Algoritmo de agrupamiento con traslape en flujos de datos

por

Argenis Aldair Aroche Villarruel

Tesis sometida como requisito parcial para obtener el grado de

Maestro en Ciencias en el Área de Ciencias Computacionales

por el

Instituto Nacional de Astrofísica, Óptica y Electrónica

Noviembre, 2015

Tonantzintla, Puebla

Supervisada por:

José Francisco Martínez Trinidad

Jesús Ariel Carrasco Ochoa

Airel Pérez Suárez

Coordinación de Ciencias Computacionales

INAOE

©INAOE 2015

Todos los derechos reservados

El autor(a) otorga al INAOE permiso para la reproducción y distribución del presente documento

Índice general

Agradecimientos xi

Resumen xiii

Abstract xv

1. Introducción 1

1.1. Introducción 1

1.2. Problemática 3

1.3. Objetivos 6

1.4. Organización de la tesis 7

2. Conceptos preliminares 9

2.1. Flujos de datos 9

2.2. Modelos de ventaneo 12

3. Trabajo Relacionado 15

3.1. Algoritmos de Agrupamiento para flujos de datos 15

3.2. Algoritmos de Agrupamiento con traslape incrementales 21

4. Algoritmo propuesto **27**

4.1. Modelo de ventaneo utilizado 27

4.2. Adaptación de estrategias de Agrupamiento 29

4.2.1. Fase *Online* 29

4.2.2. Fase *Offline* 38

5. Análisis Experimental **47**

5.1. Medida de evaluación 47

5.2. Conjuntos de datos de prueba y parámetros 50

5.3. Comparación en calidad 53

5.3.1. Mediamill 53

5.3.2. NUSWIDE-cVLAD+ 55

5.3.3. NUSWIDE-BoW 57

5.3.4. Prueba de significancia estadística 59

5.4. Comparación en tiempo de ejecución 61

6. Conclusiones y trabajo futuro **65**

6.1. Sumario 65

6.2. Conclusiones 66

6.3. Aportaciones 67

6.4. Trabajo futuro 67

6.5. Publicaciones 68

Índice de figuras

3.1. Agrupamiento basado en densidad, utilizando *grids*. Fuente: [Tu and Chen, 2009a] 18

3.2. micro-clusters obtenidos con DenStream. Fuente: [Cao et al., 2006] . . 21

4.1. Resultado obtenido después de aplicar DenStream con la expresión 4.2 para calcular el radio de los *micro-clusters*. 30

4.2. Resultado obtenido después de aplicar DenStream con la expresión 4.3 para calcular el radio de los *micro-clusters*. 32

4.3. Conjunto de datos con evolución en la distribución de objetos, cada imagen representa una ventana de 20,000 objetos distribuidos en 3 grupos. 32

4.4. Proceso de adición de nuevas aristas en el grafo. 41

5.1. Resultados obtenidos en la partición 1 y 2 del conjunto Mediamill utilizando FBcubed, DClustR (cuadros) y OCDS (líneas). 54

5.2. Resultados obtenidos en la partición 3 y 4 del conjunto Mediamill utilizando FBcubed, DClustR (cuadros) y OCDS (líneas). 54

5.3. Resultados obtenidos en la partición 5 del conjunto Mediamill utilizando FBcubed, DClustR (cuadros) y OCDS (líneas). 55

5.4. Resultados obtenidos en la partición 1 y 2 del conjunto NUSWIDE-cVLAD+ utilizando FBcubed, DClustR (cuadros) y OCDS (líneas). . 56

5.5. Resultados obtenidos en la partición 3 y 4 del conjunto NUSWIDE-cVLAD+ utilizando FBcubed, DClustR (cuadros) y OCDS (líneas). . 56

5.6. Resultados obtenidos en la partición 5 del conjunto NUSWIDE-cVLAD+ utilizando FBcubed, DClustR (cuadros) y OCDS (líneas). 57

5.7. Resultados obtenidos en la partición 1 y 2 del conjunto NUSWIDE-BoW utilizando FBcubed, DClustR (cuadros) y OCDS (líneas). . . . 58

5.8. Resultados obtenidos en la partición 3 y 4 del conjunto NUSWIDE-BoW utilizando FBcubed, DClustR (cuadros) y OCDS (líneas). . . . 58

5.9. Resultados obtenidos en la partición 5 del conjunto NUSWIDE-BoW utilizando FBcubed, DClustR (cuadros) y OCDS (líneas). 59

5.10. Tiempo de ejecución en el conjunto de datos Mediamill, DClustR (cuadros) y OCDS (triángulos). 62

5.11. Tiempo de ejecución en el conjunto de datos NUSWIDEcVLAD+, DClustR (cuadros) y OCDS (triángulos). 62

5.12. Tiempo de ejecución en el conjunto de datos NUSWIDEBoW, DClustR (cuadros) y OCDS (triángulos). 63

Índice de tablas

4.1. Resultados obtenidos con DenStream utilizando un valor de $\lambda = 0.25$ 33

4.2. Resultados obtenidos con DenStream utilizando un valor de $\lambda = 0.5$. 34

4.3. Resultados obtenidos con la estrategia de eliminación de *micro-clusters* propuesta utilizando un valor de $\lambda = 0.25$ 37

4.4. Traslape obtenido utilizando OClustR como algoritmo para la fase *Offline*. 40

5.1. Conjuntos de datos utilizados para la evaluación de los algoritmos . . 51

5.2. Promedio de calidad obtenido en cada partición con DClustR y OCDS 60

5.3. Velocidad de OCDS comparado con DClustR. 63

Agradecimientos

He concluido este trabajo de tesis no solo gracias a mi esfuerzo, sino también gracias al apoyo de varias personas, quienes me brindaron sus consejos y estuvieron al pendiente de mi avance durante mi desarrollo de tesis y durante mi formación como maestro en ciencias.

A mis padres por haberme forjado como la persona que soy en la actualidad; muchos de mis logros se los debo a ustedes. Gracias madre y padre.

A mis asesores el Dr. Francisco Martínez Trinidad y el Dr. Ariel Carrasco Ochoa, por su apoyo para el desarrollo de este trabajo de investigación y por la paciencia que tuvieron en las revisiones del presente documento de tesis. Gracias por ayudarme en mi formación como investigador.

Agradezco también a mi asesor por parte del CENATAV, el Dr. Airel Pérez Suárez, por su ayuda durante el desarrollo de esta tesis y por su apoyo durante mi estancia en Cuba. También quiero extender un agradecimiento al Centro de Aplicaciones de Tecnologías de Avanzada (CENATAV), en especial a los colaboradores del departamento de minería de datos que fue en donde realice mi estancia de investigación.

Quiero agradecer a mis sinodales por sus comentarios para mejorar el presente trabajo de tesis y por el tiempo que dedicaron a la revisión de esta tesis.

Agradezco al CONACyT por la beca que me proporciono durante mis estu-

dios de maestría en el INAOE y por la beca mixta para realizar una estancia de investigación en Cuba.

Sin duda alguna sé que he omitido a varias personas que me apoyaron de alguna manera para poder concluir con mis estudios en el INAOE, y me disculpo por no tener sus nombres escritos en estas hojas, pero eso no les resta importancia y les agradezco por todo.

Resumen

Debido al auge que han tenido las tecnologías de la información en los últimos años, en diferentes aplicaciones se generan continuamente una gran cantidad de datos (flujos de datos). Estos flujos de datos pueden ser generados por sistemas de redes de comunicaciones, redes de sensores remotos, experimentos científicos, Internet y otro gran número de entornos de carácter dinámico. El agrupamiento es una de las técnicas que se ha utilizado para extraer conocimiento a partir de los flujos de datos, convirtiéndose de esta forma en un tema de interés activo.

En diversos problemas prácticos es común encontrar flujos de datos en donde se requiere construir agrupamientos con traslape, algunos ejemplos de este tipo de problemas son el análisis de flujos de noticias, detección de intrusos en redes, aplicaciones para la detección de fraudes bancarios, entre otros. Sin embargo, los algoritmos de agrupamiento propuestos en la literatura para flujos de datos no contemplan la obtención de grupos con traslape.

Debido a lo antes mencionado, en este trabajo de tesis se propone un algoritmo de agrupamiento con traslape para flujos de datos, este algoritmo cumple con una serie de restricciones, las cuales todo algoritmo de agrupamiento en flujos de datos debe cumplir. De los experimentos realizados en conjuntos de datos con traslape, se puede concluir que el algoritmo propuesto es una mejor alternativa para procesar flujos de datos en aplicaciones donde es común que los objetos pertenezcan a más de un grupo.

Abstract

Due to the boom that information technologies have had in recent years, in different applications a lot of data are continuously generated (data streams). These data streams can be generated by network systems, remote sensors networks, scientific experiments, Internet and many others dynamic environments. Clustering is one of the techniques used to extract knowledge from data streams, in this way becoming a topic of active interest.

There are several problems dealing with data streams, like for instance new stream analysis, intrusion detection and bank fraud detection, among others, in which it is desirable to build an overlapping clustering. However, the data stream clustering algorithms proposed in the literature do not allow to build overlapping clusters.

Due to the above mentioned, in this thesis an overlapping clustering algorithm for data streams is proposed, this algorithm satisfies all the constraints, that any clustering algorithm for data streams must satisfy. From the experiments on overlapping datasets, we can conclude that OCDS is a suitable alternative for processing data streams in applications where it is common finding objects belonging to more than one cluster.

Capítulo 1

Introducción

En este capítulo se describe el tema principal de esta tesis, el cual es el agrupamiento en flujos de datos. Se empieza mencionando algunas aplicaciones en las que se han utilizado los algoritmos de agrupamiento en flujos de datos, así como las principales restricciones que tienen dichos algoritmos. Posteriormente, se introduce la problemática que se abordará en el marco de esta tesis, en donde se resalta el hecho de que los algoritmos de agrupamiento para flujos de datos actualmente reportados en la literatura, no contemplan el traslape entre los grupos construidos. A partir de la problemática, se definen los objetivos que se deben cumplir en el desarrollo de esta tesis. Por último, se describe la organización del presente documento.

1.1 Introducción

Con el auge que han tenido las tecnologías de la información en los últimos años, en diferentes aplicaciones se generan una gran cantidad de flujos de datos. Estos flujos de datos pueden ser generados por sistemas de redes de comunicaciones, redes de sensores remotos, experimentos científicos, Internet y otro gran número de entornos de carácter dinámico. Un flujo de datos es considerado como una secuencia ordenada de datos que llegan de manera continua en cada instante de tiempo. En el contexto de esta tesis se entenderá por dato un objeto descrito por un conjunto de atributos o características.

Como se ha mencionado, en los últimos años, el análisis de flujos de datos se ha convertido en un tema de investigación relevante en varios campos de las ciencias computacionales y su aplicación puede verse en distintas aplicaciones como: sistemas de detección de intrusos en redes [Cao et al., 2006], observación de entornos [Ruiz et al., 2009], sistemas médicos [Plant et al., 2010, Mete et al., 2011], análisis de la bolsa de valores [Yang et al., 2011], análisis de redes sociales [Lee, 2012] y aplicaciones de seguimiento de objetos en video [Yu et al., 2013], entre otras.

El agrupamiento es una de las técnicas que se ha utilizado para extraer conocimiento a partir de los flujos de datos, convirtiéndose de esta forma en un tema de interés activo. El agrupamiento, también conocido como clasificación no supervisada, es una técnica del Reconocimiento de Patrones que consiste en organizar en grupos un conjunto de objetos, en nuestro caso descritos en términos de un conjunto de atributos o características, de forma tal que objetos de un mismo grupo sean más similares entre sí que objetos de grupos distintos.

Dada la naturaleza de un flujo de datos, existe una serie de restricciones que un algoritmo de agrupamiento debe cumplir para considerarlo capaz de procesar un flujo de datos [Amini et al., 2014b]. Estas restricciones son: la cantidad de objetos en el flujo debe ser considerada infinita, el tiempo para procesar un solo objeto es limitado, la memoria es limitada, existen objetos que pueden ser ruido y puede existir evolución en la distribución de los objetos. Adicionalmente a estos requerimientos, también se deben tener en cuenta los problemas inherentes a los algoritmos de agrupamiento: no se sabe a priori el número de grupos que existen en el conjunto de objetos, los grupos pueden tener formas arbitrarias y el ruido puede afectar en el proceso de agrupamiento.

Para procesar los flujos de datos en las computadoras actuales comúnmente se sigue el modelo de ventaneo [Ng and Dash, 2010], este modelo consiste en retener cierta cantidad de información del flujo de datos en un buffer, el cual mantiene

ordenados los datos conforme se van agregando y les puede asignar pesos de acuerdo a ese orden.

A pesar de los buenos resultados obtenidos por los algoritmos de agrupamiento para flujos de datos que se han reportado en la literatura, estos todavia tienen un conjunto delimitaciones que reducen su aplicación en problemas reales. A continuación se describen estos problemas.

1.2 PROBLEMÁTICA

En la literatura se han reportado diversos algoritmos de agrupamiento para flujos de datos [Cao et al., 2006, Ruiz et al., 2009, Aggarwal et al., 2005, Aggarwal et al., 2003, Kranen et al., 2011, Tu and Chen, 2009b, Guha et al., 2002, Dang et al., 2009, Wan et al., 2009]. Estos algoritmos son comúnmente clasificados en cinco categorías [Han and Kamber, 2006]: particionales, jerárquicos, basados en densidad, basados en *grid* (cuadrícula) y basados en modelo.

Los algoritmos de agrupamiento basado en particiones organizan una colección de objetos en un número de particiones, donde cada partición representa a un grupo. Para construir los agrupamientos se utiliza comúnmente una función de distancia. De forma analoga al algoritmo k-means [MacQueen, 1967], este tipo de algoritmos encuentra agrupamientos con forma esférica y es sensible a los objetos que son ruido. Los algoritmos para flujos de datos, que son basados en particiones y que han resultado más citados en la literatura son STREAM [Guha et al., 2002] que está basado en las k-medianas [Kaufman and Rousseeuw, 1987] y CluStream [Aggarwal et al., 2003] que está basado en el k-means.

Los algoritmos de agrupamiento jerárquico hacen divisiones o uniones sucesivas de grupos en distintos niveles dentro de una jerarquía. Se han utilizado combinaciones de este tipo de algoritmos con otras técnicas, para mejorar la ca-

lidad de los agrupamientos. Como ejemplo se tiene el algoritmo de agrupamiento ClusTree [Kranen et al., 2011] que está basado en el algoritmo jerárquico BIRCH [Zhang et al., 1996].

Los algoritmos de agrupamiento basados en densidad, construyen grupos como áreas densas que están separadas por regiones dispersas. La idea más comúnmente utilizada es hacer crecer un grupo hasta que la densidad (número de objetos) en el vecindario exceda cierto umbral. Este tipo de agrupamiento permite el manejo de objetos que son ruido, además es capaz de descubrir grupos con formas arbitrarias. El agrupamiento basado en densidad es adecuado para flujos de datos con evolución en la distribución de objetos, ya que el número y la forma de los grupos cambian constantemente. El algoritmo basado en densidad más citado es DenStream [Cao et al., 2006], el cual está basado en el algoritmo DBSCAN [Ester et al., 1996]. Del algoritmo DenStream se derivan varias extensiones [Ruiz et al., 2009, Lin and Lin, 2009, Ntoutsi et al., 2012, Forestiero et al., 2013, Hassani et al., 2012, Amini et al., 2014a].

Los algoritmos de agrupamiento basados en *grid* particionan el espacio de los objetos en un cierto número de celdas, las cuales forman un *grid*. Un *grid* (cuadrícula) es una división de cada atributo de los objetos en intervalos de igual longitud, dichas divisiones parten el espacio m-dimensional en *grids* de igual tamaño, y en cada *grid* se tiene un represente. Los objetos se asignan a cada *grid* utilizando una función. Se ha reportado que los algoritmos de este tipo son rápidos en tiempo de ejecución, ya que no dependen del número de objetos. Algunos de los algoritmos de agrupamiento para flujos de datos basados en *grid* pueden caer dentro de la categoría de los basados en densidad, entre ellos se encuentran D-Stream [Tu and Chen, 2009b] y MR-Stream [Wan et al., 2009], que además son de los algoritmos más citados en la literatura.

Los algoritmos de agrupamiento basados en modelo optimizan una función de aptitud, para lo cual utiliza el conjunto de objetos y algún modelo matemático a

optimizar, como ejemplo de este tipo de algoritmos se tiene el algoritmo SWEM [Dang et al., 2009] que es una extensión de EM (Expectation Maximization) [Dempster et al., 1977] para el agrupamiento en flujos de datos.

Recientemente, en [Amini et al., 2014b], se ha reportado un estudio de algoritmos de agrupamiento para flujos de datos basados en densidad. En dicho trabajo se argumenta que debido a que en un flujo de datos la cantidad de grupos y la forma de los mismos está en constante cambio. Los algoritmos de agrupamiento basados en densidad constituyen una buena alternativa para trabajar en flujos de datos.

En diversos problemas prácticos es común encontrar flujos de datos en donde se requiere construir agrupamientos con traslape, algunos ejemplos de este tipo de problemas son el análisis de flujos de noticias [Pons-Porrata et al., 2002], detección de intrusos en redes, aplicaciones para la detección de fraudes telefónicos, bancarios, entre otros. Sin embargo, los algoritmos de agrupamiento propuestos en la literatura para flujos de datos [Cao et al., 2006, Guha et al., 2002, Aggarwal et al., 2003, Aggarwal et al., 2005, Zhang et al., 2009, Lu et al., 2005, Gao et al., 2005, Bhatnagar and Kaur, 2007, Aggarwal and Yu, 2008, Lühr and Lazarescu, 2009] no contemplan la obtención de grupos con traslape.

En la literatura se han propuesto algoritmos de agrupamiento incrementales, en los cuales es posible procesar nuevos objetos después de que se haya construido el agrupamiento. Estos algoritmos utilizan el agrupamiento ya formado para determinar a qué grupo pertenece un nuevo objeto o para decidir si es necesario formar un nuevo grupo o dividir un grupo previamente creado, sin tener que agrupar todo el conjunto de datos desde el inicio. De igual forma, existen algoritmos de agrupamiento incrementales que permiten formar grupos con traslape, por lo que podría pensarse que estos algoritmos construyen una alternativa para construir agrupamientos con traslape en flujos de datos. No obstante, el problema con los algoritmos de agrupamiento incrementales es que necesitan mantener todos los objetos en memoria para

agrupar nuevos objetos, lo cual hace que tengan un alto costo computacional. Adicionalmente, debido a que la cantidad de memoria es limitada, estos algoritmos no son capaces de procesar un flujo de datos, ya que en el caso de los flujos de datos se parte del supuesto de que el flujo de datos es infinito y el tiempo para procesar cada objeto nuevo es limitado ya que los objetos no se pueden mantener indefinidamente en la memoria.

Con base en lo mencionado anteriormente, el problema que se aborda en esta tesis es el desarrollo de un algoritmo de agrupamiento para flujos de datos que permita la obtención de grupos con traslape. Contar con un algoritmo de este tipo para flujos de datos permitiría obtener mejores respuestas en aquellos problemas de flujos de datos en los cuales, la propia naturaleza del problema impone obtener grupos con traslape.

1.3 Objetivos

Con base en lo planteado anteriormente, la presente tesis de maestría tiene los siguientes objetivos:

Objetivo general

Desarrollar un algoritmo de agrupamiento para flujos de datos, que permita formar grupos con traslape, que sea más rápido y mantenga similar calidad que los algoritmos incrementales que se han desarrollado para el agrupamiento con traslape.

Objetivos específicos

1. Adaptar una estrategia de paginado para el manejo de flujos de datos, que sea adecuada para el agrupamiento con traslape.
2. Adaptar una estrategia de agrupamiento con traslape para la estrategia de paginado propuesta en 1.

3. Crear un algoritmo de agrupamiento con traslape en flujos de datos, combinando las estrategias adaptadas en 1 y 2.

1.4 Organización de la tesis

El contenido de este documento se encuentra organizado en 5 capítulos. Además del presente capítulo, en el capítulo 2, se describen algunos conceptos necesarios para comprender cómo se representan y procesan los flujos de datos. También se describen algunas de las estrategias de ventaneo que son comúnmente utilizadas para procesar flujos de datos.

En el capítulo 3, se describen los algoritmos propuestos en la literatura que se encuentran relacionados con la presente tesis. Se describen algoritmos de agrupamiento que procesan flujos de datos, así como los algoritmos de agrupamiento con traslape incrementales, contra los cuales se comparará la calidad de los agrupamientos del algoritmo propuesto en esta tesis.

En el capítulo 4, se introduce el algoritmo propuesto para el agrupamiento en flujos de datos, que permite formar agrupamientos con traslape. Primero se introduce la fase *online* del algoritmo y posteriormente se describe la fase *offline*.

En el capítulo 5, se evalúa el algoritmo propuesto en términos de eficiencia y eficacia, en conjuntos de datos reales.

Finalmente, en el capítulo 6, se presentan las conclusiones y el trabajo a futuro que se derivan de la presente tesis.

Capítulo 2

Conceptos preliminares

En este capítulo se introducen los conceptos básicos necesarios para entender los capítulos 3 y 4. Primero, se describe el concepto de flujo de datos y se discute sobre los problemas que se tienen que considerar para su análisis. Adicionalmente, se mencionan algunas de las representaciones compactas comúnmente utilizadas para hacer un procesamiento rápido para el agrupamiento en flujos de datos. Posteriormente, se mencionan algunos de los modelos de ventaneo comúnmente utilizados para el manejo de los flujos de datos.

2.1 Flujos de datos

En esta tesis se utilizará la definición de flujo de datos dada en [Giraldo et al., 2013], la cual dice que un flujo de datos es una secuencia continua, potencialmente infinita, de objetos $x_1, x_2, ..., x_k, ...$ etiquetados con tiempos $T_1, T_2, ..., T_k, ...$, donde cada objeto x_i del flujo se describe por m características, y se denota por $x_i = (x_{i1}, x_{i2}, ..., x_{im})$.

Un flujo de datos comúnmente contiene una gran cantidad de objetos y éstos a su vez pueden estar descritos por una gran cantidad de atributos, sin embargo, éstos no son los únicos problemas. También puede haber ruido en el flujo de datos y es posible que en un instante de tiempo llegue más de un objeto para ser analizado.

Producto de las características de los flujos de datos, resulta poco práctico almacenar todos los objetos de un flujo de datos para posteriormente hacer un análisis de los mismos. Algunas de las consideraciones que se deben tener en cuenta en un flujo de datos son:

- No se tiene control sobre el orden en que llegan los objetos, por lo que el acceso a los mismos para su procesamiento no puede ser de manera aleatoria.
- Los flujos de datos son potencialmente infinitos.
- Los objetos que han sido procesados pueden ser eliminados, o almacenados en el mejor de los casos. Sin embargo, la recuperación de objetos ya procesados no es sencilla, partiendo de que, para realizarla, los objetos deben estar en memoria y esta se considera pequeña en relación al tamaño del flujo de datos.

Una de las formas en que comúnmente se procesan los flujos de datos es utilizando una representación compacta de un conjunto de objetos. Dicha representación hace que sea posible procesar los objetos una sola vez, es decir, una vez que se obtiene un objeto del flujo de datos, éste se procesa y no es necesario volver a acceder a él. En el agrupamiento, la representación compacta ayuda a determinar a qué grupo pertenece un nuevo objeto. Las representaciones compactas comúnmente utilizadas para representar conjuntos de objetos en el agrupamiento en flujos de datos basado en densidad, son los *micro-clusters* y los *grid* [Amini et al., 2014b]. Sin embargo, en la literatura se ha reportado que los *grid* tienen problemas para procesar objetos con alta dimensionalidad [Tan et al., 2006], por lo cual, en el presente trabajo se optó por utilizar los *micro-clusters* como representación compacta para conjuntos de objetos.

Un *micro-cluster* [Aggarwal et al., 2003] para un conjunto de objetos $x_1, x_2, ..., x_n$, con etiquetas de tiempo $T_1, T_2, ..., T_n$, descritos por m atributos está definido por la tupla $(CF2^x, CF1^x, CF2^t, CF1^t, n)$, donde $CF2^x$ y $CF1^x$

corresponden a un vector de m elementos cada uno.

- $CF2^x$: contiene la suma de los cuadrados de cada uno de los atributos de los n objetos que se encuentran en el *micro-cluster*, por lo que si los objetos tienen dimensión m, $CF2^x$ también tiene dimensión m. El j-ésimo atributo de $CF2^x$ es:

$$CF2_j^x = \sum_{i=1}^{n} (x_{ij})^2$$

- $CF1^x$: contiene la suma de cada uno de los atributos de los n objetos que se encuentran en el *micro-cluster*, por lo que si los objetos tienen dimensión m, $CF1^x$ también tiene dimensión m. El j-ésimo atributo de $CF1^x$ es:

$$CF1_j^x = \sum_{i=1}^{n} x_{ij}$$

- $CF2^t$: es la suma de los cuadrados de las etiquetas de tiempo $T_1, T_2, ..., T_n$ con $T_i \in \mathbb{N}$.

- $CF1^t$: es la suma de las etiquetas de tiempo $T_1, T_2, ..., T_n$.

- n: es el número de objetos.

Los *micro-clusters* son utilizados para representar un conjunto de objetos y de esta manera evitar almacenar todos los objetos del flujo de datos en memoria. Finalmente el agrupamiento se realiza sobre el conjunto de *micro-clusters* resultante del flujo de datos.

Del *micro-cluster* se deriva el concepto de *core-micro-cluster*, el cual es utilizado en el algoritmo DenStream y todas sus variantes. El *core-micro-cluster* utiliza una función de desvanecimiento, para dar un peso en relación a la antigüedad que tienen dichos *micro-clusters* en el flujo de datos. La función de desvanecimiento utilizada en los *core-micro-clusters* es $f(t) = 2^{-\lambda t}$, donde $\lambda > 0$.

Un *core-micro-cluster* [Cao et al., 2006] en un tiempo t para un conjunto de objetos $x_1, x_2, ..., x_n$, con etiquetas de tiempo $T_1, T_2, ..., T_n$, descritos por m atributos está definido como una tupla $CMC(w, c, r)$:

- $w = \sum_{i=1}^{n} f(t - T_i)$
- $c = (c_1, c_2, ..., c_m)$, donde $c_j = \frac{\sum_{i=1}^{n} f(t-T_i)x_{ij}}{w}$
- $r = \frac{\sum_{i=1}^{n} f(t-T_i)dist(x_i,c)}{w}$, donde $dist(x_i, c)$ es la distancia Euclidiana entre el objeto x_i y el centro c

Se debe tener en cuenta que el peso w debe ser mayor o igual que un umbral μ, el cual determina la cantidad de objetos, ponderados por la función de desvanecimiento, que debe tener un *micro-cluster* para ser considerado un *core-micro-cluster*. Además, el radio r debe ser menor o igual que un umbral ϵ. Este umbral determina el tamaño máximo que puede alcanzar el radio del *micro-cluster*; de esta manera, se trata de no incluir ruido en los *core-micro-clusters*.

Ya que en esta tesis se aborda el problema del agrupamiento se define un grupo como un conjunto de objetos obtenido a partir de algún algoritmo de agrupamiento. Un agrupamiento es el conjunto de grupos obtenido una vez que todos los objetos fueron agrupados. Una clase es un conjunto de objetos etiquetado por expertos en el área a la que corresponde el conjunto de datos.

2.2 Modelos de ventaneo

Los modelos de ventaneo constituyen uno de los métodos más ampliamente utilizados en el procesamiento de flujos de datos. Con el objetivo de reducir el espacio de procesamiento, los modelos de ventaneo son comúnmente empleados para las consultas que se hacen cada determinado tiempo en los flujos de datos, ya sea para hacer una evaluación del agrupamiento o, en una aplicación real, cuando el usuario

requiere de una respuesta por parte del sistema. En lugar de analizar un flujo de datos infinito, los modelos de ventaneo permiten analizar subconjuntos finitos del flujo de datos. Varios modelos de ventaneo de tiempo han sido usados en la literatura, siendo los siguientes los más representativos.

Modelo de ventaneo de punto de referencia

En el modelo de punto de referencia [Ng and Dash, 2010], el espacio de búsqueda y/o procesamiento es un intervalo de tiempo que se extiende desde un tiempo inicial l hasta el momento j, siendo j el tiempo actual en el flujo de datos. Ventanas sucesivas compartirán el mismo tiempo inicial e irán creciendo con el progreso del tiempo. Debido al aumento de la cantidad de objetos en el tiempo y la utilización de todo el histórico del flujo de datos; este modelo es poco aplicable, ya que se mantienen todos los objetos en memoria y si el flujo de datos es muy grande, es imposible procesarlo.

Modelo de ventaneo deslizante

En el modelo de ventaneo deslizante [Ng and Dash, 2010] solo la parte del flujo de datos que está acotada por una ventana de dimensión fija, será almacenada y procesada. En este modelo, la ventana se desplazará según fluyan los objetos en el tiempo. Este tipo de ventanas sigue un comportamiento similar a una estructura de datos tipo FIFO (*first in, first out*), donde una vez procesado e insertado en la ventana el elemento j, otro elemento $i = (j - w)$ es descartado de la ventana, siendo w el ancho de la ventana deslizante.

Modelo de ventaneo de desvanecimiento

En el modelo de ventaneo de desvanecimiento [Ng and Dash, 2010] el tamaño de ventana varía en dependencia de una función de desvanecimiento definida. En este modelo los objetos se *desvanecen* con el tiempo perdiendo así relevancia hasta llegado el punto en que son removidos de la ventana. La función de desvanecimiento

permite dar más relevancia a los objetos más recientes en el flujo; convirtiéndolo en un modelo adecuado en aplicaciones donde el efecto de los datos históricos en los resultados del procesamiento disminuya con el transcurso del tiempo. La función de desvanecimiento exponencial $f(t) = 2^{-\lambda t}$, donde $\lambda > 0$, es comúnmente utilizada en aplicaciones en donde el tiempo influye y es deseable dar menor importancia a la información con mayor antigüedad. Esta función es utilizada en la mayoría de los algoritmos de agrupamiento para flujos de datos basados en densidad y que utilizan micro-clusters.

Capítulo 3

Trabajo Relacionado

En este capítulo se describen los algoritmos de agrupamiento para flujos de datos y los algoritmos de agrupamiento con traslape más relevantes propuestos en la literatura. Dentro de los algoritmos de agrupamiento para flujos de datos se describe más detalladamente el algoritmo DenStream [Cao et al., 2006], ya que es el que ha reportado los mejores resultados. También se describe con detalle el algoritmo de agrupamiento con traslape incremental DClustR [Pérez-Suárez et al., 2013a] ya que ha reportado muy buenos resultados en este tipo de problemas y es el algoritmo contra el que se compará el algoritmo propuesto en esta tesis.

3.1 Algoritmos de Agrupamiento para flujos de datos

En la literatura se han propuesto varios algoritmos de agrupamiento para flujos de datos, la mayoría de ellos son extensiones de algoritmos de agrupamiento para conjuntos de datos estáticos. A continuación se describen los algoritmos para flujos de datos más citados en la literatura:

STREAM [Guha et al., 2002] es un algoritmo que parte del hecho de que el flujo de datos llega en subconjuntos de objetos $z_1, z_2, ..., z_n...$ donde cada subconjunto z_i se puede almacenar en memoria principal. El algoritmo funciona de la siguiente

manera: se agrupa el subconjunto z_i utilizando una variante del algoritmo k-medoids llamado LKM [Zhang et al., 1999], para obtener k objetos representativos de cada subconjunto. Posteriormente, se asigna un peso a cada objeto representativo, teniendo en cuenta los elementos pertenecientes a cada grupo. Luego, se eliminan los objetos de z_i y se almacenan los k objetos representativos ponderados. Por último, se aplica el algoritmo LKM sobre los objetos representativos obtenidos de cada $z_1, z_2, ..., z_n...$, de esta manera se obtienen los objetos representativos de todo el flujo de datos. En [Babcock et al., 2003] se extiende el algoritmo STREAM para que trabaje con ventanas deslizantes, de esta manera solo son procesados los n objetos más recientes del flujo de datos.

En [Aggarwal et al., 2003] se propone el algoritmo CluStream, retomando las ideas de BIRCH [Zhang et al., 1996]. En particular se extiende el concepto de CF (Cluster Feature) a partir del cual se propone el concepto de *micro-cluster*. Este algoritmo separa el proceso de agrupamiento en dos fases, en línea (*online*) y fuera de línea (*offline*). En un principio se crean k grupos utilizando k-means con un conjunto de objetos del flujo de datos. A partir de estos grupos se crean k *micro-clusters*, es decir, para cada grupo se calculan los vectores $CF1^x, CF2^x, CF1^t, CF2^t$, utilizando las fórmulas mostradas en el capítulo 2. En la fase *online* se toma el siguiente objeto x_i del flujo de datos y se asigna a un *micro-cluster*. Para hacer la asignación se calcula la distancia entre x_i y los centroides de los *micro-clusters* y se asigna x_i al *micro-cluster* con el centroide más cercano a él. La asignación se realiza si no se supera una cierta cantidad de objetos (esta cantidad se maneja mediante un umbral) que puede contener el *micro-cluster*. Si no es posible asignar x_i a un *micro-cluster*, se debe crear un nuevo *micro-cluster*. Para crear dicho *micro-cluster*, se utilizan los valores de x_i para calcular $CF1^x = x_i, CF2^x = SS(x_i)$, donde $SS(x_i) = (x_{i1}^2, x_{i2}^2, ..., x_{im}^2)$, $CF1^t = T_i, CF2^t = T_i^2$ y $n = 1$. Una vez asignados los objetos a los *micro-clusters* no es necesario mantenerlos en memoria.

En la fase *offline* se agrupan los *micro-clusters* para formar el agrupamien-

to final, para esto se utiliza el algoritmo k-means, utilizando los centroides de los *micro-clusters* como los objetos a agrupar. De esta manera, los *micro-clusters* con centroides que pertenecen al mismo grupo representan a todos los objetos del flujo de datos que pertenecen a dicho grupo.

En [Aggarwal et al., 2004] se propone el algoritmo HPStream, cuyo esquema general es similar al propuesto en el algoritmo CluStream; sin embargo, se hacen algunas modificaciones para tratar el problema de normalización de los datos y para utilizar solo un subconjunto de atributos para evaluar la distancia entre los objetos y los centroides de los grupos. Al inicio del agrupamiento del flujo de datos y con el fin de ponderar los diferentes atributos, se normaliza cada uno de ellos utilizando la desviación estándar de cada atributo. Al construir los *micro-clusters* iniciales utilizando k-means, para cada micro-cluster se calcula el peso del *micro-cluster*. Este peso está determinado por una función de desvanecimiento y es actualizado cada vez que se adicionan nuevos elementos al *micro-cluster*. Además, se determinan los atributos que serán utilizados, para realizar el cálculo de la distancia entre los nuevos objetos y el centroide, es decir, para cada grupo, solo un subconjunto de atributos es utilizado para el cálculo de la distancia.

Los algoritmos STREAM, CluStream y HPStream están basados en k-means o en variantes de k-means y por lo tanto, forman agrupamientos con formas esféricas. Esto hace que solo funcionen adecuadamente si se sabe a priori que el flujo de datos tiene grupos con dicha forma. Para solucionar este problema se ha propuesto utilizar algoritmos de agrupamiento basados en densidad, los cuales son capaces de formar grupos con formas arbitrarias.

DStream [Jia et al., 2008] es un algoritmo basado en *grids* y en densidad que retoma las ideas de CluStream; es decir, usa una fase *online* y una fase *offline*. En la fase *online* mapea los objetos de entrada a un *grid* conformado por celdas. Cada celda del *grid* tiene asociada una medida de densidad que es igual a

la suma de los objetos mapeados en la celda. Adicionalmente, la densidad de las celdas se desvanece continuamente de forma exponencial con un factor de desvanecimiento, el cuál es un parámetro de entrada. La fase *offline* forma un conjunto de grupos utilizando algún algoritmo de agrupamiento basado en *grids*, como pueden ser: STING [Wang et al., 1997], WaveCluster [Sheikholeslami et al., 1998] o CLIQUE [Agrawal et al., 1998]. Las celdas densas son diferenciadas de las dispersas mediante una función de umbral de densidad que depende de un parámetro, el factor de desvanecimiento y del número total de celdas en el *grid*. Un ejemplo del agrupamiento basado en *grids* se puede ver en la figura 3.1.

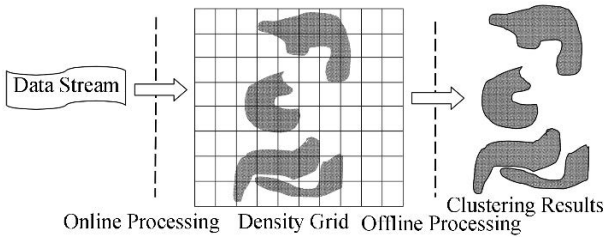


Figura 3.1: Agrupamiento basado en densidad, utilizando *grids*. Fuente: [Tu and Chen, 2009a]

La fase *online* obtiene nuevos objetos, mapea cada ojeto en un *grid* y actualiza el vector de características, el cual almacena un resumen de información del *grid*. Ya que las celdas más dispersas se desvanecen más rápido, DStream detecta periódicamente este tipo de celdas, y ya que las considera como ruido son eliminadas.

DenStream [Cao et al., 2006] es un algoritmo basado en densidad, que considera el problema de agrupamiento de un flujo de datos como un modelo de ventanas de desvanecimiento. En este modelo, el peso de cada objeto decrece exponencialmente en el tiempo t, utilizando una función de desvanecimiento. La función utilizada en DenStream es: $f(t) = 2^{-\lambda t}$, con $\lambda > 0$. Mientras más alto sea el valor de λ menor importancia se dará a los elementos más antiguos. DenStream utiliza el concepto de *core-micro-cluster* (cmc), para determinar cuándo un *micro-cluster* es cmc se utilizan dos conceptos importantes: *c-micro-cluster potencial* (mcp) y *micro-cluster*

aislado (mca).

Un mcp en un tiempo t para un grupo de objetos cercanos $x_1, x_2, ..., x_n$ con tiempos $T_1, T_2, ..., T_n$, está definido como la tupla $(\overline{CF^1}, \overline{CF^2}, w)$.

- $w = \sum_{i=1}^{n} f(t - T_i)$, $w \geq \beta\mu$, es el peso
- $\overline{CF^1} = \sum_{i=1}^{n} f(t - T_i)x_i$, es la suma lineal ponderada de los objetos.
- $\overline{CF^2} = \sum_{i=1}^{n} f(t - T_i)SS(x_i)$, donde $SS(x_i) = (x_{i1}^2, x_{i2}^2, ..., x_{im}^2)$.

A partir de los datos anteriores, se pueden calcular el centro y radio de un mcp, denotados como c y r, respectivamente, a través de las siguientes fórmulas:

$$c = \frac{\overline{CF^1}}{w} \tag{3.1}$$

$$r = \sqrt{\frac{|\overline{CF^2}|}{w} - (\frac{|\overline{CF^1}|}{w})^2} \tag{3.2}$$

Un mca en un tiempo t para un grupo de objetos cercanos $x_1, x_2, ..., x_n$ con tiempos $T_1, T_2, ..., T_n$, está definido como la tupla $(\overline{CF^1}, \overline{CF^2}, w, T_0)$. Donde $\overline{CF^1}$, $\overline{CF^2}$, w, centro y radio están definidos como en un mcp. $T_0 = T_i$ es el tiempo de creación del mca, que es utilizado para determinar el tiempo de vida del mca. Sin embargo, en este caso $w \leq \beta\mu$ ya que éste es el parámetro que determina cuándo un mca cambia a mcp.

Los mcp y mca se pueden actualizar en dependencia de cómo llega el flujo de datos. Por ejemplo, dado un mcp $c_p = (\overline{CF^1}, \overline{CF^2}, w)$. Si no se han agregado nuevos objetos al c_p por un intervalo de tiempo δt, entonces la representación de este *micro-cluster* sería $c_p = (2^{-\delta t}\overline{CF^1}, 2^{-\delta t}\overline{CF^2}, 2^{-\delta t}w)$. Por el contrario, si un objeto p es agregado a c_p, entonces $c_p = (\overline{CF^1} + p, \overline{CF^2} + SS(p), w + 1)$, donde $SS(p) = (p_1^2, p_2^2, ..., p_m^2)$. De manera similar sucede en el caso de los mca.

Cuando inicia la ejecución del algoritmo DenStream, éste tiene una fase de inicialización, en la que se toman k objetos y se aplica el algoritmo DBSCAN

[Ester et al., 1996]. Una vez que se agruparon los objetos, se calculan los mcp de manera similar que en CluStream, pero en este caso se verifican los umbrales que deben mantenerse para ser considerados como mcp. El algoritmo de agrupamiento se divide en dos fases: una fase en línea (*online*) para el mantenimiento de los *micro-clusters*, y una parte fuera de línea (*offline*), para generar el agrupamiento final; esta última fase se lleva a cabo cuando el usuario lo solicita.

Durante la fase *online* se obtiene un objeto x_i del flujo de datos, se calcula la distancia entre x_i y los centros de los mcp. Luego se calcula el radio r_j del mcp que fue más cercano a x_i pero incluyendo a x_i en el cálculo, si $r_j \leq \epsilon$ entonces se asigna x_i a ese mcp. En otro caso, se calcula la distancia entre x_i y los centros de los mca y se determina si el mca al que fue asignado cumple con las características para convertirse en mcp; es decir, se verifica si $w \geq \beta\mu$. Si x_i no se pudo agregar a ningún *micro-cluster* entonces se genera un nuevo mca a partir de x_i, creando los vectores $\overline{CF^1} = x_i$, $\overline{CF^2} = SS(x_i)$, donde $SS(x_i) = (x_{i1}^2, x_{i2}^2, ..., x_{im}^2)$ y con un peso $w = 1$ y tiempo de creación $T_0 = T_i$.

En la fase *offline*, para generar el agrupamiento final, se toman los centros de los mcp y se aplica una variante de DBSCAN que toma en cuenta el radio de los mcp y la distancia entre sus centros para determinar si son alcanzables de forma densa. Considérese dos mcp con centros c_p y c_q y con sus respectivos radios r_p y r_q. Se dice que c_p y c_q son alcanzables de forma densa si $d(c_p, c_q) \geq r_p + r_q$, donde $d(c_p, c_q)$ es la distancia Euclidiana entre c_p y c_q; todos los mcp que son alcanzables de manera densa forman un cluster. Un ejemplo de los micro-clusters resultantes a partir de un conjunto de objetos (parte izquierda de la figura 3.2) se puede observar en la parte derecha de la figura 3.2.

De los algoritmos analizados en esta sección se puede observar que ninguno contempla que los grupos puedan estar traslapados, por lo que cada algoritmo solo pueden formar grupos disjuntos. Sin embargo, actualmente hay plataformas como

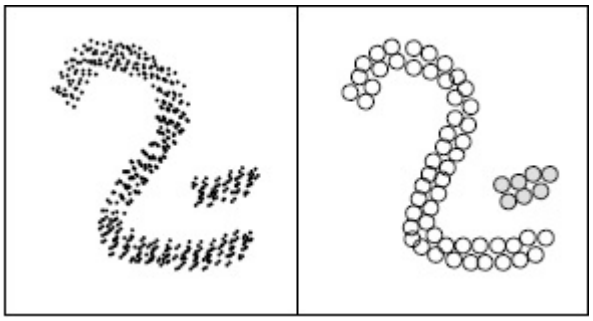


Figura 3.2: micro-clusters obtenidos con DenStream. Fuente: [Cao et al., 2006]

Twitter que genera muchos tuits cada minuto, es decir, un flujo de datos. Si se tiene en cuenta que una de las tareas donde es común que los grupos se encuentren traslapados es la detección de tópicos, ninguno de los algoritmos mencionados anteriormente podría generar una clasificación con traslape para esta tarea.

3.2 Algoritmos de Agrupamiento con traslape incrementales

En la literatura se han propuesto varios algoritmos de agrupamiento con traslape incrementales, entre los que mejores resultados han reportado, se encuentran: SHC [Hammouda and Kamel, 2004], Star [Aslam et al., 1998], ICSD [Pérez Suárez et al., 2009] y DClustR [Pérez-Suárez et al., 2013a]. Estos algoritmos siguen diferentes estrategias para obtener un agrupamiento con traslape.

SHC [Hammouda and Kamel, 2004] es un algoritmo de agrupamiento con traslape incremental que utiliza el concepto de histograma de similitud de un grupo. El histograma de similitud de un grupo c, denotado como H_c, es una representación estadística de la distribución de similitudes existente entre los objetos pertenecientes a c. H_c está dividido en un número de celdas que representan intervalos de similitud. En cada celda se almacena el número de pares de objetos con una similitud que se encuentra en el intervalo representado por dicha celda.

Para agrupar un conjunto de objetos, el algoritmo SHC utiliza el cociente del histograma de similitud de un grupo, el cual es una medida de cohesión, que determina mediante un umbral qué tan similares son los objetos de un grupo. Cada vez que un nuevo objeto x es agregado, SHC determina en qué grupos debe ser incluido x. SHC evalúa la adición de x a cada grupo c_i, calculando el cociente de H_{c_i} antes (oH_{c_i}) y después (nH_{c_i}) de incluir a x en c_i. x es incluido en c_i si y sólo si $nH_{c_i} \geq oH_{c_i}$ o $oH_{c_i} - nH_{c_i} \leq \gamma$ y $nH_{c_i} > HR_{min}$; HR_{min} y γ son parámetros del algoritmo. Si x no es incluido en ningún grupo, entonces se forma un nuevo grupo conteniendo a x como único objeto.

Los algoritmos Star [Aslam et al., 1998], ICSD [Pérez Suárez et al., 2009] y DClustR [Pérez-Suárez et al., 2013a], que se describen a continuación, utilizan el concepto de sub-grafos tipo estrella. Un sub-grafo tipo estrella está formado por un vértice v junto con sus adyacentes (también llamados satélites).

El algoritmo Star [Aslam et al., 1998] es un algoritmo basado en grafos, algunas de las tareas en las que se ha utilizados son el filtrado [Aslam et al., 2000] y la organización de información [Aslam et al., 2004]. Star representa al conjunto de objetos por medio de un grafo de β-similitud G_β. Dicho grafo es construido calculando la similitud entre todos los objetos del conjunto, los objetos son representados por vértices y las aristas conectan a aquellos objetos que superan cierto umbral de similitud. Para obtener los grupos Star busca sub-grafos, cada sub-grafo representa a un grupo, en el grafo G_β.

Star construye una lista L, en la cual están contenidos todos los vértices del grafo. Posteriormente, Star selecciona iterativamente de L, al vértice que forma al sub-grafo que contiene el mayor número de vértices adyacentes. Una vez que se selecciona un vértice, éste se incluye en un conjunto R y es eliminado de la lista L, junto con sus satélites. Este proceso termina cuando la lista L queda vacía. Los sub-grafos formados por cada vértice en R, junto con sus satélites, representan el

agrupamiento final.

Sea $C = \{S_1, S_2, ..., S_k\}$ el conjunto de sub-grafos representando a los grupos actuales de G_β. Cuando se incluye un nuevo objeto en el conjunto de objetos, algunos sub-grafos en C pueden ser eliminados o modificados, e incluso se pueden crear nuevos. Para hacer la actualización, el algoritmo Star inicializa una lista L con el vértice nuevo y con los satélites adyacentes a él. Star extrae iterativamente de L al vértice v de mayor grado y comprueba las siguientes condiciones:

1. v no pertenece a ningún sub-grafo de C.

2. El sub-grafo determinado por v tiene más satélites que todos los sub-grafos de C, que contienen a v como satélite

Si v satisface al menos una de las condiciones anteriores, entonces el sub-grafo determinado por v se adiciona al conjunto C. Si además v satisface (2), todos los sub-grafos de C que contienen a v como satélite son eliminados de C; los satélites de dichos sub-grafos son incluidos en L. El proceso de actualización termina cuando no hay más vértices en L.

ICSD [Pérez Suárez et al., 2009] es un algoritmo de agrupamiento con traslape incremental que al igual que Star utiliza un grafo de β-similitud G_β para representar al conjunto de objetos a agrupar. ICSD utiliza un criterio distinto al de Star para ordenar y seleccionar los sub-grafos necesarios para formar los grupos. Este criterio llamado *strength* tiene en cuenta el número de vértices adyacentes a un vértice v que tienen un grado menor o igual a él.

ICSD inicializa una lista L con los vértices de G_β que tienen un valor de *strength* mayor que cero, La cual se procesan orden descendente del valor de *strength*. Sea S el conjunto de sub-grafos que forman los grupos para G_β, inicialmente $S = \emptyset$. En cada iteración se obtiene de L al vértice v con mayor *strength* y se verifica si:

1. v no pertenezca a ningún sub-grafo que esté en S.

2. Existe al menos un vértice u adyacente a v, tal que u no pertenece a ningún sub-grafo contenido en S.

Si v satisface alguna de las condiciones anteriores entonces el sub-grafo determinado por v y sus vértices adyacentes (satélites) es incluido en S. Esto se repite hasta que la lista L queda vacía. Una vez construido S, este conjunto de sub-grafos se ordena ascendentemente con respecto al número de satélites de cada sub-grafo. Posteriormente cada sub-grafo $S_i \in S$ es visitado, y si el conjunto $S \setminus S_i$ contiene a todos los vértices de G_β, entonces se elimina S_i de S. Los sub-grafos que queden en S después de este proceso forman el agrupamiento final. Cuando nuevos objetos son agregados, el valor de *strength* de los vértices puede cambiar, por lo tanto es necesario actualizar los grupos de G_β. ICSD construye el conjunto $U = \{C_1, C_2, ..., C_k\}$ de componentes conexas de G_β que contienen a los grupos que pueden haberse modificado y actualiza los grupos asociados a cada componente conexa $C_j \in U$ a la cual se le hayan adicionado nuevos objetos. Para actualizar una componente conexa se aplica la estrategia descrita anteriormente.

DClustR [Pérez-Suárez et al., 2013a] es un algoritmo de agrupamiento con traslape incremental que al igual que Star e ICSD, utiliza un grafo de β-similitud G_β para representar al conjunto de objetos a agrupar. DClustR utiliza un criterio distinto a los de Star e ICSD para ordenar y seleccionar los sub-grafos necesarios para formar los grupos. Este algoritmo utiliza los conceptos de *densidad relativa* y *compacidad relativa*. La *densidad relativa* toma un valor mayor en aquellos vértices que tienen un mayor número de satélites y que además dichos satélites tengan un número menor de adyacentes. La *compacidad relativa* toma un valor mayor en aquellos vértices que tengan una mayor similitud con sus satélites y dichos satélites tengan una menor similitud con sus adyacentes. A partir de los conceptos de *densidad relativa* y *compacidad relativa* se define la *relevancia* de un vértice v como la combinación

lineal de ambas. Además, una vez formados los sub-grafos, DClustR incluye una fase de mejoramiento para reducir el número de grupos y eliminar sub-grafos poco útiles.

DClustR inicializa una lista L con los vértices no aislados de G_β que tienen un valor de *relevancia* mayor que cero, los vértices aislados son incluidos directamente en un conjunto C. Se procesa la lista L en orden descendente de acuerdo a la *relevancia* de los vértices. Sea S el conjunto de sub-grafos que forman los grupos para G_β, inicialmente $S = \emptyset$. En cada iteración se obtiene de L al vértice v con mayor *relevancia* y se verifica si:

1. v no esté cubierto todavía, i.e. $v \notin C$ y $\nexists c \in C$ tal que c y v son adyacentes.

2. v está cubierto pero tiene al menos un vértice adyacente u tal que $u \notin C$ y $\nexists c \in C$, tal que c y u sean adyacentes.

Si v satisface alguna de las condiciones anteriores entonces el sub-grafo determinado por v y sus vértices adyacentes (satélites) es incluido en C. Esto se repite hasta que la lista L queda vacía.

Para mejorar el agrupamiento, DClustR analiza el conjunto de sub-grafos en C para eliminar los vértices que son poco útiles. La utilidad de un sub-grafo se determina en función del número total de satélites que conforman el sub-grafo y del número de satélites que están contenidos solamente en dicho sub-grafo.

Cuando nuevos objetos son agregados, el valor de *relevancia* de los vértices puede cambiar, por lo tanto es necesario actualizar los grupos de G_β. DClustR construye el conjunto $U = \{C_1, C_2, ..., C_k\}$ de componentes conexas de G_β que contienen los grupos que pueden haberse modificado y actualiza los grupos de cada componente $C_j \in U$ a la cual se le hayan adicionado nuevos objetos. Para actualizar una componente conexa se aplica la estrategia descrita anteriormente, incluyendo la fase de mejora, en la cual los sub-grafos poco útiles son eliminados.

Como se puede observar, en los algoritmos antes mencionados es necesario utilizar los objetos previamente agrupados cada vez que se adiciona un nuevo objeto. Además, es necesario verificar una serie de condiciones cuando se agrega un objeto a un grupo, esto hace que el proceso de agregar objetos sea muy costoso. Aunado a esto se debe tener una gran cantidad de información en memoria, ya que no solo los objetos son almacenados sino también las similitudes o distancias entre ellos, en varios casos representadas por un grafo. Por todo esto, este tipo de algoritmos no son capaces de procesar un flujo de datos, ya que en este caso se parte del supuesto de que el flujo de datos es infinito y el tiempo para procesar cada objeto nuevo es limitado.

Capítulo 4

Algoritmo propuesto

En este capítulo se describe el algoritmo propuesto para el agrupamiento con traslape en flujos de datos. Primero se menciona la estrategia de paginado utilizada. Posteriormente se describe la fase *online*, la cual está basada en la que se propone en el algoritmo DenStream [Cao et al., 2006], sin embargo, se proponen algunas modificaciones para resolver algunos problemas que se encontraron en dicha fase. También, se describen la fase *offline* en la cual se proponen modificaciones al algoritmo OClustR [Pérez-Suárez et al., 2013b] para poder utilizarlo en la construcción de grupos con traslape mediante *micro-clusters*. El algoritmo propuesto llamado OCDS (Overlapping Clustering over Data Streams) está basado en las modificaciones antes mencionadas que se hicieron a los algoritmo DenStream y OClustR.

4.1 Modelo de ventaneo utilizado

En el capítulo 2 se hace mención de los tres principales modelos de ventaneo utilizados para el manejo de flujos de datos. Para seleccionar el modelo que se utilizó en el algoritmo propuesto en esta tesis se hizo un análisis de las ventajas y desventajas de cada modelo. Además, se tomó en cuenta cuál es el modelo de ventaneo que más se ha utilizado en la literatura para el agrupamiento en flujos de datos.

El modelo de ventaneo de *punto de referencia* rara vez es utilizado en el agru-

pamiento en flujos de datos, ya que este modelo mantiene en memoria a todos los objetos que llegan. Es muy poco recomendable para flujos de datos en los cuales se tiene una gran cantidad de información, por lo que se optó por no utilizar este modelo.

El modelo *ventana deslizante* es útil cuando se sabe a priori el número de objetos que se analizarán en cierto periodo de tiempo. Sin embargo, los algoritmo de agrupamiento en flujos de datos que utilizan este modelo de ventaneo no aprovechan la información antigua que aún sigue siendo útil, pues cuando la ventana ha sido analizada todos los objetos en dicha ventana son eliminados, así como los grupos que se formaron con esos objetos.

El modelo de ventaneo de *desvanecimiento* es el más utilizado en el agrupamiento en flujos de datos, pues éste asigna un valor de importancia a cada objeto en el flujo de datos de acuerdo a la antigüedad, para ello utiliza una función de desvanecimiento. La función de desvanecimiento exponencial mencionada en el capítulo 3 es la que se utiliza en la mayoría de los algoritmos de agrupamiento en flujos de datos, esta función junto con representaciones compactas como *micro-clusters* o *grids*, ayuda a retener los grupos que constantemente reciben nuevos objetos, aún cuando en dichos grupos existan objetos muy antiguos.

DenStream es uno de los algoritmos de agrupamiento en flujos de datos que utiliza el modelo de ventaneo de *desvanecimiento*, además de que es un algoritmo de agrupamiento basado en densidad que ha reportado buenos resultados en la literatura. La fase *online* de este algoritmo se tomó como base para procesar los objetos, sin embargo, su fase *offline* tuvo que ser modificada utilizando un algoritmo que tuviera la posibilidad de crear grupos con traslape a partir de los *micro-clusters* resultantes.

4.2 Adaptación de estrategias de Agrupamiento

Para el desarrollo del algoritmo propuesto en esta tesis, se utiliza la misma idea de DenStream; es decir, se tiene una fase *online* y una fase *offline*, para la fase *online* se proponen algunas modificaciones para tratar algunos problemas que tiene DenStream. En la fase *offline* se proponen algunas modificaciones al algoritmo OClustR, para que pueda utilizar *micro-clusters* en el proceso de construcción del grafo de similitud G_β. Además se propone una estrategia adicional para crear más aristas en el grafo, pues en un principio se obtienen grafos con muy pocas aristas, lo cual hace que se produzcan muchos grupos con poco traslape.

4.2.1 Fase *Online*

A pesar de que se propone utilizar la fase *online* de DenStream, hay ciertas consideraciones que se deben tener en cuenta pues estas afectarán en la construcción de grupos con traslape, en esta sección se mencionan las modificaciones propuestas en la fase *online* del algoritmo DenStream. Las modificaciones propuestas se enfocan en facilitar la construcción de un agrupamiento con traslape en la fase final, además de que facilitan la selección del parámetro de desvanecimiento.

Existe un problema en el algoritmo DenStream original que está relacionado con el cálculo del radio de los *micro-clusters*. Si bien la expresión 3.1 para calcular el radio de los *micro-clusters* es reportada en todas las variantes del algoritmo DenStream, en la práctica se detectó que el argumento que se obtiene dentro de la raíz cuadrada puede ser negativo, por lo cual esa expresión no puede ser utilizada en estos casos. El aplicar un valor absoluto del argumento no soluciona el problema ya que el valor obtenido es muy pequeño y al final se obtienen muchos grupos, por lo que este valor no es bueno para utilizarse como radio. En [Aggarwal et al., 2003] se reportó la siguiente expresión (4.1) para obtener el radio:

$$r = \sqrt{\sum_{i=1}^{m} \left(\frac{\overline{CF^2}_i}{w} - (\frac{\overline{CF^1}_i}{w})^2 \right)} \tag{4.1}$$

En esta expresión se elimina el uso de la norma y únicamente se hace la diferencia de cada vector resultante después de dividirlo entre el peso, finalmente se suman los resultados de las diferencias. Otra consideración que se debe tener en cuenta es que varios de los algoritmos derivados de DenStream son implementados dentro de un Framework llamado MOA (Massive Online Analysis), el cual utiliza una expresión distinta para el cálculo del radio de los *micro-clusters*, la cual es:

$$r = \max_{1 \leq i \leq m} \sqrt{\frac{\overline{CF_i^2}}{w} - (\frac{\overline{CF_i^1}}{w})^2} \tag{4.2}$$

El problema de utilizar cualquiera de las dos expresiones antes mencionadas es el hecho de que los radios obtenidos son muy pequeños, debido a que estas medidas aproximan el promedio de distancias que hay entre el centro y los objetos que se fueron agregando al *micro-cluster*, lo cual tiene un impacto negativo al momento construir el agrupamiento final, generando muchos grupos pues los radios son muy pequeños y hay poca intersección entre ellos. Algo que no se menciona en [Cao et al., 2006] es que utilizando la expresión 3.1 los *micro-clusters* rara vez se intersectan, este problema se ilustra en la figura 4.1.

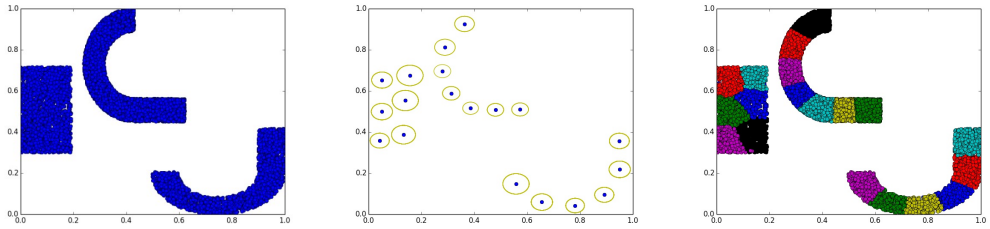


Figura 4.1: Resultado obtenido después de aplicar DenStream con la expresión 4.2 para calcular el radio de los *micro-clusters*.

En MOA se utiliza una constante con valor de 1.8 para aumentar el tamaño de los *micro-clusters*, sin embargo, no se menciona por qué se utiliza dicho valor. Si se utiliza un valor para incrementar el tamaño del radio se puede obtener un agrupamiento similar al mostrado en la figura 3.2, sin embargo en [Cao et al., 2006] no se menciona nada acerca de la constante utilizada para incrementar el tamaño de los radios.

Los *micro-clusters* con radios pequeños tienen un impacto negativo en la construcción de agrupamientos con traslape, debido a que los grupos con traslape comparten objetos y los *micro-clusters* con radios pequeños usualmente no permiten tener objetos en común, pues sus radios están muy separados. Para resolver el problema antes mencionado, se propone calcular el radio de un *micro-cluster* como la distancia entre el centro del *micro-cluster* y el objeto más lejano a éste. De esta forma, se puede eliminar a $\overline{CF^2}$ de la representación de los *micro-clusters* y en su lugar se debe guardar al objeto más lejano, al cual se llamará x_f. Además de guardar al objeto x_f, también se necesita guardar su etiqueta de tiempo T_f, ya que con el tiempo la relevancia del objeto también influye en el radio. Considerando esta modificación, la fórmula utilizada para calcular el radio es la siguiente:

$$r = \sqrt{\left(\sum_{i=1}^{m}(c_i - x_{f_i})^2\right)2^{-\lambda\delta t}} \tag{4.3}$$

donde $\delta t = t - T_f$ es la diferencia entre el tiempo actual y el tiempo en que llegó el objeto x_f; c es el centro del *micro-cluster*. En la figura 4.2 se muestra el resultado de utilizar esta expresión en el mismo conjunto de datos que la figura 4.1.

Como se puede observar el la figura 4.2, utilizando la expresión propuesta se obtienen más *micro-clusters*, muchos de ellos traslapados, esto ayuda a que en la fase final se obtenga un mejor agrupamiento y facilita la detección del traslape entre los grupos. El uso de esta expresión añade un costo extra en los cálculos, ya que

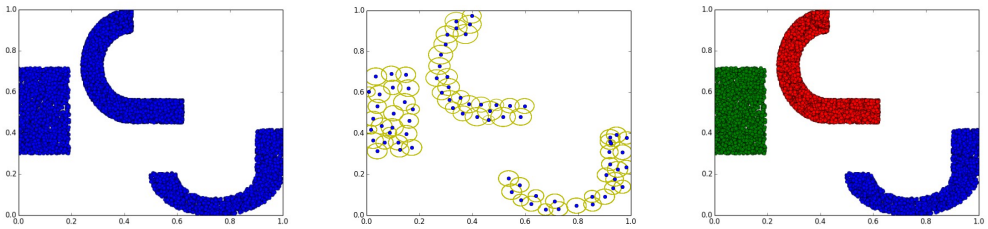


Figura 4.2: Resultado obtenido después de aplicar DenStream con la expresión 4.3 para calcular el radio de los *micro-clusters*.

cada vez que un nuevo objeto es añadido a un *micro-cluster*, se debe verificar si el nuevo objeto es más lejano que el objeto x_f actual. Sin embargo, en la práctica la diferencia en costo es muy poca con respecto a la expresión original.

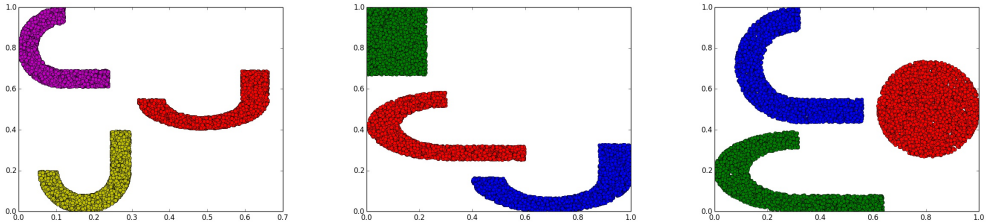
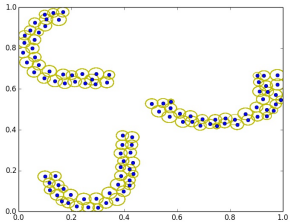
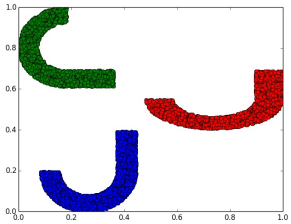
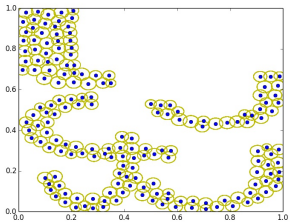
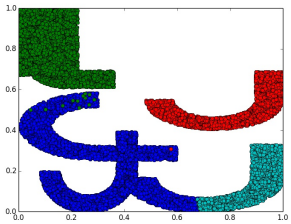
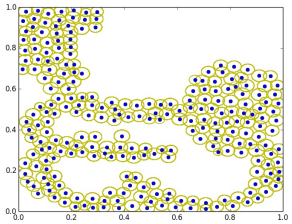
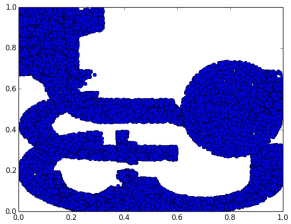


Figura 4.3: Conjunto de datos con evolución en la distribución de objetos, cada imagen representa una ventana de 20,000 objetos distribuidos en 3 grupos.

Otra modificación que se hizo a DenStream está relacionada con la evolución en el flujo de datos; es decir, cuando la forma y el número de grupos cambian con el paso del tiempo. El problema que tiene DenStream en este caso es que, en algunas ocasiones, tarda demasiado en eliminar los *mcp* que no han recibido elementos (dependiendo del valor del parámetro λ). Si se da un valor muy alto para λ se puede solucionar el problema mencionado pero surge otro: el radio de los *mcp* decrece demasiado rápido y, por lo tanto, en la fase *Offline*, al solicitar un agrupamiento final, se generan más grupos. La figura 4.3 muestra un conjunto de datos con evolución en la distribución de los objetos, que está dividido en tres ventanas de 20,000 objetos, con tres grupos en cada ventana.

Tabla 4.1: Resultados obtenidos con DenStream utilizando un valor de $\lambda = 0.25$

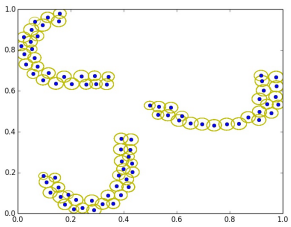
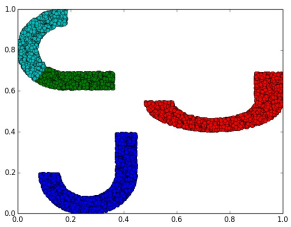
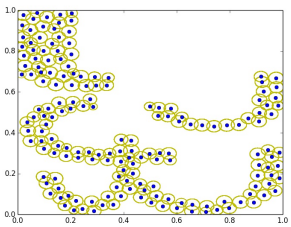
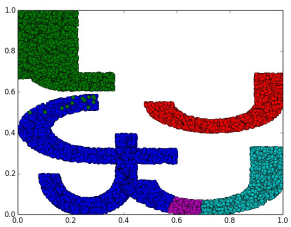
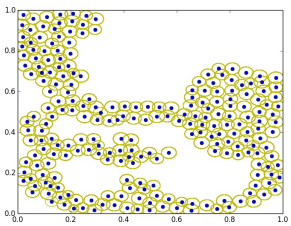
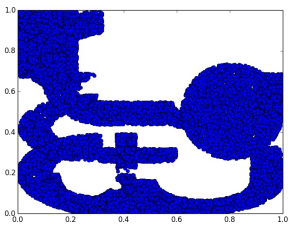
Clusters y *mcp* obtenidos	*mcp*	Grupos obtenidos
Ventana 1 - Grupos = 3, *mcp* = 86		
Ventana 2 - Grupos = 4, *mcp* = 180		
Ventana 3 - Grupos = 1, *mcp* = 251		

Al aplicar DenStream con $\lambda = 0.25$ sobre el conjunto mostrado en la figura 4.3 el algoritmo no es capaz de eliminar los *mcp* de las ventanas anteriores, esto hace que dichos *mcp* sean considerados aún como grupos, lo cual afecta el resultado del agrupamiento final. En la segunda columna de la tabla 4.1 se puede notar que tanto en la segunda como en la tercera ventana se retienen *micro-clusters* que ya no son útiles para dichas ventanas, esto hace que se incremente el costo en cálculos al momento de asignar objetos nuevos a un *micro-cluster*, pues para hacer dicha asignación se debe calcular la distancia entre el objeto nuevo y los centros de los *micro-clusters*. Además, como aún se retienen algunos *micro-clusters* de las ventanas anteriores, estos hacen que se genere un agrupamiento muy distinto al que se espera.

Por otra parte, al aplicar DenStream con valor de $\lambda = 0.5$ sobre el mismo

conjunto de datos, los *mcp* de la ventana actual se ven afectados en su peso y radio, ya que estos valores decrecen muy rápido y esto afecta en la calidad del agrupamiento final; en la tabla 4.2 se puede apreciar que los *micro-clusters* de la ventana 1 decrecieron más rápido, por lo que se encontró un grupo extra. Además, aún no es capaz de eliminar todos los *micro-clusters* que no son útiles en las siguientes ventanas. Es por este problema que se debe proponer alguna otra forma de eliminar aquellos *micro-clusters* que ya no son útiles, pues el usuario debe estar probando distintos parámetros para poder determinar cuándo estos *micro-clusters* deberían ser eliminados. Lo cual es poco práctico.

Tabla 4.2: Resultados obtenidos con DenStream utilizando un valor de $\lambda = 0.5$

Clusters y *mcp* obtenidos	*mcp*	Grupos obtenidos
Ventana 1 - Grupos = 4, *mcp* = 82		
Ventana 2 - Grupos = 5, *mcp* = 172		
Ventana 3 - Grupos = 1, *mcp* = 236		

Para resolver este problema se propuso modificar la etapa intermedia en la que DenStream elimina a los *micro-clusters* antiguos. Además, se propuso considerar a

aquellos *micro-clusters* que han recibido pocos objetos en las últimas actualizaciones como poco importantes, y en tal sentido se les trata como ruido, es decir, se convierten en *mca*. Para determinar cuándo un *micro-cluster* recibe pocos objetos se calcula la media de las asignaciones realizadas en cada *mcp* en un intervalo de tiempo, además se calcula la media de los pesos de los *mcp*. Aquellos *mcp* que tengan un número de asignaciones y un peso por debajo de ambas medias son convertidos en *mca*.

Algoritmo 1 Asignar

Entrada: x

1: Intentar incluir a x dentro de su *mcp* más cercano c_p
2: **si** x puede ser incluido en c_p **entonces**
3: $counter_p = counter_p + 1$
4: **si no**
5: Intentar incluir a x dentro de su *mca* más cercano c_a
6: **si** x puede ser incluido en c_a **entonces**
7: $counter_a = counter_a + 1$
8: **si** w (el nuevo peso de c_a)$> \beta\mu$ **entonces**
9: Convertir a c_a en un nuevo *mcp*
10: **fin si**
11: **si no**
12: Crear un nuevo *mca* para x
13: $counter_a = 1$
14: **fin si**
15: **fin si**

Otra modificación propuesta a DenStream consiste en fusionar un *mca* con su *mcp* más cercano antes de intentar eliminarlos. De esta manera se permite retener cierta información que aún puede ser útil. Para fusionar dos *micro-clusters* la distancia entre sus centros debe ser menor que el radio del *mcp*; es decir, el centro del *mca* debe estar contenido en el *mcp*. Además, se debe cumplir que el nuevo radio no debe ser mayor que ϵ, que es el parámetro utilizado para limitar el tamaño del radio.

La última modificación propuesta en DenStream consiste en no eliminar los *mcp*, pues en algún momento fueron relevantes y podrían estar en un periodo de

Algoritmo 2 DenStream mejorado

Entrada: Flujo de datos DS, radio máximo ϵ, tolerancia al ruido β, número mínimo de objetos para un *mcp* μ, parámetro para la función de desvanecimiento λ

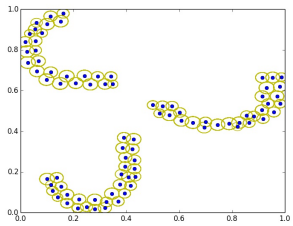
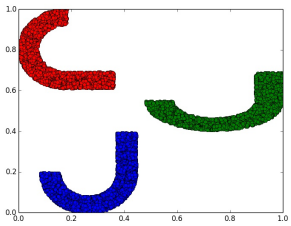
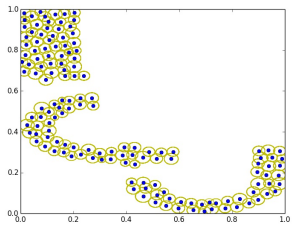
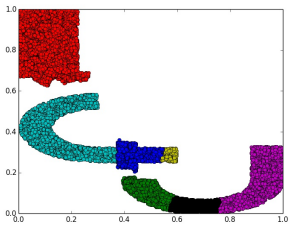
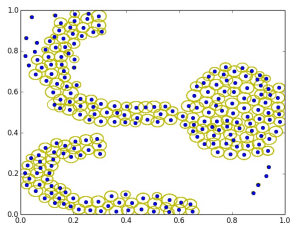
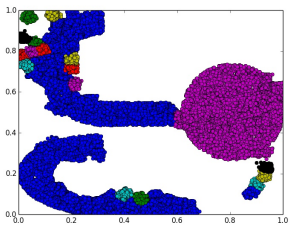
Salida: Agrupamiento C

1: $T_p = [\frac{1}{\lambda} \log \frac{\beta\mu}{\beta\mu - 1}]$ (Valor utilizado para iniciar la fase de eliminación de *micro-clusters*)
2: **repetir**
3: Obtener el siguiente objeto x en el tiempo actual t del flujo de datos DS;
4: Asignar(x)
5: **si** $((t \mod T_p) = 0)$ {Inicia la fase de eliminación de *micro-clusters*} **entonces**
6: $n =$ número de *mcp*
7: $\bar{x}_a = \frac{1}{n} \sum_{i=1}^{n} counter_i$
8: $\bar{x}_w = \frac{1}{n} \sum_{i=1}^{n} w_i$
9: **para todo** c_a en los *mca* **hacer**
10: $\xi = \frac{2^{-\lambda(t - t_a + T_p) - 1}}{2^{-\lambda T_p - 1}}$
11: **si** w_a (el peso de c_a)$< \xi$ **entonces**
12: Intentar fusionar a c_a con su *mcp* más cercano c_p
13: **si** c_a no se pudo fusionar con c_p **entonces**
14: Eliminar a c_a
15: **fin si**
16: **fin si**
17: **fin para**
18: **para todo** c_p en los *mcp* **hacer**
19: **si** w_p (el peso de c_p)$< \bar{x}_w$ y $counter_p < \bar{x}_a$ **entonces**
20: Convierte a c_p en un *mca*
21: **fin si**
22: $counter_p = 0$
23: **fin para**
24: **para todo** c_p en los *mcp* **hacer**
25: **si** w_p (el peso de c_p)$< \beta\mu$ **entonces**
26: Convierte a c_p en un *mca*
27: **fin si**
28: **fin para**
29: **fin si**
30: **si** Llega una petición para formar los grupos **entonces**
31: Generar el agrupamiento con DBSCAN
32: **fin si**
33: **fin repetir**

inactividad. En lugar de eliminarlos se convierten en *mca* para así darles la oportunidad de volver a recibir objetos.

El algoritmo con las modificaciones propuestas se muestra en el Algoritmo 2, éste a su vez utiliza al Algoritmo 1 para hacer la asignación de los nuevos objetos en los *micro-clusters*.

Tabla 4.3: Resultados obtenidos con la estrategia de eliminación de *micro-clusters* propuesta utilizando un valor de $\lambda = 0.25$

Clusters y *mcp* obtenidos	*mcp*	Grupos obtenidos
Ventana 1 - Grupos = 3, *mcp* = 104		
Ventana 2 - Grupos = 7, *mcp* = 144		
Ventana 3 - Grupos = 20, *mcp* = 230		

En la tabla 4.3 se muestra el resultado obtenido utilizando la nueva estrategia de eliminación de *micro-clusters*, en este caso se utilizó un valor de $\lambda = 0.25$. A diferencia de la tabla 4.1, en este caso sí se eliminaron los *micro-clusters* que ya no eran útiles en las diferentes ventanas. Además, debido a que no se utilizó un valor grande para λ los radios de los *micro-clusters* no decrecieron demasiado rápido.

Aunque en este ejemplo aún quedaron algunos *micro-clusters* que son considerados como grupos separados, éstos no tardarán en ser removidos pues ya no son útiles para dicha ventana.

De las modificaciones propuestas en la fase *online* de DenStream, la nueva expresión para el cálculo del radio se utiliza para poder facilitar la forma de construir grupos con traslape, pues con las expresiones antes utilizadas es muy difícil determinar cuándo dos grupos están lo suficientemente cercanos para decir que existe traslape, ya que éstos nunca se intersectan en sus radios. A pesar de las modificaciones propuestas en la fase *online* si se usa la fase *offline* de DenStream, en el resultado final se obtendrían grupos disjuntos, debido a que, en la fase final se utiliza DBSCAN para agrupar los *micro-clusters* y DBSCAN es un algoritmo de agrupamiento que no considera la existencia de traslape entre los grupos.

4.2.2 Fase *Offline*

Como se mencionó en un principio, el objetivo de este trabajo es proponer un algoritmo de agrupamiento con traslape para flujos de datos. Para esto, a partir de las mejoras propuestas en la fase *online* del algoritmo DenStream, se hizo un análisis de cuáles algoritmos pudieran adaptarse para construir un agrupamiento con traslape en la fase *offline*. En tal sentido, se analizaron distintos algoritmos de agrupamiento con traslape que no necesitan como parámetro el número de grupos y que pudieran aprovechar los *mcp* que produce la fase *online* sin requerir parámetros adicionales, entre los algoritmos analizados se encuentran Star [Aslam et al., 1998], ICSD [Pérez Suárez et al., 2009], EStar [Gil-García et al., 2003], OClustR [Pérez-Suárez et al., 2013b]. Teniendo en cuenta los resultados reportados en la literatura, se optó por el algoritmo OclustR [Pérez-Suárez et al., 2013a], ya que ha reportado muy buenos resultados, comparado contra los otros algoritmos de agrupamiento con traslape, además no necesita a priori el número de grupos a

formar.

Para generar grupos con traslape en la fase *offline* se propone seguir la misma estrategia de agrupamiento que utiliza OClustR, pero haciendo algunas adaptaciones.

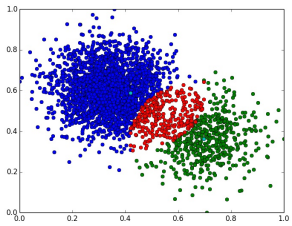
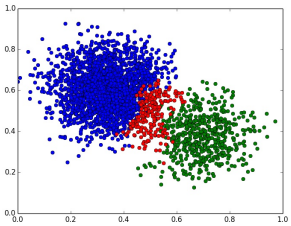
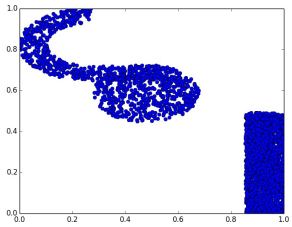
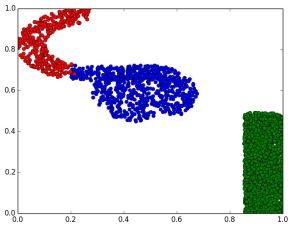
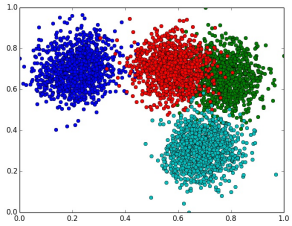
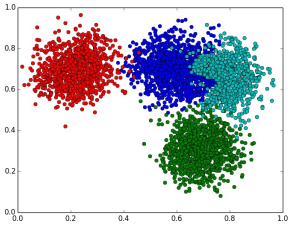
Debido a que OClustR parte de un grafo donde los vértices del mismo representan a los objetos a agrupar, en nuestro caso los vértices del grafo serán los centros de los micro-clusters. Mientras que para determinar las aristas se utiliza el radio de los *micro-clusters*, para esto, aquellos centros de *mcp* (vértices) cuyos radios se intersectan se conectan con una arista, cuyo peso es igual a $1 - d(c_i, c_j)$, con $d(c_i, c_j) \in [0, 1]$, siendo $d(c_i, c_j)$ la distancia entre los centros de los *micro-clusters* c_i y c_j. En el agrupamiento resultante, aquellos *mcp* que pertenecen a más de un grupo representan el conjunto de objetos que están en grupos traslapados, permitiendo así el traslape entre los grupos de un flujo de datos.

En la tabla 4.4 se muestra el resultado de utilizar la fase *online* propuesta de DenStream (descrita en la sección 4.2.1) y OClustR en la fase *offline*. En la imagen del primer renglón de la tabla, la región en color rojo representa el traslape entre los grupos. En la imagen del segundo renglón de la tabla, aunque no se tiene etiquetado el conjunto original, por la separación que existe entre los grupos, intuitivamente se puede determinar que el resultado no es malo. Finalmente, en la imagen del tercer renglón de la tabla, se tienen cuatro grupos, de los cuales dos están muy juntos, en este conjunto OClustR pudo determinar que existen cuatro grupos.

A pesar de que en la tabla 4.4 se puede apreciar que el resultado obtenido utilizando OClustR es similar al original, existe un problema cuando el conjunto de objetos es muy grande. En este caso, OClustR encontrará más grupos de los que realmente hay, debido a que los *micro-clusters* generan pocas aristas en el grafo de OClustR.

Como solución al problema antes mencionado, se deben generar más aristas

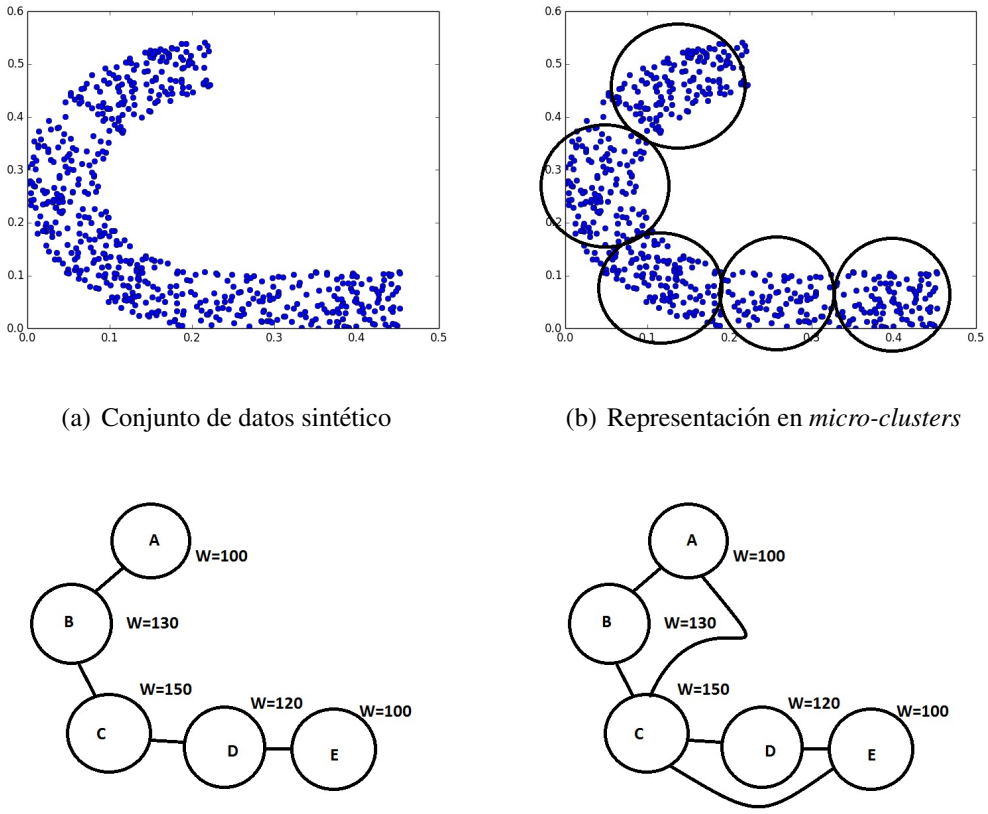
Tabla 4.4: Traslape obtenido utilizando OClustR como algoritmo para la fase *Offline*.

Conjunto original	Resultado obtenido de DenStream mejorado + OClustR
	
	
	

entre los *micro-clusters* (vértices), de esta manera se puede hacer un mejor aprovechamiento de la fase en la que el algoritmo OClustR elimina sub-grafos poco útiles.

La idea que se propuso para generar más aristas en el grafo de OClustR es primero buscar los vértices más importantes, en términos del *peso* del *micro-cluster* al que representa un vértice y los satélites de dicho vértice. El peso refleja la antigüedad y el número de objetos del *micro-cluster* representado por un vértice, mientras que los satélites determinan la cercanía del *micro-cluster* con los demás *micro-clusters*. Finalmente, aquellos vértices más importantes son conectados por medio de una arista con los adyacentes de sus satélites menos importantes, pues es muy probable que los vértices más importantes se conviertan en representativos de los grupos

que posteriormente se construirán y al tener como adyacentes a los vértices de sus satélites menos importantes, estos últimos serán considerados como poco útiles.



(a) Conjunto de datos sintético

(b) Representación en *micro-clusters*

(c) Representación en un grafo de similitud utilizando los centros de los *micro-clusters*

(d) Grafo después de añadir aristas

Figura 4.4: Proceso de adición de nuevas aristas en el grafo.

Para ejemplificar la situación antes mencionada consideremos la figura 4.4.a donde se tiene un conjunto de datos sintético en dos dimensiones. Después de representar el conjunto de datos utilizando *micro-clusters* se obtiene lo mostrado en la figura 4.4.b. Posteriormente, al representar los *micro-clusters* por un grafo, se obtiene lo que se puede ver en la figura 4.4.c, como se mencionó anteriormente los vértices solo están conectados con sus vecinos directos, pues estas aristas se crean solo entre los *micro-clusters* que se intersectan. El problema en este caso es que

al tener vértices adyacentes sin compartir, el algoritmo OClustR considerará como semillas o representativos a los vértices B y D, formando un total de dos grupos. Si agregamos más aristas según la idea propuesta se obtendrá lo que se muestra en la figura 4.4.d. En este caso, los vértices B y D comparten todos sus vértices y por lo tanto, serán considerados como poco útiles por el algoritmo OClustR y finalmente, se formará un solo grupo.

Para crear aristas entre *micro-clusters* que no son vecinos directos, se propuso una fase posterior a la creación del grafo utilizado en OClustR. En esta fase se crea una lista S, que almacenan los vértices en orden descendente con respecto al valor de importancia, aquellos vértices con mayor peso y número de satélites son considerados como más importantes. Una vez creada y ordenada la lista S, se procesa cada vértice v de S en el orden establecido. Si v conectado a los vértices adyacentes del satélite u tiene un valor de importancia mayor que el de u conectado los adyacentes de v, entonces se le considera más importante que u, y en tal caso los adyacentes a u pasan a convertirse en satélites de v, pero no se eliminan las aristas que ya existían entre u y sus adyacentes.

Finalmente el Algoritmo 3, llamado OClustRmc, es utilizado para hacer el agrupamiento con traslape sobre los *micro-clusters* en la fase *offline*. Las modificaciones propuestas son apreciables en los pasos del 1 al 9, éstas se enfocan principalmente en crear más aristas para aprovechar mejor la fase en la que los sub-grafos útiles toman objetos adyacentes de los sub-grafos poco útiles, con el objetivo de reducir el número de grupos.

De esta forma, el algoritmo propuesto para el agrupamiento con traslape en flujos de datos denominado OCDS *(Overlapping Clustering over Data Streams)* trabaja de la siguiente forma. Durante la fase *online* de OCDS se aplica el algoritmo DenStream mejorado, mientras que en la fase *offline* se aplica el algoritmo OClustRmc para agrupar los *micro-clusters* resultantes.

El algoritmo OCDS mantiene las características de la fase *online* de DenStream, como son, la tolerancia al ruido, el mantenimiento incremental de los *micro-clusters* y el *desvanecimiento* de los objetos para su eliminación cuando ya no sean útiles. Además, este algoritmo no requiere el número de grupos como parámetro y genera un agrupamiento con traslape.

Algoritmo 3 OClustR para *micro-clusters* (OClustRmc)

Entrada: conjunto de *micro-clusters* mc
Salida: Agrupamiento con traslape SC

1: Construir grafo G_β utilizando el radio de los *micro-clusters*
2: Calular *importancia* de todos los vértices en G_β
3: $S = \{v \in G_\beta\}$
4: Ordenar S en orden descendente de acuerdo a la *importancia*
5: **para todo** vértice $v \in S$ **hacer**
6: **si** v conectado por aristas con los adyacentes de sus satélites es más *importante* que sus satélites conectados a los adyacentes de v **entonces**
7: Crear aristas de v con los adyacentes de los satélites menos importantes que él
8: **fin si**
9: **fin para**
10: Calcular la relevancia de cada vértice en G_β
11: $SC = \emptyset$
12: $C = \{v \in G_\beta | v$ tenga al menos un satélite$\}$
13: $L = \{v \in G_\beta |$La relevancia de v sea mayor que $0\}$
14: Ordenar L en orden descendente de acuerdo a la relevancia
15: **para todo** vértice $v \in L$ **hacer**
16: **si** v y sus adyacentes no están en C o v está en C pero alguno de sus satélites no **entonces**
17: $C = C \cup \{v\}$
18: **fin si**
19: **fin para**
20: Ordenar C en orden descendente de acuerdo al *grado*
21: Marcar a cada vértices en C como *no-analizado*
22: **para todo** vértice c en C **hacer**
23: $c.Linked = \emptyset$
24: **para todo** vértice $v \in$ satélites de c **hacer**
25: **si** $v \in C$ y v está marcado como *no-analizado* **entonces**
26: **si** v es útil **entonces**
27: $c.Linked = c.Linked \cup$ satélites no compartidos de v
28: $C = C \setminus \{v\}$
29: **si no**
30: Marcar v como *analizado*
31: **fin si**
32: **fin si**
33: **fin para**
34: Marcar c como *centro*
35: **fin para**
36: **para todo** vértice $v \in G_\beta$ **hacer**
37: **si** v está marcado como centro **entonces**
38: $SC = SC \cup \{\{c\} \cup \{$satélites de $c\} \cup \{c.Linked\}\}$
39: **fin si**
40: **fin para**

Algoritmo 4 OCDS

Entrada: Flujo de datos DS, radio máximo ϵ, tolerancia al ruido β, número mínimo de objetos para un *mcp* μ, parámetro para la función de desvanecimiento λ

Salida: Agrupamiento C

1: $T_p = [\frac{1}{\lambda}\log\frac{\beta\mu}{\beta\mu-1}]$ (Valor utilizado para iniciar la fase de eliminación de *micro-clusters*)
2: **repetir**
3: Obtener el siguiente objeto x en el tiempo actual t del flujo de datos DS;
4: Asignar(x) {Algoritmo 1}
5: **si** $((t \mod T_p) = 0)$ {Inicia la fase de eliminación de *micro-clusters*} **entonces**
6: $n =$ número de *mcp*
7: $\bar{x}_a = \frac{1}{n}\sum_{i=1}^{n} counter_i$
8: $\bar{x}_w = \frac{1}{n}\sum_{i=1}^{n} w_i$
9: **para todo** c_a en los *mca* **hacer**
10: $\xi = \frac{2^{-\lambda(t-t_a+T_p)-1}}{2^{-\lambda T_p-1}}$
11: **si** w_a (el peso de c_a)$< \xi$ **entonces**
12: Intentar fusionar a c_a con su *mcp* más cercano c_p
13: **si** c_a no se pudo fusionar con c_p **entonces**
14: Eliminar a c_a
15: **fin si**
16: **fin si**
17: **fin para**
18: **para todo** c_p en los *mcp* **hacer**
19: **si** w_p (el peso de c_p)$< \bar{x}_w$ y $counter_p < \bar{x}_a$ **entonces**
20: Convierte a c_p en un *mca*
21: **fin si**
22: $counter_p = 0$
23: **fin para**
24: **para todo** c_p en los *mcp* **hacer**
25: **si** w_p (el peso de c_p)$< \beta\mu$ **entonces**
26: Convierte a c_p en un *mca*
27: **fin si**
28: **fin para**
29: **fin si**
30: **si** Llega una petición para formar los grupos **entonces**
31: Generar el agrupamiento con OClustRmc (Algoritmo 3)
32: **fin si**
33: **fin repetir**

Capítulo 5

Análisis Experimental

En este capítulo se presenta una comparación experimental entre el algoritmo DClustR y el algoritmo OCDS, propuesto en esta tesis. Primero, se explica la medida de evaluación utilizada en los experimentos para determinar la calidad de los agrupamientos obtenidos. Después, se da una breve descripción de la selección de los parámetros de los algoritmos, además se describen los conjuntos de datos utilizados para la evaluación experimental. Posteriormente, se hace una comparación en términos de calidad entre ambos algoritmos. Finalmente, se compara el tiempo de ejecución sobre los conjuntos de datos completos. Todos los experimentos se realizaron en una computadora con procesador Core i7 de cuatro núcleos a una velocidad de 2.90 GHz, 16 GB de memoria RAM y sistema operativo GNU/Linux.

5.1 Medida de evaluación

Para determinar qué tan bueno es un algoritmo de agrupamiento se debe hacer uso de alguna métrica que evalúe su calidad. En la literatura han sido reportadas dos tipos de medidas de calidad: externas e internas [Pfitzner et al., 2009].

Las medidas externas utilizan un conjunto de datos, previamente etiquetado. Mientras mayor sea la similitud entre las clases originales del conjunto de datos y los grupos generados por el algoritmo de agrupamiento, mejor será el algoritmo con

el que se construyeron dichos grupos. Algunos ejemplos de este tipo de medidas son Purity [Zhao and Karypis, 2001], Jaccard coefficient [Halkidi et al., 2001] y F1-measure [Larsen and Aone, 1999].

Las medidas internas evalúan un algoritmo de agrupamiento utilizando la información contenida en los grupos a evaluar, como por ejemplo: la compacidad y separación de los grupos, la conectividad de los mismos, etc. Algunos ejemplos de este tipo de medidas son los índices David-Bouldin y Silhouette [Halkidi et al., 2001] y la medida Λ [zu Eissen and Wißbrock, 2003].

De los dos tipos de medidas mencionadas, las más utilizadas en la literatura para evaluar algoritmos de agrupamiento son las medidas externas [Amigó et al., 2009]. Para el caso del agrupamiento con traslape no se han reportado medidas internas, por lo que en el presente trabajo se utilizó una medida externa.

Para evaluar agrupamientos con traslape, existen trabajos en los que se han utilizado medidas basadas en conteo de pares de objetos, como Fmeasure y Jaccard coefficient [Banerjee et al., 2005, Pérez Suárez et al., 2009, Cleuziou, 2008, Cleuziou, 2009, Gago Alonso et al., 2007], este tipo de medidas toma pares de objetos y verifica si pertenecen al mismo grupo y clase. Otros trabajos, en los cuales se aborda el problema del agrupamiento con traslape, para evaluar los resultados han utilizado medidas basadas en cotejo de conjuntos como F1-measure [Gil-García et al., 2003, Gil-García and Pons-Porrata, 2010]. Sin embargo, estas medidas tienen el problema de no considerar que, en un agrupamiento con traslape, los objetos que comparten n clases en el conjunto de datos etiquetado deberían compartir n agrupamientos en el resultado de un algoritmo de agrupamiento con traslape.

Para comparar los resultados de los algoritmos de agrupamiento en nuestros experimentos, se optó por utilizar FBcubed [Amigó et al., 2009] que explícitamente contempla el caso en que los grupos están traslapados. A continuación se describe la

métrica FBcubed:

FBcubed se calcula utilizando variaciones de las medidas Bcubed precision y Bcubed recall, propuestas en [Bagga and Baldwin, 1998]. Para calcular la FBcubed precisión y FBcubed Recall se utilizan las medidas de Multiplicity Precision y Multiplicity Recall, respectivamente. A continuación se describen estas medidas:

$$MultiplicityPrecision(x_i, x_j) = \frac{Min(|\pi(x_i) \cap \pi(x_j)|, |C(x_i) \cap C(x_j)|}{|\pi(x_i) \cap \pi(x_j)|} \tag{5.1}$$

$$MultiplicityRecall(x_i, x_j) = \frac{Min(|\pi(x_i) \cap \pi(x_j)|, |C(x_i) \cap C(x_j)|}{|C(x_i) \cap C(x_j)|} \tag{5.2}$$

donde: x_i y x_j son dos objetos, $C(x_i)$ son las clases asociadas a x_i y $\pi(x_i)$ son los grupos en los que se encuentra x_i. Estas expresiones solo están definidas cuando x_i y x_j comparten al menos un grupo (expresión 5.1) y cuando comparten al menos una clase (expresión 5.2). Estas funciones buscan penalizar el hecho de que dos objetos pertenezcan a un número distinto de grupos con respecto a las clases con las que originalmente fueron etiquetados, finalmente los valores son normalizados entre $[0, 1]$.

Sea $D(x_i)$ el conjunto de objetos que comparten al menos un grupo con x_i. La medida Bcubed Precision de x_i se calcula a partir de la siguiente expresión:

$$Bcubed_{precision}(x_i) = \frac{\sum_{x_j \in D(x_i)} MultiplicityPrecision(x_i, x_j)}{|D(x_i)|} \tag{5.3}$$

Sea $H(x_i)$ el conjunto de objetos que comparten al menos una clase con x_i. La medida Bcubed Recall de x_i se calcula a partir de la siguiente expresión:

$$Bcubed_{recall}(x_i) = \frac{\sum_{x_j \in H(x_i)} MultiplicityRecall(x_i, x_j)}{|H(x_i)|} \tag{5.4}$$

Finalmente, la medida FBcubed se define como:

$$FBcubed(X) = \frac{2(\frac{1}{|X|}\sum_{x \in X} Bcubed_{precision}(x))(\frac{1}{|X|}\sum_{x \in X} Bcubed_{recall}(x))}{(\frac{1}{|X|}\sum_{x \in X} Bcubed_{precision}(x)) + (\frac{1}{|X|}\sum_{x \in X} Bcubed_{recall}(x))} \tag{5.5}$$

Esta medida satisface una serie de restricciones que fueron propuestas en [Amigó et al., 2009], las cuales ayudan a determinar qué tan buena es una medida de evaluación para algoritmos de agrupamiento. En FBcubed un valor mayor supone una mejor evaluación para el agrupamiento y por lo tanto, mejor es el algoritmo con el que fue generado dicho agrupamiento.

Una consideración a tener en cuenta, al evaluar los resultados de los algoritmos de agrupamiento para flujos de datos es el hecho de que la mayoría de estos algoritmos remueven objetos en distintos periodos de tiempo, por lo que la evaluación de resultados se hace en distintos intervalos de tiempo.

5.2 Conjuntos de datos de prueba y parámetros

Dado que el algoritmo propuesto en esta tesis de maestría está ideado para construir grupos con traslape, éste debe ser evaluado sobre conjuntos de datos en los que existe traslape. Por otra parte, ya que el algoritmo propuesto también está ideado para trabajar en flujos de datos, debe ser evaluado en conjuntos de gran tamaño que permiten simular un flujo de datos. Por lo anterior, se obtuvieron conjuntos de prueba previamente etiquetados con tamaños que van desde los 40,000 hasta los 200,000 objetos; los cuales han sido utilizados para evaluar algoritmos de agrupamiento con traslape reportados en la literatura y se pueden encontrar en el repositorio de la página http://mulan.sourceforge.net/datasets-mlc.html. Los conjuntos de datos utilizados se encuentran descritos en la tabla 5.1. En esta tabla, la columna "Traslape" reporta el número promedio de clases a las cuales pertenece cada objeto en el conjunto de datos.

Un problema al utilizar DClustR, en conjuntos con gran cantidad de objetos, es que tiene un alto costo en espacio y tiempo de ejecución. Debido a este problema se optó por dividir los tres conjuntos de datos mostrados en la tabla 5.1 en cinco subconjuntos cada uno. Además, sólo se hizo la evaluación en cada subconjunto

Tabla 5.1: Conjuntos de datos utilizados para la evaluación de los algoritmos

Nombre	Objetos	Atributos	Traslape
Mediamill [Snoek et al., 2006]	43,907	120	4.376
NUSWIDE-cVLAD+ [Xioufis et al., 2014]	200,000	128	1.869
NUSWIDE-BoW [Chua et al., 2009]	200,000	500	1.869

hasta el punto en que DClustR era capaz de procesarlo. Por último, las evaluaciones se hicieron cada vez que se agrupaban 2000 objetos, esto último para determinar qué tanto variaba el comportamiento de los algoritmos DClustR y OCDS.

Para los experimentos, los atributos fueron normalizados al intervalo $[0, 1]$ utilizando la expresión 5.6. Normalizar es importante para evitar que variables que toman valores muy grandes sesguen la evaluación de la distancia entre objetos.

$$x_{ij}normalizado = \frac{x_{ij} - min(x_{ij})}{max(x_{ij}) - min(x_{ij})} \tag{5.6}$$

Parámetros

En el caso de DClustR se tiene como único parámetro la similitud mínima que deben tener dos objetos para que exista una arista que los conecte en el grafo de similitud G_β. En este caso se determinó el mejor valor para este parámetro de manera experimental haciendo una evaluación en el rango de valores $[0.5, 0.9]$ con incrementos de 0.1, dicha evaluación se hizo sobre los primeros dos intervalos de 2000 objetos para cada conjunto de datos; es decir, se evaluó el resultado después de agrupar 2000 objetos, después se hizo lo mismo con los siguientes 2000 objetos y posteriormente, se promediaron las evaluaciones obtenidas. Para este parámetro se eligió el valor con el que mejor evaluación se obtuvo. La función de similitud utilizada fue la distancia Euclidiana.

El algoritmo propuesto OCDS tiene distintos parámetros, todos ellos se heredan

del algoritmo DenStream. El parámetro ϵ sirve para delimitar el máximo tamaño que puede tener un *micro-cluster*, esto es para evitar que se incluyan objetos que se encuentran muy lejos del centro del *micro-cluster*. Este parámetro también afecta en el número de *micro-clusters* creados, ya que a menores valores para este parámetro son necesarios más *micro-clusters* para cubrir el conjunto de objetos. El parámetro ϵ está fuertemente ligado al parámetro μ. A mayor valor de μ mayor debe ser el valor de ϵ, pues μ determina el número mínimo de objetos que debe contener un *mcp* y si el valor de ϵ es pequeño, puede que nunca existan suficientes objetos en un *micro-cluster* para que se convierta en *mcp*, por lo que se consideraría a la mayoría de *micro-clusters* como *mca*.

En [Cao et al., 2006] se menciona una heurística para asignar valores a los parámetros ϵ y μ. Sin embargo, también se menciona que es difícil elegir valores apropiados, ya que se necesitaría mucha información a priori.

Para seleccionar el valor del parámetro ϵ se probaron diferentes valores en el rango de $[0.2, 4]$ con incrementos de 0.2 y se evaluó OCDS en los primeros 6000 objetos, en algunos casos no fue necesario seguir probando más valores pues a partir de cierto valor solo se creaba un *micro-cluster* con todos los objetos. Finalmente, se seleccionaron los valores para este parámetro que mejores resultados obtuvieron.

Para el parámetro μ se probaron los valores (20, 30, 40, 50 y 60) en los primeros 6000 objetos de cada conjunto de datos. Finalmente se seleccionó el valor que mejor resultados obtuvo en cada conjunto.

El parámetro λ se elige de acuerdo a la exigencia de la aplicación real. Si se tiene un flujo de datos en el que llegan muchos objetos por unidad de tiempo y que además cambia constantemente la forma y número de sus grupos, se debe elegir un valor de λ alto. En [Cao et al., 2006] se prueba un rango de valores para el parámetro λ y se muestra que el rango en el que se obtienen los mejores resultados es $[0.125, 1]$. En nuestros experimentos, se utilizó el valor 0.25, pues dicho valor es el que se

recomienda en la literatura, ya que permite no eliminar *micro-clusters* demasiado rápido ni mantenerlos demasiado tiempo.

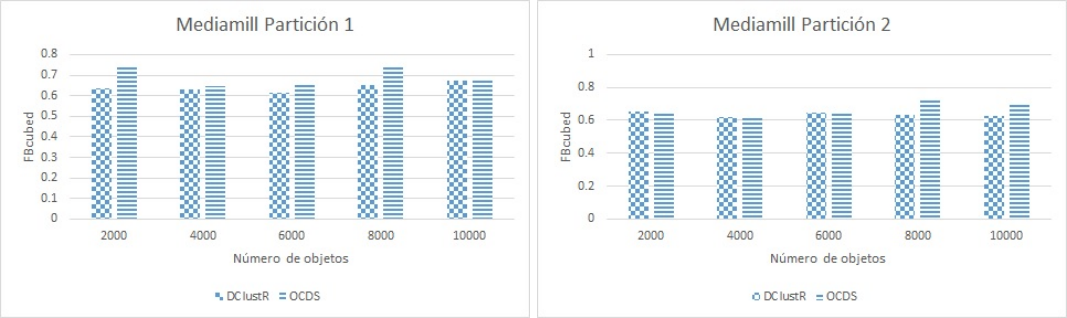
El último parámetro utilizado es β, el cual determina la tolerancia al ruido en los *micro-clusters*, y la frecuencia con que son removidos los *mca*. En [Cao et al., 2006] se prueban distintos valores y finalmente, se sugiere utilizar valores en el rango $[0.2, 0.6]$. En los experimentos se probaron los valores (0.2, 0.3, 0.4, 0.5, 0.6), al final se optó por el valor 0.2 que fue el que mejores resultados obtuvo, sin embargo, cabe aclarar que no hay mucha diferencia en los resultados obtenidos con los demás valores.

5.3 Comparación en calidad

En esta sección se presenta una comparación en términos de la calidad de los grupos construidos por los algoritmos OCDS y DClustR. Como se mencionó anteriormente, cada conjunto de prueba fue dividido en subconjuntos. Estos subconjuntos son evaluados en distintas ventanas de tiempo. Una ventana en este caso representa un subconjunto de 2000 objetos. Cabe aclarar que en todas las particiones DClustR solo llegó a agrupar un máximo de 10,000, pues al llegar a esa cantidad la memoria se agotó.

5.3.1 Mediamill

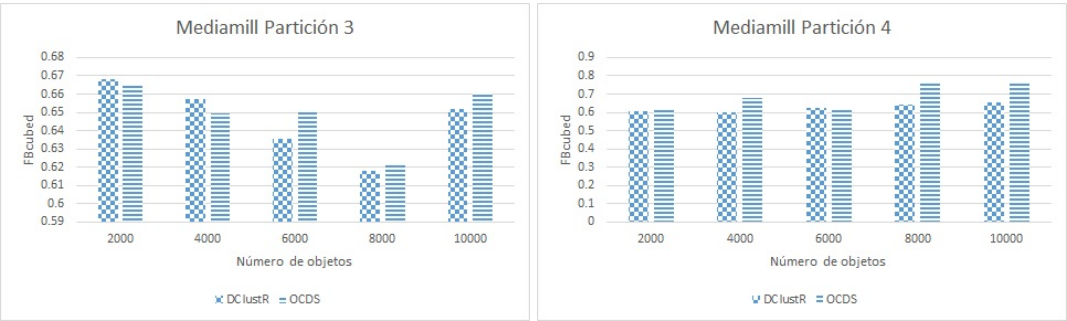
A partir del conjunto Mediamill [Snoek et al., 2006] se obtuvieron 5 subconjuntos de casi 9000 objetos. Las figuras 5.1, 5.2 y 5.3 muestran los resultados obtenidos por los algoritmos DClustR y OCDS sobre las particiones creadas a partir del conjunto Mediamill. Una ventana en este caso representa un subconjunto de 2000 objetos de una partición. Para DClustR se utilizó un valor de 0.75 como similitud mínima para conectar dos objetos por una arista. En el caso de OCDS se utilizaron



(a) (b)

Figura 5.1: Resultados obtenidos en la partición 1 y 2 del conjunto Mediamill utilizando FBcubed, DClustR (cuadros) y OCDS (líneas).

los parámetros $\epsilon = 1.43$, $\beta = 0.2$, $\mu = 30$, $\lambda = 0.25$.



(a) (b)

Figura 5.2: Resultados obtenidos en la partición 3 y 4 del conjunto Mediamill utilizando FBcubed, DClustR (cuadros) y OCDS (líneas).

En la gráfica de la figura 5.1.a se puede apreciar que el algoritmo OCDS obtuvo una mejor evaluación en todas las ventanas. En la gráfica de la figura 5.1.b DClustR solo obtuvo una mejor evaluación en la ventana 1, en las ventanas 2, 3, 4 y 5 OCDS logró una mejor evaluación; sin embargo, en las ventanas 2 y 3 el resultado en ambos algoritmos fue muy cercano.

En la gráfica de la figura 5.2.a DClustR obtuvo una mejor evaluación en las

ventanas 1 y 2, en las demás ventanas OCDS obtuvo un mejor resultado. En la gráfica de la figura 5.2.b, DClustR y OCDS obtuvieron el mismo resultado en la ventana 1, en la ventana 3 DClustR obtuvo un mejor resultado y en las ventanas 2, 3 y 5 OCDS logró una mejor evaluación.

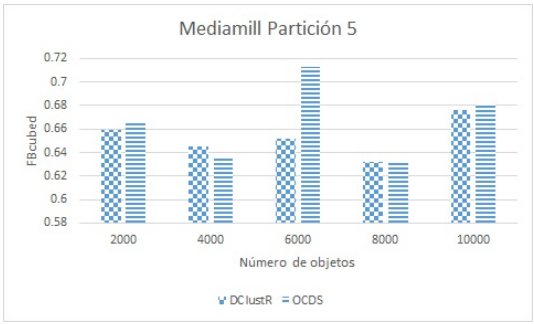


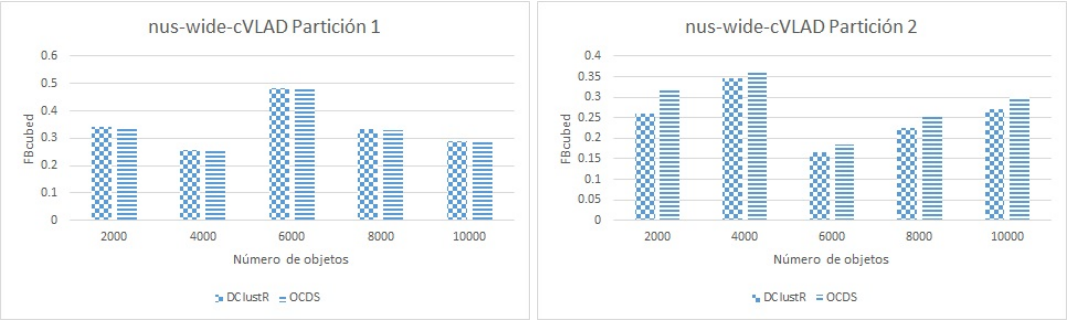
Figura 5.3: Resultados obtenidos en la partición 5 del conjunto Mediamill utilizando FB-cubed, DClustR (cuadros) y OCDS (líneas).

Por último, en la gráfica de la figura 5.3 se puede apreciar que el algoritmo DClustR obtuvo un mejor resultado en la ventana 4. En las demás ventanas OCDS logro una mejor evaluación.

5.3.2 NUSWIDE-cVLAD+

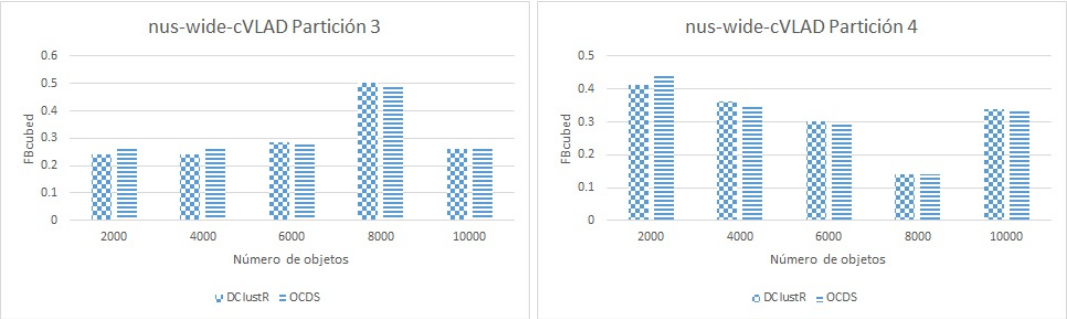
Con el conjunto NUSWIDE-cVLAD+ [Xioufis et al., 2014] se construyeron 5 subconjuntos de alrededor de 40,000 objetos. Las figuras 5.4, 5.5 y 5.6 muestran los resultados obtenidos por los algoritmos DClustR y OCDS sobre las particiones creadas a partir del conjunto NUSWIDE-cVLAD+. Para DClustR se utilizó un valor de 0.82 como similitud mínima para conectar dos objetos por una arista. En el caso de OCDS se utilizaron los parámetros $\epsilon = 1.45$, $\beta = 0.2$, $\mu = 50$, $\lambda = 0.25$.

En la gráfica de la figura 5.4.a se puede apreciar que ambos algoritmos obtuvieron resultados muy similares; sin embargo, a pesar de que no es fácilmente apreciable en la gráfica, DClustR obtuvo mejores resultados en todas las ventanas. En la gráfica de la figura 5.4.b OCDS logró un mejor resultado en todas las ventanas.



(a) (b)

Figura 5.4: Resultados obtenidos en la partición 1 y 2 del conjunto NUSWIDE-cVLAD+ utilizando FBcubed, DClustR (cuadros) y OCDS (líneas).



(a) (b)

Figura 5.5: Resultados obtenidos en la partición 3 y 4 del conjunto NUSWIDE-cVLAD+ utilizando FBcubed, DClustR (cuadros) y OCDS (líneas).

En la gráfica de la figura 5.5.a OCDS obtuvo un mejor resultado en las ventanas 1 y 2, en las demás ventanas DClustR logró una mejor evaluación. En la gráfica de la figura 5.5.b OCDS solo logró obtener una mejor evaluación en la ventana 1, DClustR obtuvo mejores resultados en las demás ventanas.

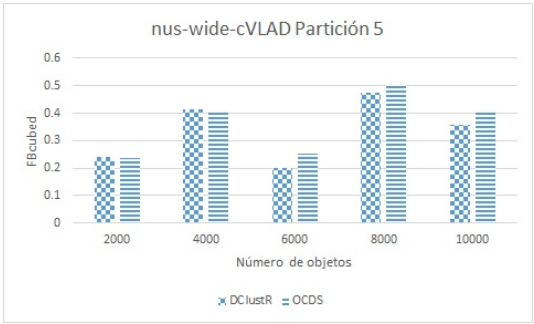


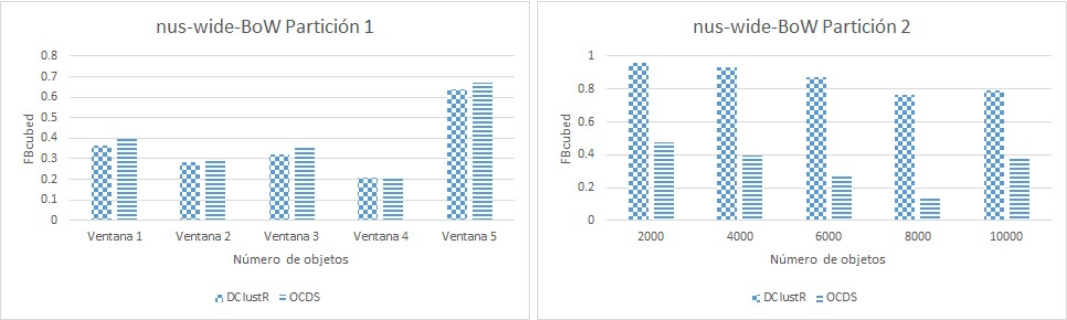
Figura 5.6: Resultados obtenidos en la partición 5 del conjunto NUSWIDE-cVLAD+ utilizando FBcubed, DClustR (cuadros) y OCDS (líneas).

Por último, en la gráfica de la figura 5.6 se puede apreciar que el algoritmo DClustR obtuvo un mejor resultado en las ventanas 1 y 2. En las demás ventanas OCDS logró una mejor evaluación. El comportamiento de los algoritmos en todas las particiones de este conjunto de objetos fue muy parecido.

5.3.3 NUSWIDE-BoW

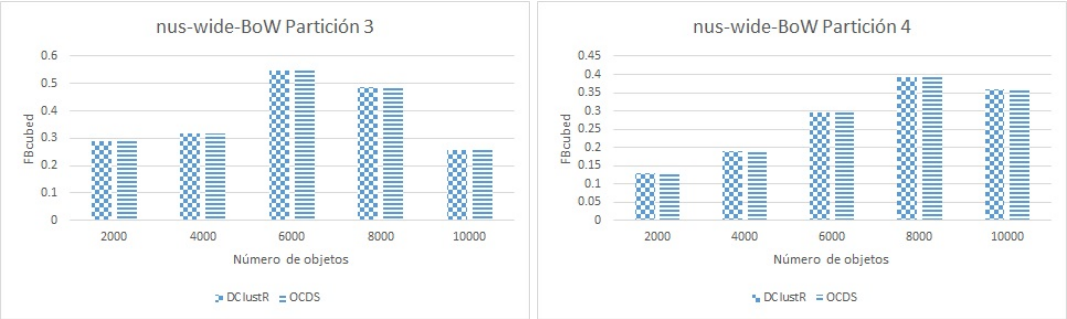
A partir del conjunto NUSWIDE-BoW [Chua et al., 2009] se generaron 5 subconjuntos de alrededor de 40,000 objetos, las figuras 5.7, 5.8 y 5.9 muestran los resultados obtenidos por los algoritmos DClustR y OCDS sobre las particiones creadas a partir del conjunto NUSWIDE-BoW. Para DClustR se utilizó un valor de 0.85 como similitud mínima para conectar dos objetos por una arista. En el caso de OCDS se utilizaron los parámetros $\epsilon = 2.75$, $\beta = 0.2$, $\mu = 40$, $\lambda = 0.25$.

En la gráfica de la figura 5.7.a se puede ver que el algoritmo OCDS logró obtener una mejor evaluación en todas las ventanas de la partición 1. En la gráfica de la figura 5.7.b fue DClustR el que obtuvo mejores resultados en todas las ventanas.



(a) (b)

Figura 5.7: Resultados obtenidos en la partición 1 y 2 del conjunto NUSWIDE-BoW utilizando FBcubed, DClustR (cuadros) y OCDS (líneas).



(a) (b)

Figura 5.8: Resultados obtenidos en la partición 3 y 4 del conjunto NUSWIDE-BoW utilizando FBcubed, DClustR (cuadros) y OCDS (líneas).

Algo que es importante recalcar aquí es que en algunas ventanas DClustR obtuvo un valor de evaluación de más del doble con respecto a OCDS.

En la gráfica de la figura 5.8.a se puede ver que ambos algoritmos lograron resultados muy similares; sin embargo, OCDS solo obtuvo un mejor resultado en la ventana 1, en las demás ventanas DClustR logró una mejor evaluación. En la gráfica de la figura 5.8.b no se aprecia mucha diferencia en los resultados obtenidos en las distintas ventanas, en este caso OCDS solo logró obtener una mejor evaluación en la ventana 5, DClustR obtuvo mejores resultados en las ventanas restantes.

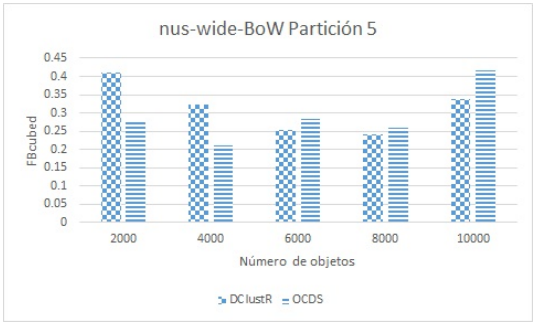


Figura 5.9: Resultados obtenidos en la partición 5 del conjunto NUSWIDE-BoW utilizando FBcubed, DClustR (cuadros) y OCDS (líneas).

Por último, en la gráfica de la figura 5.9 se puede apreciar que el algoritmo DClustR obtuvo un mejor resultado en las ventanas 1 y 2. En las demás ventanas OCDS logró una mejor evaluación.

5.3.4 Prueba de significancia estadística

Finalmente, la tabla 5.2 presenta el promedio de calidad por cada partición, el valor mayor para cada partición aparece resaltado en negritas, esta tabla se utilizó para hacer la prueba de significancia estadística y así poder determinar si existe diferencia significativa entre los resultados obtenidos por ambos algoritmos.

En la tabla 5.2 se puede apreciar que en promedio OCDS obtuvo mejores

Tabla 5.2: Promedio de calidad obtenido en cada partición con DClustR y OCDS

Nombre	DClustR	OCDS
Mediamill Partición 1	0.6409943003	**0.6916577579**
Mediamill Partición 2	0.6364532403	**0.6694717998**
Mediamill Partición 3	0.6462861157	**0.6495283958**
Mediamill Partición 4	0.6264687796	**0.686319537**
Mediamill Partición 5	0.6528311958	**0.6659672895**
NUSWIDE-cVLAD+ Partición 1	**0.3398539604**	0.3376907886
NUSWIDE-cVLAD+ Partición 2	0.2539534884	**0.2860894096**
NUSWIDE-cVLAD+ Partición 3	0.306411321	**0.3139214327**
NUSWIDE-cVLAD+ Partición 4	0.3116148869	**0.3131141093**
NUSWIDE-cVLAD+ Partición 5	0.3375263836	**0.3611718649**
NUSWIDE-BoW+ Partición 1	0.3625413921	**0.3862541321**
NUSWIDE-BoW Partición 2	**0.8646041282**	0.3295582635
NUSWIDE-BoW Partición 3	**0.3804120332**	0.378752102
NUSWIDE-BoW Partición 4	**0.2730701437**	0.2724828174
NUSWIDE-BoW Partición 5	**0.3128840581**	0.2895166403

resultados en el conjunto Mediamill, pues en todas las particiones logró una mejor evaluación de FBcubed. En el caso de NUSWIDE-cVLAD+ DClustR obtuvo una mejor evaluación para la primera partición y en las demás OCDS logró mejores resultados. Finalmente en NUSWIDE-BoW OCDS solo obtuvo un mejor resultado en la primera partición, mientras que en las demás particiones DClustR fue el que mejores resultados obtuvo.

Para determinar la significancia estadística entre los valores de la tabla 5.2, se utilizó el test de Mann-Whitney, el cual también fue utilizado en [Pérez-Suárez et al., 2013a], en este caso se utilizó un 95 % de certeza. A partir de los resultados obtenidos por el test, se concluye que no hay diferencia significativa

entre los resultados obtenidos por DClustR y OCDS. Una información detallada acerca de este test, así como una implementación del test pueden encontrarse en el sitio http://vassarstats.net/utest.html.

De los resultados obtenidos en términos de calidad se puede concluir que OCDS tiene un comportamiento similar al de DClustR, esto puede deberse a que en la fase *offline* de OCDS utiliza el algoritmo OClustR, el cual es la versión estática de DClustR, pero en este caso OClustR es ajustado para trabajar con la representación de objetos que utiliza OCDS.

5.4 Comparación en tiempo de ejecución

Para la comparación de tiempo de ejecución se utilizaron los conjuntos originales; es decir, los mostrados en la tabla 5.1 y se ejecutaron ambos algoritmos sobre dichos conjuntos. En el caso de OCDS se le hizo una solicitud para ejecutar la fase *offline* cada que llegaban 2000 objetos y se reporta el tiempo acumulado, mientras que en DClustR los objetos se agruparon de 2000 en 2000 y se reporta el tiempo de agrupar el conjunto ampliado. En las figuras 5.10, 5.11 y 5.12 se muestran los resultados que obtuvieron ambos algoritmos en intervalos de 2000 objetos, para cada conjunto de datos.

En la figura 5.10 se puede observar que el algoritmo OCDS tiene un menor tiempo de ejecución que DClustR, pues para procesar todo el conjunto de datos le toma casi el mismo tiempo que a DClustR le tomó procesar tan solo 4000 objetos del conjunto.

En la figura 5.11 se puede ver que para este conjunto sucede lo mismo que en Mediamill pues para cuando OCDS termina de procesar todo el conjunto, DClustR apenas ha procesado poco más de 4000 objetos.

En los resultados de la figura 5.12 sucede lo mismo que en los resultados an-

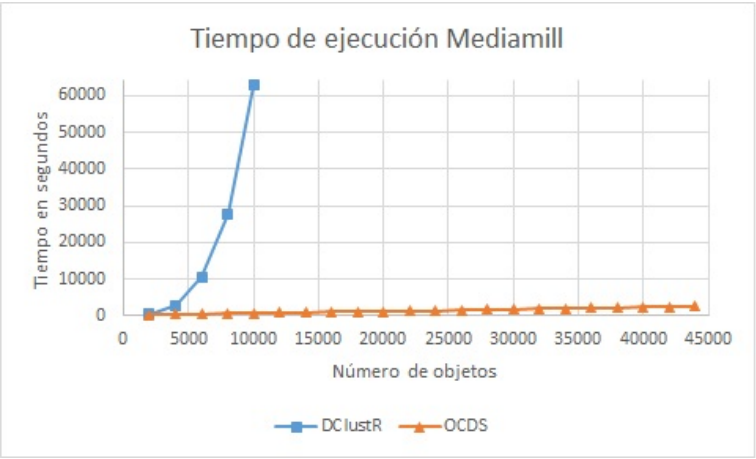


Figura 5.10: Tiempo de ejecución en el conjunto de datos Mediamill, DClustR (cuadros) y OCDS (triángulos).

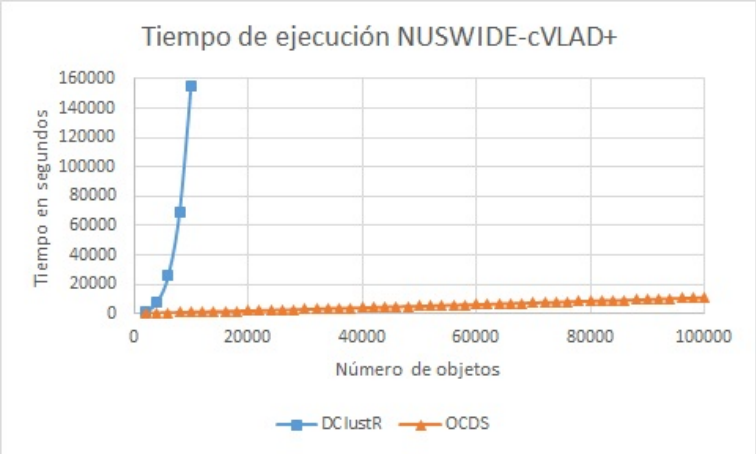


Figura 5.11: Tiempo de ejecución en el conjunto de datos NUSWIDEcVLAD+, DClustR (cuadros) y OCDS (triángulos).

Figura 5.12: Tiempo de ejecución en el conjunto de datos NUSWIDEBoW, DClustR (cuadros) y OCDS (triángulos).

teiores. En este caso DClustR fue capaz de procesar 16,000 objetos, posiblemente debido que no se crearon muchas aristas en el grafo y esto ayudó a que se consumiera menos memoria del sistema. Sin embargo, en términos de tiempo OCDS sigue siendo muy superior pues al igual que en los otros casos, este fue capaz de procesar todo el conjunto en un tiempo mucho menor.

Tabla 5.3: Velocidad de OCDS comparado con DClustR.

# de objetos / Conjunto de datos	2000	4000	6000	8000	10,000	**Promedio**
Mediamill	1.84	7.91	22.02	48.76	93.19	**34.75**
NUSWIDE-cVLAD+	3.21	13.52	33.63	70.87	128.12	**49.87**
NUSWIDE-BoW	0.91	3.39	8.62	16.26	30.43	**11.92**

En la tabla 5.3 se muestra la velocidad de OCDS comparado con DClustR (tiempo de $DClustR$/tiempo de $OCDS$), de lo obtenido se puede observar que el algoritmo OCDS es más rápido mientras se tengan más objetos a procesar, en este caso solo se muestra cuán más rápido fue para los primeros 10,000 objetos ya que a DClustR le tomaba demasiado tiempo procesar más de esa cantidad de objetos. En

promedio el algoritmo OCDS fue 32 veces más rápido que DClustR; sin embargo, si se tomaran más objetos el algoritmo OCDS sería aún más rápido, ya que los tiempos del algoritmo DClustR se disparan cuando se añaden más objetos como se mostró en las gráficas de comparación de tiempo.

La razón por la cual el algoritmo propuesto (OCDS) es mucho más rápido que DClustR es por el hecho de que OCDS utiliza una fase *online* en la cual los objetos son representados por *micro-clusters* y cada objeto solo es procesado una vez. El agrupamiento final es construido en la fase *offline* utilizando el algoritmo OClustRmc, el cual agrupa los *micro-clusters* obtenidos de la fase *online*. En [Cao et al., 2006] se demuestra que el crecimiento del número de *micro-clusters* con respecto a un número n de objetos es de $\log n$; entonces si tomamos el número de objetos a procesar, DClustR es de orden $O(n^2)$, mientras que el algoritmo OClustRmc es de orden $O(m^2)$, siendo m el número de *micro-clusters* de entrada para OClustRmc, además $m << n$. Cabe mencionar que el algoritmo propuesto solo hace la llamada a OClustRmc cuando se hace una solicitud para hacer el agrupamiento de los *micro-clusters* que en el caso de los experimentos aquí mostrados, fue cada 2000 objetos.

Finalmente, a partir de los resultados obtenidos se puede concluir que OCDS es capaz de procesar flujos de datos, además de conjuntos de datos de gran tamaño, pues DenStream solo necesita procesar los objetos cuando éstos llegan al flujo de datos. Además, se cumplió con el objetivo planteado pues el algoritmo propuesto en esta tesis es más rápido y mantiene una calidad similar que DClustR que es uno de los mejores algoritmos incrementales que se han desarrollado para el agrupamiento con traslape.

Capítulo 6

Conclusiones y trabajo futuro

En este capítulo se expone un breve sumario del trabajo realizado en esta tesis. Posteriormente se dan las conclusiones y aportaciones que se obtuvieron. Finalmente, se menciona el trabajo futuro derivado de la investigación.

6.1 Sumario

El desarrollo de algoritmos de agrupamiento para flujos de datos es una línea de investigación activa debido a su gran potencial para el análisis de datos en una amplia variedad de aplicaciones, principalmente en aquellas en las que la información es generada continuamente y se requiere un análisis en distintos períodos de tiempo. Existen varios tipos de aplicaciones como análisis de flujos de noticias, detección de intrusos en redes, detección de fraudes telefónicos, entre otras, donde es común que los objetos puedan pertenecer a varios grupos; no obstante, hasta la fecha no han sido reportados en la literatura algoritmos de agrupamiento para flujos de datos que permita la construcción de grupos con traslape, solo se han desarrollado algoritmos de agrupamiento para flujos de datos que construyen grupos disjuntos. Debido a lo mencionado anteriormente, en este trabajo se desarrolló un algoritmo de agrupamiento con traslape para flujos de datos.

Se realizaron distintos experimentos utilizando conjuntos de datos previamente

etiquetados, en los cuales existe traslape entre sus clases. Los experimentos se enfocaron en comparar la calidad y el tiempo de ejecución entre el algoritmo propuesto y DClustR, el cual es un algoritmo incremental para el agrupamiento con traslape, que ha reportado los mejores resultados en la literatura.

6.2 CONCLUSIONES

Se realizaron distintos experimentos con conjuntos de datos estándar utilizando la métrica FBcubed para determinar qué tan bueno es el algoritmo propuesto, comparándolo con el algoritmo de agrupamiento incremental DClustR que es el algoritmo incremental que mejores resultados ha reportado en la literatura. A partir de los resultados obtenidos se puede concluir que el algoritmo propuesto construye agrupamientos con una calidad similar a los construidos por DClustR, pero con la ventaja de que el algoritmo propuesto puede trabajar en flujos de datos mientras que DClustR no.

Con los mismos conjuntos de datos de hizo una comparación del tiempo de ejecución del algoritmo propuesto y con lo obtenido se concluye que el algoritmo propuesto es mucho más rápido que el algoritmo incremental DClustR.

Finalmente, se puede concluir que se cumplió con el objetivo de esta investigación y en tal sentido, el algoritmo propuesto es una alternativa adecuada para procesar flujos de datos en aplicaciones donde es común que los objetos pertenezcan a más de un grupo. Sin embargo, el ajuste de parámetros es algo que se debe mejorar, pues en particular para los parámetros ϵ y μ no hay un criterio definido para guiar la selección de sus valores.

6.3 APORTACIONES

Las aportaciones de esta investigación son las siguientes:

- Un algoritmo de agrupamiento para flujos de datos, que permite obtener un agrupamiento con traslape, con una calidad de agrupamiento similar a la de los algoritmos incrementales de agrupamiento con traslape, pero en un tiempo mucho menor.

- Otra aportación es la fase *online* del algoritmo propuesto ya que permite obtener mejores resultados que el algoritmo DenStream, utilizando la misma fase *offline* de DenStream. Esta variante permite remover *micro-clusters* poco útiles independientemente del valor del parámetro de desvanecimiento.

6.4 TRABAJO FUTURO

Se estudiará el problema de la selección de los parámetros ϵ y μ para proponer alguna heurística (sin tener que ejecutar el algoritmo) que ayude a seleccionar valores para estos parámetros.

El algoritmo OCDS fue diseñado para trabajar con conjuntos de datos descritos por atributos numéricos, por lo tanto, como trabajo futuro se propone la extensión del mismo para poder trabajar con atributos mezclados (numéricos y no numéricos). Otro trabajo futuro es el desarrollo de algoritmos de agrupamiento con traslape para flujos de datos haciendo uso de GPUs o arquitecturas en las cuales se pueda hacer procesamiento paralelo, ésta es una línea de investigación muy importante, pues esto permitiría hacer un procesamiento aún más rápido de los flujos de datos.

6.5 PUBLICACIONES

Como producto de esta investigación se tiene la siguiente publicación en congreso internacional arbitrado:

1. Aroche-Villarruel, A. A., et al. (2015). A Different Approach for Pruning Microclusters in Data Stream Clustering. In Pattern Recognition, LNCS 9116, pp. 33-43, 2015, Springer.

Bibliografía

[Aggarwal et al., 2003] Aggarwal, C. C., Han, J., Wang, J., and Yu, P. S. (2003). A framework for clustering evolving data streams. In *VLDB*, pages 81–92.

[Aggarwal et al., 2004] Aggarwal, C. C., Han, J., Wang, J., and Yu, P. S. (2004). A framework for projected clustering of high dimensional data streams. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 852–863. VLDB Endowment.

[Aggarwal et al., 2005] Aggarwal, C. C., Han, J., Wang, J., and Yu, P. S. (2005). On high dimensional projected clustering of data streams. *Data Min. Knowl. Discov.*, 10(3):251–273.

[Aggarwal and Yu, 2008] Aggarwal, C. C. and Yu, P. S. (2008). A framework for clustering uncertain data streams. In Alonso, G., Blakeley, J. A., and Chen, A. L. P., editors, *ICDE*, pages 150–159. IEEE.

[Agrawal et al., 1998] Agrawal, R., Gehrke, J., Gunopulos, D., and Raghavan, P. (1998). *Automatic subspace clustering of high dimensional data for data mining applications*, volume 27. ACM.

[Amigó et al., 2009] Amigó, E., Gonzalo, J., Artiles, J., and Verdejo, F. (2009). A comparison of extrinsic clustering evaluation metrics based on formal constraints. *Inf. Retr.*, 12(5):613.

[Amini et al., 2014a] Amini, A., Saboohi, H., Wah, T. Y., and Herawan, T. (2014a). Dmm-stream: A density mini-micro clustering algorithm for evolving data streams. In *Proceedings of the First International Conference on Advanced Data and Information Engineering (DaEng-2013)*, pages 675–682. Springer.

[Amini et al., 2014b] Amini, A., Teh, Y. W., and Saboohi, H. (2014b). On density-based data streams clustering algorithms: A survey. *J. Comput. Sci. Technol.*, 29(1):116–141.

[Aslam et al., 2004] Aslam, J. A., Pelekhov, E., and Rus, D. (2004). The star clustering algorithm for static and dynamic information organization. *J. Graph Algorithms Appl.*, 8:95–129.

[Aslam et al., 1998] Aslam, J. A., Pelekhov, K., and Rus, D. (1998). Static and dynamic information organization with star clusters. In Gardarin, G., French, J. C., Pissinou, N., Makki, K., and Bouganim, L., editors, *CIKM*, pages 208–217. ACM.

[Aslam et al., 2000] Aslam, J. A., Pelekhov, K., and Rus, D. (2000). Using star clusters for filtering. In *CIKM*, pages 306–313. ACM.

[Babcock et al., 2003] Babcock, B., Datar, M., Motwani, R., and O'Callaghan, L. (2003). Maintaining variance and k-medians over data stream windows. In Neven, F., Beeri, C., and Milo, T., editors, *PODS*, pages 234–243. ACM.

[Bagga and Baldwin, 1998] Bagga, A. and Baldwin, B. (1998). Entity-Based Cross-Document Coreferencing Using the Vector Space Model. In *Proceedings of the 17th International Conference on Computational Linguistics*, pages 79–85, Morristown, NJ, USA. Association for Computational Linguistics.

[Banerjee et al., 2005] Banerjee, A., Krumpelman, C., Ghosh, J., Basu, S., and Mooney, R. J. (2005). Model-based overlapping clustering. In Grossman, R., Bayardo, R., and Bennett, K. P., editors, *KDD*, pages 532–537. ACM.

[Bhatnagar and Kaur, 2007] Bhatnagar, V. and Kaur, S. (2007). Exclusive and complete clustering of streams. In Wagner, R., Revell, N., and Pernul, G., editors, *DEXA*, volume 4653 of *Lecture Notes in Computer Science*, pages 629–638. Springer.

[Cao et al., 2006] Cao, F., Ester, M., Qian, W., and Zhou, A. (2006). Density-based clustering over an evolving data stream with noise. In Ghosh, J., Lambert, D., Skillicorn, D. B., and Srivastava, J., editors, *SDM*. SIAM.

[Chua et al., 2009] Chua, T.-S., Tang, J., Hong, R., Li, H., Luo, Z., and Zheng, Y. (2009). Nus-wide: a real-world web image database from national university of singapore. In Marchand-Maillet, S. and Kompatsiaris, Y., editors, *CIVR*. ACM.

[Cleuziou, 2008] Cleuziou, G. (2008). An extended version of the k-means method for overlapping clustering. In *ICPR*, pages 1–4. IEEE Computer Society.

[Cleuziou, 2009] Cleuziou, G. (2009). Two variants of the okm for overlapping clustering. In Guillet, F., Ritschard, G., Zighed, D. A., and Briand, H., editors, *EGC (best of volume)*, volume 292 of *Studies in Computational Intelligence*, pages 149–166. Springer.

[Dang et al., 2009] Dang, X. H., Lee, V., Ng, W. K., Ciptadi, A., and Ong, K.-L. (2009). An em-based algorithm for clustering data streams in sliding windows. In Zhou, X., Yokota, H., Deng, K., and Liu, Q., editors, *DASFAA*, volume 5463 of *Lecture Notes in Computer Science*, pages 230–235. Springer.

[Dempster et al., 1977] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38. Series B.

[Ester et al., 1996] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with

noise. In *Proc. of 2nd International Conference on Knowledge Discovery and*, pages 226–231.

[Forestiero et al., 2013] Forestiero, A., Pizzuti, C., and Spezzano, G. (2013). A single pass algorithm for clustering evolving data streams based on swarm intelligence. *Data Min. Knowl. Discov.*, 26(1):1–26.

[Gago Alonso et al., 2007] Gago Alonso, A., Pérez Suárez, A., and Medina-Pagola, J. E. (2007). Acons: A new algorithm for clustering documents. In Rueda, L., Mery, D., and Kittler, J., editors, *CIARP*, volume 4756 of *Lecture Notes in Computer Science*, pages 664–673. Springer.

[Gao et al., 2005] Gao, J., Li, J., Zhang, Z., and Tan, P.-N. (2005). An incremental data stream clustering algorithm based on dense units detection. In Ho, T. B., Cheung, D. W.-L., and Liu, H., editors, *PAKDD*, volume 3518 of *Lecture Notes in Computer Science*, pages 420–425. Springer.

[Gil-García et al., 2003] Gil-García, R., BadÃa-Contelles, J. M., and Pons-Porrata, A. (2003). Extended star clustering algorithm. In Sanfeliu, A. and Ruiz-Shulcloper, J., editors, *CIARP*, volume 2905 of *Lecture Notes in Computer Science*, pages 480–487. Springer.

[Gil-García and Pons-Porrata, 2010] Gil-García, R. and Pons-Porrata, A. (2010). Dynamic hierarchical algorithms for document clustering. *Pattern Recognition Letters*, 31(6):469–477.

[Giraldo et al., 2013] Giraldo, F. A., León, E., and Gómez, J. (2013). Caracterización de flujos de datos usando algoritmos de agrupamiento. *Tecnura*, 17:153 – 166.

[Guha et al., 2002] Guha, S., Meyerson, A., Mishra, N., Motwani, R., and O'Callaghan, L. (2002). Streaming-data algorithms for high-quality clustering.

In Agrawal, R. and Dittrich, K. R., editors, *ICDE*, pages 685–694. IEEE Computer Society.

[Halkidi et al., 2001] Halkidi, M., Batistakis, Y., and Vazirgiannis, M. (2001). On clustering validation techniques. *Journal of Intelligent Information Systems*, 17:107–145.

[Hammouda and Kamel, 2004] Hammouda, K. M. and Kamel, M. S. (2004). Efficient phrase-based document indexing for web document clustering. *IEEE Trans. Knowl. Data Eng.*, 16(10):1279–1296.

[Han and Kamber, 2006] Han, J. and Kamber, M. (2006). *Data Mining. Concepts and Techniques*. Morgan Kaufmann, 2nd ed. edition.

[Hassani et al., 2012] Hassani, M., Spaus, P., Gaber, M. M., and Seidl, T. (2012). Density-based projected clustering of data streams. In HÃ$\frac{1}{4}$llermeier, E., Link, S., Fober, T., and Seeger, B., editors, *SUM*, volume 7520 of *Lecture Notes in Computer Science*, pages 311–324. Springer.

[Jia et al., 2008] Jia, C., Tan, C., and Yong, A. (2008). A grid and density-based clustering algorithm for processing data stream. In *Genetic and Evolutionary Computing, 2008. WGEC'08. Second International Conference on*, pages 517–521. IEEE.

[Kaufman and Rousseeuw, 1987] Kaufman, L. and Rousseeuw, P. (1987). *Clustering by means of medoids*. North-Holland.

[Kranen et al., 2011] Kranen, P., Assent, I., Baldauf, C., and Seidl, T. (2011). The clustree: indexing micro-clusters for anytime stream mining. *Knowl. Inf. Syst.*, 29(2):249–272.

[Larsen and Aone, 1999] Larsen, B. and Aone, C. (1999). Fast and effective text mining using linear-time document clustering. In *ACM SIGKDD*.

[Lee, 2012] Lee, C.-H. (2012). Mining spatio-temporal information on microblogging streams using a density-based online clustering method. *Expert Syst. Appl.*, 39(10):9623–9641.

[Lin and Lin, 2009] Lin, J. and Lin, H. (2009). A density-based clustering over evolving heterogeneous data stream. In *Computing, Communication, Control, and Management, 2009. CCCM 2009. ISECS International Colloquium on*, volume 4, pages 275–277. IEEE.

[Lu et al., 2005] Lu, Y., Sun, Y., Xu, G., and Liu, G. (2005). A grid-based clustering algorithm for high-dimensional data streams. In Li, X., Wang, S., and Dong, Z. Y., editors, *ADMA*, volume 3584 of *Lecture Notes in Computer Science*, pages 824–831. Springer.

[Lühr and Lazarescu, 2009] Lühr, S. and Lazarescu, M. (2009). Incremental clustering of dynamic data streams using connectivity based representative points. *Data Knowl. Eng.*, 68(1):1–27.

[MacQueen, 1967] MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. In Cam, L. M. L. and Neyman, J., editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press.

[Mete et al., 2011] Mete, M., Kockara, S., and Aydin, K. (2011). Fast density-based lesion detection in dermoscopy images. *Comp. Med. Imag. and Graph.*, 35(2):128–136.

[Ng and Dash, 2010] Ng, W. and Dash, M. (2010). Discovery of frequent patterns in transactional data streams. *T. Large-Scale Data- and Knowledge-Centered Systems*, 2:1–30.

[Ntoutsi et al., 2012] Ntoutsi, I., Zimek, A., Palpanas, T., Kröger, P., and Kriegel, H.-P. (2012). Density-based projected clustering over high dimensional data streams. In *SDM*, pages 987–998. SIAM / Omnipress.

[Pérez-Suárez et al., 2013a] Pérez-Suárez, A., Martínez-Trinidad, J. F., Carrasco-Ochoa, J. A., and Medina-Pagola, J. E. (2013a). An algorithm based on density and compactness for dynamic overlapping clustering. *Pattern Recognition*, 46(11):3040–3055.

[Pérez-Suárez et al., 2013b] Pérez-Suárez, A., Martínez-Trinidad, J. F., Carrasco-Ochoa, J. A., and Medina-Pagola, J. E. (2013b). Oclustr: A new graph-based algorithm for overlapping clustering. *Neurocomputing*, 121:234–247.

[Pérez Suárez et al., 2009] Pérez Suárez, A., Martínez-Trinidad, J. F., and Carrasco-Ochoa, Jesús Ariel an Medina-Pagola, J. E. (2009). A new incremental algorithm for overlapped clustering. In Bayro-Corrochano, E. and Eklundh, J.-O., editors, *CIARP*, volume 5856 of *Lecture Notes in Computer Science*, pages 497–504. Springer.

[Pfitzner et al., 2009] Pfitzner, D., Leibbrandt, R., and Powers, D. M. W. (2009). Characterization and evaluation of similarity measures for pairs of clusterings. *Knowl. Inf. Syst.*, 19(3):361–394.

[Plant et al., 2010] Plant, C., Teipel, S. J., Oswald, A., Böhm, C., Meindl, T., Miranda, J. M., Bokde, A. L. W., Hampel, H., and Ewers, M. (2010). Automated detection of brain atrophy patterns based on mri for the prediction of alzheimer's disease. *NeuroImage*, 50(1):162–174.

[Pons-Porrata et al., 2002] Pons-Porrata, A., Ruiz-Shulcloper, J., Berlanga-Llavori, R., and Santiesteban-Alganza, Y. (2002). Un algoritmo incremental para la obtención de cubrimientos con datos mezclados. In *Proceedings of CIARP*, pages 405–416.

[Ruiz et al., 2009] Ruiz, C., Ruiz, E. M., and Spiliopoulou, M. (2009). C-denstream: Using domain knowledge on a data stream. In Gama, J., Costa, V. S., Jorge, A. M., and Brazdil, P., editors, *Discovery Science*, volume 5808 of *Lecture Notes in Computer Science*, pages 287–301. Springer.

[Sheikholeslami et al., 1998] Sheikholeslami, G., Chatterjee, S., and Zhang, A. (1998). Wavecluster: A multi-resolution clustering approach for very large spatial databases. In *VLDB*, volume 98, pages 428–439.

[Snoek et al., 2006] Snoek, C. G., Worring, M., van Gemert, J. C., Geusebroek, J.-M., and Smeulders, A. W. (2006). The challenge problem for automated detection of 101 semantic concepts in multimedia. In *Proceedings of ACM Multimedia*, pages 421–430, Santa Barbara, CA.

[Tan et al., 2006] Tan, P., Steinbach, M., and Kumar, V. (2006). Introduction to data mining, ser.

[Tu and Chen, 2009a] Tu, L. and Chen, Y. (2009a). Stream data clustering based on grid density and attraction. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(3):12.

[Tu and Chen, 2009b] Tu, L. and Chen, Y. (2009b). Stream data clustering based on grid density and attraction. *TKDD*, 3(3).

[Wan et al., 2009] Wan, L., Ng, W. K., Dang, X. H., Yu, P. S., and Zhang, K. (2009). Density-based clustering of data streams at multiple resolutions. *TKDD*, 3(3).

[Wang et al., 1997] Wang, W., Yang, J., and Muntz, R. R. (1997). Sting: A statistical information grid approach to spatial data mining. In Jarke, M., Carey, M. J., Dittrich, K. R., Lochovsky, F. H., Loucopoulos, P., and Jeusfeld, M. A., editors, *Twenty-Third International Conference on Very Large Data Bases*, pages 186–195. Morgan Kaufmann.

[Xioufis et al., 2014] Xioufis, E. S., Papadopoulos, S., Kompatsiaris, Y., Tsoumakas, G., and Vlahavas, I. P. (2014). A comprehensive study over vlad and product quantization in large-scale image retrieval. *IEEE Transactions on Multimedia*, 16(6):1713–1728.

[Yang et al., 2011] Yang, D., Rundensteiner, E. A., and Ward, M. O. (2011). Summarization and matching of density-based clusters in streaming environments. *PVLDB*, 5(2):121–132.

[Yu et al., 2013] Yu, Y., Wang, Q., Wang, X., Wang, H., and He, J. (2013). Online clustering for trajectory data stream of moving objects. *Comput. Sci. Inf. Syst.*, 10(3):1293–1317.

[Zhang et al., 1999] Zhang, B., Kleyner, G., and Hsu, M. (1999). A local search approach to k-clustering. *HP LABORATORIES TECHNICAL REPORT HPL*, (119).

[Zhang et al., 2009] Zhang, C., Jin, C., and Zhou, A. (2009). Efficiently clustering probabilistic data streams. In *Advances in Data and Web Management, Joint International Conferences, APWeb/WAIM 2009, Suzhou, China, April 2-4, 2009, Proceedings*, pages 273–284.

[Zhang et al., 1996] Zhang, T., Ramakrishnan, R., and Livny, M. (1996). BIRCH: an efficient data clustering method for very large databases. In *SIGMOD*, pages 103–114.

[Zhao and Karypis, 2001] Zhao, Y. and Karypis, G. (2001). Criterion functions for document clustering: Experiments and analysis. Technical report.

[zu Eissen and Wißbrock, 2003] zu Eissen, B. S. S. M. and Wißbrock, F. (2003). On cluster validity and the information need of users.