTAL time to process a single I/P file:  %s seconds ---
" % (t2- t0))
584.
585.     # Analyzing DBSCAN results: this to be commented out if using withihn MapRed
Hadoop
586.     #-----
587.     # DBSCAN_result(param_scores, dist_matrix, data_search, center, radius)
588.
589.
590.     #####
591.     ##### IMPORTANT: Uncommnet following lines to execute in MapRed in Hadoop
592.     #####
593.
594.     #-----
595.     # >> FOR MAPRED in HAdoop: this prints results into stdout as requested
596.     #   by Hadoop Streaming!!
597.     #

```



```

598.     # Printing O/P on stderr to generate results file in HFS
599.     # (when executed from within cluster with Hadoop)
600.     #-----
601.     for k, v in param_scores.items():
602.         print(ip_file, ",", ra, ",", dec, ",", l, ",", b, ",", radius, ",", g_lim
603.             ", ", sample_factor, ", ", v[0], ", ", v[1], ", ", v[2], end=', ')
604.         if v[2] > 0:
605.             for coord in v[3]:
606.                 print(coord[0], ',', coord[1], end=', ')
607.             print()
608.
609.
610.     # %%
611.     # %%
612.     # %%     REDUCER     #####
613.     #
614.     # Reducer goes here!!
615.     #-----
616.     for value in sys.stdin:
617.         run_ip_file(value)

```

7.5. process_MapRed.py

El objetivo de esta rutina es la de recuperar los resultados salvados como ficheros csv por MapReduce y cargarlos como DataFrames de Pandas sobre los que realizar el post-procesado/análisis/visualización.

```

1.     #!/usr/bin/env python3
2.     # -*- coding: utf-8 -*-
3.     """
4.     Created on Sun Jun 21 12:35:06 2020
5.
6.     @author: bruno.femenia@gmail.com
7.
8.     This code takes the cvs files produced during the MapRed process with Hadoop
9.     and reformats the data within the cvs file into the DataFrames expected by the
10.    post-processing routines generated in RL19.
11.    """
12.
13.    # %%     IMPORT SECTION
14.
15.    import pandas as pd
16.    import numpy as np
17.    import datetime
18.    import matplotlib.pyplot as plt
19.
20.    from sklearn.metrics.pairwise import pairwise_distances
21.    from sklearn.cluster import DBSCAN
22.
23.    from reducerTFM import *     #Recall DiasCatalog and GaiaData classes were
24.                                #defined within reducerTFM.py
25.
26.

```

```

27. # %%
28. # Wrapper to match structure of O/P from MapRed to format in mainBL20.py
29.
30. def results_to_df(ipfile= 'results_DC_l00_l05.csv',
31.                   ipath = '/home/TFM/Results_DC/',
32.                   file_to_extract= 'ip_file_DC_00748.csv'):
33.     '''
34.     Bruno Femenia Castella
35.
36.     This routine loads into a Pandas DataFrame the results of the execution
37.     of the code in MapRed in Hadoop. This acts as a wrapper between what
38.     reducerTFM.py produces and what the rest of post-processing and plotting
39.     routines in RL19 are expecting
40.
41.     Parameters
42.     -----
43.     ipfile : string,
44.         The overall file storing ALL the part-00* files generated by MapRed
45.         during the bath processing.
46.         The default is 'results_DC_l00_l05.csv'
47.
48.     ipath : string,
49.         Path to where ip file is. The default is '/home/TFM/Results_DC/'.
50.
51.     file_to_extract : string,
52.         specifies which file in the batch process is to be processed.
53.         The default is 'ip_file_DC_00748.csv'.
54.
55.     Returns
56.     -----
57.     df      : Pandas DataFrame,
58.         df follows same structure as in original RL19 and BF20 codes.
59.
60.     '''
61.
62.
63.     #Reading full file but only keeping the rows with file_to_extract
64.     #Also reading a variable number of columns.
65.     #Solutions obtained from:
66.     #
67.     # stackoverflow.com/questions/13651117/how-can-i-filter-lines-on-load-in-
68.     #pandas-read-csv-function
69.     #
70.     # stackoverflow.com/questions/15242746/handling-variable-number-of-columns
71.     #-with-pandas-python
72.
73.     c1 = ['ipfile', 'ra', 'dec', 'l', 'b', 'r', 'glim', 'sample',
74.           'epsilon', 'minpts', 'local_score']
75.     c2 = []
76.     for i in range(3000): #10000
77.         dummy = 'clstr_' + str(i).zfill(4)
78.         c2.append(dummy+'_ra')
79.         c2.append(dummy+'_dec')
80.
81.     c_names = c1 + c2

```

```

82.     iter_csv = pd.read_csv(ipath+ipfile,iterator=True,chunksize=9100,
83.                             names=c_names, engine='python')
84.
85.     df_tmp = pd.concat([chunk[chunk['ipfile'] == file_to_extract]
86.                         for chunk in iter_csv])
87.     df_tmp.reset_index(drop=True, inplace=True)
88.
89.     sscores = df_tmp[['epsilon', 'minpts', 'local_score']]
90.     sscores_meta = df_tmp.iloc[0][c1[0:8]]
91.
92.     return(sscores, sscores_meta)
93.
94.
95.     #sscores, sscores_meta = results_to_df('results_test.csv')
96.
97.
98.     # %% DATA PLOTTING: taken directly from nain_BF20.py
99.     def plot_clusters(X, labels, op_name):
100.         # Black removed and is used for noise instead.
101.         size = 6.0
102.         f = plt.figure(figsize=(20,20))
103.         unique_labels = set(labels)
104.         colors = [plt.cm.Spectral(each)
105.                  for each in np.linspace(0, 0.5, len(unique_labels))]
106.         for k, col in zip(unique_labels, colors):
107.             if k == -1:
108.                 # Black used for noise.
109.                 col = [0, 0, 0, 1]
110.                 size = 3.5
111.
112.             class_member_mask = (labels == k)
113.
114.             xy = X[class_member_mask]
115.             plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
116.                     markeredgecolor='k', markersize=size)
117.             size=6.0
118.
119.         plt.title('Estimated number of clusters: %5d' % int(len(set(labels))-1))
120.         plt.xlabel("Right Ascension (deg)")
121.         plt.ylabel("Declination (deg)")
122.
123.         f.savefig(op_name)
124.
125.
126.     # %%
127.     def plot_score(param_scores, op_name):
128.         np_param_scores = param_scores.values
129.         eps = np.sort(np.array(list(set(param_scores['epsilon']))))
130.         Nmin = np.sort(np.array(list(set(param_scores['minpts']))))
131.         Z = np.empty((len(Nmin), len(eps)))
132.         fig, ax = plt.subplots(constrained_layout = True)
133.         X, Y = np.meshgrid(eps, Nmin)
134.         for i, n in enumerate(Nmin):
135.             for j, e in enumerate(eps):
136.                 Z[i,j] = np_param_scores[np.where((param_scores['epsilon'] == e) &
137.                                                     (param_scores['minpts'] == n)),2]

```

```

138.     extend = "neither"
139.     cmap = plt.cm.get_cmap('hot')
140.     CS = ax.contourf(X,Y,Z, cmap=cmap, extend=extend)
141.     fig.colorbar(CS)
142.     ax.set_xlabel('Epsilon')
143.     ax.set_ylabel('Nmin')
144.     ax.set_title('DBSCAN matching M')
145.
146.     fig.savefig(op_name)
147.
148.
149. # %%
150. # Using the above as a wrapper between MapRed O/P and expected by BL20
151. # this resumes where reducerTFM left it at.
152.
153. def process_reducerTFM(ipfile = 'results_DC_l00_l05.csv',
154.                        ipath = '/home/TFM/Results_DC/', plot=False,
155.                        file_to_extract='ip_file_DC_00748.csv'):
156.     ....
157.     Bruno Femenia Castella
158.
159.     Using results_to_df as a WRAPPER to put the MapReduce O/P data in the
160.     format expected by BL20, this routine resumes BL20 from where reducerTFM
161.     left it at.
162.
163.     Parameters
164.     -----
165.     ipfile : string,
166.         The overall file storing ALL the part-00* files generated by MapRed
167.         during the bath processing.
168.         The default is 'results_DC_l00_l05.csv'
169.
170.     ipath : string,
171.         Path to where ip file is. The default is '/home/TFM/Results_DC/'.
172.
173.     file_to_extract : string,
174.         specifies which file in the batch process is to be processed.
175.         The default is 'ip_file_DC_00748.csv'.
176.
177.     Returns
178.     -----
179.     None.
180.
181.     ...
182.
183.     # 1/ Loading results from MapRed for this ip_file_DC
184.     #-----
185.     param_scores, param_meta = results_to_df(ipfile, ipath, file_to_extract)
186.
187.
188.     # 2/ Launching the Gaia query to retrieve the data in this ip_file_DC
189.     # This is entirely for visual purposes displaying the stars and
190.     # and clusters detected in each field sampled by ip_file_DC
191.     #-----
192.     coord_icrs =[ param_meta['ra'], param_meta['dec'] ] # RA & Dec
193.     coord_gal  =[ param_meta['l'], param_meta['b'] ]  # l & b

```

```

194.     radius      = param_meta['r' ]                # radius
195.     g_lim       = param_meta['glim' ]             # g_lim
196.     sample      = param_meta['sample']
197.
198.     data, center = extract_data(radius, 100, g_lim,
199.                                coord_icrs=coord_icrs, coord_gal=coord_gal)
200.
201.
202.     # 3/ Last to be at where reducerTFM left wrt BF20 -> we need dist_matrix
203.     #-----
204.     data_sphe, data_cart, data_all = preprocessing(data, sample)
205.     data_search = data_sphe[:,2]                  #For TFM no scale -> no NORM.
206.     dm = pairwise_distances(data_search,          #For TFM always ftype='sphe'
207.                             metric='euclidean', #Metric always euclidean
208.                             n_jobs=-1)
209.     dist_matrix = dm
210.
211.
212.     # 4/ Running same sequence of steps as in DBSCAN_result
213.     #-----
214.     dias_catalog = DiasCatalog('Cluster', 'RAJ2000', 'DEJ2000', 'l', 'b', 'Class',
215.                                'Diam', 'Dist', 'pmRA', 'pmDE', 'Nc', 'RV', 'o_RV',
216.                                l0=coord_gal[0], b0=coord_gal[1], radius=radius)
217.
218.     cluster_centers = []
219.     sorted_scores= param_scores.sort_values(by=['local_score'],ascending=False)
220.
221.     opt_epsilon = sorted_scores.iloc[0,0]
222.     opt_min_pts = sorted_scores.iloc[0,1]
223.     best_M      = sorted_scores.iloc[0,2]
224.
225.     # 5/ Launching again DBSCAN with optimal pars. This to retrieve the
226.     #     detected clusters. We ca do this faster but it takes shorter to
227.     #     copy the lines (to improve: this is already in ip_file)
228.     #-----
229.     id_str = ((file_to_extract.split('_')[3]).split('.')[0])
230.     id_nb  = int(id_str)
231.     ra     = coord_icrs[0]
232.     dec    = coord_icrs[1]
233.     l      = coord_gal[ 0]
234.     b      = coord_gal[ 1]
235.
236.     if (best_M > 0) and plot:
237.         db = DBSCAN(eps=opt_epsilon, min_samples=opt_min_pts,
238.                     metric='precomputed', n_jobs=-1).fit(dist_matrix)
239.
240.         labels = db.labels_
241.         for i in range(len(set(labels))-1):
242.             cluster = data[np.where(labels == i)]
243.             cluster_center = cluster.mean(axis=0)
244.             cluster_centers.append(cluster_center)
245.
246.         matches = dias_catalog.get_match(center, radius, cluster_centers)
247.         for i, m in enumerate(matches):

```

```

248.         if (i==0):
249.             print(f'\n\nResults for {file_to_extract}:\n'+33*'=')
250.             print(f'\tBox (RA, Dec) = ({ra}. {dec}) degs , '
251.                   f'side={2*radius} degs\n')
252.
253.             print('\t\t>> Cluster found: %s'%(m[0]))
254.
255.             plot_clusters(data, labels, 'plt_ID'+id_str+'_clstrs.png')
256.             plot_score(param_scores, 'plt_ID'+id_str+'_scores.png')
257.
258.             return(id_nb, ra, dec, l, b, opt_epsilon, opt_min_pts, best_M)
259.
260.
261.     # %%
262.     #     Creating maps of opt_epsilon, opt_min_pts, best_M
263.
264.     def get_df(ipfile='results_DC_l00_l05.csv',
265.               ipath='/home/TFM/Results_DC/',
266.               batch='batch_DC_l_00_l_05.txt'):
267.
268.         cols = ['id', 'ra', 'dec', 'l', 'b', 'eps', 'n_pts', 'M']
269.         df = pd.DataFrame(columns=cols)
270.
271.         with open(ipath+batch, 'r') as f:
272.             files = f.read().splitlines()
273.
274.         for file in files:
275.             id_nb, ra, dec, l, b, opt_epsilon, opt_min_pts, best_M = \
276.                 process_reducerTFM(ipfile=ipfile, ipath=ipath, file_to_extract=file)
277.
278.             tmp = pd.Series([id_nb, ra, dec, l, b, opt_epsilon, opt_min_pts, best_M],
279.                             index=df.columns)
280.
281.             df = df.append(tmp, ignore_index=True)
282.
283.         return(df)
284.
285.
286.     ipath = '/home/TFM/Results_DC/'
287.
288.     ipfile = 'results_DC_l00_l05.csv' #Analysing files l=[0,5]
289.     batch = 'batch_DC_l_00_l_05.txt'
290.     df_l00_l05 = get_df(ipfile=ipfile, ipath=ipath, batch=batch)
291.
292.     ipfile = 'results_DC_l05_l14.csv' #Analysing files l=[5,14]
293.     batch = 'batch_DC_l_05_l_14.txt'
294.     df_l05_l14 = get_df(ipfile=ipfile, ipath=ipath, batch=batch)

```

7.6. Código completo BF20.

Aquí se proporciona el código completo **BF20** que puede ser ejecutado en una máquina individual. El código se distribuye en tres módulos:

- `DiasCatalog.py`: define la clase `DiasCatalog` que permite hacer consultas al catálogo Dias et al. (2002) y otras funcionalidades específicas cuando se usa esta clase.
- `GaiaData.py`: define la clase `GaiaDat` que se encarga de hacer la consulta a Gaia DR2.
- `main_BF20.py`: módulo principal de código donde se ejecuta el proceso ETL completo y el análisis de los resultados obtenidos.

7.6.1. `DiasCatalog.py`

```

1.  #!/usr/bin/env python3
2.  # -*- coding: utf-8 -*-
3.  """
4.  Created on Sun Oct 21 14:57:32 2018
5.
6.  @author: rafa
7.  """
8.
9.  from astroquery.vizier import Vizier
10. from astropy.coordinates import Angle, Longitude, Latitude, SkyCoord
11. from astropy.table import Column, Table
12. from astropy import units as u
13. import numpy as np
14. import matplotlib.pyplot as plt
15.
16. #To time code execution
17. import time
18.
19. def parse_radec(ra, dec):
20.     tmp = ra.split()
21.     ra_hms = tmp[0]+'h'+tmp[1]+'m'+tmp[2]+'s'
22.
23.     tmp = dec.split()
24.     dec_deg= tmp[0]+'d'+tmp[1]+'m'+tmp[2]+'s'
25.
26.     return ra_hms, dec_deg
27.
28.
29. def load_full_DiasCatalog(file='/home/bfemenia/TFM/DiasCatalog.pkl'):
30.
31.     table=0
32.
33.     try:
34.         with open(file,'rb') as f:
35.             table=pickle.load(f)
36.
37.     except IOError as e:
38.         print(f'Error loading file with catalog. Error is{e}')
```



```

39.
40.     except:
41.         print(f'Error reading file with catalog')
42.
43.     finally:
44.         if len(table) <= 2:
45.             print('No catalog loaded!')
46.         else:
47.             print(f'\tCatalog in {file} loaded succesfully')
48.
49.     return(table)
50.
51.
52.     class DiasCatalog(object):
53.
54.         # %%
55.         def __init__(self, *attributes, ra_0=0, dec_0=0, r_0=0.5, entire=False, galactic=False):
56.             """
57.             Bruno Femenia Castella:
58.
59.             in order to speed up calcularions it does not make sense to scan
60.             the whole catalog but only those clusters found within 1.5*r_0 [deg]
61.             around coordinates [ra_0, dec_0]
62.             """
63.             self.attributes = attributes
64.             Vizier.ROW_LIMIT = -1
65.
66.             if entire:
67.                 dias_catalog = Vizier.get_catalogs('B/ocl/clusters')
68.             else:
69.                 center_coord = SkyCoord(ra=ra_0, dec=dec_0, unit=(u.deg, u.deg), frame='icrs')
70.                 dias_catalog = Vizier.query_region(center_coord, radius=1.5*r_0*u.deg, catalog='B/ocl/clusters')
71.
72.
73.             if attributes == ():
74.                 self.table = dias_catalog['B/ocl/clusters']
75.             else:
76.                 self.table = dias_catalog['B/ocl/clusters'][attributes]
77.
78.
79.             if galactic:
80.                 nel = len(self.table)
81.                 Col_l = Column(name='l', unit=u.deg, dtype=np.float32,
82.                                description='Galactic longitude', length=nel)
83.                 Col_b = Column(name='b', unit=u.deg, dtype=np.float32,
84.                                description='Galactic latitude', length=nel)
85.                 for i,row in enumerate(self.table):
86.                     ra = Angle(row['RAJ2000'], unit=u.hourangle)
87.                     dec = Angle(row['DEJ2000'], unit=u.deg)
88.                     cl_radec = SkyCoord(ra, dec, frame='icrs')
89.                     # ra_hms, dec_deg = parse_radec(row['RAJ2000'], row['DEJ2000'])
90.                     # cl_radec=SkyCoord(ra_hms, dec_deg, frame='icrs')
91.                     Col_l[i] = cl_radec.galactic.l.deg

```

```

92.         Col_b[i] = cl_radec.galactic.b.deg
93.
94.         self.table.add_columns([Col_l, Col_b])
95.
96.
97.     # %%
98.     def get_ra_dec(self, cluster):
99.         """Method to get right ascension and declination from a cluster
100.
101.         Parameters
102.         -----
103.         cluster: Name of the cluster to get its position
104.         Returns
105.         -----
106.         SkyCoord object containing ra and dec in degrees
107.         """
108.         try:
109.             row = self.table[np.where(self.table['Cluster'] == cluster)]
110.             ra = Angle(row['RAJ2000'], unit=u.hourangle)
111.             dec = Angle(row['DEJ2000'], unit=u.deg)
112.             return([float(ra.deg), float(dec.deg)])
113.         except:
114.             print('Cluster does not exist')
115.
116.
117.     # %%
118.     def get_cluster_names(self):
119.         """Method to get the name of the clusters in the catalog
120.         Returns
121.         -----
122.         print out name of the clusters in the catalog
123.         """
124.         print(self.table['Cluster'])
125.
126.
127.     # %%
128.     def get_diam(self, cluster):
129.         """Method to get right ascension and declination from a cluster
130.
131.         Parameters
132.         -----
133.         cluster: Name of the cluster to get its diameter
134.         Returns
135.         -----
136.         SkyCoord object containing ra and dec in degrees
137.         """
138.         try:
139.             row = self.table[np.where(self.table['Cluster'] == cluster)]
140.             diam = float(row['Diam'])*u.arcmin
141.             return(diam.to('deg').value)
142.         except:
143.             print('Cluster does not exist')
144.
145.
146.     # %%

```

```

147.     def get_cluster_coordinates(self):
148.         ra = Angle(self.table['RAJ2000'], unit=u.hourangle)
149.         ra = ra.deg
150.         ra = Longitude(ra, unit=u.deg)
151.         dec = Latitude(self.table['DEJ2000'], unit=u.deg )
152.         return(SkyCoord(ra, dec))
153.
154.
155.     # %%
156.     def plot_all(self):
157.         coordinates = self.get_cluster_coordinates()
158.         ra = coordinates.ra
159.         dec = coordinates.dec
160.         fig, ax = plt.subplots()
161.         ax.scatter(ra, dec, s=0.5)
162.         #for i, txt in enumerate(np.array(self.table['Cluster'], dtype='S')):
163.         #    ax.annotate(txt, (ra[i].value, dec[i].value))
164.         plt.show()
165.
166.
167.     # %%
168.     def get_clusters(self, center, r):
169.         # print('\t\tDentro de DiasCatalog.get_clusters() method.')
170.
171.         # tA= time.time()
172.         clusters = []
173.         ra_center = Longitude(center[0], unit=u.deg)
174.         dec_center = Latitude(center[1], unit=u.deg)
175.         # print('\t\tDentro de DiasCatalog.get_clusters() method.')
176.         center = SkyCoord(ra_center, dec_center)
177.         # tB= time.time()
178.         # print(f'\t\t\t >> Seconds before internal loop: {tB-tA}')
179.
180.         # tC= time.time()
181.         # print(f"\t\t\t >> Starting internal loop with #clusters in Dias Catalog
: {self.table['Cluster'].shape[0]}")
182.         for i in range(self.table['Cluster'].shape[0]):
183.             tD= time.time()
184.             ra = Angle(self.table[i]['RAJ2000'], unit=u.hourangle)
185.             ra = ra.deg
186.             ra = Longitude(ra, unit=u.deg)
187.             dec = Latitude(self.table[i]['DEJ2000'], unit=u.deg)
188.             cluster = SkyCoord(ra, dec)
189.             diam = float(self.table[i]['Diam'])*u.arcmin
190.             if(np.isnan(diam)):
191.                 diam = 0.05
192.             else:
193.                 diam = diam.to('deg').value
194.             if(cluster.separation(center).value <= r):
195.                 clusters.append((self.table['Cluster'][i], cluster, diam))
196.
197.         #     if i==0:
198.         #         print(f"\t\t\t\t 1st iter in internal loop takes: {time.time()-
tD} seconds")
199.

```

```

200.         # print(f"\t\t\t >> Done with internal loop. Seconds to scan all clusters
: {time.time()-tC} ")
201.         # print(f"\t\t >> Done with DiasCatalog.get_clusters() in {time.time()-
tA} seconds")
202.
203.
204.
205.         return(clusters)
206.
207.
208.     # %%
209.     def get_rate(self, center, r, clusters):
210.         """
211.         Method to get a metric between 0 and 1 to measure the success of our
212.         algorithm.
213.
214.         Parameters
215.         -----
216.         center : List
217.             Center of the sky
218.         r : float
219.             Radius of the sky
220.
221.         cluster: List
222.             Center point of a cluster detected
223.
224.         Returns
225.         -----
226.         rate: float
227.             Metric between 0 and 1 that measures how good our algorithm worked
228.         """
229.         num_cum = int(0)
230.         matches = int(0)
231.         rate = 0.0
232.         ra = Longitude(center[0], unit=u.deg)
233.         dec = Latitude(center[1], unit=u.deg)
234.         center = SkyCoord(ra, dec)
235.         #     ra_cluster = Longitude(cluster[0], unit=u.deg)
236.         #     dec_cluster = Latitude(cluster[1], unit=u.deg)
237.         #     cluster = SkyCoord(ra_cluster, dec_cluster)
238.         for i in range(self.table['Cluster'].shape[0]):
239.             ra_dias = Angle(self.table[i]['RAJ2000'], unit=u.hourangle)
240.             ra_dias = ra_dias.deg
241.             ra_dias = Longitude(ra_dias, unit=u.deg)
242.             dec_dias = Latitude(self.table[i]['DEJ2000'], unit=u.deg)
243.             cluster_dias = SkyCoord(ra_dias, dec_dias)
244.             diam = float(self.table[i]['Diam'])*u.arcmin
245.             if(np.isnan(diam)):
246.                 diam = 0.05
247.             else:
248.                 diam = diam.to('deg').value
249.             if(cluster_dias.separation(center).value <= r):
250.                 num_cum += 1
251.             for cluster in clusters:
252.                 ra_cluster = Longitude(cluster[0], unit=u.deg)
253.                 dec_cluster = Latitude(cluster[1], unit=u.deg)

```

```

254.         cluster = SkyCoord(ra_cluster, dec_cluster)
255.         if(cluster_dias.separation(cluster).value <= diam/2.0):
256.             matches += 1
257.             break
258.     try:
259.         rate = matches/num_cum
260.         print('Number of cluster in the sky %5d' % num_cum)
261.         print('Number of matches: %5d' % matches)
262.     except:
263.         print('No star cluster detected in Dias catalog in the part of\n the sky specified')
264.     return(rate)
265.
266.
267. # %%
268. def get_match(self, center, r, clusters_detected):
269.     # print('\t\tDentro de DiasCatalog.get_match() method.')
270.
271.     # tA= time.time()
272.     existing_clusters = self.get_clusters(center, r)
273.     # tB= time.time()
274.     # print(f'\t\t\t >> Seconds to run .get_clusters(): {tB-tA}')
275.
276.     tot_matches = []
277.     for cluster in clusters_detected:
278.         ra_cluster_detected = Longitude(cluster[0], unit=u.deg)
279.         dec_cluster_detected = Latitude(cluster[1], unit=u.deg)
280.         cluster_detected = SkyCoord(ra_cluster_detected, dec_cluster_detected)
281.
282.         l_tmp = [match for match in existing_clusters if match[1].separation(cluster_detected).value <= match[2]/2.0]
283.         tot_matches = list(set(tot_matches + l_tmp))
284.
285.         # tC= time.time()
286.         # print(f'\t\t\t >> Seconds to run internal loop : {tC-tB}')
287.         # print(f'\t\t\t >> Seconds to run .get_match() =>{tC-tA}')
288.     return tot_matches

```

GaiaData.py

```

1.     #!/usr/bin/env python3
2.     # -*- coding: utf-8 -*-
3.     """
4.     Created on Sat Oct 6 10:59:51 2018
5.
6.     @author: rafa
7.
8.     This class fetches data from Gaia depending on certain paramters:
9.     """
10.
11.     from astroquery.gaia import Gaia
12.
13.     class GaiaData(object):
14.
15.     # %%

```

```

16.     def __init__(self, attributes, coordsys = 'ICRS'):
17.         """Constructo of an object representing an ADQL query
18.
19.         Parameters
20.         -----
21.         attributes: List of fields to query (projection)
22.         point: point in the sky to query
23.         extent: length of the box centered in the point
24.         """
25.         self.attributes = attributes
26.         self.coordsys = coordsys
27.
28.         # %%
29.         def astrometric_query_box(self, point, length, err_pos, g_lim):
30.             """Construct a query of a sky box centered in point specified when creati
ng this object
31.
32.             Parameters:
33.             -----
34.             point: Point of the vertex of the box
35.             extent: lenght of the arms of the bos
36.
37.             Returns
38.             -----
39.             ADQL Query
40.             data retrieve from the query
41.
42.             """
43.             query = "SELECT %s"%', '.join(self.attributes)
44.             query += " FROM gaiadr2.gaia_source"
45.             query += " WHERE CONTAINS(POINT('%s', ra, dec), BOX('%s', %s, %s, %s))=1"
%(self.coordsys,
46.                                     self.coordsys, ', '.join([str(n) for n in point]), str(len
gth), str(length))
47.             query += " AND parallax IS NOT NULL"
48.             query += " AND parallax >= 0.0"
49.             query += " AND ra IS NOT NULL"
50.             query += " AND dec IS NOT NULL"
51.             query += " AND pmra IS NOT NULL"
52.             query += " AND pmdec IS NOT NULL"
53.             query += " AND ABS(ra_error) < %s"%str(err_pos)
54.             query += " AND ABS(dec_error) < %s"%str(err_pos)
55.             query += " AND ABS(parallax_error) < %s"%str(err_pos)
56.             query += " AND ABS(phot_g_mean_mag) < %s"%str(g_lim)
57.             #print('\n\t'+query)
58.
59.             data=[[-999]]
60.             while data[0][0] == -999:
61.                 data = self.get_results(query)
62.             return(data)
63.
64.         # %%
65.         def astrometric_query_circle(self, point, radius, err_pos, g_lim):
66.             """Construct a query of a sky circle centered in point specified when cre
ating this object
67.

```

```

68.     Parameters:
69.     -----
70.     radius: Radius of the circle to query
71.
72.     Returns
73.     -----
74.     ADQL Query
75.     data retrieved from the query
76.     """
77.     query = "SELECT %s"%', '.join(self.attributes)
78.     query += " FROM gaiadr2.gaia_source"
79.     query += " WHERE CONTAINS(POINT('%s', ra, dec), CIRCLE('%s', %s, %s))=1"%
(self.coordsys,
80.                                     self.coordsys, ', '.join([str(n) for n in point]), str(radius))
81.     query += " AND parallax IS NOT NULL"
82.     query += " AND parallax >= 0.0"
83.     query += " AND ra IS NOT NULL"
84.     query += " AND dec IS NOT NULL"
85.     query += " AND pmra IS NOT NULL"
86.     query += " AND pmdec IS NOT NULL"
87.     query += " AND ABS(ra_error) < %s"%str(err_pos)
88.     query += " AND ABS(dec_error) < %s"%str(err_pos)
89.     query += " AND ABS(parallax_error) < %s"%str(err_pos)
90.     query += " AND ABS(phot_g_mean_mag) < %s"%str(g_lim)
91.     print(query)
92.     data = self.get_results(query)
93.     return(data)
94.
95.     # %%
96.     def get_results(self, query):
97.         """Runs a job in GAIA archive to retrieve data
98.
99.         Parameters.
100.         -----
101.         query: Query for GAIA database
102.
103.         Returns
104.         -----
105.         data: Data retrieved from ADQL query
106.         """
107.         data = [[-999]]
108.         try:
109.             job = Gaia.launch_job_async(query)
110.         except:
111.             print('Query has failed')
112.         else:
113.             data = job.get_results()
114.         return(data)

```


7.6.2. main_BF20.py

```

1.  #!/usr/bin/env python3
2.  # -*- coding: utf-8 -*-
3.  """
4.  Created on Sun Feb 10 11:23:54 2019
5.
6.  @author: rafa
7.
8.  Modified by: Bruno Femenía Castellá (bruno.femenia@gmail.com)
9.  """
10. import sys
11. import numpy as np
12. import pandas as pd
13. import GaiaData as gd
14. import DiasCatalog as dc
15. import datetime
16.
17. # %%      IMPORT Section      #####
18. #
19. # Third-party dependencies
20. from astropy import units as u
21. from astropy.coordinates import Angle, Longitude, Latitude, Distance
22. from astropy.coordinates import SkyCoord
23. from astropy.table import Table
24.
25.
26. # Set up matplotlib and use a nicer set of plot parameters
27. import matplotlib.pyplot as plt
28. from matplotlib import cm
29. from astropy.visualization import astropy_mpl_style
30. from mpl_toolkits.mplot3d import Axes3D
31. import matplotlib.gridspec as gridspec
32.
33.
34. #Sklearn algorithm
35. from sklearn.metrics.pairwise import pairwise_distances
36. from sklearn.cluster import DBSCAN
37. from sklearn.preprocessing import MinMaxScaler
38. from sklearn.metrics import silhouette_score
39.
40.
41. #To suppress warnings
42. import warnings
43. warnings.filterwarnings("ignore")
44.
45.
46. #To time code execution
47. import time
48.
49.
50. # %%      DATA EXTRACTION      #####
51. #
52. def extract_data(r, err_pos, g_lim, cluster=None, coordinates=None):
53.     #Define variables needed to make the query
54.     data = None

```

```

54.     dias_catalog = dc.DiasCatalog('Cluster', 'RAJ2000', 'DEJ2000', 'Class', 'Diam',
55.                                   'Dist', 'pmRA', 'pmDE', 'Nc', 'RV', 'o_RV',
56.                                   ra_0=coordinates[0], dec_0=coordinates[1], r_0=r)
57.     if (coordinates != None) or (cluster != None):
58.         if coordinates == None:
59.             point = dias_catalog.get_ra_dec(cluster)
60.         else:
61.             point = coordinates
62.             #Create the GAIA query
63.             attributes = ['source_id', 'ra', 'ra_error', 'dec', 'dec_error', 'parallax', 'parallax_error', 'pmra', 'pmdec', 'phot_g_mean_mag']
64.             gaia_query = gd.GaiaData(attributes)
65.             data = gaia_query.astrometric_query_circle(point, r, err_pos, g_lim)
66.             return(data, point)
67.
68.
69.     # %%      DATA PREPROCESSING      #####
70.     #
71.     def preprocessing(data, sample_factor = None):
72.         #Preprocessing
73.         data_metrics = data
74.         data_metrics['parallax'] = data_metrics['parallax'].to(u.parsec, equivalencies=u.parallax())
75.         #Adapt data metrics to a numpy array
76.         np_data_metrics = np.transpose(np.array([data_metrics['ra'], data_metrics['dec'], data_metrics['parallax'],
77.                                                  data_metrics['pmra'], data_metrics['pmdec']]))
78.         #Sample data
79.         if(sample_factor != None):
80.             np.random.seed(0)
81.             idx = np.random.choice(np_data_metrics.shape[0], size=int(sample_factor*np_data_metrics.shape[0]),
82.                                   replace=False)
83.             np_data_metrics = np_data_metrics[idx,:]
84.             #Change coordinates from Spherical to Cartesian coordinate system
85.             ra = Longitude(np_data_metrics[:,0], unit=u.deg)
86.             ra.wrap_angle = 180 * u.deg
87.             dec = Latitude(np_data_metrics[:,1], unit=u.deg)
88.             dist = Distance(np_data_metrics[:,2], unit=u.parsec)
89.             sphe_coordinates = SkyCoord(ra, dec, distance = dist, frame='icrs', representation_type='spherical')
90.             cart_coordinates = sphe_coordinates.cartesian
91.             #Adapt data to normalize it correctly
92.             data_sphe_adapted = np.transpose(np.array([sphe_coordinates.ra, sphe_coordinates.dec, sphe_coordinates.distance]))
93.             data_cart_adapted = np.transpose(np.array([cart_coordinates.x, cart_coordinates.y, cart_coordinates.z]))
94.             data_pm_adapted = np_data_metrics[:,3:5]
95.             data_all_adapted = np.append(data_cart_adapted, data_pm_adapted, axis=1)
96.             return(data_sphe_adapted, data_cart_adapted, data_all_adapted)
97.
98.     # %%
99.     def get_distances_for_ML(X, Y):
100.         distance = 0
101.         ra1 = X[0]*u.deg
102.         ra2 = Y[0]*u.deg

```

```

102.     dec1 = X[1]*u.deg
103.     dec2 = Y[1]*u.deg
104.     if(len(X) == 3):
105.         dist1 = X[2]*u.parsec
106.         dist2 = Y[2]*u.parsec
107.         point1 = SkyCoord(ra1, dec1, dist1)
108.         point2 = SkyCoord(ra2, dec2, dist2)
109.         distance = point1.separation_3d(point2)
110.     else:
111.         point1 = SkyCoord(ra1, dec1)
112.         point2 = SkyCoord(ra2, dec2)
113.         distance = point1.separation(point2)
114.     return(distance.value)
115.
116.
117. def get_distance_matrix(data, scale=False, metric='euclidean'):
118.     if scale:
119.         scaler = MinMaxScaler()
120.         data_scaled = scaler.fit_transform(data)
121.     else:
122.         data_scaled = data
123.     if metric != 'euclidean':
124.         dist_matrix = pairwise_distances(data_scaled, metric=get_distances_for_ML
, n_jobs=-1)
125.     else:
126.         dist_matrix = pairwise_distances(data_scaled, metric='euclidean', n_jobs=
-1)
127.     return(dist_matrix)
128.
129.
130. # %%      DATA EVALUATON      #####
#
131.
132. def generate_ip_dias(l_max=5, r_max=1.5, n_min=3):
133.     """
134.     This routine generates the selection of ip files over the Galactic plane
135.     containing at least 3 clusters in Dias catalog and not beyond r_max degs of
136.     size. Each of these sky patches witll be analyzed to derived eps=eps(l,b)
137.     and min_pts=min_pets(l,b) which will guide the search for clusters in a
138.     subsequent exploration of Gaia data.
139.
140.     Based on these files we evaluate the (eps,Nmin) to be used over different re
gions of whole
141.     Galactic plane to find new candidates with DBSCAN.
142.
143.     Returns
144.     -----
145.     None.
146.
147.     """
148.
149.     full_dc=dc.DiasCatalog('Cluster','RAJ2000','DEJ2000','Class','Diam',entire=T
rue,galactic=True)
150.
151.     gal_disk=full_dc.table[ (abs(full_dc.table['b']) < 20) & (full_dc.table['l']
< l_max) ]

```

```

152.
153.     patches=[]
154.
155.     for i, this_cluster in enumerate(gal_disk):    #Iterating over ALL clusters
156.
157.         #in selected region |b|< 20 and l < l_max
158.         c1      = SkyCoord(this_cluster['l']*u.deg,
159.                             this_cluster['b']*u.deg,
160.                             frame='galactic')
161.         clusters = gal_disk[ (abs(gal_disk['b']-this_cluster['b']) <= r_max)]
162.
163.         if len(clusters) < n_min:                #Requesting a minimum number of clusters in this patch!!
164.             continue
165.
166.         candidates=[{'name':this_cluster['Cluster'], 'ang_sep':0.,
167.                      'l':this_cluster['l'], 'b':this_cluster['b']}]
168.
169.         for cluster in clusters:
170.             c2      = SkyCoord(cluster['l']*u.deg, cluster['b']*u.deg, frame='galactic')
171.             ang_sep = c1.separation(c2).deg
172.
173.             if (ang_sep <= r_max) and (ang_sep > 0):
174.                 new = dict({'name':cluster['Cluster'], 'ang_sep':ang_sep,
175.                             'l':cluster['l'], 'b':cluster['b']})
176.                 candidates.append(new)
177.
178.             if len(candidates) >= n_min:
179.                 df = pd.DataFrame.from_records(candidates)
180.                 df.sort_values('ang_sep', inplace=True)
181.                 patch = df[0:n_min].mean()                #Use only n_min clusters at a time!!
182.                 patches.append(patch)
183.
184.
185.     patches_df = pd.DataFrame.from_records(patches)
186.     patches_df.drop_duplicates(subset=['l','b'], inplace=True) #Removing duplicates
187.     patches_df.reset_index(drop=True, inplace=True)
188.
189.     #Adding rest of fields as defined by Rafa in his Thesis. Then save csv.
190.     #-----
191.     op_dict={'ra':[0.],
192.              'dec':[0.],
193.              'l':[0.],
194.              'b':[0.],
195.              'r':[0.],
196.              'err_pos':[100],
197.              'g_lim':[18.5],
198.              'norm':[None],
199.              'sample':[1.],
200.              'dim3':[None],
201.              'distance':['euclidean'],
202.              'eps_min':[0.008],

```

```

203.         'eps_max':[0.03],
204.         'eps_num':[100],
205.         'min_pts_min':[10],
206.         'min_pts_max':[100],
207.         'min_pts_num':[91]}
208.     op_df=pd.DataFrame(data= op_dict)
209.
210.     for index, row in patches_df.iterrows():
211.         fn = 'ip_file_DC_'+str(index).zfill(5)+'.csv'
212.         c1 = SkyCoord(row['l']*u.deg, row['b']*u.deg, frame='galactic')
213.
214.         op_df['ra']= c1.icrs.ra.deg
215.         op_df['dec']= c1.icrs.dec.deg
216.         op_df['l']= row['l']
217.         op_df['b']= row['b']
218.         op_df['r']= row['ang_sep']
219.
220.         op_df.to_csv(fn, index=False, header=True)
221.
222.     return patches_df
223.
224.
225.     # %%
226.     def DBSCAN_result(param_scores, dist_matrix, data, center, radius):
227.         #dias_catalog = dc.DiasCatalog()
228.         dias_catalog = dc.DiasCatalog('Cluster', 'RAJ2000', 'DEJ2000', 'Class', 'Diam',
229.
230.                                     'Dist', 'pmRA', 'pmDE', 'Nc', 'RV', 'o_RV',
231.                                     ra_0=center[0], dec_0=center[1], r_0=radius)
232.         cluster_centers = []
233.         sorted_param_scores = param_scores.sort_values(by=['local_score'], ascending
234.                                                         =False)
235.         opt_epsilon = sorted_param_scores.iloc[0,0]
236.         opt_min_pts = sorted_param_scores.iloc[0,1]
237.         db = DBSCAN(eps=opt_epsilon, min_samples=opt_min_pts, metric='precomputed',
238.                     n_jobs=-1).fit(dist_matrix)
239.         labels = db.labels_
240.         for i in range(len(set(labels))-1):
241.             cluster = data[np.where(labels == i)]
242.             cluster_center = cluster.mean(axis=0)
243.             cluster_centers.append(cluster_center)
244.         matches = dias_catalog.get_match(center, radius, cluster_centers)
245.         for m in matches:
246.             print('Cluster found: %s'%(m[0]))
247.         plot_clusters(data, labels)
248.         plot_score(param_scores)
249.
250.         # %%
251.         def DBSCAN_eval(data, center, r, sample_factor, ftype, dim3, eps_range, min_sa
252.                         mples_range, scale=False, metric = 'euclidean', pm=False):
253.             data_sphe, data_cart, data_all = preprocessing(data, sample_factor)
254.             data_search = data_sphe[:, :2]
255.             if ftype == 'cart':
256.                 if dim3:
257.                     data = data_cart

```

```

255.         else:
256.             data = data_cart[:, :2]
257.     elif ftype == 'sphe':
258.         if dim3:
259.             data = data_sphe
260.         else:
261.             data = data_sphe[:, :2]
262.     elif ftype == 'pm':
263.         data = data_all
264.     else:
265.         print('Specify tpye of dataset: cart, sphe, pm')
266.         sys.exit()
267.
268.     dist_matrix = get_distance_matrix(data, scale, metric)
269.
270.     if pm:
271.         data = data[:, :3]
272.
273.     #dias_catalog = dc.DiasCatalog()
274.     dias_catalog = dc.DiasCatalog('Cluster', 'RAJ2000', 'DEJ2000', 'Class', 'Diam',
275.                                   'Dist', 'pmRA', 'pmDE', 'Nc', 'RV', 'o_RV',
276.                                   ra_0=center[0], dec_0=center[1], r_0=r)
277.     num_cum = len(dias_catalog.get_clusters(center, r))
278.     tmp_sscores = []
279.     matches = []
280.     sscores = pd.DataFrame(columns=['epsilon', 'minpts', 'local_score'])
281.     cluster_centers = []
282.
283.     # idx_eps=0
284.
285.     for eps in eps_range:
286.         tA= time.time()
287.         # idx_pts=0
288.         print(f'\nCase eps: {eps}\n'+9*'-')
289.
290.         for min_samples in min_samples_range:
291.             # tB=time.time()
292.             db = DBSCAN(eps=eps, min_samples=min_samples, metric="precomputed", n_
obs=-1).fit(dist_matrix)
293.             # tC=time.time()
294.             # print(f'\n\tSeconds to execute DBSCAN:{tC-tB}')
295.
296.             labels = db.labels_
297.             for i in range(len(set(labels))-1):
298.                 cluster = data_search[np.where(labels == i)]
299.                 cluster_center = cluster.mean(axis=0)
300.                 cluster_centers.append(cluster_center)
301.
302.             # tD=time.time()
303.             matches = dias_catalog.get_match(center, r, cluster_centers)
304.             # tE=time.time()
305.             # print(f'\n\tSeconds to execute dias_catalog.get_match:{tE-tD}')
306.
307.             tmp_sscores.append(eps)

```

```

308.         tmp_sscores.append(min_samples)
309.         if(len(matches) > 0 and len(set(labels)) > 1):
310.             tmp_sscores.append(len(matches)/(num_cum + (len(cluster_centers)-
len(matches))))
311.         else:
312.             tmp_sscores.append(0.0)
313.             sscores.loc[len(sscores)] = tmp_sscores
314.             tmp_sscores = []
315.             cluster_centers = []
316.
317.             # print(f'\tSeconds to execute (eps,Npts)=({eps},{min_samples}): {time.
time()-tB}')
318.             # idx_pts +=1
319.             # if idx_pts==2:
320.                 # break
321.
322.             print(f'\t>> Seconds to execute eps={eps}: {time.time()-tA}')
323.             # idx_eps += 1
324.             # if (idx_eps == 2):
325.                 # break
326.
327.         return(sscores, dist_matrix, data_search)
328.
329.
330.
331.     # %% DATA PLOTTING #####
#
332.     def plot_clusters(X, labels):
333.         # Black removed and is used for noise instead.
334.         size = 6.0
335.         f = plt.figure(figsize=(20,20))
336.         unique_labels = set(labels)
337.         colors = [plt.cm.Spectral(each)
338.                   for each in np.linspace(0, 0.5, len(unique_labels))]
339.         for k, col in zip(unique_labels, colors):
340.             if k == -1:
341.                 # Black used for noise.
342.                 col = [0, 0, 0, 1]
343.                 size = 3.5
344.
345.             class_member_mask = (labels == k)
346.
347.             xy = X[class_member_mask]
348.             plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
349.                     markeredgecolor='k', markersize=size)
350.             size=6.0
351.
352.             plt.title('Estimated number of clusters: %5d' % int(len(set(labels))-1))
353.             plt.xlabel("Right Ascension (deg)")
354.             plt.ylabel("Declination (deg)")
355.
356.             dummy = datetime.datetime.now()
357.             op_name= f'plt_clstr_exec_{dummy.year}-{dummy.month}-
{dummy.day} {dummy.hour}-{dummy.minute}-{dummy.second}.png'
358.             f.savefig(op_name)
359.

```



```

360.
361. # %%
362. def plot_score(param_scores):
363.     np_param_scores = param_scores.values
364.     eps = np.sort(np.array(list(set(param_scores['epsilon']))))
365.     Nmin = np.sort(np.array(list(set(param_scores['minpts']))))
366.     Z = np.empty((len(Nmin), len(eps)))
367.     fig, ax = plt.subplots(constrained_layout = True)
368.     X, Y = np.meshgrid(eps, Nmin)
369.     for i, n in enumerate(Nmin):
370.         for j, e in enumerate(eps):
371.             Z[i,j] = np_param_scores[np.where((param_scores['epsilon'] == e) & (param_scores['minpts'] == n)),2]
372.             extend = "neither"
373.             cmap = plt.cm.get_cmap('hot')
374.             CS = ax.contourf(X,Y,Z, cmap=cmap, extend=extend)
375.             fig.colorbar(CS)
376.             ax.set_xlabel('Epsilon')
377.             ax.set_ylabel('Nmin')
378.             ax.set_title('DBSCAN matching M')
379.
380.             dummy = datetime.datetime.now()
381.             op_name= f'plt_score_exec_{dummy.year}-{dummy.month}-{dummy.day}.png'
382.             fig.savefig(op_name)
383.             #fig.savefig('plot_score_execution_%.png'%(datetime.date.today()))
384.
385. # %%
386. def check_errors(f):
387.     def check(*args, **kwargs):
388.         try:
389.             f(*args, **kwargs)
390.         except:
391.             print("An error has occurred")
392.     return(check)
393.
394.
395. # %%
396. @check_errors
397. def plot_bar(*args):
398.     if args[1] == 'eps':
399.         df = args[0][['epsilon', 'local_score']]
400.         df['bucket'] = pd.cut(df.epsilon, args[2])
401.         title = 'Máximo M en función de epsilon'
402.         x_label = 'epsilon'
403.     else:
404.         df = args[0][['minpts', 'local_score']]
405.         df['bucket'] = pd.cut(df.minpts, args[2])
406.         title = 'Máximo M en función de Nmin'
407.         x_label = 'Nmin'
408.     newdf = df[['bucket', 'local_score']].groupby('bucket').max()
409.     ax = newdf.plot(kind='bar', title=title, colormap = 'jet', fontsize=7, legend=False)
410.     ax.set_xlabel(x_label)
411.     ax.set_ylabel("Max(M)")
412.     ax.grid()
413.

```

```

414.
415.  ###      MAIN      #####
416.
417.  def process_line(ftype, params):
418.      ra = params['ra']
419.      dec = params['dec']
420.      radius = params['r']
421.      err_pos = params['err_pos']
422.      g_lim = params['g_lim']
423.      sample_factor = params['sample']
424.      scale = params['norm'] == 'X'
425.      metric = params['distance']
426.      dim3 = params['dim3'] == 'X'
427.      eps_min = params['eps_min']
428.      eps_max = params['eps_max']
429.      eps_num = params['eps_num']
430.      min_pts_min = params['min_pts_min']
431.      min_pts_max = params['min_pts_max']
432.      min_pts_num = params['min_pts_num']
433.      eps_range = np.linspace(eps_min, eps_max, eps_num)
434.      min_pts_range = np.linspace(min_pts_min, min_pts_max, min_pts_num)
435.
436.      center = [ra, dec]
437.      data, center = extract_data(radius, err_pos, g_lim, coordinates = center)
438.      param_scores, dist_matrix, data_search = DBSCAN_eval(data, center, radius, s
439.      ample_factor, ftype, dim3, eps_range, min_pts_range, scale, metric)
440.
441.      return(param_scores, dist_matrix, data_search, center, radius)
442.
443.  # %%
444.  def main():
445.      if len(sys.argv) == 3:
446.          file = sys.argv[1]
447.          ftype = sys.argv[2]
448.          try:
449.              params = pd.read_csv(file)
450.          except:
451.              print('The file does not exist')
452.              sys.exit(0)
453.          for i in range(len(params)):
454.              param_scores, dist_matrix, data_search, center, radius = process_line(f
455.              type, params.iloc[i,:])
456.              param_scores.to_csv('execution%s_%s.csv'%(str(i), datetime.date.today()
457.              ))
458.              DBSCAN_result(param_scores, dist_matrix, data_search, center, radius)
459.
460.  # %%
461.  # This is just to launch the mainn code from a session with Spyder IDE
462.
463.  def main_spyder(file='ip_file_0001.csv', ftype='sphe'):
464.      file=file.split(sep='\n')[0]
465.      print('\n\nStarting main_spyder!!\n'+22*('='+'\n\n'))
466.      try:

```

```

467.     print(f'\tReading file {file}\n')
468.     params = pd.read_csv(file)
469. except:
470.     print('The file does not exist')
471.     return
472.
473.
474.     for i in range(len(params)):
475.         t0 = time.time()
476.         print('\nStarting Iter# {0:1.0f}'.format(i+1))
477.         print(16*'=')
478.         print("  Sending query to GAIA\n ", 21*'-')
479.
480.         param_scores, dist_matrix, data_search, center, radius = process_line(fty
pe, params.iloc[i,:])
481.         t1= time.time()
482.         print("\n\n      Execution of process_line      --- %s seconds ---
" % (t1 - t0))
483.
484.         param_scores.to_csv('execution%s_%s.csv'%(str(i), datetime.date.today()))
485.
486.         t2= time.time()
487.         print("      Saving results into csv      --- %s seconds ---
" % (t2- t1))
488.         DBSCAN_result(param_scores, dist_matrix, data_search, center, radius)
489.         t3= time.time()
490.         print("      Execution of DBSCAN_result      --- %s seconds ---" % (t3-t2))
491.
492.         print("  >> Execution of this iterarion      --- %s seconds ---
" % (t3- t2))
493.
494.
495.     #main_spyder()
496.
497.
498.     # %%
499.     if __name__ == "__main__":
500.         main()

```