Universidad Zaragoza

Trabajo Fin de Grado

GeoCluster: Librería de algoritmos de agrupamiento para objetos geoespaciales

GeoCluster: A clustering library for geospatial objects

Autor

Javier Beltrán Jorba

Director

Francisco Javier López Pellicer

Escuela de Ingeniería y Arquitectura

2016

GeoCluster: Librería de algoritmos de agrupamiento para objetos geoespaciales

RESUMEN

Este trabajo consiste en el desarrollo de una librería que, a partir de una fuente de datos geoespaciales, sea capaz de generar conjuntos utilizando algoritmos de agrupamiento. En concreto, se ha implementado el algoritmo DBSCAN y varias versiones de K-Means.

Como la información geoespacial es compleja y puede tener atributos de todo tipo, la librería incluye diversas distancias que se pueden utilizar en el proceso de agrupamiento. Se ha comprobado que la distancia de Hausdorff es la mejor opción para medir la lejanía entre las geometrías de los objetos geoespaciales, ya que éstas suelen ser polígonos.

Para poder aprovechar al máximo la complejidad de los datos geoespaciales, se ha construido una distancia combinada que tiene en cuenta diversos criterios al formar los conjuntos. Esto permite agrupar ciudades en base a su localización y a su número de habitantes, al mismo tiempo.

Una vez ejecutado el algoritmo, el usuario debería interpretar los resultados para entender qué significan los conjuntos generados. Para ayudarle en esta tarea, la librería anota los conjuntos. El proceso consiste en analizar la información textual de cada elemento, utilizando un nomenclátor, y extraer los términos más repetidos. Así las anotaciones consistirán en información que es común al conjunto.

Como la tarea de anotación está sujeta a fallos (por ejemplo, ambigüedades en la información de los objetos), se ha desarrollado una técnica de validación de las anotaciones generadas. Consiste en recurrir a una base de datos de unidades administrativas globales, y utilizarla para comparar la geometría de cada elemento con la que corresponde a sus anotaciones. Una anotación se considerará válida si ambas geometrías son aproximadamente iguales.

La librería construida gestiona la lectura de datos geoespaciales desde ficheros ESRI Shapefile y desde bases de datos MySQL. También permite almacenar las agrupaciones generadas en ficheros ESRI Shapefile y en bases de datos MySQL.

Por último se han realizado una serie de experimentos con conjuntos de datos reales y sintéticos. Éstos se han utilizado para resolver problemas sencillos de carácter geográfico y para extraer conclusiones sobre los algoritmos y distancias desarrollados.

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Javier Beltrán Jorba,

con nº de DNI 76917763-J en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) GeoCluster: Librería de algoritmos de agrupamiento para objetos geoespaciales,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 12 de junio de 2016

Fdo: Javier Beltrán Jorba

Tabla de contenido

1. Introducción

En primer lugar se va a presentar la librería GeoCluster, indicando los problemas que busca resolver y el contexto en el que se ha realizado.

1.1 Contexto y objetivos

En la actualidad las empresas e instituciones manejan una gran cantidad de datos, que en muchos casos contienen información espacial con la que hay que tratar. Esta información es relevante porque nos ayuda a entender mejor nuestro mundo.

Trabajar con datos geoespaciales tiene una serie de dificultades que los algoritmos de agrupamiento tradicionales no tienen en cuenta. En particular, la información geográfica es compleja: está definida por una localización en unas coordenadas geográficas y por una serie de propiedades textuales, numéricas o de cualquier tipo.

Este trabajo pretende ofrecer un método para agrupar datos geoespaciales teniendo en cuenta su complejidad. No es el primer trabajo que busca una solución a este problema, pero sí que intenta ofrecer un enfoque diferente.

En la actualidad existen multitud de herramientas que permiten ejecutar algoritmos de agrupamiento, como la librería ELKI[1]. Sin embargo, los productos más populares que afrontan nuestro problema tan solo resuelven una simplificación de este. Es habitual encontrar librerías que asumen que la localización de un dato geoespacial es un punto o, en el caso de que no lo sea, utilizan su centroide. Es lo que hacen, por ejemplo, los algoritmos de Apache Commons Math[2]. En su lugar, este trabajo asume que los datos pueden tener geometrías complejas (en general, polígonos), y permite ejecutar algoritmos de agrupamiento con ellas.

Además de su localización, los datos geográficos suelen incluir información, en forma de atributos, sobre la entidad que representan. La posibilidad de utilizar esta información como criterio de agrupamiento, así como poder combinar criterios dispares (por ejemplo, la localización de una ciudad y su número de habitantes) son parte de los objetivos de este trabajo.

La generación de agrupamientos geográficos requiere un análisis manual posterior para determinar a qué elemento del mundo real se corresponde cada grupo. Otro de los objetivos del trabajo es automatizar esta tarea: a partir de la información textual de los datos geográficos, utilizar técnicas de anotación de objetos para obtener información representativa de los agrupamientos.

---

[1] *Environment for Developing KDD-Applications Supported by Index-Structures* http://elki.dbs.ifi.lmu.de/

[2] http://commons.apache.org/proper/commons-math/

Este trabajo se enmarca en el grupo de investigación en Sistemas de Información Avanzados (IAAA) de la Universidad de Zaragoza, dedicado a los sistemas de información geoespacial y servicios web.

1.2 La librería GeoCluster

El resultado de este trabajo es la creación de una librería de algoritmos de agrupamiento escrita en Java denominada GeoCluster. Se ha construido sobre la librería GeoTools, que proporciona herramientas para trabajar con Sistemas de Información Geográficos en Java. A partir de una fuente de datos (por ejemplo, en formato ESRI Shapefile[3]) la librería realiza un algoritmo de agrupamiento y anota los conjuntos obtenidos, y produce sus resultados en el mismo formato de entrada.

1.3 Evaluación

La tarea de evaluar la corrección de los resultados obtenidos por un algoritmo de agrupamiento no es sencilla. La realidad es que no existe una solución correcta a este problema, sino que depende de la interpretación que se le dé por parte del usuario de esta librería. Por ello, la mejor forma de evaluar este trabajo es mediante la visualización de los resultados obtenidos al ejecutar los algoritmos sobre diversos conjuntos de datos.

Esta tarea de visualización se ha desarrollado utilizando la aplicación web Jupyter Notebook, que permite ejecutar scripts (denominados *notebooks*) escritos en el lenguaje Python 3.3. La librería *matplotlib* ha facilitado la tarea de visualización de los datos.

1.4 Organización de la memoria

En esta memoria se explican las técnicas utilizadas para desarrollar la herramienta GeoCluster, así como la investigación previa y las conclusiones obtenidas. En el capítulo 2 se analizan los algoritmos de agrupamiento más populares y se determinan sus ventajas e inconvenientes, haciendo énfasis en las relacionadas con la información geoespacial. El capítulo 3 trata el estudio de las diferentes distancias a utilizar por los algoritmos de agrupamiento, y la elección de las más apropiadas para este trabajo. A continuación, en el capítulo 4 se analiza la necesidad de anotar los conjuntos obtenidos para facilitar la tarea de interpretación de resultados, y se explican las técnicas utilizadas para ello y las fuentes de datos a las que se ha recurrido. El capítulo 5 de la memoria se dedica a explicar la arquitectura de la librería, y en el capítulo 6 se exponen ejemplos de uso de GeoCluster y los resultados más destacables que se han obtenido, repasando las técnicas utilizadas e indicando si han funcionado. La memoria termina con un resumen de la gestión del proyecto en el capítulo 7, y con las conclusiones expuestas en el capítulo 8.

---

[3] Formato de almacenamiento de información geográfica que guarda la localización de los datos geográficos y sus atributos. https://doc.arcgis.com/en/arcgis-online/reference/shapefiles.htm

2. Algoritmos de agrupamiento

Un algoritmo de agrupamiento realiza grupos a partir de unos datos de entrada, en base a un determinado criterio. En general ese criterio es una distancia, de modo que los elementos más cercanos entre sí quedan agrupados.

Un ejemplo básico de distancia es la distancia euclídea, según la cual los elementos se agruparían en base a su cercanía espacial. En la Figura 1 se muestra un ejemplo de esta distancia en el algoritmo K-Means. Sin embargo, en el mundo de los datos geoespaciales la información es compleja y tiene atributos. Una distancia igualmente válida sería aquella que compare la población entre ciudades.

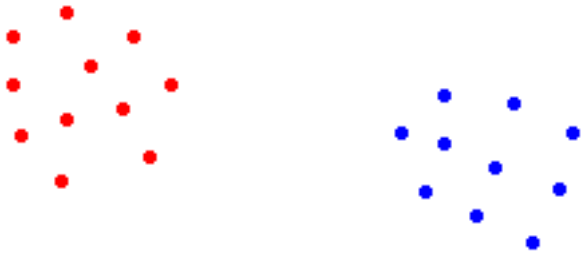


Figura 1: dos conjuntos resultantes de agrupar utilizando la distancia euclídea.

Se puede ver que la definición de una distancia y la elección de la más adecuada para cada problema es un aspecto importante, y será explicado en profundidad más adelante. De momento la distancia elegida será tratada como un parámetro del algoritmo.

A continuación se van a explicar dos tipos de algoritmos de agrupamiento, y se va a analizar su utilidad en el trabajo con información geográfica.

2.1 Algoritmos de la familia K-Means

Uno de los algoritmos de agrupamiento más populares es K-Means. Es un método que consiste en generar k grupos de modo que cada dato pertenece al grupo cuya media es más cercana.

Este problema es NP-duro [1] y por tanto intratable en términos de computación, pero existen técnicas iterativas que van refinando la solución obtenida paso a paso. En general, el algoritmo es el siguiente:

$\boldsymbol{algoritmo}\ kmeans(e_1..e_n, k)$:
$\boldsymbol{variables}\ c_1..c_n$
$\boldsymbol{variables}\ \mu_1..\mu_k$
$\boldsymbol{for}\ i \leftarrow 1\ \boldsymbol{hasta}\ k\ \boldsymbol{hacer}$
    $\mu_i := asignarInicial(e_1..e_n)$ (elige un centroide de entre e_1 y e_n)
$\boldsymbol{repetir}$

$$\textbf{for } i \leftarrow 1 \textbf{ hasta } n \textbf{ hacer}$$
$$c_i := \mu_j \; tal \; que \; mín \left(dist(e_i, \mu_j) \right)$$
$$\textbf{for } i \leftarrow 1 \textbf{ hasta } k \textbf{ hacer}$$
$$m := \#e_j \; tal \; que \; c_j = \mu_i$$
$$\mu_i := \frac{\sum_{c_j = \mu_i} e_j}{m}$$
$$\textbf{hasta que } \mu_1 .. \mu_k \; converjan$$
$$\textbf{devolver } c_1 .. c_n$$

El criterio de convergencia es variable, pero en general se busca que los centroides no cambien de una iteración a otra. Esta es la estrategia que se ha utilizado en GeoCluster.

2.1.1 Elección de centroides iniciales

Diferentes implementaciones de K-Means utilizan técnicas distintas para seleccionar los centroides iniciales del algoritmo. Se trata de un aspecto importante desde el punto de vista de la eficiencia, ya que una elección adecuada requerirá menos pasos para que el algoritmo converja a una solución.

GeoCluster implementa dos versiones de K-Means que difieren en la selección inicial de los centroides:

1. K-Means básico: es la implementación más simple del algoritmo. Se eligen como centroides *k* elementos de la colección, de forma aleatoria. Si en una ejecución concreta los centroides elegidos son muy cercanos entre sí, al algoritmo le puede costar más converger o incluso puede no dar un buen resultado. Sirve de ejemplo la Figura 2.

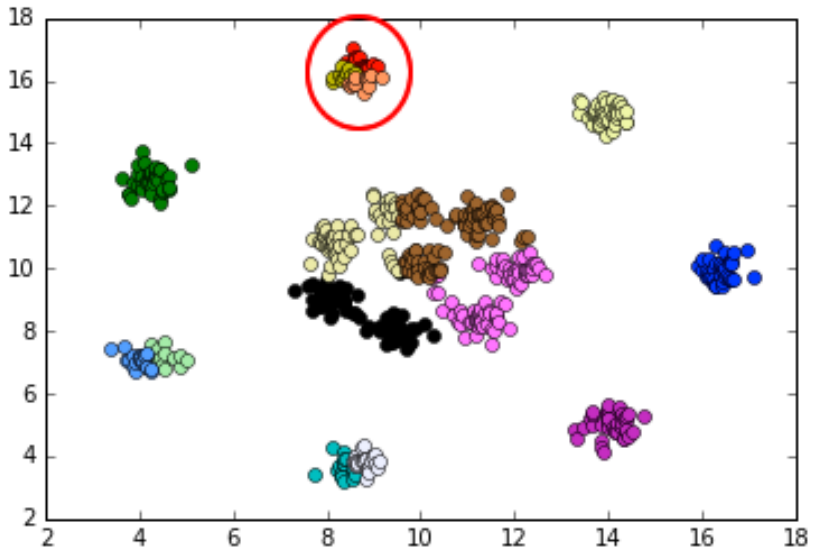


Figura 2: el conjunto de puntos superior aparece dividido en tres grupos, debido a que tres centroides se han inicializado en esa parte del espacio.

2. K-Means++ [2] intenta evitar que los centroides elegidos sean muy cercanos entre sí. Para ello, empieza eligiendo el primero como un elemento cualquiera de manera equiprobable. A partir de ahí la probabilidad de que un nuevo elemento sea seleccionado como centroide es directamente proporcional a su distancia a los demás.

Esta selección de centroides iniciales implica un coste temporal añadido en K-Means++. Sin embargo, su convergencia es más rápida que la de K-Means, especialmente para grandes volúmenes de datos o para valores grandes de *k*. Esta es la causa de que, en general, se puede considerar que K-Means++ es más eficiente. [2]

2.2 Algoritmo DBSCAN

DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*) es un algoritmo de agrupamiento basado en el concepto de densidad: los conjuntos se generarán en las áreas del espacio más densamente pobladas [3]. Para ello, DBSCAN busca los vecinos de cada elemento, y genera grupos en aquellos lugares del espacio donde existen más vecinos.

El algoritmo requiere dos parámetros:

- Un valor denominado ε (*eps*) que indica la distancia máxima en la que se buscan los vecinos de un elemento.
- Un número mínimo de puntos vecinos entre sí (*minPts*) para que estos se agrupen.

Estos parámetros permiten ser flexible con el concepto de densidad que maneja el algoritmo, pero dificultan su uso porque deben ajustarse. A continuación se describe en pseudocódigo el algoritmo. En él se maneja el estado de cada elemento como *No visitado, Visitado o Ruido* (si no pertenece a ningún grupo):

$\underline{\textbf{algoritmo}\ DBSCAN(e_1..e_n, eps, minPts):}$
$\textbf{variables}\ C, clusters, neighbors$
$\textbf{for}\ i \leftarrow 1\ \textbf{hasta}\ n\ \textbf{hacer}:$
    $\textbf{if}\ e_i\ visitado:$
        $continue$
    $\textbf{else}:$
        $neighbors = getNeighbors(e_i, e_1..e_n)$
        $\textbf{if}\ \#neighbors \geq minPts:$
            $(añade\ los\ vecinos\ al\ cluster\ y\ los\ marca\ como\ visitados)$
            $C = expandCluster(e_i, neighbors, \varepsilon)$
            $\textbf{añadir}\ C\ \textbf{a}\ clusters$
        $\textbf{else}:$
            $\textbf{marcar}\ e_i\ \textbf{como}\ RUIDO$
$\textbf{devolver}\ clusters$

2.3 Comparación desde el punto de vista geoespacial

GeoCluster implementa DBSCAN y las dos versiones de K-Means. Si bien no hay una regla de oro para decidir qué algoritmo va a funcionar mejor, conviene conocer sus principales diferencias.

Al utilizar K-Means el usuario debe indicar el número de grupos que desea que se formen. Esto es un inconveniente en contextos en los que esta medida sea difícil de estimar. DBSCAN, por su parte, requiere indicar los parámetros ε y *minPts* que definen la densidad de los grupos.

En el mundo de la información geográfica es posible encontrar casos de uso adecuados para ambos algoritmos. Sin embargo DBSCAN tiene una ventaja fundamental: puede encontrar grupos no linealmente separables [3]. Esto es muy útil debido a los dispares criterios según los cuales se puede llegar a agrupar información compleja. La Figura 3 compara el comportamiento de ambos algoritmos.

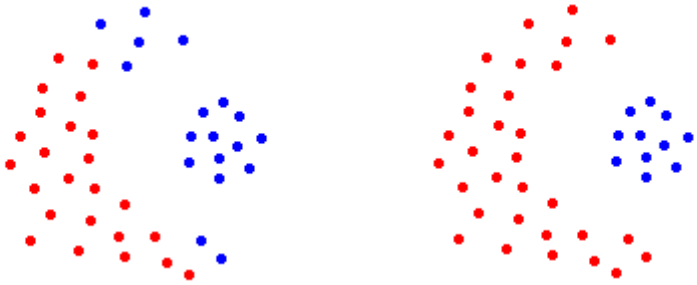


Figura 3: a la izquierda se ve que K-Means no puede separar correctamente los dos grupos, pues estos no son linealmente separables. A la derecha se ve que DBSCAN sí puede hacerlo.

Además, DBSCAN es robusto a la existencia de datos espurios, es decir, aquellos que no pertenecen a ningún grupo. Mientras que K-Means los asociaría al grupo más cercano, DBSCAN considera que si un elemento no es vecino de ninguno, entonces no debe pertenecer a ningún grupo.

Es fácil imaginar ejemplos en los que la robustez frente a espurios es útil en el contexto geoespacial. Aquí se propone el problema de detectar áreas metropolitanas a partir de un conjunto de poblaciones, teniendo en cuenta su localización y otros atributos.

Una parte de ese conjunto serán poblaciones pequeñas que no forman parte del área metropolitana de ninguna ciudad. Aun así, K-Means las agruparía con aquella más cercana, mientras que DBSCAN las detectaría como datos espurios, ofreciendo unas agrupaciones más fieles a la realidad que se busca representar.

3. Elección de distancias

Los algoritmos de agrupamiento indicados en el capítulo 2 de este documento tienen algo en común: manejan el concepto de distancia. El algoritmo busca agrupar en base a la cercanía entre los elementos. Por lo tanto, cabe responder a la pregunta ¿qué distancia utilizar en un algoritmo de agrupamiento enfocado a datos geoespaciales? La respuesta más simple es utilizar la distancia euclídea entre las coordenadas de los datos geoespaciales, con el objetivo de agrupar en base a su localización. Pero esta aproximación ignora el hecho de que la información geoespacial es compleja: puede tener múltiples atributos de tipo textual, numérico, etc. Si los datos a tratar son poblaciones, una distancia perfectamente válida sería la diferencia en número de habitantes.

En este capítulo se van a estudiar las diferentes distancias implementadas en GeoCluster, explicando su motivación desde el punto de vista geoespacial; y se va a proponer una técnica para utilizar múltiples distancias al trabajar con información compleja.

3.1 Distancia euclídea

La distancia euclídea se utiliza para calcular distancias entre puntos en el mundo real, por lo que es adecuada para gran variedad aplicaciones geográficas en las que las localizaciones se representen como puntos. Sin embargo, GeoCluster busca soluciones para geometrías complejas como líneas o polígonos. Como la distancia euclídea solo está definida entre puntos, se ha buscado una solución sencilla que consiste en el concepto de punto representativo.

Se ha considerado que el punto representativo de una línea es su punto medio, y en el caso de un polígono es su centroide. Afortunadamente, la Java Topology Suite (JTS)[4] contiene implementaciones eficientes para ambos cálculos. Esta solución permite utilizar la distancia euclídea con cualquier geometría. Esto no significa que sea recomendable hacerlo para obtener buenos resultados: es una simplificación que no tiene en cuenta la forma real de la geometría.

3.2 Distancia de Hausdorff

La distancia de Hausdorff mide la lejanía entre dos conjuntos de puntos. Por esta razón, es mucho más conveniente que la distancia euclídea al trabajar con líneas o polígonos, pues estos se reducen a los conjuntos de puntos que los forman si se ignoran los segmentos que los conectan.

La distancia de Hausdorff entre un conjunto X y un conjunto Y se calcula como sigue:

[4] JTS es una API para trabajar con geometrías bidimensionales en Java. Ha sido de gran ayuda en el desarrollo de GeoCluster.

$$d_H(X,Y) = \max\{sup_{x\epsilon X}\ inf_{y\epsilon Y}\ d(x,y), sup_{y\epsilon Y}\ inf_{x\epsilon X}\ d(x,y)\}$$

Donde *sup* representa el supremo e *inf* el ínfimo, y d(x,y) es la distancia euclídea entre los puntos x e y.

Es decir, es la mayor de las distancias desde un punto de un conjunto a su punto más cercano en el otro conjunto. Es una distancia dirigida (los resultados que se obtienen de X a Y son distintos que los de Y a X), por lo que elegimos la mayor en ambos sentidos para tener una buena medida de la distancia entre dos conjuntos. La Figura 4 la compara con la distancia euclídea.

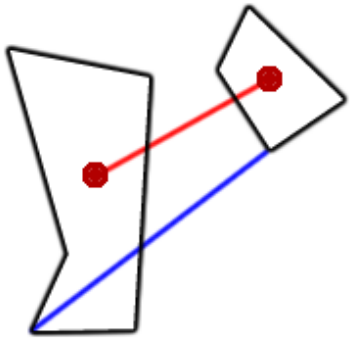


Figura 4: comparación entre la distancia euclídea, pintada en rojo y definida entre los centroides de los polígonos; y la distancia de Hausdorff, pintada en azul.

3.2.1 Implementación y optimizaciones

Pese a funcionar mejor cuando se trabaja con polígonos, esta distancia no sería útil si su cálculo tuviese un coste elevado. De hecho, su implementación básica (consistente en comparar todos los puntos de un conjunto con todos los puntos del otro, como se describe en el algoritmo siguiente) se puede considerar costosa.

algoritmo $NaiveHausdorffDistance(a_1..a_m, b_1..b_n)$:
$cmax := 0$
for a_i **in** $a_1..a_m$ **hacer**:
    $cmin := \infty$
    for b_i **in** $b_1..b_n$ **hacer**:
        $d := ||a_i, b_i||$
        if $d < cmin$:
            $cmin := d$
    if $cmin > cmax$:
        $cmax := cmin$
return $cmax$

Su tiempo de ejecución para dos conjuntos A y B es $O(m \cdot n)$, donde m=|A| y n=|B|.

En su lugar, GeoCluster utiliza una implementación eficiente obtenida de Taha y Hanbury [4]. Se realizan dos mejoras sobre el algoritmo original, que se van a explicar por su importancia en el contexto geoespacial:

Early Break: el algoritmo original solo se queda con la distancia obtenida en una ejecución del bucle interior si ésta es mayor que la máxima distancia encontrada por el algoritmo. Por ello, en cuanto el bucle interior obtenga una distancia mínima que sea menor que la distancia máxima obtenida hasta el momento, es innecesario seguir recorriendo el bucle.

Esta optimización se justifica fácilmente: un algoritmo enfocado a datos geográficos debe estar listo para trabajar con muchos datos, y en ocasiones un solo polígono está formado por una gran cantidad de puntos que recorrer.

Random Sampling: esta mejora está relacionada con el orden en que se eligen los elementos de un conjunto a la hora de comparar distancias. En muchos contextos (el geográfico es claramente uno de ellos) aparecerá el principio de localidad espacial: los datos almacenados de forma consecutiva estarán localizados, en general, a poca distancia entre sí. Este hecho está fuertemente relacionado con la probabilidad de que se produzca un *early break*. Si éste no se produce para un dato determinado, es poco probable que ocurra para los siguientes elementos si estos cumplen el principio de localidad espacial. Recorrer los elementos en orden aleatorio aumenta, por tanto, la probabilidad de *early break*. Esto es especialmente cierto al trabajar con polígonos, cuyos vértices estarán almacenados en el orden en el que están conectados.

A continuación se expone el algoritmo optimizado, cuyo coste tiende a lineal:

$\underline{\textbf{algoritmo}\ EfficientHausdorffDistance(a_1..a_m, b_1..b_n):}$
$x_1..x_m = randomize(a_1..a_m) \qquad (random\ sampling)$
$y_1..y_n = randomize(b_1..b_n)$
$cmax := 0$
for a_i **in** $a_1..a_m$ **hacer**:
    $cmin := \infty$
    for b_i **in** $b_1..b_n$ **hacer**:
        $d := ||a_i, b_i||$
        if $d < cmax$: $\qquad (early\ break)$
            break
        if $d < cmin$:
            $cmin := d$
    if $cmin > cmax$:
        $cmax := cmin$
return $cmax$

3.3 Diferencia en valor absoluto

En este trabajo se hace especial hincapié en el hecho de que los datos geoespaciales suelen ser complejos y almacenan todo tipo de contenido en forma de atributos. Aquí se ha realizado un acercamiento orientado al caso de atributos numéricos.

La diferencia en valor absoluto mide la distancia entre valores unidimensionales. Es fácil imaginar ejemplos de utilización en el contexto geográfico: dada una serie de datos que representan poblaciones, éstos pueden tener atributos como el número de habitantes o su densidad de población.

3.4 Distancia combinada para datos complejos

Con las distancias presentadas hasta ahora se puede agrupar en base al valor de atributos numéricos o en base a la localización de los datos geoespaciales. También se podrían construir fácilmente otras distancias para atributos textuales o para tipos de datos más específicos como fechas. Sin embargo la existencia de múltiples distancias no resuelve un problema. Al tratar de agrupar información compleja, puede resultar interesante utilizar múltiples criterios al mismo tiempo. Lo explicado hasta ahora solo nos permite utilizar una distancia en el algoritmo de agrupamiento.

Nos interesa poder resolver problemas como agrupar ciudades teniendo en cuenta su localización y a la vez su población, con la intención de detectar áreas metropolitanas. La solución propuesta es la siguiente: dado que a los algoritmos de agrupamiento implementados solo se les puede pasar una distancia, se va a hacer una **combinación lineal de varias distancias**. Además de especificar qué distancias intervienen, el usuario puede darles los pesos que desee. Es decir, la distancia combinada se define como sigue:

$$d_C(d_1..d_n, w_1..w_n) = \sum_{i=1}^{n} w_i d_i$$

Volviendo al ejemplo de antes, se puede agrupar teniendo en cuenta la localización en un 60% y el número de habitantes en un 40%, dándoles pesos de 0.6 y 0.4 respectivamente. Hay que aclarar que los pesos no tienen por qué sumar 1, pues se trata de una combinación lineal y no de una media ponderada.

4. Anotación de conjuntos

Generalmente la información geoespacial está acompañada de datos textuales que describen las localizaciones que se representan. Por ello, resulta interesante la posibilidad de dotar de información textual a las agrupaciones generadas.

En este trabajo se han llevado a cabo tareas de anotación de objetos para extraer la información textual relevante de los elementos que forman un conjunto, y a partir de ella generar información que lo describa. Su construcción está basada en la propuesta de Silva y otros [5].

4.1 Motivación y objetivos

Existe un paso posterior a la ejecución de un algoritmo de agrupamiento: consiste en que el usuario observe los resultados y sea capaz de entenderlos. Es decir, debe responder a la pregunta ¿qué significa este conjunto?

Lo que se busca es que sea el propio programa el que responda esa pregunta. Así se facilita la tarea al usuario de esta herramienta, que podrá comprender mejor los resultados que ha obtenido.

4.2 Extracción de nombres relevantes

El primer paso para resolver este problema es anotar los datos geoespaciales que va a utilizar el algoritmo. Este proceso de anotación consiste en analizar los atributos textuales y compararlos con un nomenclátor para detectar coincidencias. Así se puede obtener información del nomenclátor y anotar con ella el dato de entrada.

Decidir qué información anotar depende del contexto: en el caso de GeoCluster, se han realizado las pruebas con el nomenclátor español NGCE[5], y se han anotado las localizaciones con el municipio, provincia y Comunidad Autónoma correspondiente al elemento geográfico encontrado. Se ha elegido este nomenclátor por su completitud: está formado por 3667 topónimos normalizados, y está etiquetado en base a unidades administrativas[6].

4.3 Robustez: validación de anotaciones a través de la geometría

En el proceso de anotación de los datos pueden aparecer inconsistencias: por ejemplo, tomar un dato denominado "Villanueva" y que representa a *Villanueva de Gállego*, pero anotarlo como *Villanueva de Córdoba*.

Para corregir estos errores se ha añadido un proceso de validación de las anotaciones a partir de la geometría del dato anotado. En el ejemplo anterior, se comprueba que

---

[5] Nomenclátor Geográfico Conciso de España.
[6] http://www.ign.es/ign/layoutIn/actividadesToponimia.do

la geometría del objeto anotado como *Villanueva de Córdoba* no coincide con la geometría real de dicha localidad.

Para realizar esta comprobación se recurre a GADM[7], una base de datos de unidades administrativas a nivel global. Así se consigue descartar aquellas anotaciones incorrectas.

4.3.1 Comparación de geometrías

La principal dificultad del proceso de validación es encontrar una técnica que permita comparar dos geometrías, y decir si tienen aproximadamente la misma forma. La noción de "aproximadamente iguales" es importante, ya que es deseable que las dos geometrías de la Figura 5 se consideren válidas:

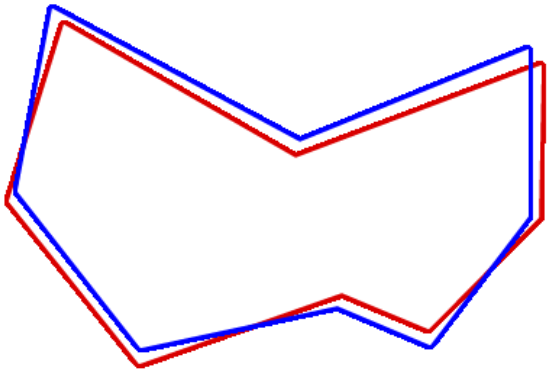


Figura 5: geometrías que deberían considerarse aproximadamente iguales.

A continuación se explican los diferentes enfoques que se han estudiado para resolver este problema:

1. Reconocimiento de formas 2D: si se toman las geometrías (en general, polígonos) como imágenes binarias 2D, se pueden calcular los momentos de imagen que actuarán como descriptores de la geometría.

Se ha descartado esta idea por su complejidad, pero se ha aprovechado una idea de ella, utilizar los momentos de imagen que sí tenemos disponibles: área y perímetro.

2. Distancia entre conjuntos de puntos[8]: este enfoque consiste en fijarse únicamente en los puntos que definen cada geometría, y calcular la media de las distancias punto-a-conjunto. Así obtenemos una medida de la lejanía entre los conjuntos. La fórmula es la siguiente para dos conjuntos de puntos A y B:

---

[7] GADM database of Global Administrative Areas.
[8] http://math.stackexchange.com/questions/1292345/way-to-measure-the-similarity-difference-of-2d-point-clouds

$$d_i(A,B) = \frac{1}{|A|+|B|}\left(\sum_{a\epsilon A} dist(a,B) + \sum_{b\epsilon B} dist(b,A)\right)$$

Esta solución tiene un inconveniente fundamental: no tiene en cuenta cómo están conectados los puntos entre sí, así que podría cometer errores con geometrías como las de la Figura 6, considerándolas válidas.

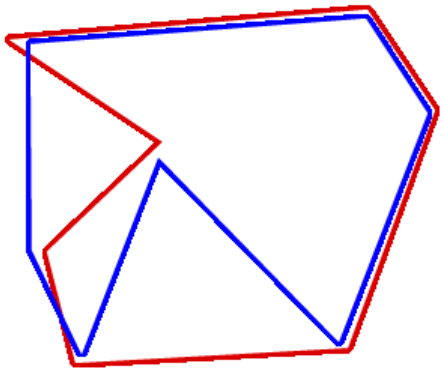


Figura 6: geometrías definidas por los mismos puntos, pero conectadas de manera diferente. No deberían considerarse aproximadamente iguales.

Solución implementada: se ha optado por utilizar las distancias punto-a-conjunto, por ser la opción más rápida de calcular. A la vez, se ha corregido su debilidad utilizando algunos momentos de imagen básicos, como el área y perímetro, por tratarse de dos descriptores que cambian sustancialmente en base a la forma en que está construida la geometría del objeto.

1. **Comprobar si las geometrías son exactamente iguales**. Para ello se utiliza una función de la librería JTS que observa la topología de ambas geometrías. Si este paso devuelve cierto, el validador las acepta. Si devuelve falso, se sigue haciendo comprobaciones.
2. **Momentos de imagen básicos**. Se comprueba si las áreas y perímetros de ambas geometrías son parecidos. En concreto, el valor menor debe ser al menos un 90% del valor mayor en ambos casos. Si este paso devuelve cierto, sigue haciendo comprobaciones. Si no, el validador las rechaza.
3. **Media de las distancias punto-a-conjunto**. Si la distancia entre ambas supera un determinado umbral, se considera que las geometrías son los suficientemente distintas y el validador las rechaza. Si no, son aceptadas.

Si bien este proceso no soluciona todos los casos de error de la media de distancias punto-a-conjunto, sí que reduce su número gracias a calcular previamente las áreas y perímetros.

4.4 Técnicas de anotación de conjuntos

Una vez anotados los elementos individuales se ejecuta el algoritmo de agrupamiento y se generan los conjuntos. Éstos también se pueden anotar a partir de las anotaciones de sus componentes.

Se pueden diseñar diferentes técnicas para resolver este problema. Aquí se ha optado por reunir las anotaciones de todos los elementos del conjunto, indicando el número de apariciones de cada una, y quedarse con las más comunes.

De este modo un conjunto se describirá a partir de la información más repetida entre sus elementos. Esta es solo una de las soluciones a este problema, que se ha elegido por ser sencilla y eficaz. Otro posible enfoque consistiría en resolverlo como un problema de Recuperación de Información y utilizar el índice *tf-idf*[9] para detectar los términos más repetidos en el conjunto y que al mismo tiempo sean poco comunes dentro de la colección formada por todos los elementos. Esto permitiría omitir anotaciones redundantes que se encuentran en la mayoría de datos que se van a agrupar, y que por tanto no aportan información útil.

[9] *Term Frequency – Inverse Document Frequency*, https://en.wikipedia.org/wiki/Tf%E2%80%93idf

5. Arquitectura y diseño de GeoCluster

La librería creada está compuesta por una serie de módulos, cuya construcción e integración se va a explicar a continuación. También se describe cuál es el flujo de trabajo, es decir, los pasos que seguirá el usuario al utilizar esta herramienta.

5.1 Herramientas utilizadas

En la actualidad existen librerías para trabajar con datos geoespaciales desde Java. De ellas se ha podido aprovechar la infraestructura básica para desarrollar GeoCluster. En concreto se han utilizado las siguientes, todas ellas de código abierto:

- **GeoTools[10]:** se ha utilizado esta librería como base para construir GeoCluster, por disponer de facilidades para trabajar con objetos geoespaciales y para acceder a ellos desde fuentes externas. Además, GeoTools implementa las especificaciones del OGC[11].
- **Java Topology Suite (JTS)[12]:** se han reutilizado algunas de las operaciones con geometrías 2D que implementa. Se ha elegido esta librería por ser una referencia en algoritmos espaciales.
- **Apache Commons Math[13]:** se han utilizado sus implementaciones de algunos algoritmos de agrupamiento como inspiración para construir los que incorpora GeoCluster.

5.2 Módulos desarrollados

La funcionalidad que proporciona esta librería está dispuesta en una serie de paquetes que se explican a continuación. Esta explicación se apoya en el diagrama de paquetes de la Figura 7. Una versión más detallada se encuentra en el Anexo I, junto con más información de diseño de software.

Algoritmos de agrupamiento (paquete *clustering*): se ha utilizado el patrón *Factory* para poder crear el algoritmo que se desee con unos parámetros específicos.

Distancias (*distance*): implementa todas las distancias con las que se pueden ejecutar los algoritmos de agrupamiento, así como otras distancias que utiliza la librería.

Parámetros (*parameter*): este paquete otorga flexibilidad a la construcción de algoritmos de agrupamiento. Permite utilizar la distancia combinada indicando qué atributos de los objetos geoespaciales estarán implicados.

---

[10] http://www.geotools.org/
[11] *Open Geospatial Consortium*, http://www.opengeospatial.org/
[12] http://www.vividsolutions.com/jts/JTSHome.htm
[13] http://commons.apache.org/proper/commons-math/

Anotaciones (*annotation*): contiene la lógica que decide qué anotaciones añadir a objetos y agrupaciones, así como la interfaz para acceder al nomenclátor que proporciona las anotaciones. Se asume que las fuentes de datos utilizadas están alojadas localmente.

Validación (*validation*): implementa los algoritmos de validación de anotaciones a partir de la geometría. Se ha asegurado la robustez de esta funcionalidad mediante excepciones que avisan si se está intentando validar un dato no anotado.

Centro (*center*): los algoritmos de agrupamiento de tipo K-Means funcionan con los centroides de los objetos geoespaciales. Este paquete calcula el centroide de geometrías complejas como polígonos, a partir de las funciones de JTS.

Utilidades (*util*): incluye las interfaces que comunican GeoCluster con fuentes de datos externas. En particular, se ha implementado la lectura y escritura en shapefiles y bases de datos MySQL.

Adapter (*adapter*): este paquete implementa el patrón *Adapter* para poder establecer una relación adecuada entre las implementaciones de objetos espaciales que ofrece GeoTools y los tipos de datos concretos que necesita GeoCluster.

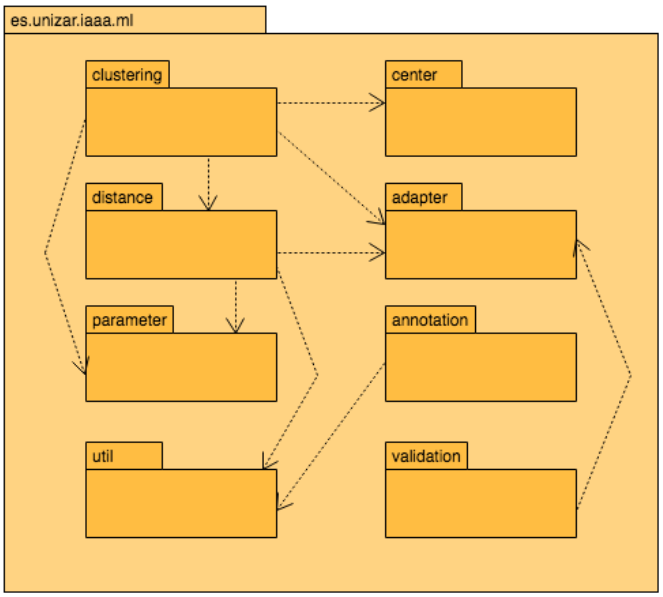


Figura 7: diagrama de paquetes de GeoCluster.

5.3 Interfaces

La funcionalidad de la librería está distribuida en interfaces, de las que se han construido algunas implementaciones principales. Por lo tanto, GeoCluster se puede ampliar añadiendo nuevas implementaciones.

- **Cluster y Clusterable**: son la pieza fundamental de la librería, ya que funcionan como adaptador entre los objetos geoespaciales de GeoTools y las herramientas implementadas en GeoCluster. Un *Clusterable* es cualquier elemento que se puede utilizar en un algoritmo de agrupamiento, y un *Cluster* es el conjunto resultante.
- **FeatureClusterer**: define un algoritmo de agrupamiento para objetos geoespaciales a partir de una distancia (*DistanceMeasure*). Encapsula en un método la funcionalidad del algoritmo.
- **DistanceMeasure**: define una distancia para los algoritmos de agrupamiento. De este modo se le puede pasar cualquier distancia a cualquier algoritmo.
- **Parameter**: define un parámetro del algoritmo. Actualmente solo tiene una implementación relacionada con los atributos que se utilizan en el algoritmo, pero es ampliable a nuevas funcionalidades.
    - **Attribute**: define un atributo sobre el que se va a realizar el agrupamiento. Tiene implementaciones numéricas y geométricas. De este modo, no importa que los objetos sean complejos y tengan múltiples atributos, el algoritmo sabrá sobre cuáles hacer los cálculos.
- **GeometryValidator**: permite efectuar la validación de anotaciones basada en la geometría, como paso previo al algoritmo de agrupamiento. Se pueden desarrollar múltiples técnicas de validación.

5.4 Algoritmos implementados

Los interfaces explicados en el apartado anterior no estarían completos sin sus respectivas implementaciones. Aquí se indican aquellas que forman parte de la librería creada, pero ésta se puede ampliar:

- **Distancias**: implementan el interfaz *DistanceMeasure* como se observa en la Figura 8.
    - **EuclideanDistance**: como se ha indicado en el capítulo 3, la distancia euclídea se ha definido para cualquier geometría. En el caso de polígonos, se calcula en base a su centroide.
    - **DiscreteHausdorffDistance**: hace un cálculo aproximado de la distancia de Hausdorff en base a un umbral. Se ha tomado de la librería JTS.
    - **ExactHausdorffDistance**: implementación propia de la distancia de Hausdorff, optimizada para datos geoespaciales como se ha explicado en el capítulo 3.
    - **AbsoluteDifferenceDistance**: compara la cercanía entre dos elementos numéricos, mediante la resta en valor absoluto. Es, por tanto, la herramienta básica para agrupar por atributos numéricos.
    - **CombinedDistance**: es la distancia combinada explicada en el capítulo 3. Se crea a partir de varias distancias, siguiendo el patrón *Composite* (véase Figura 8).

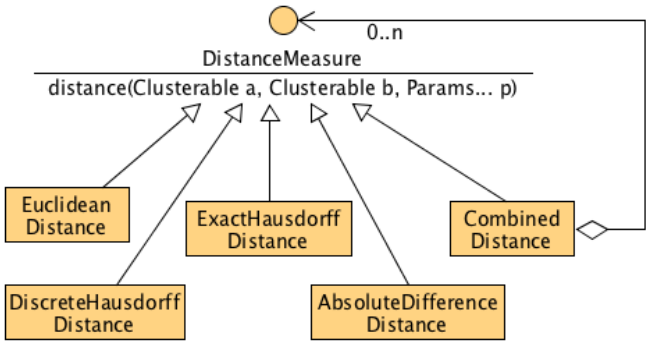


Figura 8: estructura de clases de las distancias.

- **Algoritmos de agrupamiento**: implementan el interfaz *FeatureClusterer*.
  - **DBSCANClusterer**: algoritmo DBSCAN explicado en el capítulo 2.
  - **KMeansClusterer**: algoritmo K-Means explicado en el capítulo 2.
  - **KMeansPlusPlusClusterer**: hace una reimplementación del primer paso de *KMeansClusterer*, consistente en la inicialización de centroides para obtener mejores resultados y facilitar la convergencia del algoritmo.

5.5 Mecanismos de anotación

El anotador de objetos y el de conjuntos recurren a un extractor, que recupera términos de un nomenclátor para que el anotador los utilice. La estructura de clases aparece en la Figura 9.

- **Extractor**: define un sistema de extracción de nombres para el proceso de anotación. Se puede adaptar a múltiples nomenclátores.
  - **DefaultExtractor**: a partir de una fuente de datos, la recorre entera buscando coincidencias con los atributos de cada objeto.
  - **NGCEExtractor**: es un extractor adaptado al nomenclátor NGCE. Solo analiza los campos que son interesantes para el proceso de anotación. Como NGCE almacena las coordenadas de cada elemento, puede comprobar si la anotación es correcta o no en base a estas coordenadas.

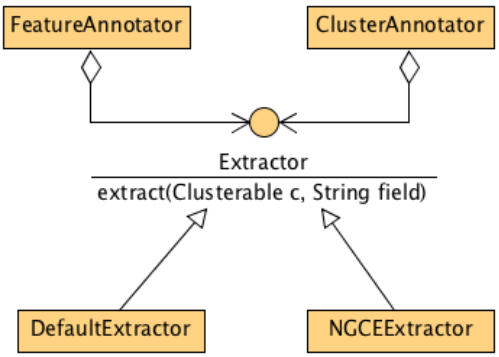


Figura 9: estructura de clases de los anotadores y el extractor.

5.6 Manipulación de datos

Los interfaces *Cluster* y *Clusterable* permiten adaptar las clases de GeoTools al uso que se les da en los algoritmos desarrollados. En particular, se utilizan los *SimpleFeature* como objeto geoespacial básico, y los *SimpleFeatureCollection* como agrupación de objetos.

- **Clusterable**:
  - **SimpleFeatureClusterable**: adaptador entre un *SimpleFeature* de GeoTools y un *Clusterable* de GeoCluster.
- **Cluster**:
  - **SimpleFeatureCollectionCluster**: actúa como adaptador entre un *SimpleFeatureCollection* de GeoTools y un *Cluster* de GeoCluster.

También se ha utilizado la noción de punto significativo de una geometría. Su objetivo es poder utilizar con polígonos aquellos algoritmos o distancias que solo están definidos para puntos.

- **CenterSelector**: devuelve un punto significativo de un objeto, que actuará como su centro.
  - **CentroidSelector**: calcula el centroide de un polígono, permitiendo utilizar *EuclideanDistance* con cualquier geometría.

5.7 Mecanismos de validación

GeoCluster utiliza la base de datos de unidades administrativas globales GADM para validar anotaciones, como se ha explicado en el capítulo 4. Sin embargo se puede ampliar para hacer funcionar la validación con otras fuentes de datos.

- **GeometryValidator**:
  - **GADMValidator**: el proceso de validación se adapta a la estructura de la base de datos GADM, por lo que solo estudia los campos que aportan información útil.

5.8 Entrada/Salida de datos

Se ha aprovechado la infraestructura de la librería GeoTools para desarrollar un sistema que pueda leer los datos que se van a utilizar y permita escribir los resultados obtenidos.

En particular se ha implementado la comunicación con las siguientes fuentes externas de datos:

- **Ficheros shapefile:** permite leer los objetos geoespaciales contenidos en el fichero, así como escribir los conjuntos resultantes en un nuevo fichero shapefile.

- **Bases de datos MySQL:** se pueden leer los datos de una tabla y escribir los resultados en una tabla nueva. Como MySQL soporta el tipo *Geometry*, almacenar objetos geoespaciales no entraña dificultad.

También se ha implementado una herramienta denominada *ClusterWriter*, para realizar la escritura de manera automática. Se encarga de producir dos salidas, y funciona tanto en MySQL como en shapefiles:

- Los conjuntos generados junto con sus anotaciones.
- Los mismos datos de entrada, junto con las anotaciones generadas para ellos y el número de conjunto al que pertenecen.

5.9 Flujo de trabajo con GeoCluster

Con todos los componentes creados, se expone a continuación el funcionamiento de la librería de manera conjunta, utilizando todas sus funcionalidades. La Figura 10 representa de forma visual este procedimiento.

0. Obtención de las fuentes de datos con las que vamos a trabajar, en formato shapefile o como base de datos MySQL.
1. Lectura de datos utilizando un DataStoreReader.
2. Anotación de los datos de entrada utilizando un nomenclátor como NGCE.
3. Validación de anotaciones comprobando si la geometría de los datos coincide con la anotación correspondiente, utilizando la base de datos GADM.
4. Ejecución del algoritmo de agrupamiento, especificando una distancia para el algoritmo y sobre qué atributos se miden las distancias.
5. Anotación de conjuntos a partir de las anotaciones generadas en el paso 2.
6. Escritura de los datos utilizando un DataStoreWriter. Se puede escribir en un shapefile o en una base de datos MySQL.

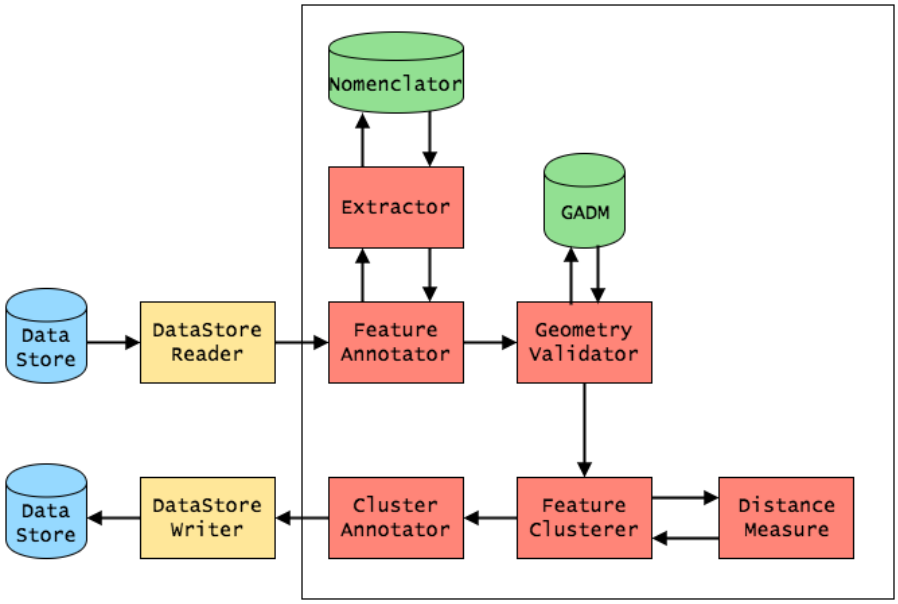


Figura 10: resumen del proceso de trabajo con GeoCluster, paso a paso.

6. Resultados obtenidos

Una vez desarrollada la librería, se han realizado una serie de experimentos con ella para comprobar los resultados que se producen. A continuación se explica el proceso empleado y un resumen de los resultados, que se exponen más extensamente en el Anexo II.

6.1 Metodología utilizada

En primer lugar se han recogido conjuntos de datos sobre los que poder hacer las pruebas. En particular se han utilizado los siguientes, todos ellos en formato shapefile:

- Diversos conjuntos de datos puntuales sintéticos[14].
- Un conjunto que contiene las Comunidades Autónomas españolas, almacenando su geometría con forma de polígono.
- Un conjunto que contiene todas las localidades españolas, almacenando su número de habitantes y su geometría poligonal. Se han utilizado únicamente subconjuntos de éste, por ser demasiado grande.

Una vez obtenidos los datos, se ha creado una aplicación que realiza diversos experimentos con ellos. Esta aplicación lee un conjunto y produce dos salidas mediante la utilidad ClusterWriter: un shapefile con los datos anotados y etiquetados a un conjunto, y otro shapefile con los agrupamientos anotados.

Uno de los objetivos de este trabajo es la visualización de estos resultados. Para ello se ha utilizado la herramienta Jupyter Notebook, que permite crear scripts interactivos escritos en Python. Se han creado varios de estos scripts (llamados *notebooks*) para visualizar los resultados de los experimentos realizados.

Cada *notebook* ejecuta la aplicación indicada antes, que está contenida en un jar, con unos parámetros determinados que indican el conjunto sobre el que se trabaja y las opciones del algoritmo de agrupamiento. A continuación lee los shapefiles obtenidos y procesa sus datos para visualizar cada conjunto de un color distinto, de modo que sean distinguibles.

Jupyter Notebook permite añadir fragmentos de texto en formato Markdown, por lo que los *notebooks* creados están documentados e incluyen las conclusiones obtenidas.

6.2 Resultados sobre conjuntos sintéticos

Los conjuntos sintéticos obtenidos consisten en puntos dispuestos en un espacio bidimensional. Sobre ellos se han realizado pruebas básicas que, además de

[14] Obtenidos de la sección *shapesets* de https://cs.joensuu.fi/sipu/datasets/

demostrar que los algoritmos funcionan, exponen las bondades y debilidades de cada alternativa implementada.

6.2.1 La distancia euclídea como medida de similaridad

Una de las cuestiones tratadas en este trabajo es la búsqueda de una distancia que funcione correctamente para comparar localizaciones de datos geoespaciales. Se han presentado la distancia euclídea y la de Hausdorff, siendo esta última una solución para geometrías complejas como polígonos.

Por ello, la distancia euclídea debería ser perfectamente válida al trabajar con geometrías puntuales. Para comprobarlo se ha ejecutado el algoritmo DBSCAN sobre el mismo conjunto y con los mismos parámetros, pero variando la elección de la distancia. Se muestran los resultados en la Figura 11.

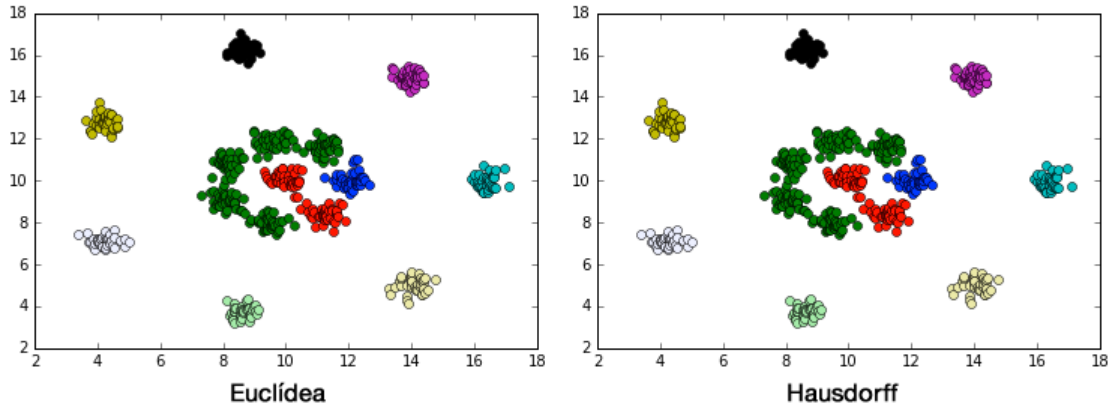


Figura 11: comparación entre la distancia euclídea y la de Hausdorff. Por tratarse de geometrías de tipo puntual, Hausdorff funciona igual que la distancia euclídea.

Estos resultados demuestran que el algoritmo se comporta igual con las dos alternativas, concluyendo que la distancia euclídea es la mejor opción al trabajar con puntos, pues es menos costosa de calcular y ofrece los mismos resultados.

6.2.2 DBSCAN y el concepto de densidad

El algoritmo de agrupamiento DBSCAN genera grupos en los lugares del espacio donde haya más densidad de elementos. Son los dos parámetros del algoritmo los que determinan cómo es esta densidad, ya que definen la distancia máxima a la que se puede encontrar un vecino y el mínimo número de vecinos que forman un grupo.

Modificar estos parámetros supone cambiar sustancialmente los resultados obtenidos, como se demuestra en la Figura 12. Por ello, es probable que ejecutar DBSCAN exija una fase de ajuste de los parámetros para cada problema concreto. Esta es su principal debilidad, que se ve corregida en versiones como GDBSCAN [6], HDBSCAN [7] o el algoritmo OPTICS [8].

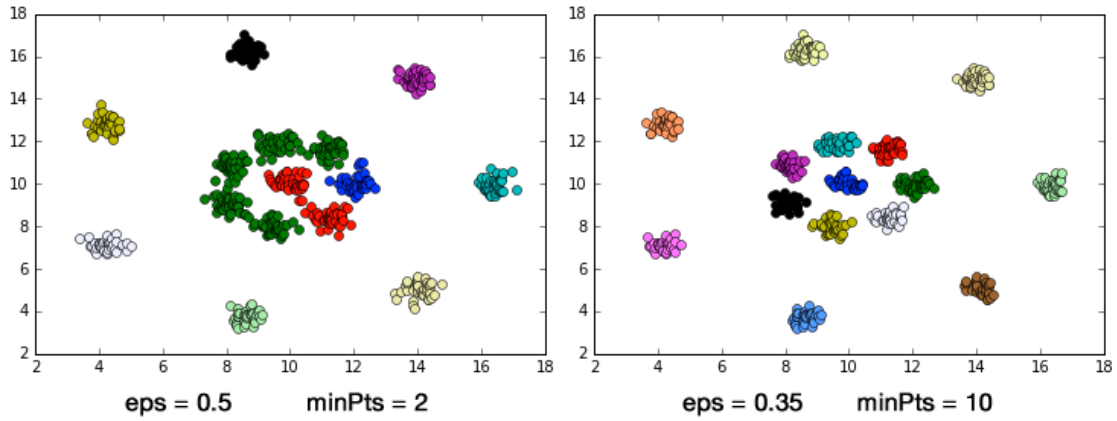


Figura 12: comparación entre dos ejecuciones de DBSCAN con parámetros diferentes.

6.3 Resultados sobre datos reales

Los conjuntos de datos reales obtenidos tienen formas poligonales, por lo que nos han permitido estudiar, entre otros, las ventajas de la distancia de Hausdorff. También se han realizado otras pruebas más avanzadas, o que resuelven algún problema de carácter geográfico.

6.3.1 La distancia de Hausdorff como medida de similaridad

En el apartado 6.2.1 se ha expuesto que la distancia euclídea funciona correctamente para datos puntuales. Sin embargo no está definida para polígonos, hecho que se ha resuelto en GeoCluster utilizando el centroide como punto representativo de una geometría.

Esta solución es una simplificación que no tiene en cuenta la forma real de los datos. Los resultados de la Figura 13 demuestran que la distancia de Hausdorff produce mejores resultados al trabajar con polígonos.

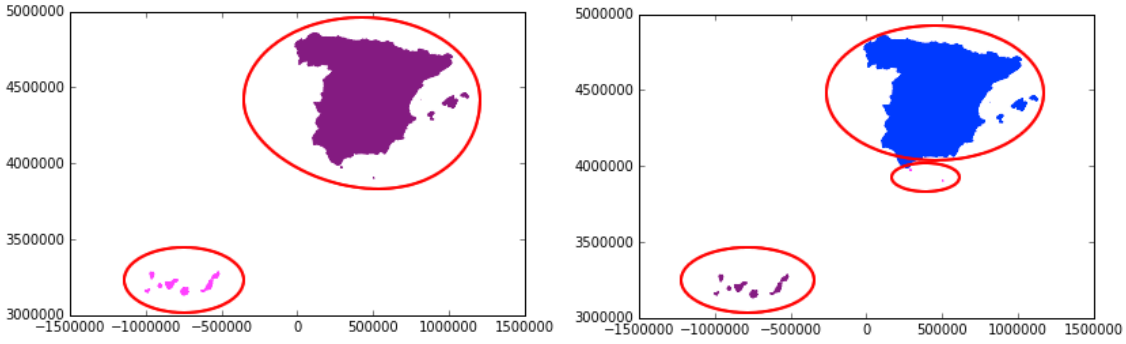


Figura 13: a la izquierda, la distancia euclídea ha producido dos conjuntos; a la derecha, la distancia de Hausdorff ha producido tres.

En la comparación anterior se aprecia cómo el mismo algoritmo genera salidas distintas según la medida de similaridad empleada. En concreto, el dato correspondiente a las ciudades autónomas de Ceuta y Melilla forma su propio grupo al utilizar la distancia de Hausdorff. Esto es así porque tiene en cuenta toda su

geometría en lugar de simplificar el problema recurriendo a su centroide. En general, esta aproximación es más cercana al punto de vista de un observador, que puede encontrar agrupaciones en conjuntos de datos homogéneos o contiguos. En estos casos la distancia de Hausdorff destaca frente a la simplificación de la euclídea.

6.3.2 La distancia combinada: un caso de ejemplo

Para poder abordar la complejidad de la información geográfica se ha implementado una distancia combinada, que se define como la combinación lineal de diversas distancias que se calculan sobre diferentes atributos de los datos de entrada. Una explicación más detallada se encuentra en el apartado 3.4.

Para comprobar su funcionamiento se ha realizado un experimento con datos de las poblaciones ubicadas en las áreas de influencia de las principales ciudades españolas. Estos datos están formados por la geometría poligonal de la población y el número de habitantes.

Se quiere comprobar si ciudades similares en población y localización forman grupos. Para conseguirlo se utiliza el algoritmo DBSCAN con la siguiente distancia combinada (cuyos pesos se han elegido de forma arbitraria, por ser el primer experimento con esta distancia):

- La distancia de Hausdorff entre las geometrías, con un peso de 0.6.
- La distancia de la diferencia en valor absoluto entre los enteros que representan el número de habitantes, con un peso de 0.4. Esta distancia será cercana a 0 cuando ambas ciudades tengan un número de habitantes similar.

Aquí se muestra el caso del área de influencia de la ciudad de Barcelona. Es un caso paradigmático porque existen dos poblaciones vecinas de población muy parecida como son Terrassa y Sabadell. El resultado esperado es que ambos formen un grupo propio.

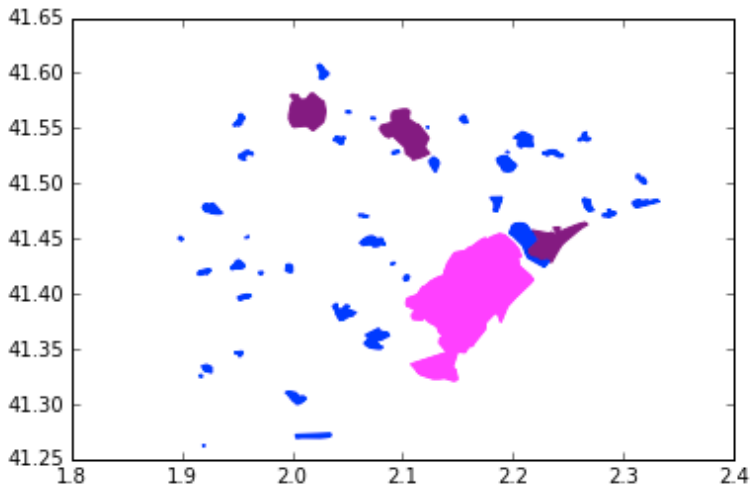


Figura 14: resultados de combinar la distancia de Hausdorff entre geometrías y la distancia en valor absoluto entre el número de habitantes.

Los conjuntos que se observan en la Figura 14 son los siguientes:

- Uno formado únicamente por la ciudad de Barcelona, por ser la de mayor población con una gran diferencia respecto al resto.
- La unión de las poblaciones de Terrassa, Sabadell y Badalona.
- Otro grupo formado por el resto de poblaciones, con un número de habitantes mucho menor que cualquier otra localidad.

Se ha creado el conjunto esperado, si bien incluye una tercera población como es Badalona, más alejada pero muy similar en población. Se han detectado dos causas para esto:

- La diferencia del número de habitantes está en una escala mucho mayor que la distancia entre localidades. Estas divergencias en la escala de los atributos provocan que unos tengan mucho más peso que otros, como sucede en este caso con el número de habitantes.
- Los pesos de la distancia combinada tienen una gran influencia en los resultados, y puede ser necesario modificarlos repetidamente hasta obtener los resultados buscados.

De este experimento se concluye que la distancia combinada puede dar resultados inesperados si los datos no han sido normalizados previamente. Se pueden utilizar los pesos con una función regularizadora, como se demuestra en la continuación de este experimento en el Anexo II.

6.3.3 Elección del algoritmo de agrupamiento según el problema

En los experimentos descritos hasta ahora se ha preferido DBSCAN sobre los algoritmos de tipo K-Means. Esto demuestra la importancia del concepto de densidad en el contexto de la información geoespacial, que provoca que DBSCAN funcione bien en numerosas aplicaciones de este tipo.

Sin embargo no hay que desestimar la utilidad de K-Means al trabajar con datos geográficos. En este experimento se muestra un ejemplo de uso de este algoritmo, consistente en proponer una nueva división administrativa de Aragón, de modo que existan cinco provincias en lugar de las tres actuales.

Para ello se ha ejecutado K-Means con la distancia de Hausdorff, ya que los datos de entrada consisten en las geometrías poligonales de las poblaciones de Aragón. Los resultados se muestran en la Figura 15.

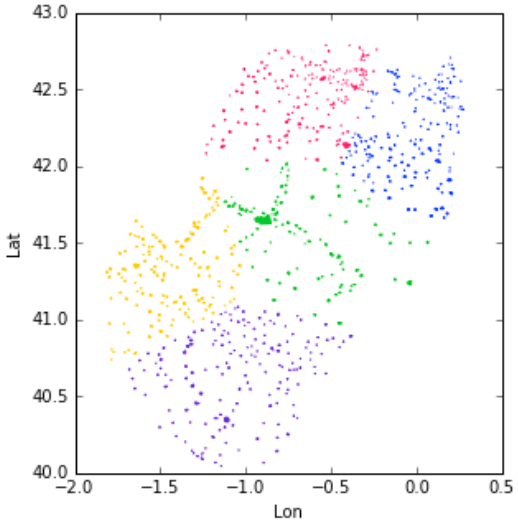


Figura 15: resultado de ejecutar K-Means sobre las localidades de Aragón con k=5.

La interpretación desde el punto de vista geográfico es interesante, ya que cada una de las cinco provincias resultantes tendría una ciudad de cierta entidad que podría actuar de cabecera provincial: Zaragoza, Huesca, Teruel, Calatayud y Monzón / Barbastro.

La razón de elegir K-Means en este caso reside en la naturaleza del problema: se busca dividir todo un espacio relativamente homogéneo en k grupos, y resulta deseable que éstos sean similares en dimensiones. Esta tendencia de K-Means, que supone una debilidad del algoritmo para resolver otros problemas, es en este caso una ventaja.

Por otra parte, la noción de densidad que utiliza DBSCAN es indeseable en este caso, ya que descartaría una inmensa cantidad de municipios pequeños ubicados en localizaciones espaciales poco pobladas. Además DBSCAN no permite indicar el número de grupos resultantes, por lo que habría sido necesario ajustar manualmente sus parámetros hasta conseguirlo.

En conclusión, no hay un algoritmo de agrupamiento que sea el mejor en todas las condiciones. En general su elección dependerá del problema a resolver.

7. Gestión del proyecto

A continuación se describe el proceso de gestión llevado a cabo durante la realización de GeoCluster. Es un aspecto importante en un Trabajo Fin de Grado, ya que una buena gestión facilita la comunicación y la coordinación entre el alumno y el director.

7.1 Control de versiones

El proyecto utiliza la herramienta de control de versiones Git y está alojado en GitHub. Se ha utilizado un repositorio privado durante el proceso de desarrollo, y finalmente se ha trasladado la librería definitiva a un repositorio público para permitir su acceso y consulta por otros desarrolladores.

En cuanto al desarrollo de la memoria, ésta se ha compartido con el director a través del mismo repositorio privado, y así él ha podido actualizarla con comentarios y sugerencias de mejora. Asimismo, la herramienta de control de versiones de Word ha ayudado en esta tarea.

7.2 Comunicación

Para garantizar la correcta comunicación entre el alumno y el director a lo largo del proyecto, se han tomado las siguientes medidas:

- Utilización de la herramienta Slack, que integra un sistema de comunicación por chat con avisos de actualizaciones en los repositorios de GitHub. Así se facilita la resolución de dudas y la realización de comentarios relacionados con actualizaciones recientes.
- Reuniones periódicas con el director para establecer las principales tareas, especialmente durante las primeras etapas del proyecto, y para repasar las tareas ya realizadas. Según el trabajo avanzaba, la carga de reuniones fue disminuyendo y se empezó a funcionar con mayor autonomía.

7.3 Planificación

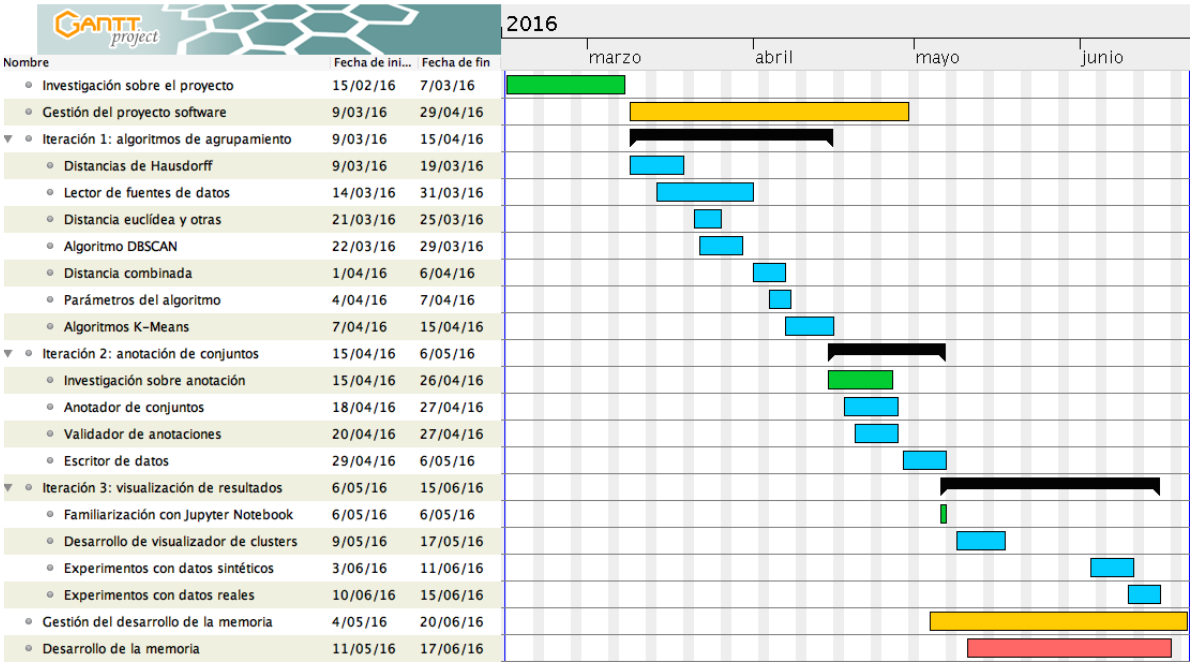
La planificación del proyecto está basada en iteraciones, teniendo cada una de ellas una tarea principal por desarrollar. Las fases de este trabajo se enumeran a continuación, y en paralelo a ellas se han llevado a cabo otras tareas de gestión del proyecto y de la memoria.

- **Investigación sobre el proyecto**: búsqueda de información sobre otras soluciones que resuelvan el mismo problema, y familiarización con las librerías que se van a utilizar.
- **Iteración 1 – algoritmos de agrupamiento**: se han desarrollado los algoritmos explicados en esta memoria, así como las distancias necesarias para que funcionen correctamente. También se ha presentado la distancia

combinada y se ha creado un sistema de lectura de datos para probar el sistema.

- **Iteración 2 – anotación de conjuntos**: en primer lugar se han buscado las fuentes de datos a utilizar, como el nomenclátor NGCE. A continuación se ha desarrollado un anotador de objetos y de conjuntos, y se ha implementado el sistema de validación que comprueba las geometrías. También se ha creado un sistema de escritura de resultados.
- **Iteración 3 – visualización de resultados**: concluida la librería GeoCluster, se han realizado diversos experimentos utilizando Jupyter Notebook, y se han alcanzado una serie de conclusiones.
- **Desarrollo de la memoria**: se ha realizado en paralelo a la Iteración 3.

El diagrama de Gantt de la Figura 16 expone la planificación entrando en mayor detalle, indicando además las fechas. En azul se indican las tareas de desarrollo y realización de experimentos, en verde la investigación y análisis, en amarillo la gestión y en rojo el desarrollo de la memoria.



Nombre	Fecha de ini...	Fecha de fin
Investigación sobre el proyecto	15/02/16	7/03/16
Gestión del proyecto software	9/03/16	29/04/16
Iteración 1: algoritmos de agrupamiento	9/03/16	15/04/16
Distancias de Hausdorff	9/03/16	19/03/16
Lector de fuentes de datos	14/03/16	31/03/16
Distancia euclídea y otras	21/03/16	25/03/16
Algoritmo DBSCAN	22/03/16	29/03/16
Distancia combinada	1/04/16	6/04/16
Parámetros del algoritmo	4/04/16	7/04/16
Algoritmos K-Means	7/04/16	15/04/16
Iteración 2: anotación de conjuntos	15/04/16	6/05/16
Investigación sobre anotación	15/04/16	26/04/16
Anotador de conjuntos	18/04/16	27/04/16
Validador de anotaciones	20/04/16	27/04/16
Escritor de datos	29/04/16	6/05/16
Iteración 3: visualización de resultados	6/05/16	15/06/16
Familiarización con Jupyter Notebook	6/05/16	6/05/16
Desarrollo de visualizador de clusters	9/05/16	17/05/16
Experimentos con datos sintéticos	3/06/16	11/06/16
Experimentos con datos reales	10/06/16	15/06/16
Gestión del desarrollo de la memoria	4/05/16	20/06/16
Desarrollo de la memoria	11/05/16	17/06/16

Figura 16: diagrama de Gantt con la planificación de tareas del proyecto.

7.3.1 Esfuerzos realizados

A continuación se resumen las horas empleadas en el proyecto, organizadas por etapas. Además, en la Figura 17 aparecen sus porcentajes de tiempo.

- **Investigación y análisis**: 45 horas.
- **Total Desarrollo**: 171 horas. De las cuales:
  - Iteración 1 – algoritmos de agrupamiento: 70 horas.
  - Iteración 2 – anotación de conjuntos: 55 horas.
  - Iteración 3 – visualización de resultados: 46 horas.
- **Tareas de gestión**: 42 horas.
- **Desarrollo de la memoria**: 64 horas.
- **Total**: 322 horas.

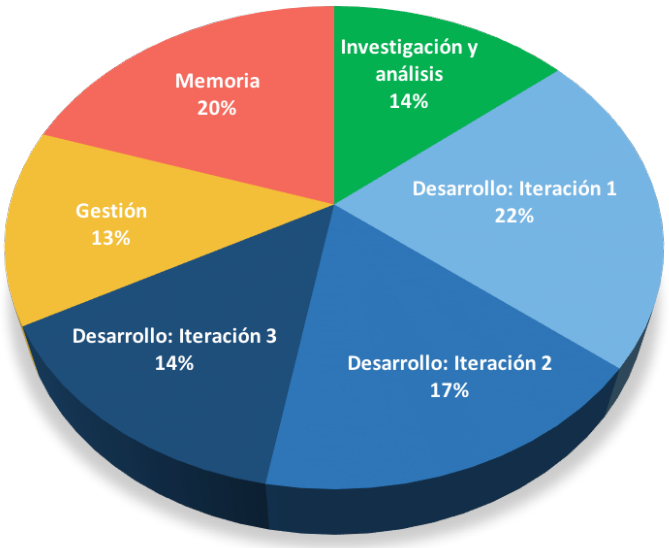


Figura 17: etapas del proyecto según su porcentaje de tiempo respecto al total del proyecto.

En el Anexo III se presentan resultados más extensos acerca de la planificación de este proyecto.

8. Conclusiones

Una vez desarrollada la librería GeoCluster y tras experimentar con ella, se han alcanzado una serie de conclusiones que se exponen a continuación.

8.1 Conclusiones sobre el trabajo

El resultado de este trabajo es una librería de algoritmos de agrupamiento diseñada específicamente para datos geoespaciales. Es decir, es capaz de calcular distancias entre geometrías complejas y tiene en cuenta los atributos numéricos con los que están etiquetados los datos.

Además, la librería incluye un mecanismo para anotar los elementos de entrada y los conjuntos obtenidos, utilizando un nomenclátor. De este modo el usuario dispone de una descripción de cada grupo, que le facilitará la tarea de interpretación de los resultados. Esta anotación es robusta gracias a las técnicas de validación implementadas.

Sobre el producto creado se han realizado una serie de experimentos que nos han permitido alcanzar las siguientes conclusiones técnicas:

- Implementar la distancia de Hausdorff para medir la similaridad entre la localización de los objetos geoespaciales ha sido una buena idea. Se ha comprobado que es mucho más precisa que la distancia euclídea para comparar polígonos.
- El desarrollo de dos tipos de algoritmos de agrupamiento (DBSCAN y los de tipo K-Means) ha permitido resolver un abanico de problemas mucho más amplio, concluyendo que no hay un algoritmo mejor para todo sino uno más adecuado para cada problema.
- La contextualización de agrupaciones mediante sus anotaciones obtenidas de un nomenclátor aporta valor añadido al producto final. Además, las técnicas de validación aumentan la fiabilidad de las anotaciones elegidas.
- Uno de los objetivos del trabajo consiste en mantener una batería de pruebas sobre la que experimentar y visualizar los resultados. Este objetivo ha sido una parte fundamental del trabajo, ya que ha permitido alcanzar conclusiones y detectar errores que de otro modo habrían permanecido ocultos.

8.2 Posibles ampliaciones y mejoras

Como el código de GeoCluster está disponible en GitHub, y al haberse construido directamente sobre la librería de código abierto GeoTools, es muy fácil que cualquier desarrollador interesado pueda desarrollar ampliaciones. Aquí se enumeran las más importantes:

- Implementar la normalización de atributos como paso previo al algoritmo de agrupamiento. En general no es necesaria, pero mejora el funcionamiento de la distancia combinada, ya que al usarla se pueden comparar atributos en escalas distintas.
- Se han expuesto las dificultades de utilizar DBSCAN, relacionadas con el ajuste de los parámetros del algoritmo. Existen versiones avanzadas que mejoran este tipo de aspectos, como GDBSCAN o HDBSCAN. Lo mismo sucede con el algoritmo OPTICS.
- Por ahora solo se han implementado las interfaces que comunican GeoCluster con bases de datos MySQL y ficheros Shapefile. Esto no significa que la librería solo pueda utilizarse con estas fuentes de datos, pero actualmente el usuario deberá encargarse de su conexión. Este es, por tanto, otro de los aspectos que se pueden ampliar.
- Las distancias desarrolladas hasta ahora cubren dos tipos de atributos: geométricos y numéricos. Los datos geográficos también suelen tener información textual. La existencia de distancias entre atributos textuales es un campo complejo, muy dependiente del contexto, pero que puede dar resultados muy interesantes.
- En la línea del punto anterior, trabajar con datos geográficos semánticos (es decir, enlazados entre sí y con más datos) aportaría más contexto. Esto permitiría hacer mejores anotaciones de los conjuntos.

8.3 Reflexión personal

Decidí dedicar mi Trabajo Fin de Grado a los algoritmos de agrupamiento y a los datos geográficos por interés personal en ambos temas. Una vez he desarrollado GeoCluster y tras utilizarlo con datos reales para extraer conclusiones, puedo decir que tomé una buena decisión.

Realizar este trabajo me ha permitido profundizar en técnicas y algoritmos que desconocía, o de los que tenía un conocimiento superficial. También me ha permitido aplicar mis conocimientos de proyectos de software para desarrollar una arquitectura extensa y compleja.

Pero llegar hasta aquí no ha estado exento de dificultades. Al no disponer de un conocimiento profundo de arquitectura software, fue costoso integrar correctamente todos los módulos construidos. En este sentido destaco la ayuda del director de este trabajo.

Del mismo modo, me he enfrentado por primera vez a la metodología de desarrollo basado en *tests*. Ha resultado especialmente útil a la hora de detectar el correcto funcionamiento de todos los casos de los algoritmos.

Además he podido apreciar la utilidad del código abierto, que ha facilitado inmensamente la tarea de desarrollar mi trabajo. Por esta razón, he dejado GeoCluster disponible en GitHub junto con su documentación[15]. Al haberlo construido como una extensión de GeoTools, este podría ser un primer paso para que mi proyecto forme parte de dicha librería si sus desarrolladores están interesados.

También quiero destacar la importancia de la visualización de resultados. Este trabajo me empezó a satisfacer en el momento en que pude probarlo con datos reales, inventar problemas de geografía y ordenación territorial, intentar resolverlos y ver los resultados obtenidos.

En definitiva, estoy orgulloso de GeoCluster y valoro todo lo que he aprendido durante este proyecto. Para una persona como yo, que quiere dedicarse a aprender más sobre la Inteligencia Artificial y el análisis de datos, creo que este ha sido un buen primer paso.

[15] Librería en https://github.com/IAAA-Lab/GeoCluster, experimentos en https://github.com/IAAA-Lab/GeoCluster-Demo

Bibliografía

[1] A. Vattani, "The hardness of k-means clustering in the plane."

[2] D. Arthur and S. Vassilvitskii, "k-means++: The Advantages of Careful Seeding."

[3] X. X. Martin Ester, Hans-Peter Kriegel, Jörg Sander, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise."

[4] A. A. Taha and A. Hanbury, "An Efficient Algorithm for Calculating the Exact Hausdorff Distance," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 11, pp. 2153–2163, 2015.

[5] M. J. Silva, B. Martins, M. Chaves, N. Cardoso, and A. P. Afonso, "Adding Geographic Scopes to Web Resources."

[6] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu, "Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications," *Data Min. Knowl. Discov.*, vol. 2, no. 2, pp. 169–194, 1998.

[7] M. Farhadur Rahman, W. Liu, S. Bin Suhaim, S. Thirumuruganathan, N. Zhang, and G. Das, "HDBSCAN: Density based Clustering over Location Based Services."

[8] J. S. Mihael Ankerst, Markus M. Breunig, Hans-peter Kriegel, "OPTICS: Ordering Points To Identify the Clustering Structure." .

Índice de figuras

Anexo I: Diseño del sistema

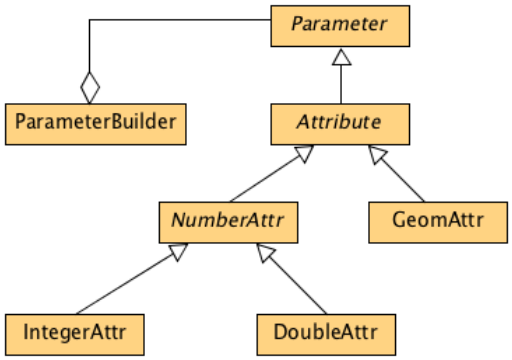
1. Patrones de diseño

Se han utilizado determinados patrones arquitecturales para facilitar el diseño del sistema y su posterior utilización.

2.1 Patrones de creación de objetos

Al utilizar GeoCluster el usuario debe crear diversos objetos: el algoritmo de agrupamiento, la distancia, los parámetros del algoritmo, el anotador y el validador, entre otros. Algunos de estos objetos son complejos, por lo que se han utilizado patrones como *Factory* o *Builder* para facilitar su construcción.

Por ejemplo, la tarea de crear algoritmos de agrupamiento se ha delegado en una clase *ClustererFactory* que está preparada para construir los diferentes tipos de algoritmo existentes. En otras situaciones el problema no es la existencia de múltiples objetos sino la complejidad de crearlos. En estos casos se ha utilizado el patrón *Builder* para encapsular el proceso de construcción. Es lo que sucede con los parámetros del algoritmo o con las distancias.

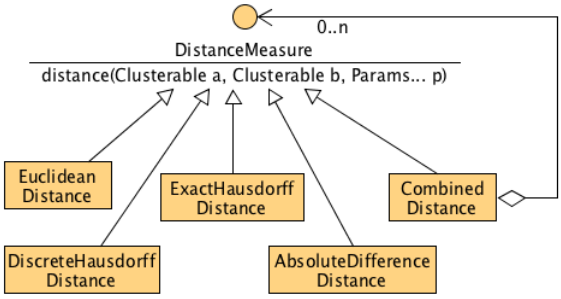


En concreto, se han utilizado estas soluciones en las siguientes clases:

- Algoritmos de agrupamiento: *ClustererFactory*.
- Distancias: *DistanceMeasureBuilder*.
- Parámetros: *ParameterBuilder*.
- Cluster: *ClusterBuilder*.

2.2 El patrón Composite en la distancia combinada

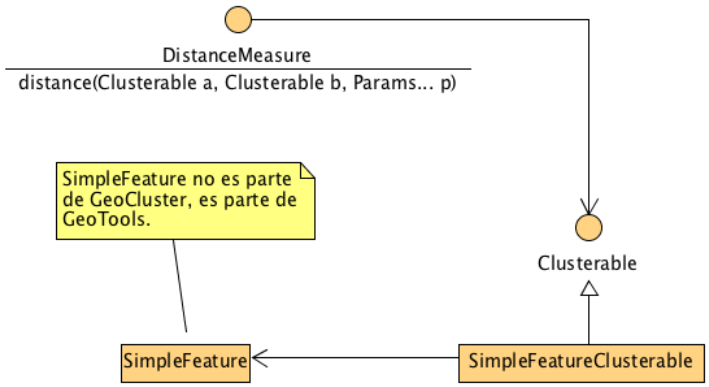
La distancia combinada es un ejemplo perfecto de aplicación del patrón *Composite*, ya que consiste en un objeto que es parte de una familia de objetos (las distancias), y a la vez está definido como un conjunto de dichos objetos. De este modo, cualquier combinación de distancias es posible siempre que hereden de *DistanceMeasure*.



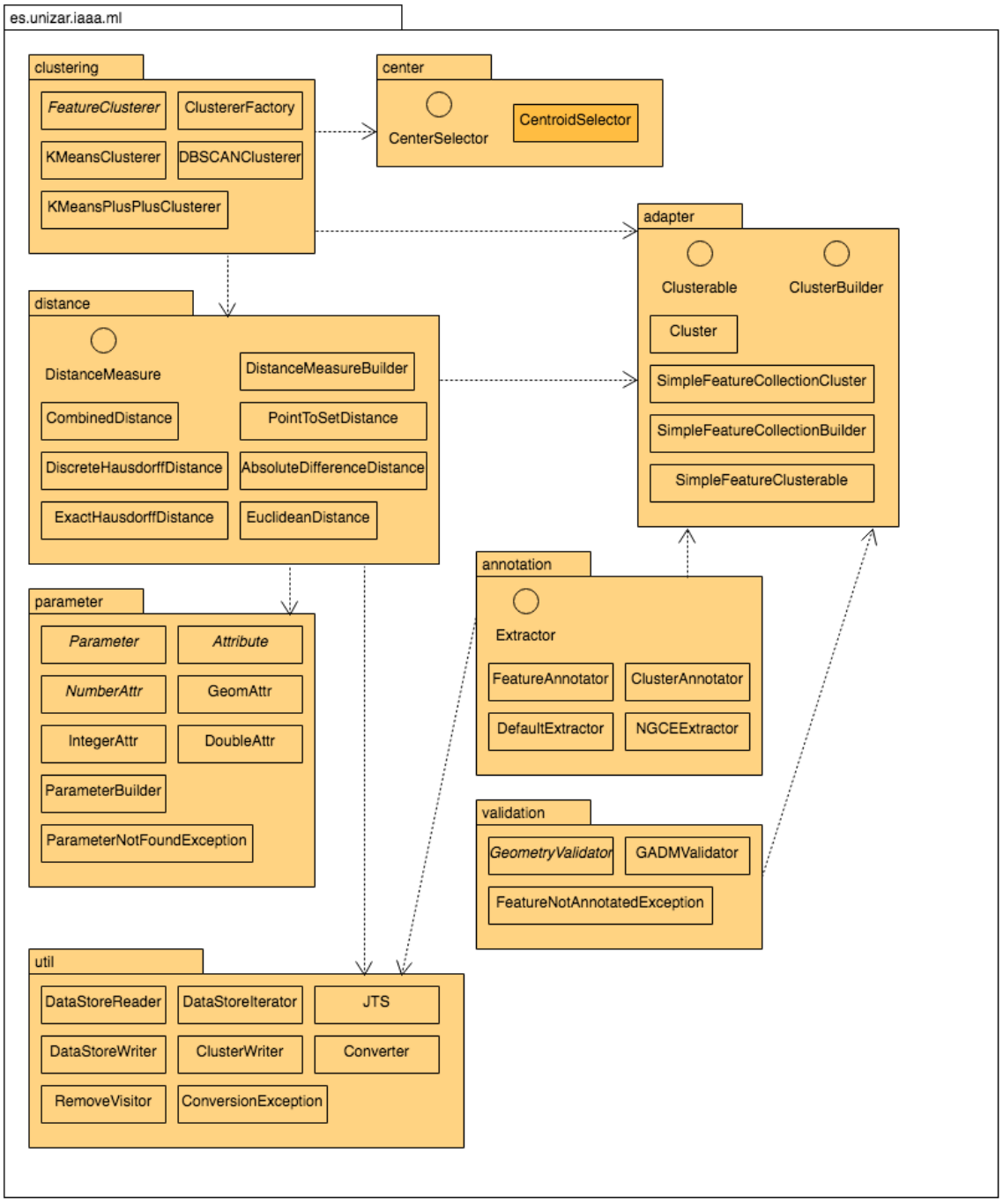
2.3 El patrón Adapter y su papel en GeoCluster

Como se ha explicado, la librería GeoCluster está construida directamente sobre GeoTools. Utiliza, entre otros, el objeto básico de GeoTools para trabajar con objetos geoespaciales: *SimpleFeature*. En lugar de utilizar directamente esta clase, se han separado ambas partes y se ha utilizado un patrón *Adapter* para conectarlas.

La interfaz *Clusterable* implementa el patrón *Adapter* en GeoCluster, como se aprecia en la imagen siguiente, que es un ejemplo de cómo las distancias acceden a los objetos de tipo *SimpleFeature* a través del patrón.



2. Estructura de paquetes

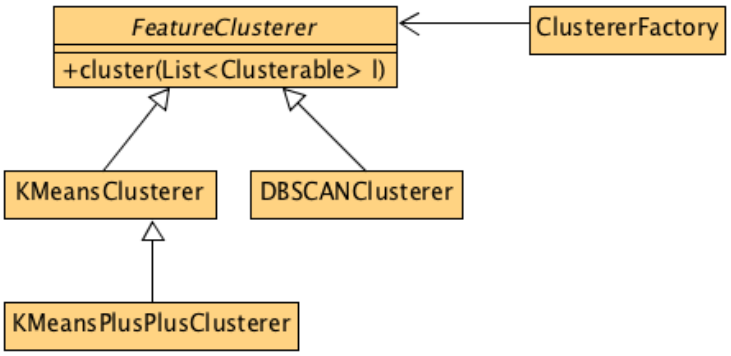


3. Estructura de clases

A continuación se presenta la estructura de clases del sistema, dividida en paquetes por cuestiones de legibilidad y extensión.

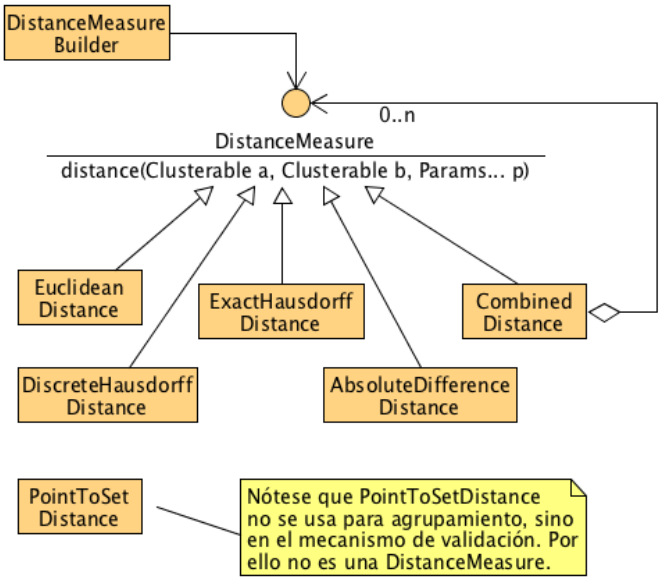
3.1 Paquete Clustering

Se ha utilizado el patrón *Factory* para poder crear el algoritmo que se desee con unos parámetros específicos.



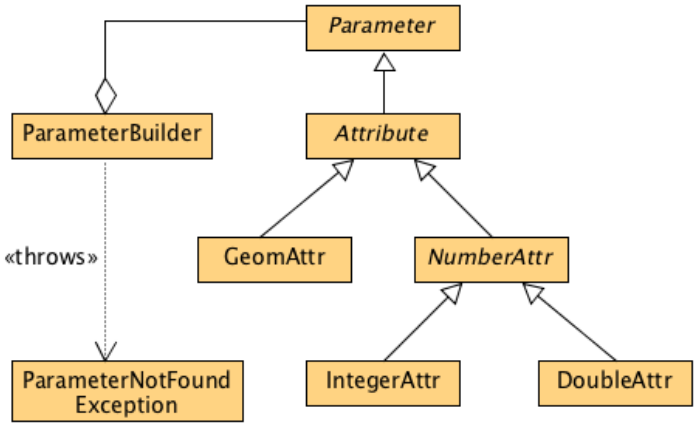
3.2 Paquete Distance

Implementa todas las distancias con las que se pueden ejecutar los algoritmos de agrupamiento, así como otras distancias que utiliza la librería.



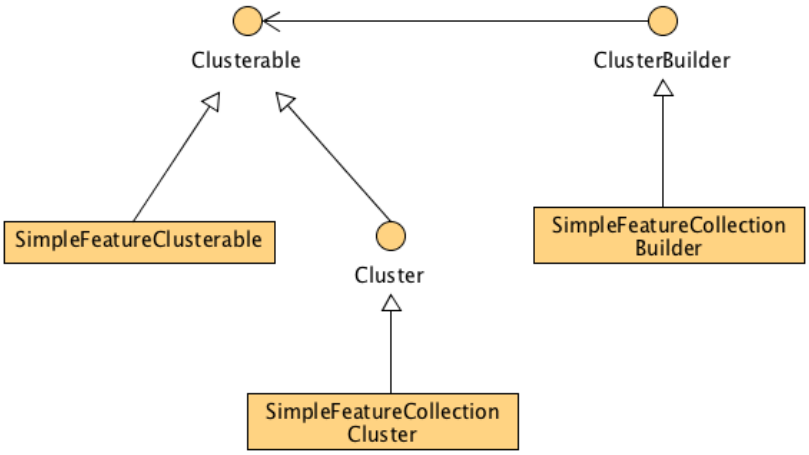
3.3 Paquete Parameter

Este paquete otorga flexibilidad a la construcción de algoritmos de agrupamiento. Permite utilizar la distancia combinada indicando qué atributos de los objetos geoespaciales estarán implicados.



3.4 Paquete Adapter

Este paquete implementa el patrón *Adapter* para poder establecer una relación adecuada entre las implementaciones de objetos espaciales que ofrece GeoTools y los tipos de datos concretos que necesita GeoCluster.

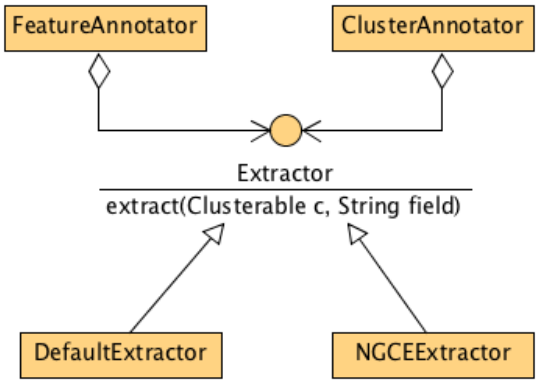


3.5 Paquete Center

Los algoritmos de agrupamiento de tipo K-Means funcionan con los centroides de los objetos geoespaciales. Este paquete calcula el centroide de geometrías complejas como polígonos, a partir de las funciones de JTS.

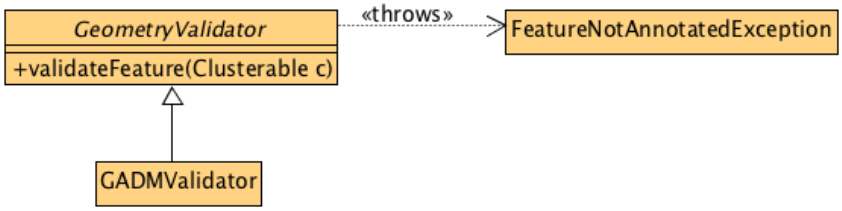
3.6 Paquete Annotation

Contiene la lógica que decide qué anotaciones añadir a objetos y agrupaciones, así como la interfaz para acceder al nomenclátor que proporciona las anotaciones. Se asume que las fuentes de datos utilizadas están alojadas localmente.



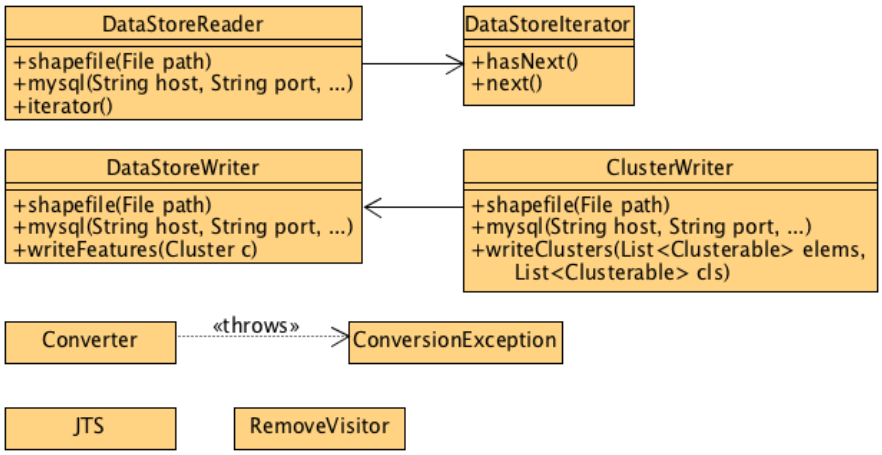
3.7 Paquete Validation

Implementa los algoritmos de validación de anotaciones a partir de las geometrías. Se ha asegurado la robustez de esta funcionalidad mediante excepciones que avisan si se está intentando validar un dato no anotado.



3.8 Paquete Util

Incluye las interfaces que comunican GeoCluster con fuentes de datos externas. En particular, se ha implementado la lectura y escritura en shapefiles y bases de datos MySQL.



Anexo II: Experimentos detallados

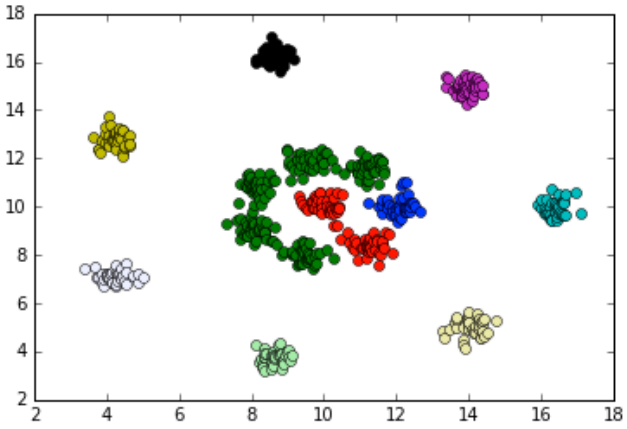
En este anexo se detallan los experimentos que se han realizado para probar la librería desarrollada, y se describen los conjuntos de datos que se han utilizado en dichos experimentos.

1. Experimento con datos puntuales

En primer lugar se han realizado una serie de pruebas básicas con conjuntos de datos sintéticos. Este experimento utiliza el conjunto denominado R15, consistente en 15 agrupaciones claramente delimitadas en áreas concretas del espacio, en las que la densidad de puntos es muy elevada. Por esta razón, es un conjunto interesante para demostrar las ventajas de DBSCAN frente a K-Means en cuanto al concepto de densidad.

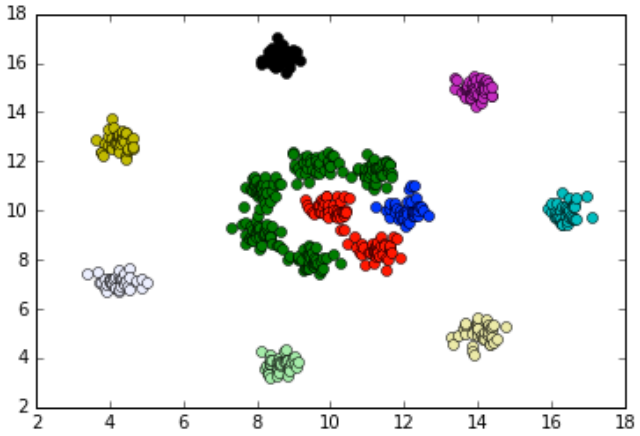
1.1 Ejecución 1

Algoritmo: DBSCAN (eps = 0.5, minPts = 2).
Distancia: Hausdorff.
Resultados: se generan 10 grupos en lugar de 15, debido a que algunos están demasiado cerca entre ellos y el *eps* indicado es muy elevado.



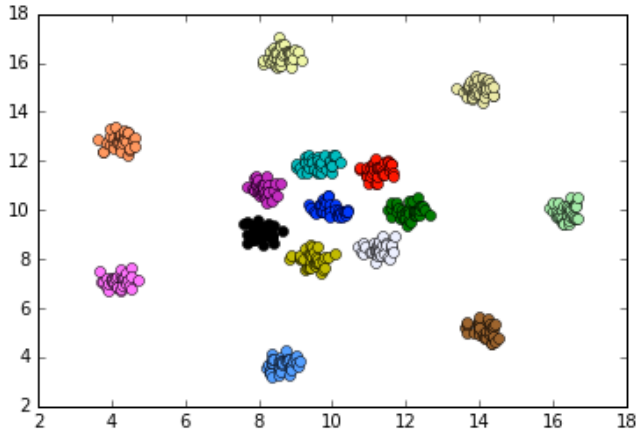
1.2 Ejecución 2

Algoritmo: DBSCAN (eps = 0.5, minPts = 2).
Distancia: Euclídea.
Resultados: los mismos que en Ejecución 1, demostrando que la distancia de Hausdorff y la distancia euclídea funcionan igual para datos de tipo puntual.



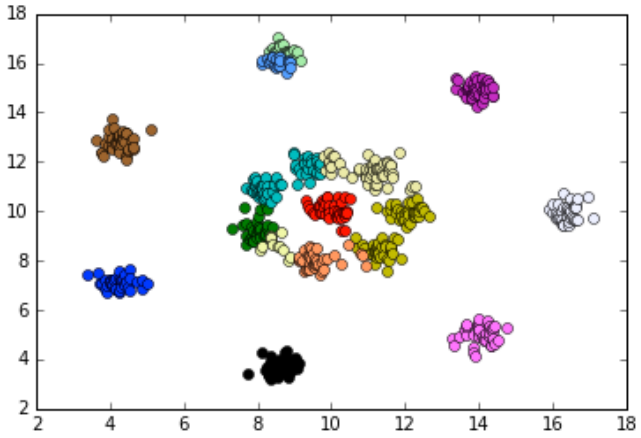
1.3 Ejecución 3

Algoritmo: DBSCAN (eps = 0.35, minPts = 10).
Distancia: Euclídea.
Resultados: al reducir *eps* se han separado los grandes conjuntos que se habían formado hasta ahora. Del mismo modo, al incrementar minPts se descartan pequeños conjuntos que se pueden crear como consecuencia de la reducción de *eps*. Ahora se obtienen los 15 conjuntos deseados, a costa de perder una pequeña cantidad de elementos que no se han clasificado, porque DBSCAN los considera ruido.



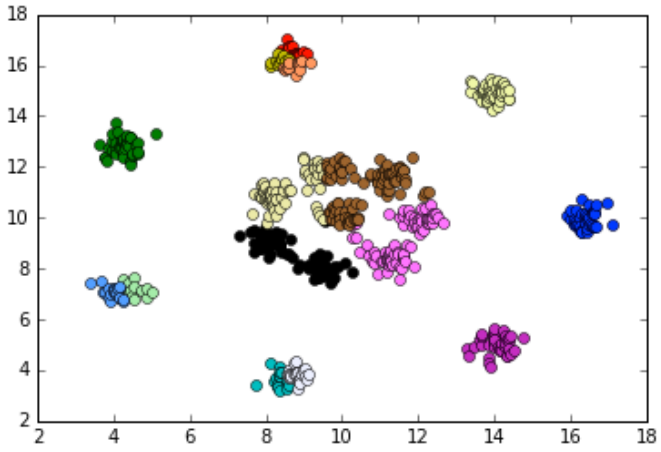
1.4 Ejecución 4

Algoritmo: K-Means (k = 15)
Distancia: Euclídea
Resultados: aunque se han especificado 15 conjuntos, éstos no son los esperados (aunque son ligeramente distintos en cada ejecución). Esta ejecución permite ver los inconvenientes de K-Means: si dos centroides se inicializan demasiado cerca en el espacio podemos obtener resultados indeseados, como sucede con los conjuntos claramente delimitados que aquí se dividen en dos.



1.5 Ejecución 5

Algoritmo: K-Means++ (k = 15)
Distancia: Euclídea
Resultados: de nuevo, no se generan los mismos 15 conjuntos en cada ejecución por tratarse de un algoritmo con un componente aleatorio. Sin embargo ahora los centroides iniciales tienden a estar separados entre sí, ya que se está usando K-Means++. Aun así los resultados son malos. Esto es debido a la naturaleza de los datos, que están muy concentrados en pequeñas porciones del espacio. Separar los centroides no garantiza que cada uno de ellos vaya a parar a una de las zonas densas.



1.6 Conclusiones

Este conjunto consiste en un espacio bidimensional con áreas de alta densidad separadas en un gran espacio vacío. Como DBSCAN detecta las áreas más densas y las agrupa, ofrece unos resultados mucho mejores que K-Means para estos datos.

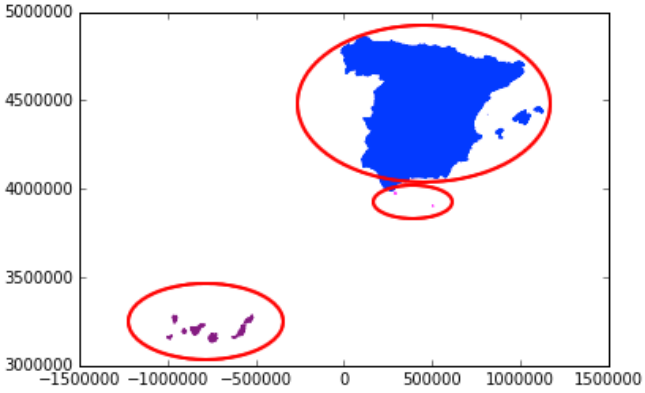
Por otro lado, la distancia euclídea se comporta igual que la de Hausdorff por tratarse de datos puntuales, por lo que la euclídea es suficiente por ser la más simple.

2. Experimento con datos poligonales

En este caso se ha utilizado un conjunto de datos consistente en los polígonos de las Comunidades Autónomas españolas. Es decir, cada dato corresponde a una comunidad, además de haber un dato correspondiente a la unión de las 2 ciudades autónomas (Ceuta y Melilla). Por tanto hay, en total, 18 datos.

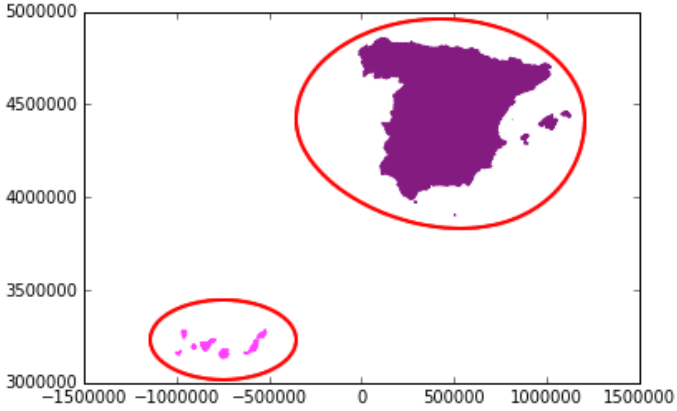
2.1 Ejecución 1

Algoritmo: DBSCAN (eps = 350000, minPts = 0).
Distancia: Hausdorff.
Resultados: se han generado tres grupos, de los que uno corresponde a las comunidades peninsulares, otro contiene únicamente Ceuta y Melilla y el último contiene únicamente las Islas Canarias. Lo más llamativo es que, pese a la cercanía de Ceuta con la península, el algoritmo no lo considera parte del mismo conjunto ya que utiliza la distancia de Hausdorff.



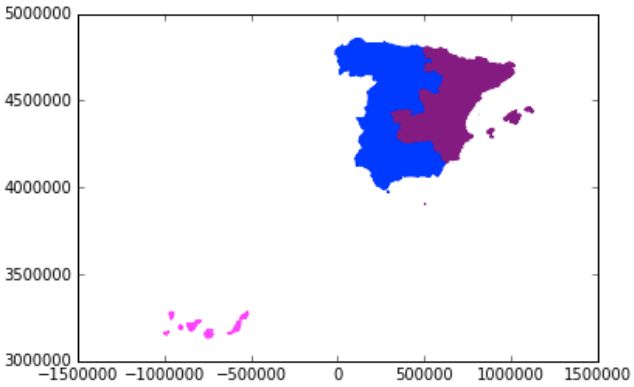
2.2 Ejecución 2

Algoritmo: DBSCAN (eps = 350000, minPts = 0).
Distancia: Euclídea.
Resultados: se han generado dos grupos, de los que uno contiene la península y las ciudades autónomas, y el otro contiene únicamente las Islas Canarias. A diferencia de la Ejecución 1, el uso de la distancia euclídea, que toma un punto central de cada dato como referencia, provoca que las ciudades autónomas no tengan su propio conjunto.



2.3 Ejecución 3

Algoritmo: K-Means (k = 3).
Distancia: Hausdorff.
Resultados: se han generado tres grupos a partir de la división del espacio en tres partes iguales, sin atender a la densidad de elementos en cada zona del espacio (concepto que K-Means no maneja).



2.4 Conclusiones

Lo más probable es que los resultados esperados por el usuario se parezcan más a los de las dos primeras ejecuciones. La razón vuelve a ser el concepto de densidad: en el contexto geoespacial, las regiones más densamente pobladas comparten más información entre sí.

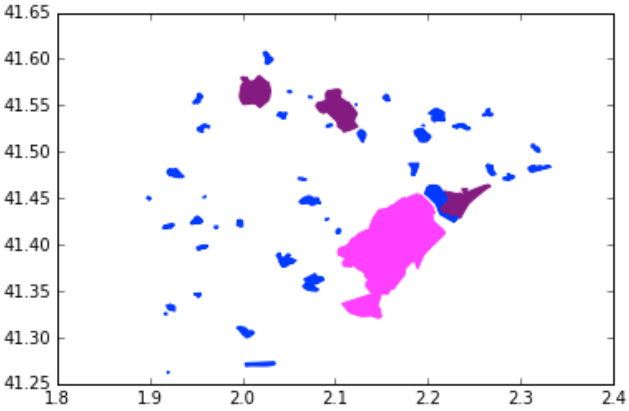
3. Experimento con la distancia combinada

Para probar la distancia combinada se han construido unos datos basados en localizaciones geográficas y datos numéricos. En concreto, se ha fusionado un conjunto formado por los polígonos de todas las localidades españolas, con los datos de población del censo del INE. El objetivo es agrupar ciudades próximas en ubicación y en población.

Del conjunto resultante se ha utilizado, por cuestiones de extensión y eficiencia, únicamente el área de Barcelona. Su elección se fundamenta en que existen dos ciudades muy próximas en localización y en población: Terrassa y Sabadell. El resultado esperado es que ambas formen su propio grupo.

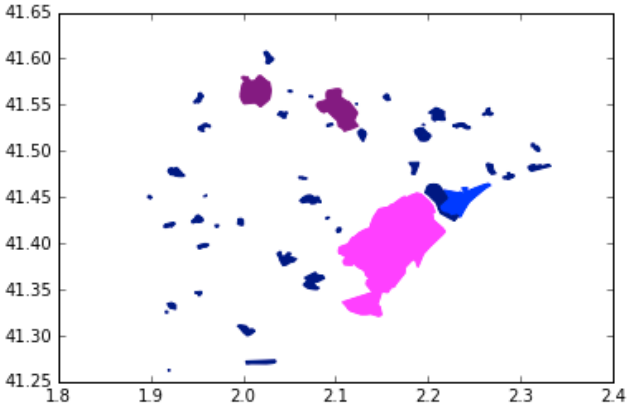
3.1 Ejecución 1

Algoritmo: DBSCAN (eps = 20000, minPts = 0)
Distancia: Combinada (Hausdorff · 0.6 + DiferenciaAbsoluta · 0.4).
Resultados: se han generado tres conjuntos: uno contiene únicamente la ciudad de Barcelona, otro contiene las ciudades de Terrassa, Sabadell y Badalona y el último agrupa al resto de poblaciones, independientemente de su ubicación. Dado que las distancias están en grados sexagesimales, son mucho más pequeñas que las diferencias en número de habitantes, que pueden alcanzar los miles de habitantes. En conclusión se está sobrestimando el número de habitantes.



3.2 Ejecución 2

Algoritmo: DBSCAN (eps = 15000, minPts = 0)
Distancia: Combinada (Hausdorff · 111000 + DiferenciaAbsoluta · 0.1).
Resultados: se está aprovechando la existencia de pesos en la distancia combinada para regularizar los datos, es decir, pasarlos a escalas comparables entre sí. Se ha optado por hacer la conversión entre grados sexagesimales y metros (1º = 111km[16]). Ahora las escalas son similares y se ha conseguido separar el conjunto indeseado en dos: uno con Terrassa y Sabadell, y otro únicamente con Badalona. Sin embargo, sigue habiendo un gran grupo al que van a parar todas las poblaciones pequeñas, independientemente de su ubicación.



3.3 Conclusiones

Los pesos de la distancia combinada se han utilizado para regularizar los atributos de modo que tengan escalas comparables. De este modo se ha conseguido separar en dos el grupo problemático, demostrando el correcto funcionamiento de la distancia combinada.

[16] Se trata de una simplificación que asume que la tierra es esférica. Dado el pequeño tamaño del espacio de los datos en comparación con el planeta, es una aproximación asumible.

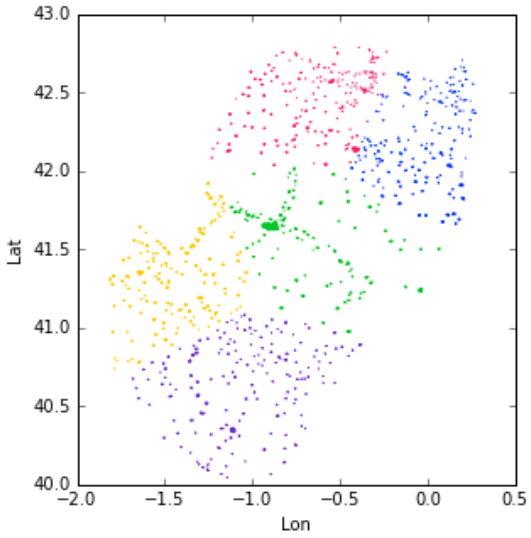
Sin embargo, la existencia de un grupo que aglomera a todas las poblaciones pequeñas independientemente de su localización demuestra que el problema no se ha corregido completamente. Se podrían seguir ajustando los parámetros del algoritmo y de la distancia combinada hasta conseguirlo, pero la verdadera solución a este tipo de problema es normalizar los datos de entrada como paso previo a la ejecución del algoritmo.

4. Experimento con un caso real: divisiones provinciales

Se han utilizado los mismos datos que en el experimento 3, pero tomando un subconjunto que contiene las localidades de la Comunidad Autónoma de Aragón. El objetivo de esta prueba es hacer una nueva división provincial en Aragón, es decir, todas las localidades deben formar parte de algún grupo. Por esta razón se ha descartado DBSCAN para resolverlo, y en su lugar se va a ejecutar K-Means++.

4.1 Ejecución 1

Algoritmo: K-Means++ (k = 5).
Distancia: Hausdorff.
Resultados: se han indicado 5 provincias al algoritmo. Estas pueden cambiar de una ejecución a otra, dada la naturaleza aleatoria de los algoritmos de K-Means. Sin embargo, en general los resultados se parecen mucho a los indicados en esta ejecución.



4.2 Conclusiones

Los resultados obtenidos no resultan destacables por su complejidad sino porque muestran un ejemplo de uso del algoritmo K-Means en el contexto geoespacial. Al determinar las provincias se busca dividir un espacio en un número determinado de zonas, y se prefiere que éstas tengan dimensiones similares. K-Means se adapta perfectamente a este problema que DBSCAN no podría resolver.

Anexo III: Planificación

A continuación se explican aquellos detalles de planificación que no se han indicado en el capítulo 7.

1. Hitos del proyecto

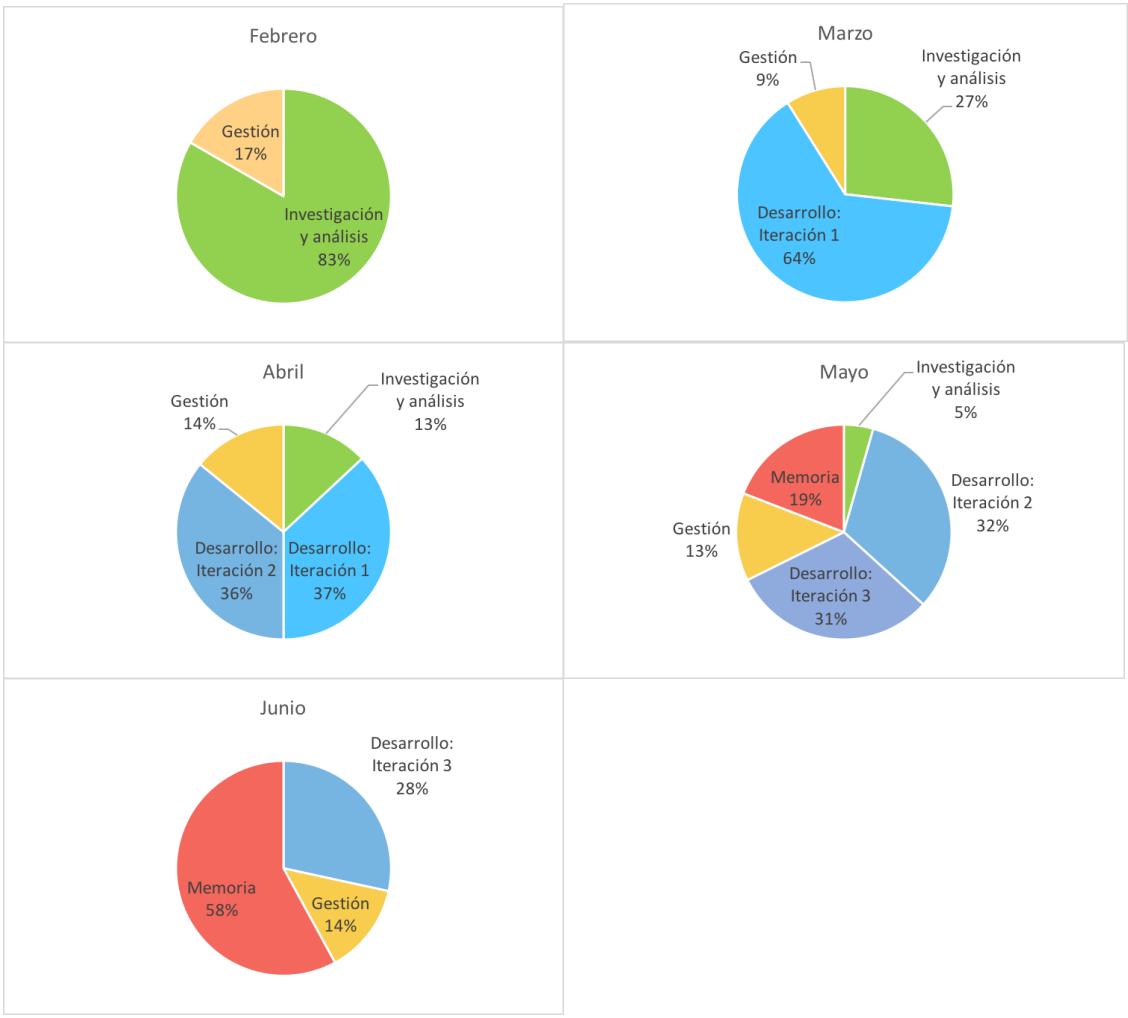
- **15 de febrero**: reunión de inicio del trabajo.
- **7 de marzo**: reunión de control de la etapa de análisis e investigación previa. Comienza la Iteración 1 – algoritmos de agrupamiento.
- **21 de marzo**: reunión de control.
- **4 de abril**: comienza la Iteración 2 – anotación de conjuntos.
- **6 de mayo**: comienza la Iteración 3 – visualización de resultados.
- **11 de mayo**: comienza el desarrollo de la memoria.
- **17 de junio**: termina el desarrollo de la memoria.

2. Análisis de esfuerzos

En el capítulo 7 se han indicado las horas dedicadas a cada tarea principal. Aquí se va a desgranar esa información en mayor detalle, empezando por el reparto de horas por meses:

- Febrero: mes de inicio del proyecto, dedicado principalmente a investigación y tareas iniciales de gestión. **Total**: 18 horas.
    - Investigación y análisis: 15 horas.
    - Gestión: 3 horas.
- Marzo: mes dedicado a la 1ª iteración del proyecto. **Total**: 56 horas.
    - Investigación y análisis: 15 horas.
    - Desarrollo – Iteración 1: 36 horas.
    - Gestión: 5 horas.
- Abril: mes con mayor carga de trabajo por estar dedicado a la finalización de la 1ª iteración y a la realización de la 2ª iteración. **Total**: 92 horas.
    - Investigación y análisis: 12 horas.
    - Desarrollo – Iteración 1: 34 horas.
    - Desarrollo – Iteración 2: 33 horas.
    - Gestión: 13 horas.
- Mayo: mes dedicado a desarrollar las bases de la 3ª iteración y a comenzar la memoria. **Total**: 68 horas.
    - Investigación y análisis: 3 horas.
    - Desarrollo – Iteración 2: 22 horas.
    - Desarrollo – Iteración 3: 21 horas.
    - Gestión: 9 horas.
    - Memoria: 13 horas.
- Junio: mes dedicado a finalizar la 3ª iteración y la memoria. **Total**: 88 horas.
    - Desarrollo – Iteración 3: 25 horas.
    - Gestión: 12 horas.
    - Memoria: 51 horas.

A continuación aparece el reparto de horas en porcentajes para cada mes.



De la observación de estos datos se extrae que, durante los meses centrales, las tareas principales fueron de desarrollo. La investigación y análisis comienza siendo la actividad principal pero va disminuyendo según el proyecto avanza, y sus horas las pasa a ocupar el desarrollo de la memoria. En cuanto al porcentaje de horas dedicado a la gestión, este se mantiene estable durante todo el proyecto.