UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO

FACULTAD DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA, INFORMÁTICA Y MECÁNICA

ESCUELA PROFESIONAL DE INGENIERÍA INFORMÁTICA Y DE SISTEMAS

TESIS

ALGORITMO PARA EL PROBLEMA DATA STREAMING CLUSTERING PARA CONJUNTOS AMORFOS Y CON OUTLIERS

Para optar al título profesional de:
INGENIERO INFORMÁTICO Y DE SISTEMAS

Presentado por:
BR. ISAAC CAMPOS ARDILES

Asesor:
DR. RONY VILLAFUERTE SERNA

Cusco - Perú
2021

Dedicado a mi madre Soraya quien guió mis primeros pasos, a mi querido hermano Fernando, con quien inicié esta tesis y por quien seré el profesional del cual quiero se sientan orgullosos, y a mi padre Edgar mi constante apoyo.

Resumen

Clustering es un campo importante dentro de *Machine Learning* y ha sido ampliamente estudiado durante varios años. Como resultado, se desarrollaron muchos algoritmos que resuelven este problema, los cuales tal como están planteados no pueden resolver el caso particular que sera objeto de este estudio. Por ello un nuevo problema llamado *Data Streaming Clustering* fue propuesto y fue objeto de investigación de muchos estudios. Este problema está definido como el *clustering* de un flujo de datos recibidos continuamente. *Data Streaming clustering* tiene como objetivo encontrar y mantener un conjunto de *clusters* válidos en un continuo y posiblemente ilimitado flujo de datos. Teniendo en cuenta las limitantes actuales en la tecnología como la capacidad de la memoria o limitaciones en el tiempo computacional. Es importante tener en cuenta que los algoritmos para el problema de *clustering* no pueden resolver eficientemente el problema en estudio sin una previa modificación, ya que no toman en cuenta estas características. Debido a las características de este problema los algoritmos planteados para resolver el problema de *Data Streaming clustering* pueden ser usados para minería de datos con características especiales como *outliers* o ruido en los datos, como grabaciones telefónicas, transacciones bancarias, información de redes sociales. En esta investigación se presenta el diseño e implementación de un algoritmo para *Data Streaming Clustering* para *datasets* con *clusters* irregulares, *outliers* y sin necesidad de un conocimiento previo del número de *clusters*, además se realiza un análisis y discusión sobre los resultados.

Palabras clave: *Clustering, Data Streaming Clustering, Estructura Disjoint-set, Feature Vector, SetClust, Micro-cluster*

Abstract

Clustering is a relevant field within *Machine Learning* and has been widely studied for several years. As a result, many algorithms were developed to solve this problem. Which as they are designed cannot elucidate the particular case that will be the object of this study. Therefore a new problem called Data Streaming Clustering was proposed and was investigated by many studies. This problem is defined as the clustering of a stream of continuously received data. Streaming data clustering aims to find and maintain a set of valid clusters in a continuous and possibly unbounded stream of data. Considering the current limitations in technology as the memory capacity or limitations on computational time. It is important to be aware of the current algorithms, that solve the problem of clustering, can not be applied to the current studied problem without changing its features. Due to the characteristics of this problem, the algorithms proposed to solve the Data Streaming clustering problem can be used for Data Mining with the special feature as outliers or noise, for instance, phone recordings, bank transactions, social network information. This research will present the design and implementation of an algorithm for Data Streaming Clustering for datasets with irregular clusters, outliers, and without needing the previous knowledge of the actual number of clusters, furthermore, it will be presented an analysis and a discussion about the results.

Keywords: *Clustering, Data Streaming Clustering, Disjoint-set structure, Feature Vector, SetClust, Micro-cluster*

Índice

Índice de figuras

Índice de tablas

Glosario de términos

- **Big data**: Es un término que describe el gran volumen de datos, estructurados y no estructurados, que una organización acumula en el tiempo.
- **Cluster:** conjunto de datos que comparten una característica en común.
- **Complejidad computacional:** Medida utilizada para dar relación entre el tiempo y la complejidad de un problema.
- **Dataset:** Es un conjunto de datos tabulados en cualquier sistema de almacenamiento de datos estructurados.
- **Diagrama de Voronoi:** Estructura intuitiva y sencilla conformada por una división en regiones de un conjunto de puntos en un plano.
- **Estructura de indexación:** Estructura de datos que permite el búsqueda rápida de información a través de indices.
- **Hiperparametro:** Es el valor de la configuración de una función independiente a los datos a ser procesados.
- **Huerística:** Elaboración de medios auxiliares, principios, reglas, estrategias y programas que faciliten la búsqueda de vías de solución a problemas.
- **Memoria principal:** Es la memoria de la computadora donde se almacenan temporalmente tanto los datos como los programas que la unidad central de procesamiento está procesando o va a procesar en un determinado momento.
- **Memoria secundaria:** Es el conjunto de dispositivos y soportes de almacenamiento de datos que conforman el subsistema de memoria externo de la computadora.
- **Minería de datos:** Es un campo de la estadística y las ciencias de la computación referido al proceso que intenta descubrir patrones en grandes volúmenes de conjuntos de datos.
- **Outlier:** Es una observación anormal y extrema en una muestra estadística o serie temporal de datos que puede afectar potencialmente a la estimación de los parámetros del mismo.
- **Ruido:** Es toda señal no deseada que se mezcla con la señal útil que se quiere transmitir.

Capítulo 1

Aspectos generales

1.1. Planteamiento del problema

1.1.1. Descripción del problema

El problema de *clustering* que tiene como objetivo agrupar un conjunto de datos de tal manera que la información que pertenece a un *cluster* tenga más correlación con este *cluster* que con información en otros, debido a los requerimientos de este problema los algoritmos que lo solucionan pueden ser utilizados en varios campos como *machine learning*, reconocimiento de patrones, análisis de imágenes, bioinformática, minería de datos, compresión de datos y gráficos computacionales.

Teniendo en cuenta el problema del *clustering*, la investigación se centrará en un subcampo de este problema el cual es *Data Streaming Clustering* este problema adopta el mismo objetivo que el problema principal, pero bajo diferentes condiciones y complicaciones.

Data Streaming Clustering busca crear y mantener un conjunto de grupos validos donde se puedan agrupar datos que son transmitidos mediante un flujo continuo e ilimitado, por esta razón se deben lidiar con algunos problemas como por ejemplo, que los datos no son conocidos previamente, es decir, no se puede suponer un número fijo de grupos *clusters* que deben ser formados, ya que este puede cambiar en el tiempo. En adición a esto la disposición de los datos en el espacio no necesariamente se describe como conjuntos convexos, por ende, formas mas complejas pueden existir y las soluciones propuestas a este problema deben ser capaces de descubrirlas.

En los problemas relacionados a la medición de la información tenemos que a causa de la precisión de los instrumentos de recopilación de datos y la transmisión de estos, la información puede tener ruido que es información falsa de los datos que están siendo medidos. Los *outliers* es otro problema a tener en cuenta, estos son información atípica de los datos observados, por lo tanto, si es información real, pero no debería ser

procesada ya que no describe un comportamiento normal de los datos que se analizan. Estos son las complicaciones que un algoritmo propuesto para este problema debería manejar.

A parte de los problemas propios de los datos, se debe tomar en cuenta el costo computacional que requiere este problema. En primer lugar, la limitación espacial, la memoria principal del computador no es capaz de almacenar toda la información que se desea procesar, y la memoria secundaria si es capaz, pero agrega tiempo de acceso a la memoria. En segundo lugar, la limitación temporal, al ser tanta información que procesar solo se puede concebir la idea de que la información sea leída solo una vez. Debido a estas condiciones especiales los algoritmos propuestos para el problema de *Clustering* no pueden ser utilizados para este problema, por lo menos no sin modificaciones.

1.1.2. Formulación del problema

¿El diseño e implementación de un algoritmo para el problema de *data streaming clustering* para conjuntos amorfos y con *outliers* optimiza los resultados de otros algoritmos propuestos?

1.2. Antecedentes

1.2.1. Ester et al., 1996, *A density-based algorithm for discovering clusters in large spatial databases with noise*, Proceedings of the Second International Conference on Knowledge Discovery and Data Mining

El algoritmo *Density-based spatial clustering of aplications with noise* más conocido como *DBSCAN* (Ester et al., 1996) es un algoritmos de *clustering* basado en densidad parametrizado, dado un conjunto de puntos en un espacio *n-dimensional*, este agrupa puntos en un mismo conjunto a los puntos que tienen muchos vecinos cercanos, convirtiendo en *outliers* los puntos que permanezcan solos en regiones de densidad baja, por ejemplo, los puntos que tengan a sus vecinos más cercanos bastante alejados de ellos.

Conclusiones: Los algoritmos de *clustering* son atractivos para tareas de identificación de clases en bases de datos espaciales. Sin embargo, los algoritmos conocidos sufren de varios inconvenientes cuando son aplicados a un gran conjunto de datos. En esta investigación, se presentó un algoritmo de *clustering DBSCAN* que depende en la noción de densidad de *clusters*. Esto solo requiere un parámetro y ayuda al usuario a identificar el apropiado. Se realizaron evaluaciones de desempeño sobre datos arti-

ficiales y sobre datos reales de *SEQUOIA* 2000 *Benchmark*. Los resultados de estos experimentos demostraron que *DBSCAN* es significativamente más eficiente en descubrir *clusters* de formas arbitrarias que el conocido algoritmo *CLARANS*. Más aún, los experimentos han mostrado que *DBSCAN* supera en rendimiento a *CLARANS* por un factor de al menos 100 en términos de eficiencia.

1.2.2. Cao et al., 2006, *Density-Based Clustering over an Evolving Data Stream with Noise*, SIAM Society for Industrial and Applied Mathematics

Clustering es una importante tarea en minería de datos en evolución. Además, la memoria limitada y la limitación de solo poder hacer una lectura de los datos requieren de algoritmos que puedan lidiar mejor con estos inconvenientes. En este artículo se presenta DenStream, un nuevo enfoque para descubrir clústeres en un flujo de datos en evolución. Para ello se definen algunos conceptos, el *micro-cluster* "denso" (denominado *núcleo-micro-cluster*) es usado para resumir los *clusters* con formas arbitrarias, mientras que las estructuras potenciales de *núcleo-micro-cluster* y *outlier-micro-cluster* son propuestos para mantener y distinguir los posibles *clusters* y *outliers*. Además es diseñada una nueva estrategia de poda basada en estos conceptos, que garantiza la precisión de los pesos de los *micro-clusteres* con memoria limitada (Cao et al., 2006).

El flujo de datos en evolución posee los siguientes requerimientos para *clustering* de flujo de datos.

- No conocimiento del número de *clusters*. El número de *clusters* es comúnmente desconocido, además, este número es cambiante en el tiempo.
- Descubrimiento de *clusters* con formas arbitrarias. Esto es muy importante para muchas aplicaciones de flujo de datos.
- La habilidad de manejar *outliers*. En los escenarios de flujo de datos debido a la influencia de muchos factores como interferencia electromagnética, fallo temporal de sensores, sensores con baja batería, etc. puede aparecer ruido de manera aleatoria en los datos.

Las características principales del algoritmo publicado en este artículo son las siguientes:

- El *core-micro-cluster* está diseñado para resumir la formación de los *clusters* con formas arbitrarias.
- Incluye una nueva estrategia de podado, la cual da oportunidad para el crecimiento de nuevos *clusters* mientras se deshace rápidamente de los *outliers*.

- Un *outlier-buffer* es introducido para separar el procesamiento de potenciales *core-micro-clusters* y *outlier-micro-clusters*, que mejora la eficiencia del algoritmo.

Conclusiones: En esta investigación, se propuso *DenStream*, un efectivo y eficiente método para *clustering* en flujos de datos en evolución. Este método puede descubrir *clusters* de formas arbitrarias en flujos de datos, y no se ve afectado por el ruido. Las estructuras de *p-micro-clusters* y *o-micro-clusters* mantienen suficiente información para realizar *clustering*, y una nueva estrategia de podado fue diseñada para limitar el consumo de memoria mientras se garantiza la precisión. La evaluación de los resultados experimentales de desempeño sobre un número de conjuntos de datos reales y artificiales demuestran la efectividad y eficiencia de *DenStream* en descubrir *clusters* de formas arbitrarias en el conjunto de datos.

Comentario: Se uso este algoritmo para la fase de pruebas y contrastación de los resultados.

1.2.3. Kranen et al., 2009, *Self-Adaptive Anytime Stream Clustering*, IEEE International Conference on Data Mining, USA

En este trabajo se propone un algoritmo *ClusTree* libre de parámetros que se adapta automáticamente a la velocidad del flujo de datos. Hace mejor uso del tiempo disponible bajo las restricciones actuales para proporcionar una agrupación de los objetos vistos hasta ese punto. Es incorporada la edad de los objetos para enfatizar la importancia de los datos más recientes. Además, *ClusTree* es capaz de detectar la variación de la velocidad de flujo de datos, la creación de nuevos *clusters* y la detección de *outliers* en el flujo de datos.

ClusTree es una estructura de índice compacta y auto adaptativa que mantiene una estructura que resume la información actual del flujo de datos (Kranen et al., 2009). Para lo cual cuenta con las siguientes características:

- *Micro-clusters*: Es una compacta representación de la distribución de los datos manteniendo las medidas de los datos cambiantes de la media y la varianza de un *micro-cluster*.
- *CluTree*: Consiste en una estructura jerarquizada en forma de árbol en la que se almacenan la información de los datos entrantes que describen las propiedades de las características de los *clusters* y los subárboles relacionados a ella.
- Envejecimiento de los datos: Para mantener actualizado la representación de los *clusters* es importante que los datos más frecuentes tengan más importancia que los datos antiguos, para ellos es utilizada una función exponencial de decaimiento

dependiente del tiempo para el envejecimiento de los datos.

$$w(\Delta t) = \beta^{-\lambda \Delta t}$$

- Agregación adaptativa: Para manejar la velocidad del flujo de datos se debe realizar una inserción de la información veloz, para lo cual se hace una inserción por grupos en la cual el grupo va descendiendo por la estructura *ClusTree* y llegando a los nodos destino de manera más eficiente que si se insertara elemento por elemento.
- Formas de los *cluster*: Utilizando la información almacenada en la estructura *ClusTree* se puede utilizar algoritmos de *clustering offline* para realizar la agrupación final de la información u otros métodos de *clustering* para manejar formas arbitrarias en los *clusters*.

Conclusiones: El *Clustering* de flujos de datos está en creciente importancia en muchas aplicaciones. En este trabajo, se propuso un enfoque basado en índices libre de parámetros que se auto adapta a la variación de velocidad de los flujos, y es capaz de realizar *clustering* en cualquier momento del flujo. *ClusTree* mantiene los valores necesarios para computar la media y la varianza de los *micro-clusters*. Al incorporar agregados locales, por ejemplo, *buffers* temporales para *"hitchhikers"*, se propuso una nueva solución para la fácil interrupción del proceso de inserción que se puede reanudar simplemente en cualquier momento posterior. Para flujos de datos rápidos, la agregación de objetos similares permite la inserción de grupos en vez de objetos singulares para un procesamiento aún más rápido. En comparación a enfoques recientes se ha mostrado que *ClusTree* puede mantener la misma cantidad de *micro-clusters* en velocidades de flujos que son más veloces por órdenes de magnitud y para iguales velocidades de flujos la granularidad es exponencial. Más aún, se discutió la compatibilidad de este enfoque para la búsqueda de *clusters* con formas arbitrarias y modelar transiciones de *clusters* y datos en evolución usando enfoques recientes.

Comentario: Se uso este algoritmo para la fase de pruebas y contrastación de los resultados.

1.2.4. Aggarwal et al., 2003, *A Framework for Clustering Evolving Data Streams*, VLDB Proceedings of the 29th international conference on Very large data bases

En este artículo se discute una forma diferente de abordar el problema de *Data Streaming Clustering*. El objetivo de dividir el proceso de *clustering* en dos fases, la primera una fase *online* que periódicamente almacenará la información estadística resumida del flujo de datos, y la segunda fase *offline* en la cual se usará esta información resumida, en esta fase se puede usar una amplia variedad de técnicas para realizar el *clustering* final de los datos. Esta nueva manera de abordar el problema será llamada *CluStream* (Aggarwal et al., 2003).

Son utilizados dos conceptos importantes para el desarrollo de *CluStream*.

- *Micro-clusters*: Es la información estadísticas de un conjunto de datos.
- *Pyramidal Time Frame*: Se realizan capturas instantáneas durante el flujo de datos de los *micro-clusters* formados las cuales fluyen en un patrón piramidal. Este patrón permite un intercambio efectivo entre los requisitos de almacenamiento y la habilidad de recuperar *micro-clusters* de otros instantes del flujo de datos.

Conclusiones: En esta investigación, se ha propuesto un efectivo y eficiente método llamado *CluStream*, para el *clustering* de gran cantidad de flujos de datos en evolución. El método tiene claras ventajas sobre recientes técnicas las cuales tratan de realizar *clustering* sobre el completo flujo de una sola vez, en vez de ver el flujo como un proceso cambiante sobre el tiempo. El modelo *CluStream* provee una amplia variedad de funcionalidades en caracterización de flujos de *clusters* sobre diferentes horizontes de tiempo en un ambiente en evolución. Esto es conseguido a través de una cuidadosa división del trabajo entre la colección *online* de datos estadísticos y el análisis *offline* de este componente. Así, el proceso provee una considerable flexibilidad al análisis en tiempo real y el cambio de ambiente. Estos objetivos fueron alcanzados por un cuidadoso diseño del proceso de almacenamiento de los datos estadísticos. El uso de *pyramidal time window* asegura que la esencia de los datos estadísticos del flujo de datos en evolución puede ser capturado sin sacrificar el espacio subyacente y la eficiencia de tiempo del proceso de *clustering* del flujo de datos. Además, la explotación de los *micro-clusters* asegura que *CluStream* pueda alcanzar una precisión más alta que *STREAM* debido a su registro de información más detallada que los k puntos usados por el enfoque *k-means*. El uso de *micro-clustering* asegura una colección de datos escalable, mientras retiene suficiente información de los datos para un *clustering* efectivo.

Comentario: Se uso este algoritmo para la fase de pruebas y contrastación de los resultados.

1.2.5. Miller et al., 2014, Twitter spammer detection using data stream clustering, Information Sciences

El rápido crecimiento de Twitter ha provocado un aumento dramático en el volumen y la sofisticación del spam. El abuso de ciertos componentes de Twitter, como "hashtags", "mencione" y URLs abreviadas, permite a los spammers operar de manera eficiente. Sin embargo, estas mismas características pueden ser un factor clave para identificar nuevas cuentas de spam, como se muestra en estudios anteriores. Este estudio proporciona tres contribuciones novedosas. En primer lugar, estudios anteriores han abordado la detección de spam como un problema de clasificación, mientras que nosotros lo vemos como un problema de detección de anomalías. En segundo lugar, se introdujeron 95 características del texto del tweet junto con la información del usuario analizada en estudios anteriores. Finalmente, para manejar de manera efectiva la

naturaleza de transmisión de tweets, se modificaron dos algoritmos de agrupamiento de transmisión, *StreamKM++* y *DenStream*, para facilitar la identificación de spam. Ambos algoritmos agruparon a los usuarios normales de Twitter, tratando a los valores atípicos como spammers.

Conclusiones: Debido a la creciente popularidad y el uso intensivo de redes sociales como Twitter, el número de spammers está creciendo rápidamente. Esto ha dado lugar al desarrollo de varias técnicas de detección de spam. Este estudio ha realizado tres nuevas aportaciones al campo de la detección de spam en Twitter. En primer lugar, consideramos la identificación de spam como un problema de detección de anomalías. En segundo lugar, presentamos 95 características independientes del texto del tweet para la de detección de spam en Twitter. Por último, utilizamos el flujo de tweets en tiempo real, así como la información del perfil del usuario con dos algoritmos de agrupación en cluster basados en flujo, *DenStream* y *StreamKM++*. Cuando se probaron, estos dos enfoques lograron un $97,1\,\%$ y un $84,2\,\%$ de precisión de *F-Measure*, y un $94,0\,\%$ y un $4,8\,\%$ de precisión de *F-Measure* respectivamente. Los hallazgos sugieren que la adición de características independientes mejora la detección de spam. Aunque estos algoritmos demostraron de forma independiente una buena detección, la combinación de los dos mejoró aún más todas nuestras métricas, en particular la recuperación y la tasa de falsos positivos al $100\,\%$ y al $2,2\,\%$, lo que muestra el valor del enfoque multicapa para la detección de spam.

1.2.6. Jiang et al., 2020, Exemplar-based data stream clustering toward Internet of Things, The Journal of Supercomputing

Lidiar con el flujo de datos dinámicos se ha convertido en uno de los campos de investigación más activos para Internet de las cosas (IoT). Específicamente, la agrupación hacia un flujo de datos dinámico es una base necesaria para numerosas plataformas de IoT. Este artículo, se enfoca en modelos de agrupamiento dinámicos basados en ejemplos. En términos del principio máximo a priori, bajo el marco de probabilidad, primero resumimos una explicación unificada para dos modelos típicos de agrupamiento basados en ejemplos, a saber, movimiento de expansión α mejorado (EEM) y propagación de afinidad (AP). Luego, se propone en consecuencia un nuevo algoritmo de agrupación de flujos de datos basado en ejemplos dinámicos llamado DSC. El mérito distintivo del algoritmo propuesto DSC es que simplemente podemos utilizar el marco del algoritmo EEM mediante la modificación de las definiciones de varias variables y no es necesario diseñar otro mecanismo de optimización. Además, el algoritmo DSC es capaz de tratar dos casos de similitudes. A diferencia de AP y EEM, los resultados experimentales indican el poder del algoritmo DSC para los flujos de datos de IoT del mundo real.

Conclusiones: Para la tecnología de IoT, sería muy útil un modelo de agrupamiento dinámico eficiente y efectivo hacia el flujo de datos. Bajo esta situación, este documento propone de nuevo un marco de agrupamiento dinámico basado en ejemplos. En primer lugar, bajo MAP y el marco de probabilidad bayesiano, se estima la relación de los

puntos de datos y el conjunto ejemplar. Y luego damos una explicación unificada para los algoritmos de agrupamiento de AP y EEM. En segundo lugar, en base en estos hallazgos, se propone el algoritmo DSC para agrupar el flujo de datos dinámicos cuyas características evolucionan con el tiempo. Especialmente, este nuevo algoritmo aprovecha la similitud entre los ejemplos anteriores y los ejemplos actuales para guiar el proceso de agrupación de puntos de datos actuales. En tercer lugar, se utiliza la estrategia de optimización de EEM para garantizar el rendimiento del nuevo modelo de agrupación en clusters, y el rendimiento del algoritmo de agrupación en cluster propuesto DSC se verifica experimentalmente en los flujos de datos sintéticos y de IoT del mundo real. Aunque DSC muestra un rendimiento de agrupación alentador para el flujo de datos de IoT, aún quedan por estudiar algunos otros problemas. Por ejemplo, el algoritmo DSC subyacente asume que el flujo de datos evoluciona lentamente y sigue siendo similar a lo largo del tiempo. Una vez que se rompe esta suposición, la introducción de puntos de datos anteriores puede tener una influencia negativa. Por tanto, se debería prestar más atención al supuesto en los estudios continuos. Otro problema es como determinar un parámetro apropiado para un conjunto de datos insuficiente.

1.3. Objetivos

1.3.1. Objetivo general

Diseñar un algoritmo para el problema *Data Streaming Clustering* para conjuntos amorfos y con *outliers*.

1.3.2. Objetivos específicos

1. Explorar de técnicas de *Data Streaming Clustering*.
2. Desarrollar una estructura capaz de almacenar la información resumida de un flujo de datos.
3. Desarrollar un algoritmo capaz de aplicar *clustering* en la estructura de información resumida.
4. Proponer *Datasets* artificiales con *clusters* amorfos para el problema *Data Streaming Clustering*.
5. Comparar y analizar los resultados obtenidos.

1.4. Justificación

Debido al creciente uso de la tecnología en todas las ramas de la ciencia y en el uso cotidiano de las personas, la cantidad de información generada excede lo que hace unos años podría llamarse manejable. Es por esto que se desarrollaron muchas teorías y estudios como Big Dada, Machine Learing, etc, para poder volver esta enorme cantidad de datos en información clara y útil.

Dentro de este conjunto de información que es recopilada, podemos distinguir dos grandes sectores, los cuales son los datos que se obtienen con una descripción o etiqueta, y los datos que no la poseen. Estos últimos serán el tema a tratar en el desarrollo de esta investigación. Estos datos no poseen un referente según el cual se los pueda clasificar, por lo tanto, se debe poder extraer la información subyacente de ellos para poder realizar una clasificación.

Más aun, se desea circunscribir el tema a la resolución del problema *Data Streaming Clustering* que fue descrito anteriormente. Debido a la naturaleza de este problema y tener requerimientos especiales que pueden ir variando en su intensidad dentro de un flujo de datos, han sido planteados una gran variedad de métodos para lidiar con este problema, por lo cual es importante realizar un análisis de estos métodos para poder extraer las mejores características de estos y poder consolidar un algoritmo eficiente con las capacidades suficientes para resolver el problema central de *clustering* y además pueda manejar las complicaciones que trae un flujo de datos.

1.5. Alcances y limitaciones

- La comparación del algoritmo propuesto se realizará con algoritmos para *Data Streaming Clustering*. Estos algoritmos tendrán el mismo funcionamiento *online* del algoritmo propuesto y otros serán *semi-online*.
- Ambas fases del algoritmo: fase de estructuración de los datos y la fase de *clustering* serán procesadas *online*.
- La construcción del algoritmo será planteada sin tomar en cuenta las características de ruido en los *datasets*.
- El uso de *datasets* será limitado a *datasets* tradicionales, esto implicada no utilizar *datasets* en evolución debido a que los *clusters* en estos van cambiando de posición y forma en el tiempo, por lo tanto, no se implemento el envejecimiento de datos como característica del algoritmo.
- El algoritmo propuesto será capaz de realizar el proceso de *clustering* sobre *datasets* que no conozca previamente el número de *clusters* existentes además también podrá adaptarse a las formas de los *clusters*, sean formas euclidianas o otras mas complejas.

- El algoritmo propuesto será capaz del manejo de *outliers* mas no la eliminación de estos, esto no impide poder agregar esta característica al algoritmo posteriormente.

1.6. Metodología

La metodología que se utiliza en el desarrollo de este proyecto para la documentación y recolección de información y la revisión de la literatura tiene como base la metodología descriptiva según (Hernández Sampieri et al., 2014), esto puede verse en la fase 1.

Debido a que no existe una metodología clara sobre el diseño e implementación de algoritmo se propone la siguiente metodología de desarrollo para esta investigación. La metodología que se presenta a continuación está en concordancia al cronograma que se presenta en la sección 1.7.

1. **Revisión de la literatura:** Se realiza un estudio de la literatura previa sobre *data streaming clustering* y los aportes que se hicieron. Esto permitirá que las siguientes etapas sean posibles.

2. **Diseño del algoritmo:** En esta etapa se busca una posible solución en base al estado del arte. Al finalizar esta etapa se debe contar con ambas partes del algoritmo formuladas, la etapa de lectura del flujo de datos y el almacenamiento de su información resumida y la etapa de *clustering*.

3. **Implementación del algoritmo:** En esta etapa se utilizo el lenguaje de programación *Python* para la implementación del algoritmo diseñado en la etapa anterior.

4. **Generación de *datasets* artificiales:** Para hacer las pruebas de los algoritmos es necesario contar con *datasets* con propiedades específicas, a pesar de que existen *datasets* con algunas características útiles para esta investigación, no poseen todas. Para ello se debe generar *datasets* con grandes cantidades de datos que posean *clusters* con formas irregulares y *outliers*.

5. **Evaluación y análisis de los resultados:** En esta fase se utilizará la implementación del algoritmo obtenida en la etapa 3, y se realizará una comparación utilizando la métrica *V measure score*. Posteriormente se analizará los resultados de esta comparación para determinar una conclusión.

1.7. Cronograma de actividades

El cronograma de actividades se muestra en la Figura 1.1.

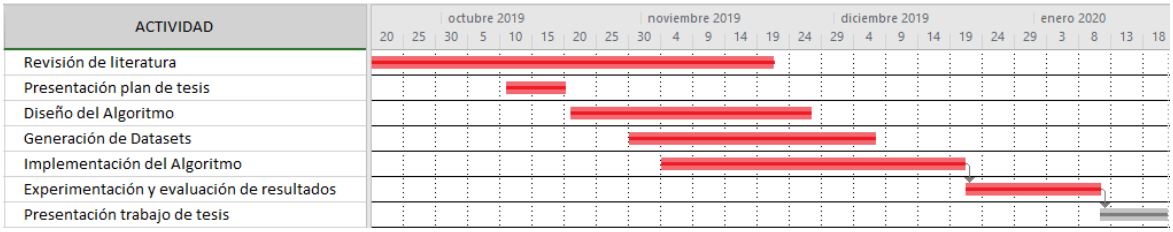

Figura 1.1: Diagrama de Gantt del cronograma de actividades. Se muestra de color rojo las actividades que han sido completadas.

Fuente: *Elaboración propia.*

Capítulo 2

Marco teórico

2.1. Machine Learning

El aprendizaje automático es el subcampo de las ciencias de la computación y una rama de la inteligencia artificial, cuyo objetivo es desarrollar técnicas que permitan que las computadoras aprendan. Se dice que un agente aprende cuando su desempeño mejora con la experiencia; es decir, cuando la habilidad no estaba presente en su genotipo o rasgos de nacimiento. De forma más concreta, los investigadores del aprendizaje de máquinas buscan algoritmos y heurísticas para convertir muestras de datos en programas de computadora, sin tener que escribir los últimos explícitamente. Los modelos o programas resultantes deben ser capaces de generalizar comportamientos e inferencias para un conjunto más amplio de datos (Russell et al., 2004).

Los diferentes algoritmos de Aprendizaje Automático se agrupan en una taxonomía en función de la salida de los mismos. Algunos tipos de algoritmos son:

- Aprendizaje supervisado: El algoritmo produce una función que establece una correspondencia entre las entradas y las salidas deseadas del sistema. Un ejemplo de este tipo de algoritmo es el problema de clasificación, donde el sistema de aprendizaje trata de etiquetar (clasificar) una serie de vectores utilizando una entre varias categorías (clases). La base de conocimiento del sistema está formada por ejemplos de etiquetados anteriores. Este tipo de aprendizaje puede llegar a ser muy útil en problemas de investigación biológica, biología computacional y bioinformática.
- Aprendizaje no supervisado: Todo el proceso de modelado se lleva a cabo sobre un conjunto de datos del cual no se tiene información sobre las categorías de esos ejemplos. Por lo tanto, en este caso, el sistema tiene que ser capaz de reconocer patrones para poder etiquetar las nuevas entradas.

2.2. Clustering

Dentro del campo del *Machine Learning*, existen dos tipos principales de aprendizaje: supervisado y no supervisado. La principal diferencia entre ambos tipos es que, en el aprendizaje supervisado, se utiliza un conocimiento previo, información a priori de las salidas correctas que deben tener los casos de ejemplo que se disponen. Por lo tanto, el objetivo del aprendizaje supervisado es estimar una función que, dados casos de aprendizaje y sus respectivas salidas, aproxime de la mejor forma posible la relación entre las entradas y salidas en los datos observados. El aprendizaje no supervisado, por otro lado, no dispone de las salidas correctas para los ejemplos disponibles, por lo tanto, el objetivo es inferir una estructura natural presente dentro de un conjunto de datos.

Clustering tiene como objetivo de dividir la población o los puntos en un número de grupos de tal manera que los puntos en los mismos grupos son más parecidos a los puntos que están en el mismo grupo que aquellos que pertenecen a otros grupos. En general, *clustering* se puede dividir en dos grupos:

- *Hard Clustering:* En este *clustering* cada punto pertenece completamente a un *cluster* o no lo hace.
- *Soft Clustering:* En *soft clustering* en vez de poner cada punto en un único *cluste*r, se calcula un probabilidad de que el punto pertenezca a ciertos *clusters*.

Debido a que el objetivo del *clustering* es subjetivo, los medios que se pueden utilizar para lograr este objetivo son muchos. Cada metodología plantea un diferente grupo de reglas para definir la similitud de los puntos a ser agrupados (Estivill-Castro, 2002). Existen más de 100 algoritmos conocidos para este problema cada uno con su propio entendimiento del problema, pero solo unos cuantos son usados comúnmente. Se agruparon estos conceptos para organizar los algoritmos de *clustering* en las siguientes categorías:

- *Modelos de conectividad:* Estos modelos están basados en la noción de que los puntos más cercanos en el espacio muestran más similitud entre ellos que los puntos que están más alejados. Estos modelos pueden seguir dos enfoques. El primer enfoque plantea comenzar con clasificar todos los puntos en *clusters* separados y luego ir uniéndolos según como decrece la distancia. En el segundo enfoque todos los puntos están clasificados en el mismo *cluster* y luego son separados a medida que la distancia aumenta. Además, la elección de la función de distancia es subjetiva.
- *Modelos basado en centros:* Estos son algoritmos de *clustering* iterativos en donde el principio de similitud deriva de la cercanía de un punto al centroide del *cluster*. El algoritmo *K-means* es el algoritmo más conocido que sigue este concepto, sin embargo, recientemente se desarrollaron algoritmos con un comportamiento

superior entre ellos *EMAX* (León et al., 2020). En este modelo, el número de *clusters* requeridos al final debe ser conocido previo a su ejecución.

- *Modelos distribuidos:* Estos modelos de *clustering* están basados en la idea de cuán probable es que todos los puntos en el *cluster* pertenezcan a la misma distribución probabilística (Normal, Gaussiana, etc.). Estos modelos comúnmente sufren de *overfitting*.

- *Modelos basados en densidad:* Estos modelos buscan en el espacio de datos por áreas que tengan una densidad inusual de puntos. Con lo cual aísla diferentes regiones densas y asigna a los puntos en esta área en el mismo *cluster*. Los algoritmos más conocidos del modelo basado en densidad son *DBSCAN* (Ester et al., 1996) y *OPTICS* (Kriegel et al., 2011).

2.3. Data Streaming Clustering

Avances recientes tanto en software como hardware han permitido un crecimiento de datos a gran escala. Sin embargo, lidiar con cantidades masivas de datos tiene un desafio para los investigadores, debido a las limitaciones físicas de los recursos computaciones actuales. En la última década se han visto un crecimiento interesante en el manejo de estas masivas e ilimitadas secuencias de datos que son generados continuamente y de manera acelerada (Silva et al., 2013), estos serán llamados *data streams* (Gama, 2010).

De manera más formal un *data stream* S es una masiva secuencia de datos $x^1, x^2, ..., x^N$, $S = \{x^i\}_{i=1}^{N}$ la cual es potencialmente ilimitada ($N \to \infty$). Cada objeto es descrito por un $n - dimensional$ vector de atributos $X^i = [x_j^i]_{j=1}^{n}$ perteneciendo a un espacio ω que puede ser continuo, categórico o una mezcla de ambos.

Las aplicaciones *data streams* incluyen minería de datos generada por sensores de redes, análisis meteorológico, análisis del mercado, monitoreo de trafico de redes computaciones, entre otros. Estas aplicaciones conllevan conjuntos de datos que son mucho más grandes que la memoria principal de un computador y son mayormente almacenados en la memoria secundaria del mismo. A este punto está probado que el uso de memoria de acceso aleatorio es prohibitivamente costoso (Guha et al., 2003), realizar exploraciones lineales de los datos es el único método de acceso aceptable en términos de eficiencia computacional.

Extraer conocimiento potencialmente útil de un *data stream* es desafiante de por sí. La mayoría de las técnicas de minería de datos y descubrimiento de conocimiento asumen que hay una cantidad finita de datos generados por una distribución de probabilidad estacionaria desconocida, la cual puede ser físicamente almacenada y analizada en múltiples pasos usando un algoritmo *batch-mode*. Para *data streaming clustering*, sin embargo, para un satisfactorio desarrollo de algoritmos se tiene que tener en cuenta las siguientes restricciones (Silva et al., 2013):

- Los objetos llegan continuamente.
- No hay control sobre el orden en el cual los datos serán procesados.
- El tamaño del *data stream* es potencialmente ilimitado.
- Los objetos son descartados después de haber sido procesados. Se puede almacenar parte de los datos por un periodo de tiempo dado, si se implementa un mecanismo de descarte.
- La desconocida generación de datos procesados puede ser no estacionaria, por ejemplo, la distribución de probabilidad puede cambiar en el tiempo.

2.4. Estructuras de datos

Desarrollar estructuras de datos adecuadas para el almacenamiento estadístico de la información resumida del *data stream* es un paso crucial para cualquier algoritmo de *data stream clustering*, debido a las limitaciones de almacenamiento asumidas en estas aplicaciones. Considerando que no puede ser almacenado en la memoria principal el *data stream* completo, estructuras especiales deben ser empleadas para el resumido del *stream*. Por ello estas son cuatro estructuras de datos comúnmente empleadas en el paso de la abstracción de datos: *feature vector*, *prototype array*, *coreset trees* y *grids*.

2.4.1. Feature vector

El uso de *feature vector* para resumir grandes cantidades de datos fue introducido por primera vez por el algoritmo *BIRCH* (Zhang et al., 1996). Este vector, llamado *Clustering Feature vector* (CF) tiene tres componentes: N como el número de objetos, LS como la suma lineal de los objetos, y SS la suma del cuadrado de los datos. Las estructuras LS y SS son arreglos $n-dimensionales$. Estas tres componentes permiten calcular la información estadística del *cluster* como: la media, radio, diámetro, entre otros.

$$media = \frac{LS}{N}$$

$$radio = \sqrt{\frac{SS}{N} - (\frac{LS}{N})^2}$$

$$diametro = \sqrt{\frac{2N * SS - 2 * LS^2}{N(N-1)}}$$

Este vector CF posee importantes propiedades incrementales y aditivas como las siguientes:

- *Incrementalidad:* Un nuevo objeto x^j puede ser fácilmente agregado en el vector CF mediante la actualización de sus valores:

$$LS = LS + x^j$$

$$SS = SS + (x^j)^2$$

$$N = N + 1$$

- *Aditividad:* Dos vectores disjuntos CF_1 y CF_2 pueden ser fácilmente unidos en un CF_3 sumando sus componentes:

$$N_3 = N_1 + N_2$$

$$LS_3 = LS_1 + LS_2$$

$$SS_3 = SS_1 + SS_2$$

Debido a la eficiencia y el natural uso que se le puede dar a esta estructura con otras estructuras, esta será usada en el desarrollo de esta investigación.

2.4.2. Prototype array

Algunos algoritmos de *data streaming clustering* usan una simplificada estructura de resumen mediante esta estructura *prototype array* (Shah et al., 2005).

Por ejemplo, *Stream* (Guha et al., 2000) utiliza un arreglo de prototipos para resumir el *stream* dividiendo el *data stream* en dos pedazos de tamaño $m = N^E$, $0 < E < 1$. Cada pedazo de m datos es resumido en $2k$ datos representativos. Luego estos m prototipos son aún más comprimidos en $2k$ prototipos y el proceso sigue a lo largo del *stream*.

2.4.3. Coreset tree

Esta es una estructura de resumen de datos significativamente diferente empleada en *StreamKM++* (M. R. Ackermann et al., 2012). Esta estructura es un árbol binario en el cual cada nodo del árbol i contiene los siguientes componentes: un conjunto de objetos E_i, un prototipo de E_i x^{p_i}, el número de objetos en E_i N_i, y la suma de las distancias al cuadrado de los objetos en E_i a X^{p_i} SSE_i. E_i solo tiene que estar almacenado en los nodos hoja del *coreset tree*, debido a que los objetos de los nodos interiores son implícitamente definidos como la unión de los objetos de sus nodos hijos.

2.4.4. Grids

Algunos algoritmos de *data streaming clustering* realizan el resumido de los datos mediante *grids* o cuadrículas (Cao et al., 2006). Partiendo un espacio de características $n - dimensional$ en cuadrículas de densidad. Por ejemplo, el algoritmo *DenStream* (Cao et al., 2006) mapea cada punto en una cuadrícula de densidad, cada objeto en un tiempo t es asociado a un coeficiente de densidad que decrece en el tiempo, $D(x^j, t) = \lambda^{t-t^j}$, donde $\lambda \in (0, 1)$ es un factor de decrecimiento. La densidad de una cuadrícula g en un tiempo t, $D(g, t)$ esta dada por la suma de las densidades ajustadas de cada objeto que esta mapeado a g en ese momento o antes del tiempo t. Cada cuadrícula está representada por una tupla $< t_g, t_m, D, label, status$ donde t_g es el último momento que la cuadrícula fue actualizada, t_m es el último momento que la cuadrícula fue removida de la tabla de *hash* que mantiene las cuadrillas validas, D es la cuadrículas de densidad en su última actualización, *label* es la etiqueta de la cuadrícula y *status* indica si la cuadrícula es *NORMAL* o *SPORADIC*.

2.5. Métodos de clustering

Una vez terminada la fase de resumen de la información, debido a que no se puede manejar la información sin un preprocesado, utilizando las estructuras de datos se puede proceder a la tarea de agrupar los datos en conjuntos que compartan información. Para realizar esto se utilizan dos métodos conocidos de clustering *K-means* y *DBSCAN*.

2.5.1. K-means

K-means es una técnica de *center based clustering* que busca particionar un conjunto de n elementos en k conjuntos en los cuales cada punto pertenece al *cluster* con el promedio más cercano. Esto resulta en un particionamiento del espacio de datos según particiones de Voronoi.

Encontrar una solución a este problema que esa un óptimo global es computacionalmente costoso, por lo tanto, existen algoritmos heurísticos que son comúnmente utilizados y que convergen rápidamente a un óptimo local. El algoritmo más común utiliza una técnica de refinamiento de la solución. Dada su popularidad, es comúnmente llamado el algoritmo *K-means*; también es conocido como el algoritmo de Lloyd (Lloyd, 1982), descrito a continuación.

Dado un conjunto inicial de k promedios $m_1^{(1)}, m_2^{(1)}, ..., m_k^{(1)}$, el algoritmo procede por alternar los siguientes pasos:

- **Paso de asignación:** Asigna cada observación al *cluster* cuyo promedio tiene la

menor distancia Euclidiana al cuadrado, esto es intuitivamente el promedio más cercano.

$$S_i^{(t)} = \left\{ x_p : \|x_p - m_i^{(t)}\|^2 \leq \|x_p - m_j^{(t)}\|^2 \forall j, 1 \leq j \leq k \right\}$$

donde cada x_p es asignado exactamente a un $S^{(t)}$, incluso si este pudiera ser asignado a dos o más de ellos.

- **Paso de actualización:** Calcular los nuevos promedios como los centroides de la nueva organización de los *clusters*.

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j$$

El algoritmo es normalmente presentado como una asignación de los objetos al *cluster* más cercano. Al utilizar una función diferente al cuadrado de la distancia Euclidiana, el algoritmo puede no converger. Se han presentado multitud de modificaciones al algoritmo *k-means* como el algoritmo *k-means* esférico y *k-medoids*, estos utilizan otras medidas de distancia.

Este método de *clustering* es comúnmente usado en algoritmos que requieren de dos fases una en tiempo real para el resumen y estructuración de los datos y la fase del *clustering* se ejecuta en la fase de espera debido que este algoritmo procesa datos en secciones.

2.5.2. Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

DBSCAN es un algoritmo de *clustering* que es fácilmente aplicado en tiempo real debido a sus características de actualización de la información. Más aún, por la forma en la que este algoritmo crea sus *clusters*, les permite a estos conformar una estructura que va creciendo, esta estructura puede adoptar formas irregulares en el espacio. Es gracias a esto que este es un algoritmo muy versátil para la implementación de muchos algoritmos de *clustering*. Pero por el otro lado esta estructura requiere hacer una comprobación de la información existente en la estructura cada vez que es agregado un dato, esto se convierte en una desventaja debido al costo computacional de esta acción.

Para poder tener una mejor comprensión de las ventajas y desventajas de este algoritmo, sera descrito a continuación.

Teniendo un conjunto de puntos a ser agrupado. Dado ϵ sea un parámetro que especifica el radio de un vecindario con respecto a algún punto. Los puntos serán clasificados como *core points*, que son puntos que poseen una cantidad mínima de vecino cercanos y *outliers* los cuales no los poseen:

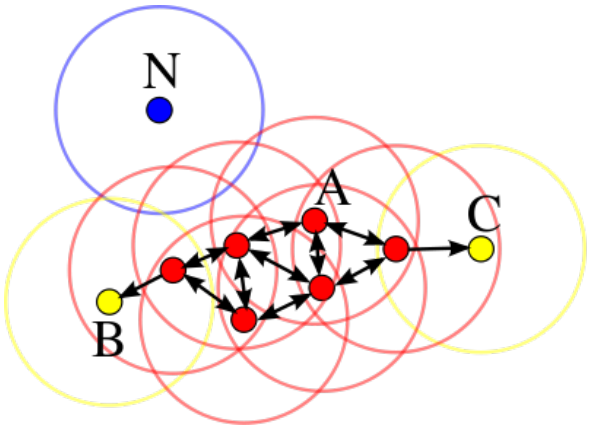

Figura 2.1: Los marcados como A son *core points*, los puntos B y C son puntos que forman aristas con los *core points*, el punto N al no ser alcanzable por ningún *core point* es un *outliers* o ruido. $MinPts = 3$.

Fuente: *Elaboración propia.*

- Un punto p es un *core point* si al menos hay $minPts$ puntos que están dentro de una distancia ϵ de él, incluyéndose a sí mismo.
- Un punto q es directamente alcanzable desde p si el punto q está dentro de la distancia ϵ del *core point* p. Los puntos son solo directamente alcanzables por *core points*.
- Un punto q es alcanzable por p si existe un camino $p_1, ..., p_n$ con $p_1 = p$ y $p_n = q$ donde cada p_{i+1} es directamente alcanzable por p_i. Todos los puntos p_i son *core points* excluyendo al punto q.
- Todos los puntos que no son alcanzable por ningún *core point* son *outliers* o ruido.

Si p es un *core point* entonces este forma un *cluster* con todos los *core points* y los que no lo son que sean alcanzables por él. Cada *cluster* contiene al menos un *core point*, los puntos que no son *core points* pueden ser parte de un *cluster*, pero ellos forman aristas, estos no se pueden utilizar para alcanzar otros puntos, ver Figura 2.1.

DBSCAN procesa cada punto de la base de datos varias veces al agregar un elemento. Esto convierte a este algoritmo dependiente del número de llamados al agregado de elementos y la cantidad de elementos que pertenecen a la estructura. *DBSCAN* ejecuta una consulta por cada punto añadido, si se utiliza una estructura de índice que ejecute la consulta dentro de la estructura en $f(n)$ la complejidad total del algoritmo seria $O(nf(n))$.

2.6. Union Find And Disjoint Sets

En ciencias de la computación el problema de conjuntos disjuntos plantea encontrar una estructura de datos que pueda mantener un seguimiento de un conjunto de datos particionado en un número de conjuntos disjuntos. Existen muchos enfoques para este problema (Galil & Italiano, 1991), sin embargo, la estructura de datos más famoso y eficiente para lidiar con este problema es *Union Find and Disjoint Sets* que está basada en una estructura *disjoint sets forest* que fue propuesta por primera vez por (Galler & Fisher, 1964).

La construcción de esta estructura es la siguiente, dado un arreglo de elementos $S_1, S_2, ..., S_k$, donde k representa el número de elemento en la estructura, S_i señala el índice al que apunta el elemento i, se puede indicar que i es el padre de S_i, en el caso que $S_i = i$ este es el elemento representativo de un conjunto. Sobre esta estructura se definen 3 operaciones.

- *Make-Set(x):* Esta operación crea un nuevo elemento que forma un nuevo conjunto con un identificador único x, este es el elemento representativo de su conjunto. Esta operación aumenta el tamaño de la estructura en uno.

Algoritmo 1: Make-set

```
1 Function Make-set(x) is
2     x.padre = x
3     x.rank = 0
```

- *Find(x):* Dado el elemento S_x, esta operación devuelve el elemento representativo del conjunto al que pertenece S_x.

Algoritmo 2: Find

```
1 Function Find(x) is
2     if x.padre ≠ x then
3         x.padre = Find(x.padre)
4     return x.padre
```

- *Union(x,y):* Dato los elementos S_x y S_y, la operación mezcla los conjuntos S_x y S_y para formar un nuevo conjunto. El elemento representativo de este nuevo conjunto es cualquier elemento de *Find(x)* $\cup$ *Find(y)*. Si los conjuntos son disjuntos esta operación reduce el número de conjuntos en uno.

La implementación de la estructura *disjoint-set forest* representa a un conjunto como un árbol con raíz en el cual cada vértice contiene un elemento y cada árbol representa un conjunto (Cormen et al., 2009). La raíz de cada árbol es el elemento

Algoritmo 3: Union

```
1  Function Union(x,y) is
2      xRaíz = Find(x)
3      yRaíz = Find(y)
4      if xRaíz == yRaíz then
5          return
6      if xRaíz.rank < yRaíz.rank then
7          xRaíz.padre = yRaíz
8      else if if xRaíz.rank > yRaíz.rank then
9          yRaíz.padre = xRaíz
10     else
11         yRaíz.padre = xRaíz
12         xRaíz.rank = xRaíz.rank + 1
```

representativo de cada conjunto, además cada raíz apunta a sí misma como su padre. Usando esta representación podemos realizar las operaciones descritas anteriormente de la siguiente manera: La operación *Make-set* crea un nuevo árbol con un solo vértice que apunta a sí mismo, la operación *Find* comienza de algún vértice del árbol y va avanzando a través de sus padres en dirección a la raíz del árbol o elemento representativo del conjunto, la operación *Union* busca las raíces de cada árbol de los elementos que quieren ser unido y une los árboles volviendo la raíz de un árbol padre del otro, Figura 2.2.

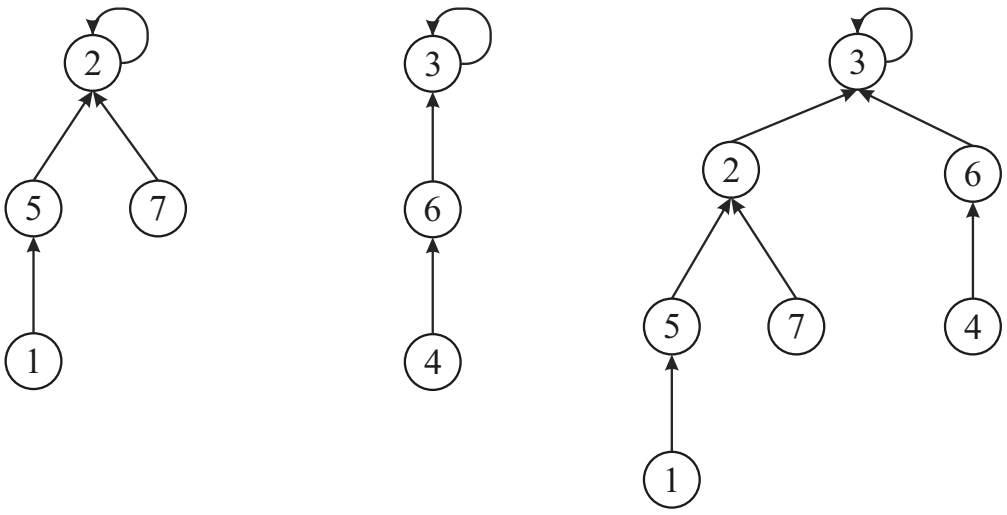

Figura 2.2: El primer árbol (comenzando por la izquierda) representa un conjunto $A = \{1, 2, 5, 7\}$ teniendo 2 como elemento representativo. El segundo árbol representa el conjunto $B = \{3, 4, 6\}$ teniendo 3 como elemento representativo. El tercer árbol representa la unión de conjuntos $A \cup B = \{1, 2, 3, 4, 5, 6, 7\}$ teniendo 3 como elemento representativo.

Fuente: *Elaboración propia.*

Para conseguir un tiempo una complejidad lineal en la ejecución de estas operaciones, es necesario aplicar dos heurísticas: *union by rank* y *path compression*. La primera heurística *union by rank* mantiene un seguimiento de la altura límite de cada elemento, luego cuando una operación *Union* se realiza la raíz con menor rango toma como padre la raíz con mayor rango por lo tanto el árbol no incremente su profundidad y la complejidad no se ve afectada. Cuando los dos árboles tienen la misma altura, la altura del nuevo árbol es como máximo un elemento más grande que los demás árboles. La segunda heurística *path compression* hace que cada nodo visitado por la operación *Find* apunte a la raíz del árbol del que pertenece.

Al aplicarse estas dos simples heurísticas permite conseguir una complejidad temporal casi constante. Dado un n como el número de operaciones *Make-set* y m como el número total de operaciones *Make-set*, *Find* y *Union*. Si solo es usada la heurística *union by rank* se puede conseguir una complejidad computacional de $O(m \log n)$. Si solo la heurística *path compression* es utilizada se puede conseguir una complejidad de $\theta(n + f(1 + \log_{2+f/n} n))$ donde f es el número de operación *Find* (Cormen et al., 2009). Usando ambas heurísticas *union by rank* y *path compression* la complejidad se reduce a $O(m\alpha(n))$ (Tarjan & van Leeuwen, 1984) donde $\alpha(n)$ es el inverso de la función de Ackermann (W. Ackermann, 1928) que crece extraordinariamente lento, por lo que este factor es 4 o menos para cualquier n que pueda escribirse en el universo físico. Esto hace que las operaciones de esta estructura se amorticen prácticamente a un tiempo constante.

2.7. Espacios L^p

Los espacios L^p son espacios funcionales definidos como una generalización natural de la norma-p para espacios vectoriales de dimensión finita. Son también llamados espacios de Lebesgue por Henri Lebesgue (Dunford & Schwartz, 1971).

La longitud de un vector $x = (x_1, \ldots, x_n)$ en el espacio vectorial real n-dimensional $\mathbb{R}^n$ es usualmente dada por la norma Euclidiana:

$$\|x\|_2 = \left(x_1^2 + x_2^2 + \cdots + x_n^2\right)^{1/2}$$

La distancia Euclidiana entre dos puntos x y y es la longitud $|x - y|_2$ de la línea recta entre los dos puntos. En muchas situaciones, la distancia Euclidiana es insuficiente para capturar la distancia real en un espacio dado. Una analogía a esto es sugerida por un conductor de taxi en un conjunto de calles con forma de cuadrícula. Este medía las distancias no en términos de la longitud de la línea recta entre los dos puntos, más bien en términos de las distancias rectilineares (las componentes de la recta), lo que tiene en cuenta que las calles son perpendiculares o paralelas unas con otras. Las clases de norma$-p$ generaliza ambos ejemplos de la siguiente manera:

Para un número real $p \geq 1$, la norma-p o norma$-L^p$ de x está definida por:

$$\|x^p\| = (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{1/p}$$

Las barras de valor absoluto son innecesarias cuando p es un número racional y, en su forma reducida, tiene un numerador par. De lo anterior, la norma Euclidiana se encuentra en esta clase y es la norma-2, y la norma-1 corresponde con la distancia rectilineal (Manhattan distance) La Figura 2.3 muestra cómo varía la forma de una circunferencia dependiendo de la norma que se utiliza para la distancia.

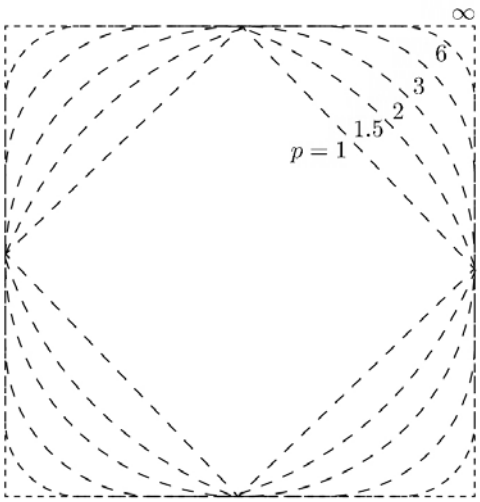

Figura 2.3: Circunferencia definida en los espacios L^p con $p = 1, 1.5, 2, 3, 6$ y ∞

Fuente: *Elaboración propia.*

La norma L^∞ o norma máxima (o norma uniforme) es el límite de las normas L^p cuando $p \to \infty$. Este límite es equivalente a la siguiente definición:

$$\|x^p\|_\infty = \lim_{p\to\infty} \|x\|_p = \max\{|x_1|, |x_2|, ..., |x_n|\}$$

Para todos los $p \geq 1$, las normas-L^p y L^∞ satisfacen las propiedades de una "función de longitud" (o norma), que son:

- Solo el vector cero tiene longitud cero.
- La longitud de un vector es positiva y homogénea con respecto a la multiplicación por un escalar (homogeneidad positiva).
- La longitud de la suma de dos vectores no es mayor a la suma de longitudes de los vectores (desigualdad triangular).

La distancia rectilineal entre dos puntos nunca es menor que la distancia del segmento entre ellos. Formalmente esto significa que la norma Euclidiana de cualquier vector está acotada por su norma-1:

$$\|x\|_2 \leq \|x\|_1$$

Esta propiedad se generaliza a las normas-p en que la norma-p $||x||_p$ de cualquier vector x no crece con p:

$||x||_{p+a} \leq ||x||_p$ para cualquier vector x y números reales $p \geq 1$ y $a \geq 0$

En la dirección opuesta, se tiene una relación entre la norma-1 y la norma-2:

$$\|x\|_1 \leq \sqrt{n}\|x\|_2$$

Esta desigualdad depende de la dimensión n del espacio vectorial subyacente y es consecuencia directa de la desigualdad de Cauchy-Schwarz.

2.8. Métricas de evaluación de clustering

Después de haber realizado todo el proceso del *clustering* es necesario realizar comprobaciones sobre los resultados obtenidos para poder garantizar que nuestro algoritmo cumple con los requerimientos de precisión. Para ellos necesitamos una métrica que pueda darnos esta información sabiendo que los datos pueden estar etiquetados de formas muy diferentes y variadas. Para ellos se tomará en cuenta la métrica *V-measure score*.

2.8.1. V-measure score

V-measure es una medida basada en entropía que mide explícitamente cuanto se satisfizo el criterio de homogeneidad y completitud. *V-measure* es calculado como la media armónica de distintos valores de homogeneidad y completitud, de la manera que precisión y *recall* son comúnmente combinados en *F-measure*. De la misma manera que los puntajes de *F-measure* se pueden ponderar, *V-measure* puede ser ponderado a favor de la contribución de la homogeneidad y la completitud (Andrew & Hirschberg, 2007).

Asuma que el conjunto de datos comprende N puntos, y dos particiones de este: un conjunto de clases, $C = \{c_i | i = 1, ..., n\}$ y un conjunto de *clusters*, $K = \{k_i | i = 1, ..., n\}$. Sea A la tabla de contingencia producida por el algoritmo de *clustering* contra la solución real del *clustering*, de modo que $A = \{a_{ij}\}$ donde a_{ij} es el número de puntos que son miembros de la clase c_i y elementos del *cluster* k_j.

Para discutir la evaluación de la medida, son introducidos dos criterios para la solución del *clustering*: homogeneidad y completitud. Un resultado de *clustering* satisface la homogeneidad si todos sus *clusters* contienen solo puntos que pertenecen a una sola clase. Un resultado del *clustering* satisface la completitud si todos los puntos que son miembros de una clase dada son elementos del mismo *cluster*. La homogeneidad y la completitud son medidas que van en diferentes direcciones, mientras se incremente la homogeneidad de la solución usualmente indica un decremento en la completitud de la solución. Se define como la medida de distancia a un *clustering* perfecto como la media armónica ponderada de la homogeneidad y la completitud.

Homogeneidad: Para satisfacer el criterio de homogeneidad, el *clustering* debe asignar únicamente los puntos que son miembros a una clase a un solo *cluster*. Dado un

clustering se determina cuán cerca está a su ideal examinando la entropía condicional de la distribución de las clases del *clustering* propuesto. En un caso perfecto $H(C|K)$ es 0. Sin embargo, en una situación imperfecta el tamaño del valor depende del tamaño del *dataset* y de la distribución del tamaño de las clases.

$$c = \begin{cases} 1 & \text{if } H(C,K) = 0 \\ 1 - \dfrac{H(C|K)}{H(C)} & \text{else} \end{cases}$$

donde

$$H(C|K) = -\sum_{k=1}^{|K|}\sum_{c=1}^{|C|}\frac{a_{ck}}{N}\log\frac{a_{ck}}{\sum_{c=1}^{|C|}a_{ck}}$$

$$H(C) = -\sum_{c=1}^{|C|}\frac{\sum_{k=1}^{|K|}a_{ck}}{n}\log\frac{\sum_{k=1}^{|K|}a_{ck}}{n}$$

Completitud: Completitud es simétrico a la homogeneidad. Para satisfacer el criterio de completitud, el *clustering* debe asignar todos los puntos que son miembros de una sola clase, a un solo *cluster*. Para evaluar la completitud, examinamos la distribución de las asignaciones de los *clusters* a cada clase. En una solución perfecta para la completitud cada una de estas distribuciones será segmentada en un solo *cluster*. Se puede calcular el grado de la segmentación calculando la entropía condicional del *cluster* propuesto. En el caso de completitud perfecta $H(C|K) = 0$. Sin embargo, en el peor escenario cada clase está representada por cada grupo con una distribución igual a la distribución del tamaño del grupo, $H(C|K)$ es máxima e igual a $H(K)$.

$$h = \begin{cases} 1 & \text{if } H(K,C) = 0 \\ 1 - \dfrac{H(K|C)}{H(K)} & \text{else} \end{cases}$$

donde

$$H(K|C) = -\sum_{c=1}^{|C|}\sum_{k=1}^{|K|}\frac{a_{ck}}{N}\log\frac{a_{ck}}{\sum_{K=1}^{|K|}a_{ck}}$$

$$H(K) = -\sum_{k=1}^{|K|}\frac{\sum_{c=1}^{|C|}a_{ck}}{n}\log\frac{\sum_{c=1}^{|C|}a_{ck}}{n}$$

Basado en estos cálculos de homogeneidad y completitud, calculamos el *V-measure* de solución de *clustering* calculando la media armónica ponderada de la homogeneidad y la completitud.

$$V_\beta = \frac{(1+\beta)\times h \times c}{(\beta \times h) + c}$$

Similar a *F-measure*, si β es mayor que 1 la completitud es ponderada más fuertemente en el cálculo, si β es menor a 1 la homogeneidad es más ponderada.

Capítulo 3

Desarrollo de la investigación

El algoritmo planteando para esta investigación llamado *SetClust*, como fue introducido en la sección 1.3.1, se enfoca en el problema explicado anteriormente *Data Streaming Clustering*, con un enfoque para conjuntos amorfos y *outliers* en los datos. Esto implica que el número de *clusters* formados con los datos puede ser incrementado o decrementado durante el tiempo de ejecución, tratando de mantener la menor información que resuma los *clusters* durante la ejecución. Además, la forma de los conjuntos que el algoritmo descubrirá no necesariamente tienen forma euclídea, y los *clusters* pueden tener datos *outliers* que no afectan al proceso de *clustering*.

Si esta tarea fuese abordada por un algoritmo clásico de *clustering* como por ejemplo *K-means* (o algún otro algoritmo basado en centroides), sería necesario calcular el número de *cluster* utilizando otros métodos cada vez que un dato sea recibido, esto claramente incrementaría el tiempo de ejecución del proceso de *clusterización* para algoritmos clásicos en condiciones especiales como puntos que arriban mientras el algoritmo se ejecuta, *outliers* que afecten la precisión del algoritmo o *clusters* amorfos que prácticamente vuelvan imposible el proceso de *clustering*.

Si bien esta propuesta no requiere una información previa del número de *clusters* que van a ser tratados como en otros algoritmos, si se requiere de otro tipo de información, que en este caso sería un ligero conocimiento sobre la distribución de los datos, y la separación de los *cluster*. Como veremos a continuación se utilizarán hiperparámetros en las funciones, que guardan relación con estos datos. Si no conocemos a la perfección estos valores probablemente tenga un mayor impacto en el tiempo de ejecución que en la precisión de los *clusters* formados.

3.1. Estructura de resumen de información

Por la definición de este problema se puede asegurar que la cantidad de información que tendrá que ser procesada puede ser bastante extensa, incluso infinita, por ende, es crucial poder manejar la información que será recibida. Pero lograr manejar esta

cantidad de información es problemático, consecuentemente es necesario reducir el tamaño de los datos perdiendo la menor cantidad posible de información para asegurar una representación fiel de los datos.

Para ello se utilizó la estructura de información conocida como *feature vector*, ver sección 2.4.1, esta es una estructura de datos de tipo vector la cual almacena la información estadística de un conjunto de datos, por lo que un solo *feature vector* puede resumir la información de un conjunto extenso de datos sin tener que almacenarlo como tal. Aparte de los estadísticos normalmente almacenados en la estructura como la media, suma de cuadrados y cantidad de datos, se almacenará una etiqueta la cual indicará el identificador de esta estructura para ser utilizado en el enlazamiento de micro-clusters, ver sección 3.3.3, además se almacena la desviación estándar que es un valor que puede ser calculado utilizando los anteriores estadísticos pero puede ser diferente según existan uno o más datos almacenados, teniendo una desviación estándar inicial σ_{ini}, ver sección 3.3.1.

Sea m_i un *feature vector* en un espacio d-dimensional con los siguientes elementos:

$$m_i = (l_i, \mu_i, \sigma_i, s_i, n_i)$$

1. l_i la etiqueta o identificador del vector, $l_i \in \mathbb{N}$.
2. μ_i la media aritmética de los puntos que pertenecen al vector, $\mu_i \in \mathbb{R}^d$.
3. σ_i la desviación estándar de los datos del vector, $\sigma_i \in \mathbb{R}^d$.
4. s_i la suma de los cuadrados de los puntos que pertenecen al vector, $s_i \in \mathbb{R}^d$.
5. n_i el número de datos, $n_i \in \mathbb{N}^+$.

$$m_i = \begin{array}{|c|c|c|c|c|} \hline & \mu_{i_1} & \sigma_{i_1} & s_{i_1} & \\ l_i & \vdots & \vdots & \vdots & n_i \\ & \mu_{i_d} & \sigma_{i_d} & s_{i_d} & \\ \hline \end{array}$$

Utilizando esta estructura podremos reducir la gran cantidad de datos recibidos en un único puñado de información que será de fácil acceso, almacenamiento y actualización.

3.2. Métodos de clustering

Una vez definida la estructura que representa a los datos en un plano d-dimensional, se debe definir bajo qué criterio las estructuras se relaciona para definir la disposición final de los clusters. Para ello existen varios métodos de clustering los cuales utilizan la información sin procesar o la información resumida para determinar las relaciones subyacentes. Los métodos a considerar serán: basados en centroides y basados en densidades.

Al haber resumido la información de los datos en estas estructuras, se tiene una nueva representación de los datos en el espacio, una representación que no favorece al uso del método basado en centroides debido a que ya no se tienen la posición original de los datos, siendo esto imprescindible para la aplicación de este algoritmo. Además, el objetivo de este método es la minimización de una función de coste definida por cada algoritmo, y este no es el objetivo de la investigación. Otro característica de este método es que particiona el espacio en un número k de particiones convexas, limitando el objetivo principal del trabajo, ver la Figura 3.1.

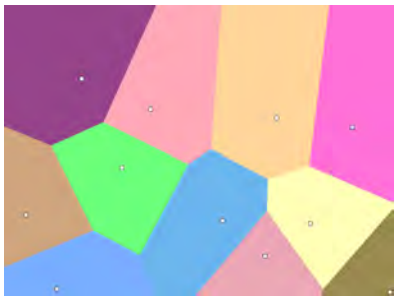

Figura 3.1: Diagrama de Voronoi, partición del espacio dados k centros

Fuente: *Elaboración propia.*

Debido al comportamiento del algoritmo *DBSCAN*, descrito en la sección 2.5.2, se puede asegurar que los algoritmos basados en densidades nos permiten descubrir el número de cluster sin información previa al respecto, además, la forma en la que se enlazan los datos le permite crear estructuras más complejas que representan de manera más precisa la forma real de los clusters. Por lo tanto, este es la propuesta que más se adaptaría al objetivo de esta investigación. Pero es importante tener en cuenta que el algoritmo objetivo de esta investigación no puede ejecutarse sobre los datos sin procesar previamente debido al alto coste computacional que eso provocaría. Por esta razón se planteó un nuevo algoritmo que mantiene las propiedades propuestas por *DBSCAN* utilizando *Union Find and Disjoint sets* como estructura de unión de los *micro-cluster* y *feature vector* como estructura para el preprocesado de los datos. De esta manera podemos mejorar la eficiencia mientras mantenemos sus ventajas.

3.3. Desarrollo del algoritmo

Durante el desarrollo del algoritmo se utilizaron los términos *cluster* y *micro-cluster* de manera independiente: el término *micro-cluster* para definir la unidad mínima de información agrupada utilizando la estructura de resumen de información *feature vector* descrita en la sección 3.1, y el término *cluster* como un conjunto de uno o más *micro-clusters* unidos bajo la misma etiqueta, la unión de estos *micro-clusters* se realizó mediante la estructura de indexación *Union Find and Disjoint sets* descrita en la sección 2.6.

La estructura *Union Find and Disjoint sets* es construida sobre todos los *micro-clusters* creados. Esto significa que para cada *micro-cluster* tenemos que guardar un

conocimiento sobre el padre y el rango del mismo, esto nos permite hallar el elemento representativo de cada *cluster*. Para la determinación de la pertenencia de cada punto a un *cluster* es necesario guardar la etiqueta de cada punto al *micro-cluster* que pertenece. Es importante aclarar que esta última parte no es necesaria para la ejecución del algoritmo debido a que tiene un costo computacional (tiempo y memoria) alto al tener que almacenar toda la información, solo debe ser realizada cuando se requiere conocer la pertenencia de los datos y puede ser desechada en cualquier momento. El objetivo del algoritmo es poder realizar un *clustering* eficaz reduciendo el uso de memoria y tiempo computacional.

Teniendo esta información, se definieron tres operaciones que redefinirán el algoritmo usado para el *clustering*, estas operaciones que representan las acciones que deben ser ejecutadas en las siguientes situaciones: Agregación de un elemento, unión de *micro-clusters*, y absorción de *micro-clusters*. Las tres operaciones serán descritas a continuación.

3.3.1. Agregación de elementos

El propósito de la operación agregación de elementos es añadir la información de un punto a un *micro-cluster*. Esto es conseguido de la siguiente manera: dado un punto $p \in \mathbb{R}^d$, se debe encontrar el *micro-cluster* con la media más cercana $\mu_i \in \mathbb{R}^d$ teniendo una desviación estándar de $\sigma_i \in \mathbb{R}^d$. Luego, definiremos una función $f : \mathbb{R}^d \mapsto \mathbb{R}$ y un número $c \in \mathbb{R}^+$ como hiperparámetro. Esta función puede ser definida como la media, la mediana, máximo, la distancia Euclídea o cualquier tipo de métrica que vaya acorde con los datos. A continuación, el punto p es añadido al *micro-cluster* si la siguiente inecuación 3.1 se cumple.

$$f\left(|\mu_i - p|./\sigma_i\right) < c \tag{3.1}$$

Donde "./" es la división elemento a elemento de dos vectores de la misma dimensión. Intuitivamente un punto es añadido a su *micro-cluster* si este se encuentra dentro de las c desviaciones estándar de la media del *micro-cluster* cuando la función f es aplicada a todos sus componentes.

Dada esta condición, existen dos posibles escenarios. Primero, el punto no pertenece a un *micro-cluster*. En este caso se creará un nuevo *micro-cluster* teniendo como único elemento el mismo punto. La media del *micro-cluster* será el mismo punto, y la desviación estándar inicial será fijada a σ_{ini}[1] siendo este último un hiperparámetro. Luego, la información de este nuevo *micro-cluster* es agregada a la estructura *disjoint set* utilizando la función propia de esta estructura *Make-Set(l_i)*. Segundo, el punto pertenece a un *micro-cluster*. En este caso, la información del *micro-cluster* debe ser actualizada, en otras palabras, actualizar la media, desviación estándar y el número de elementos. La actualización de la media con la fórmula tradicional requeriría el almacenamiento y la lectura de todo el conjunto de datos que pertenecen al *micro-cluster*. Sin embargo, podemos deducir la siguiente fórmula 3.2 para actualizar este valor en

[1]Hablando estrictamente, la desviación estándar para un único elemento debería ser 0, para fines prácticos será σ_{ini}.

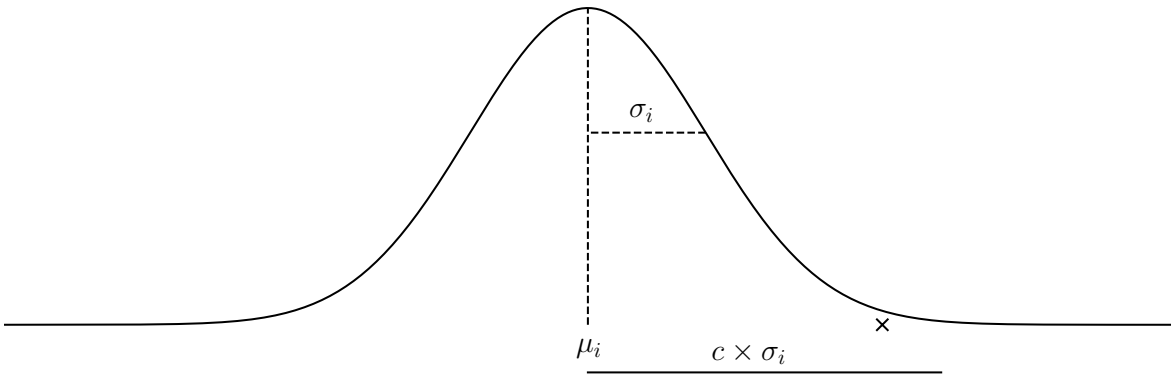

Figura 3.2: Este es un ejemplo de una dimensión de un *micro-cluster* y un punto satisfaciendo la condición de agregación de elemento. El término "x" representa un nuevo punto dentro del límite de aceptación del *micro-cluster*.

Fuente: *Elaboración propia.*

tiempo constante.

$$\begin{aligned} \mu_i^{(n+1)} &= \frac{1}{n+1} \sum_{j=1}^{n+1} x_j \\ &= \frac{1}{n+1} \left(\sum_{j=1}^{n} x_j + x_{n+1} \right) \\ &= \frac{1}{n+1} \left(\mu_i^{(n)} \times n + x_{n+1} \right) \end{aligned} \tag{3.2}$$

Con este resultado, podemos fácilmente actualizar la media cuando un punto x_{n+1} es agregado a un *micro-cluster* en tiempo constante solo usando la media previa $\mu_i^{(n)}$ y el número de elemento n. Podemos conseguir una fórmula similar para la actualización

de la desviación estándar sacando ventaja de la fórmula de la varianza.

$$\begin{aligned}
\sigma_i^{(n)} &= \sqrt{\frac{1}{n}\sum_{j=1}^{n}\left(x_j - \mu_i^{(n)}\right)^2} \\
&= \sqrt{\frac{1}{n}\sum_{j=1}^{n}\left(x_j^2 - 2x_j\mu_i^{(n)} + (\mu_i^{(n)})^2\right)} \\
&= \sqrt{\frac{1}{n}\left(\sum_{j=1}^{n}x_j^2\right) - \frac{2\mu_i^{(n)}}{n}\left(\sum_{j=1}^{n}x_j\right) + \frac{(\mu_i^{(n)})^2}{n}\left(\sum_{j=1}^{n}1\right)} \\
&= \sqrt{\frac{1}{n}\left(\sum_{j=1}^{n}x_j^2\right) - 2(\mu_i^{(n)})^2 + \frac{(\mu_i^{(n)})^2}{n}(n)} \\
&= \sqrt{\frac{1}{n}\left(\sum_{j=1}^{n}x_j^2\right) - (\mu_i^{(n)})^2} \qquad (3.3)
\end{aligned}$$

La fórmula simplificada (3.3) implica que para conseguir la desviación estándar, solo necesitamos mantener la suma de todos los elementos al cuadrado s_i que se puede actualizar en tiempo constante, y el cuadrado de la media $\mu_i^{(n)}$ (previamente calculada).

Algoritmo 4: Agregación de elementos

```
1  Function Agregacion_elementos(m_i, p, UF) is
2      if p ∈ m_i then
3          n_i = n_i+1
4          μ_i = (μ_i .* (n_i - 1) + p)./n_i
5          s_i = s_i + p^2
6          σ_i = sqrt(s_i/n_i-μ_i^2)
7          if σ_i < σ_ini then
8              σ_i = σ_ini
9      else
10         m_j = (l_j, p, σ_ini, p_i^2, 1)
11         UF.Make_set(l_j)
```

En el Algoritmo 4 observamos el proceso que se lleva a cabo durante la ejecución de la operación Agregación de elemento, donde m es el *micro-cluster* elegido para la condición de agregación, p es el punto que se pretende agregar, UF es la estructura *Union Find and Disjoint sets* que posee todas las funciones definidas en la sección 2.6. Las operaciones matemáticas que poseen un "." antes del operador corresponden a una operación de elementos de vectores uno a uno. En el momento de comprobar si la desviación estándar del *micro-cluster* es menor que σ_{ini} esto se realiza elemento por elemento en este vector, y solo se modifican los que son menores.

Ambos cálculos tienen una complejidad computacional de $O(d)$, siendo d el número de dimensiones de los puntos. La complejidad completa de la operación agregación de

elementos es $O(rd)$, siendo r el número actual de *micro-clusters* creados. Nótese que modificando de esta manera la forma en la que se agregan los datos a los *micro-cluster* se reduce de gran manera el tiempo de formación de estos debido a que ya no depende de la cantidad de datos procesados hasta el momento.

Esta operación lleva a cabo la construcción de la estructura de *micro-cluster* y el agregado de la información a estos mismos. Pero no son las únicas situaciones que se puede dar durante la ejecución de este algoritmo. Para lo cual las siguientes operaciones resuelven las demás situaciones.

3.3.2. Absorción de micro-clusters

La operación de absorción de *micro-cluster* es responsable de convertir dos *micro-clusters* cercanos en uno solo cuando se descubre que uno puede estar incluido en el otro. La operación se define de la siguiente manera: dados dos *micro-clusters* con una media y una desviación estándar μ_i, σ_i y μ_j, σ_j respectivamente y el hiperparámetro c, los dos *micro-clusters* deben convertirse en uno solo cuando un *micro-cluster* y el espacio comprendido por sus c desviaciones estándar están incluidas dentro de las c desviaciones estándar del otro *micro-cluster* en todas las componentes. Formalmente, si la siguiente ecuación se cumple:

$$|\mu_i - \mu_j| + c \times \sigma_j <^d c \times \sigma_i \tag{3.4}$$

Donde "$<^d$" representa el operador menor para todas las componentes del vector. Es importante aclarar que para la condición de esta operación no es importante la función f ya que sin importar que función se elija los límites que incluyen un *micro-cluster* en otro seguirán siendo los mismos, ver la Figura 3.3 para un ejemplo en 1 dimensión.

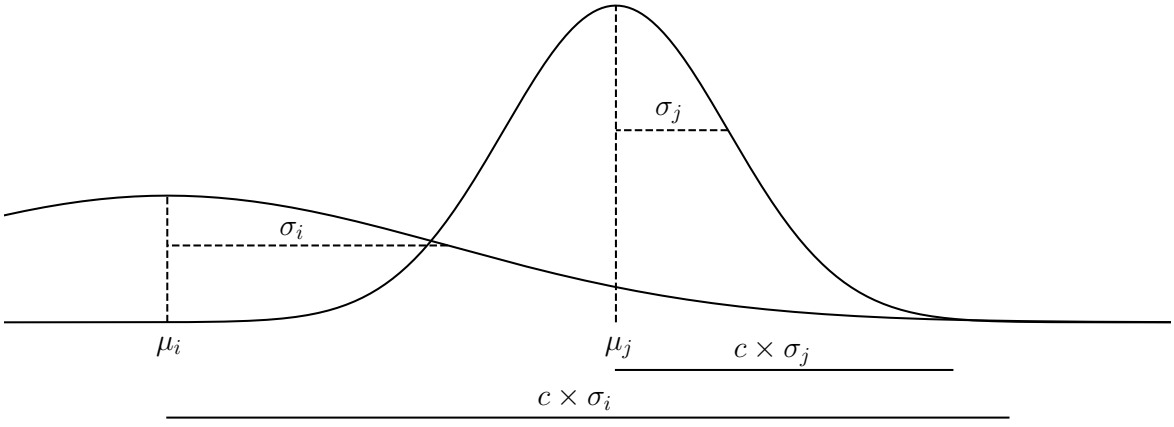

Figura 3.3: Ejemplo en una dimensión de dos *micro-clusters* que satisfacen la condición de absorción. El *micro-cluster* con media μ_j está dentro del límite de pertenencia del *micro-cluster* con media μ_i.

Fuente: *Campos et al., 2020.*

Cuando la condición de absorción es cumplida, se procede a unir ambos *micro-clusters* compartiendo la misma información. La media de ambos *micro-clusters* puede ser calculada con una generalización de la fórmula hallada anteriormente (3.2). Si tenemos dos *micro-cluster* con medias $\mu_i^{(n)}$ y $\mu_j^{(m)}$ y número de elementos n y m respectivamente, una de las dos medias $\mu_{i|j}$ es actualizada de la siguiente manera:

$$\mu_{i|j}^{(n_i+n_j)} = \frac{\mu_i^{(n_i)} \times n_i + \mu_j^{(m_j)} \times n_j}{n_i + n_j} \tag{3.5}$$

Este resultado, de la misma manera que (3.2) es útil para unir dos *micro-clusters* sin tener almacenados todos los elementos que pertenecen a ambas estructuras. La desviación estándar puede ser directamente calculada usando (3.3), solamente debe calcularse la suma de los cuadrados de los dos *micro-clusters* $s_{i|j} = s_i + s_j$. El número de elementos es también calculado fácilmente sumando el número de elementos de ambas partes $n_{i|j} = n_i + n_j$.

Una vez actualizada la información del *micro-cluster* es necesario actualizar la información de la estructura *Union Find*. Para esto debemos tener en cuenta la jerarquía de ambos *micro-clusters*, como estos no pertenecen a una misma rama se aplica la operación *Union*(l_i, l_j) como fue mostrado en la sección 2.6. De esta manera todos los *micro-clusters* que estén dentro del *cluster* se unirán al *micro-cluster* que los absorbe.

En el Algoritmo 5 observamos el proceso que se lleva a cabo durante la ejecución de la operación absorción de *micro-clusters*, donde m_i es el *micro-cluster* elegido para la condición de absorción UF es la estructura *Union Find and Disjoint sets* que posee todas las funciones definidas en esta estructura. Las operaciones matemáticas que poseen un "." antes del operador corresponden a una operación de elementos de vectores uno a uno. En el momento de comprobar si la desviación estándar del *micro-cluster* modificado es menor que σ_{ini} esto se realiza elemento por elemento en este vector, y solo se modifican los que son menores.

La función de absorción es ejecutada hasta que no existan dos *micro-clusters* que cumplan la condición. Es posible que uniendo dos estructuras se cree la necesidad de seguir ejecutando más de estas operaciones debido a que los valores que describen la forma y el tamaño del *micro-cluster* han variado y este puede cumplir esta condición con otros *micro-clusters* adyacentes. La absorción de un *micro-cluster* tiene una complejidad computacional de $O(r^2 d)$. Dependiendo de la implementación, podemos comprobar todos los *micro-clusters* una sola vez para esta condición [2]. Esta operación cumple un papel importante no solo dar una funcionalidad necesaria al algoritmo, sino que ejecutarse esta operación el número de *micro-cluster* r se reduce en 1 cada vez que se cumple.

[2]Si se aprovecha el hecho que solo el ultimo *micro-cluster* modificado puede ser inestable para esta condición, podemos conseguir un peor caso de complejidad computacional de $O(rd)$.

Algoritmo 5: Absorción de micro-clusters

```
1  Function Absorcion_micro_clusters(m_i, UF) is
2      existe_intersecion = verdad
3      while existe_intersecion do
4          existe_intersecion = falso
5          for m_j ∈ M do
6              if m_i ⊂ m_j | m_j ⊂ m_i then
7                  μ_i = (μ_i. * n_i + μ_j. * n_j)./(n_i * n_j)
8                  s_i = s_i + s_j
9                  n_i = n_i + n_j
10                 σ_i = sqrt(s_i./n_i − μ_i^2)
11                 if σ_i <^d σ_ini then
12                     σ_i:=σ_ini
13                 UF.Union(l_i, l_j)
14                 existe_intersecion = verdad
```

3.3.3. Enlazamiento de micro-clusters

El objetivo de la operación enlazamiento de *micro-clustrers* es determinar si dos *micro-clusters* están lo suficientemente cerca uno del otro para ser considerados parte de un mismo *cluster*. Dados dos *micro-cluster* con medias μ_i, μ_j y una desviación estándar σ_i, σ_j respectivamente, se dice que ambos *micro-clusters* pueden ser enlazados si se cumple la siguiente inecuación:

$$f\left(|\mu_i - \mu_j|./\sigma_i\right) < c \tag{3.6}$$

En esta función se usa el previamente definido hiperparámetro c y la función f. Intuitivamente, la condición verifica si la media del *micro-cluster* está dentro del espacio que comprende c veces la desviación estándar de otro *micro-cluster* cercano, ver la Figura 3.4 para un ejemplo en una dimensión.

Una vez determinado por la condición si dos *micro-clusters* serán enlazados, la operación *Union*(l_i,l_j) nativa de la estructura *Union Find and Disjoint sets* es aplicada sobre sus etiquetas, esto significa que ambos *micro-clusters* pertenecerán al mismo *cluster*. Ejecutando esta operación de esta manera, conseguimos que el algoritmo enlace *micro-clusters* que pueden crear formas de *clusters* más complejas que formas euclidianas clásicas.

En el Algoritmo 6 observamos el proceso que se lleva a cabo durante la ejecución de la operación enlazamiento de *micro-clusters*, donde m_i es el *micro-cluster* elegido para la condición de enlazamiento UF es la estructura *Union Find and Disjoint sets* que posee todas las funciones definidas en esta estructura. Usando la función *Find*(l_i) podemos encontrar si elemento representativo, de tal manera que si ambos *micro-clusters* tienen el mismo elemento representativo ya no es necesario enlazarlos.

La operación de enlazamiento de *micro-clusters* es ejecutada después de la opera-

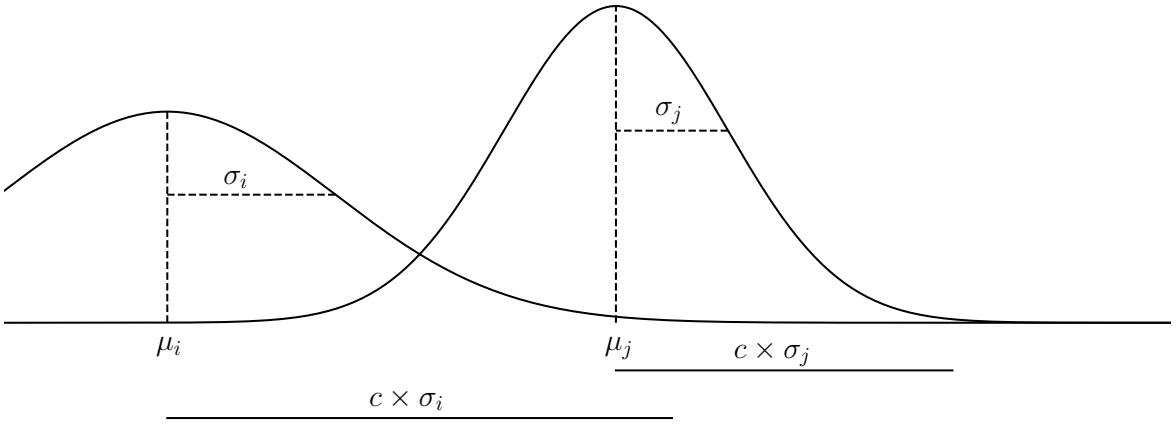

Figura 3.4: Ejemplo de una dimensión de dos *micro-cluster* que satisfacen la condición de enlazamiento. El *micro-cluster* con media μ_j está dentro del límite de cumplimento de la condición del *micro-cluster* con media μ_i. Nótese que la condición de absorción no se cumple en este caso.

Fuente: *Elaboración propia.*

Algoritmo 6: Enlazamiento de micro-clusters

```
1 Function Enlazamiento_micro_clusters(m_i, UF) is
2     for m en M do
3         if UF.Find(l_i) ≠ UF.Find(l_j) then
4             if μ_i ∈ m_j | μ_j ∈ m_i then
5                 UF.Union(l_i, l_j)
```

ción absorción de *micro-clusters* debido a que si dos *micro-cluster* satisfacen la condición de absorción, entonces estos satisfacen la condición de enlazamiento con una alta probabilidad. Se restringe el enlazamiento de *micro-cluster* que pueden ser absorbidos por algún otro *micro-cluster* para poder reducir el número de estos, ya que enlazándolos podrían tener igualmente un buen funcionamiento. Dependiendo de la implementación esta operación puede ser ejecutada en tiempo lineal sobre el número de *micro-clusters*[3], se puede ejecutar esta operación un peor caso computacional de $O(\alpha(r)rd)$.

Es importante aclarar que siempre que un nuevo punto aparece, el algoritmo ejecuta las tres operaciones (agregación de elementos, absorción de *micro-cluster*, y enlazamiento de *micro-clusters* en este orden) a toda la estructura. Las tres operaciones son independientemente ejecutadas una tras otra. Cómo fue mencionado anteriormente, el peor caso de las operaciones son: $O(rd)$, $O(rd)$ (a lo mucho $O(r)$ veces), y $O(\alpha(r)rd)$

[3]Como en el caso anterior, si se aprovecha el hecho que solo el ultimo *micro-cluster* actualizado puede generar una inestabilidad de esta condición.

respectivamente. Si asumimos que $\alpha(r)$ es una constante pequeña[4], la complejidad computacional completa del algoritmo cada vez que se añade un nuevo punto es de $O(r^2 d)$.

Al definir estas operaciónes para el tratamiento de los datos se logra mejorar las deficiencias que tiene *DBSCAN* a la hora de manejar gran cantidad de datos, ya que este algoritmo dependería de n siendo este el número total de puntos en la estructura. Pero al realizar estas modificaciones podemos cambiar la variable del número total de puntos al número total de *micro-clusters* r, teniendo esta una relación sublineal o incluso constante con la variable n. Sabiendo que los *micro-clusters* tiene formas pseudo elípticas, podemos intuir que mientras más simples sean las formas de los *cluster* a los que se vaya a aplicar este algoritmo menos *micro-cluster* formará este, debido a que con pocos *micro-cluster* se puede amoldar a las formas de los conjuntos. Pero mientras más complejas sean estas formas requerirá más *micro-clusters* para adaptarse a su forma, lo que tiene una implicación directa en el tiempo computacional.

Cuando dos o más *micro-clusters* son enlazados, se considera que estos pertenecen al mismo *cluster*. Esta propiedad fue especialmente escogida para el algoritmo debido a la ventaja que da cuando los *micro-cluster* enlazados pueden formar *clusters* con formas no euclídeas. Es fácil ver que cuando dos o más *micro-clusters* son absorbidos en uno solo la forma del *micro-cluster* resultante será pseudo elíptica. Sin embargo, cuando dos o más *micro-clusters* se enlazan unos con otros, entonces la forma resultante puede ser más sofisticada. La operación de enlazamiento puede continuar mientras se forman estructuras más exóticas, ver Figura 3.5.

En cualquier momento, se puede conocer si dos elementos etiquetados por l_i, l_j respectivamente pertenecen al mismo *cluster* ejecutando la función *Find(l_k)* sobre cada etiqueta y comparando si su elemento representativo es el mismo. Además, en cualquier momento el algoritmo puede devolver una lista de todas las etiquetas de los puntos hasta el momento (no es parte esencial del algoritmo). El algoritmo también puede devolver la estructura completa que contiene toda la información usada para definir completamente los *micro-cluster* y los *clusters* con el objetivo de poder seguir realizando *clustering* si es necesario. Se puede observar un ejemplo en pseudocódigo del algoritmo propuesto en Algoritmo 7.

Algoritmo 7: Algoritmo SetClust

```
1 while Un punto p arriba do
2     m = El micro-cluster más cercano a p
3     Agregacion_elementos(m, p, UF)
4     Sea m_i el micro-cluster donde p ∈ m_i
5     Absorcion_micro_clusters(m_i, UF)
6     Enlazamiento_micro_clusters(m_i, UF)
```

Cuando un punto aparece, la operación de agregación de elementos es ejecutada. Esto lleva a la creación de otro *micro-cluster* o al añadido del punto a uno existente.

[4]Donde $\alpha(n)$ es la inversa de la función de Ackermann, de manera que $\alpha(n) < 5$ para todos los prácticamente remotos valores de n.

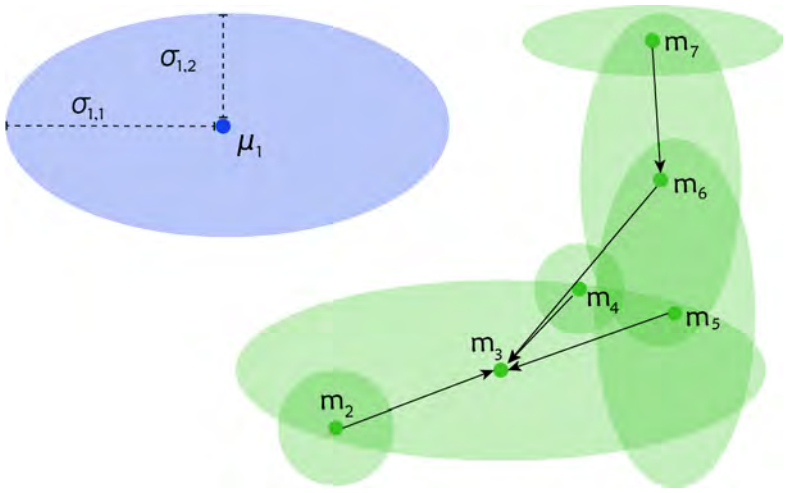

Figura 3.5: Ejemplo de la construcción de dos *clusters* en 2 dimensiones. El primero *cluster* formado solo por un *micro-cluster* m_1 es un *cluster* simple que no requiere de formas complejas para ser representado. El segundo *cluster* formado por los *micro-clusters* $\{m_2, m_3, m_4, m_5, m_6, m_7\}$ es un *cluster* que requiere de formas más complejas para poder ser representado. Este *cluster* tiene como elemento representativo el *micro-cluster* m_3, ya que este viene a ser el padre en la estructura *Union Find*. El *micro-cluster* m_6 no tiene como padre al m_5, debido a la función interna *Path Compresion* que acorta los caminos entre las hojas de la estructura con el elemento representativo m_3.

Fuente: *Elaboración propia.*

En cualquier caso, el creado o modificado *micro-cluster* es tomado como un posible candidato para la operación de absorción. Si la condición de absorción se cumple entonces la operación es ejecutada. Esta operación se ejecuta hasta que ya no se cumpla la condición de absorción. Luego, el mismo *micro-cluster* es usando como candidato para la operación de enlazamiento, comprobando por la correspondiente condición como en el caso anterior. Si la condición es cumplida, entonces la operación de enlazamiento es ejecutada una vez. Luego se repite este proceso cada vez que un punto aparece.

3.4. Selección de función de inclusión e hiperparámetros

Como se vio en la sección anterior, dos de las operaciones planteadas para el algoritmo *SetClust* requieren de una función f que llamaremos función de inclusión, al ser esta la función que determina si un elemento pertenece o no a un *micro-cluster* en caso de la operación Agregación de elementos o determinar si dos *micro-clusters* pueden ser enlazado para formar parte del mismo *cluster* en caso de la operación Enlazamiento de *micro-clusters*. Por lo tanto, es importante escoger la función correcta o que más se adapte a nuestros datos para poder obtener mejores resultados. Como vimos, las funciones a escoger pueden ser: la media, la mediana, la distancia de Manhattan (l^1), la distancia Euclídea (l^2), el máximo (l^∞), entre otras, véase la Figura 3.6.

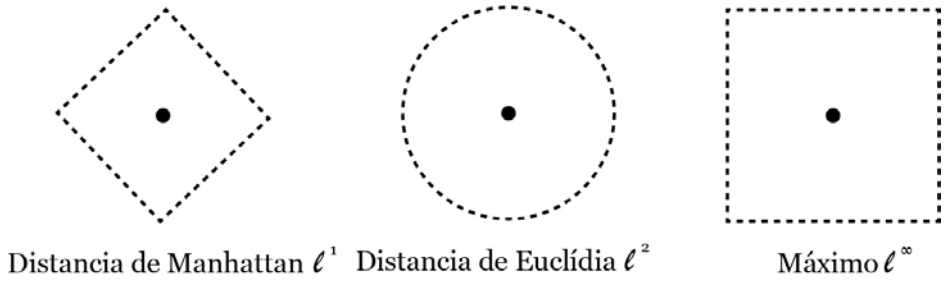

Figura 3.6: Ejemplo de la forma de los de los *micro-clusters* utilizando diferentes tipos de métricas de distancia.

Fuente: *Elaboración propia.*

Teniendo en cuenta que la métrica que usemos determinará la forma de los *micro-clusters*, por ejemplo ver la Figura 3.5 en la que los *micro-clusters* tiene formas pseudo-elípticas, podemos conocer de manera muy superficial las forma y distribución del *dataset* al que vamos a aplicar este algoritmo y suponer que métrica funcionaria mejor para ese *dataset* en particular. Como se explicó anteriormente este algoritmo cambia la necesidad de tener conocimiento del número de *cluster* que el algoritmo debe hallar en el *dataset* a la necesidad de tener un conocimiento superficial sobre la distribución de los datos.

El problema de la elección de la función de inclusión f puede ser solucionado debido a la operación de enlazamiento de *micro-cluster* ya que esto permite que se adopten formas más irregulares, de modo que en vez que los *micro-clusters* se amolden a estas formas, toda la estructura del *cluster* se adaptará a las formas. Esto salva el problema de la elección de la función de inclusión, pero incrementa la posibilidad de tener más *micro-clusters* en la estructura para adaptarse correctamente a la forma, esto influye en el tamaño de la variable r y aumente el costo del tiempo computacional.

La selección de los hiperparámetros σ_{ini} y c tiene una profunda interpretación cuando se observa cómo afecta la variación de estos parámetros durante la creación de la estructura. El hiperparámetro σ_i puede ser interpretado como la primera aproximación a la desviación estándar de un *micro-cluster*, ya que ningún *micro-cluster* creado podrá tener una desviación estándar menor a esta. Por esta razón sería una buena práctica elegir este parámetro observando en grado de separación y dispersión de los datos. El parámetro c puede ser visto de manera similar, como una suposición de cuán separados estarán los *clusters* entre sí, este valor afecta más el crecimiento de los *micro-clusters* mientras el parámetro σ_{ini} solo afecta a su construcción, también puede ser entendido como la libertad que tiene un *cluster* para absorber más datos si es que el valor es alto, o de limitar esa libertad reduciendo este valor. De la misma manera, este hiperparámetro puede ser deducido o escogido por la observación de los datos.

Como podemos observar en la Figura 3.7 manejando correctamente los parámetros se puede conseguir un buen resultado de *clustering*. Además, se puede notar que hay un tipo de relación entre estos dos parámetros, de manera que para que se tenga un buen resultado del *clustering*, si el hiperparámetro σ_{ini} se reduce el hiperparámetro c

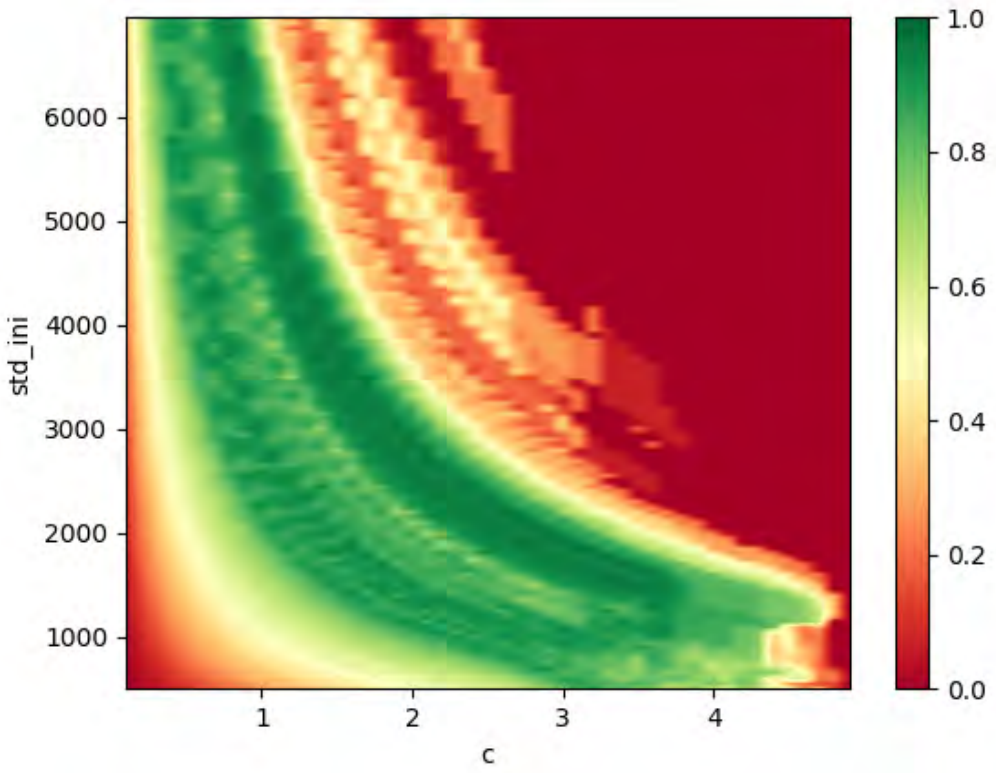

Figura 3.7: Ejemplo de la precisión del *clustering* dependiendo de los hiperparámetros σ_{ini} y c utilizando la distancia Euclídea como función de inclusión f. Teniendo en cuenta que los resultados van de 0 siendo el peor a 1 siendo un agrupamiento perfecto.

Fuente: *Elaboración propia.*

debe crecer y viceversa para mantener una buena precisión del algoritmo.

3.5. Conjuntos de datos

Para observar los resultados del algoritmo propuesto, se realizaron algunas pruebas experimentales. Es necesario utilizar conjuntos de datos clásicos para comprobar su funcionamiento en condiciones estándar, y conjuntos de datos especiales para comprobar su funcionamiento en condiciones extremas como que presenten múltiples *outliers*, formas de *clusters* irregulares, *clusters* desbalanceados, o *clusters* solapados. Para ellos se utilizarán conjuntos de datos artificiales y reales, los datos artificiales se dividirán en dos tipos, los datos artificiales existentes usado para evaluar algoritmos de aprendizaje automático, y conjuntos de datos artificiales creados para este problema en específico acentuando las características que se deben probar para este algoritmo.

3.5.1. Conjunto de datos artificiales existentes

Se utilizaron 5 *datasets* comúnmente empleados en el área de *machine learning* con fines experimentales o didácticos, estos fueron elegidos por ser *datasets* estándar para los experimentos y por tener características referentes al estudio. Los *dataset* a utilizar fueron obtenidos de los repositorios del departamento de computación de la Universidad de Joensuu (Kärkkäinen & Fränti, 2002) y la Escuela de computación de la Universidad del Este de Finlandia (Fränti & Sieranoja, 2018).

- **Dataset artificial 1:** El primer *dataset* consta de *clusters* ligeramente solapados con un *cluster* con una forma ligeramente aplanada.
- **Dataset artificial 2:** Este *dataset* contiene *cluster* separados correctamente en un espacio dimensional muy alto, se utilizó este *dataset* para comprobar la correcta funcionalidad del algoritmo en altas dimensiones.

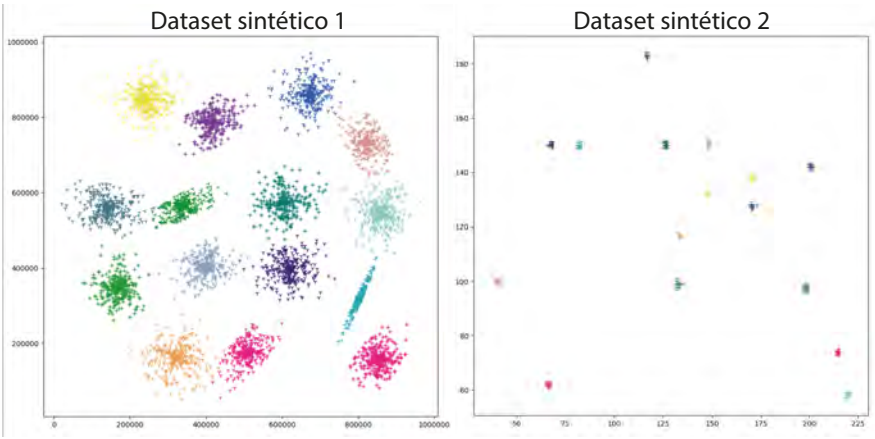

Figura 3.8: Representación de los *datasets* artificiales 1 y 2, el *dataset* artificial dos solo está representado en 2 dimensiones.

Fuente: *Elaboración propia.*

- **Dataset artificial 3:** Este *dataset* es la primera parte de un conjunto de 3 *datasets* incluidos uno dentro de otro con el fin de ver la evolución de los datos en el tiempo, los *clusters* tienen formas regulares y están solapados ligeramente, posee gran cantidad de *clusters*.
- **Dataset artificial 4:** Este es la segunda parte del conjunto de 3 *datasets*, este contiene al *dataset* artificial 3.
- **Dataset artificial 5:** Este es la tercera y última parte del conjunto de 3 *datasets*, este contiene al *Dataset* artificial 4. Utilizando este grupo de 3 *datasets* artificiales podremos comprobar el funcionamiento del algoritmo planteado en la evolución de los datos durante el tiempo incrementando la cantidad de *clusters*, manteniendo los mismos parámetros.

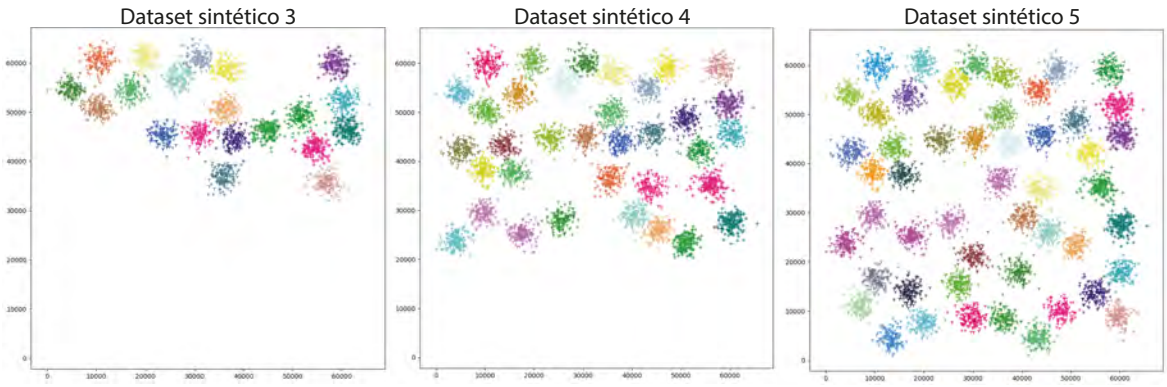

Figura 3.9: Representación de los *Datasets* artificiales 3, 4 y 5, se puede observar cómo cada *Dataset* está incluido dentro del siguiente.

Fuente: *Elaboración propia.*

Tabla 3.1: Información sobre los *datasets* artificiales existentes.

Dataset	Data 1	Data 2	Data 3	Data 4	Data 5
Dimensión	2	1024	2	2	2
Clusters	15	16	20	35	50
Puntos	5000	1024	3000	5250	7500

Fuente: *Elaboración propia.*

3.5.2. Conjunto de datos artificiales propuestos

Se generaron 4 *datasets* en los que se realzaron las características requeridas para este algoritmo tales como presencia de *outliers* y formas irregulares de los *clusters*.

- **Dataset artificial 6:** Este *dataset* posee 4 *clusters* en que están desbalanceados, cada *cluster* tiene más puntos que el siguiente pero no poseen la misma densidad de puntos debido a que cada vez los puntos están más dispersos. Los *clusters* son generados utilizando una distribución de probabilidad Gaussiana multivariable.

  La distribución Gaussiana Multivariable o Normal Multivariable es la distribución conjunta más utilizada para variables continuas. La función de densidad de probabilidad de la distribución para una dimensión D está definida por la siguiente expresión:

$$\mathcal{N}(x|\mu,\Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right) \tag{3.7}$$

  donde $\mu = \mathbb{E}[x] \in \mathbb{R}^D$ es el vector media, y $\Sigma = cov[x]$ es la matriz de covarianza $D \times D$. La constante de normalización $(2\pi)^{D/2}|\Sigma|^{1/2}$ se asegura que la función de densidad se integre a 1.

- **Dataset artificial 7:** Este *dataset* posee seis *clusters* y una gran cantidad de datos. Estos *clusters* tiene la particularidad de no tener formas regulares, los

cuales están formados por medias circunferencias $(x - x_o)^2 + (y - y_o)^2 = r^2$ que están dispuestas de tal forma que dificultan el *clustering* por algoritmos basados en centroides. Una vez creada la forma de los *clusters*, se crean los puntos que pertenecen a estos utilizando una distribución Gaussiana Multivariable.

- **Dataset artificial 8:** Este *dataset* posee 2 *clusters* los cuales poseen una gran cantidad de *outliers*, cada *cluster* esta creado con una distribución de probabilidad Gaussiana multivariable, cada *cluster* esta generado con dos matrices de covarianza diferentes, la primera genera los datos normales que pertenecerían a un *cluster* mientras la segunda genera los datos que estarían fuera del comportamiento normal.

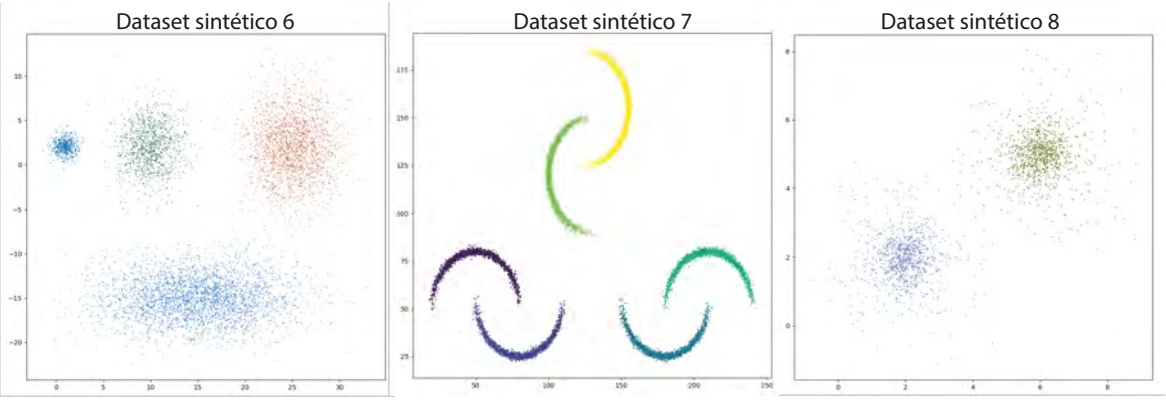

Figura 3.10: Representación de los Datasets artificiales 6, 7 y 8.

Fuente: *Elaboración propia.*

- **Dataset artificial 9:** Este *dataset* posee 4 *clusters* que poseen formas irregulares para un *cluster* clásico. Este *dataset* está en 3 dimensiones, estos *cluster* forman una doble hélice, cada *cluster* responde a una ecuación paramétrica que da la forma, y a partir de esta función se utiliza una distribución de probabilidad Gaussiana, como vimos en el *Dataset* artificial 6, para generar los puntos. El primer *cluster* responde a la siguiente ecuación paramétrica $f_1(t) = (t, sin(t), cos(t))$, $f_2(t) = (t, sin(t + \pi), cos(t + \pi))$, $f_3(t) = (t, cos(t), sin(t))$ y $f_4(t) = (t, cos(t + \pi), sin(t + \pi))$ con $t \in [0, 4]$.

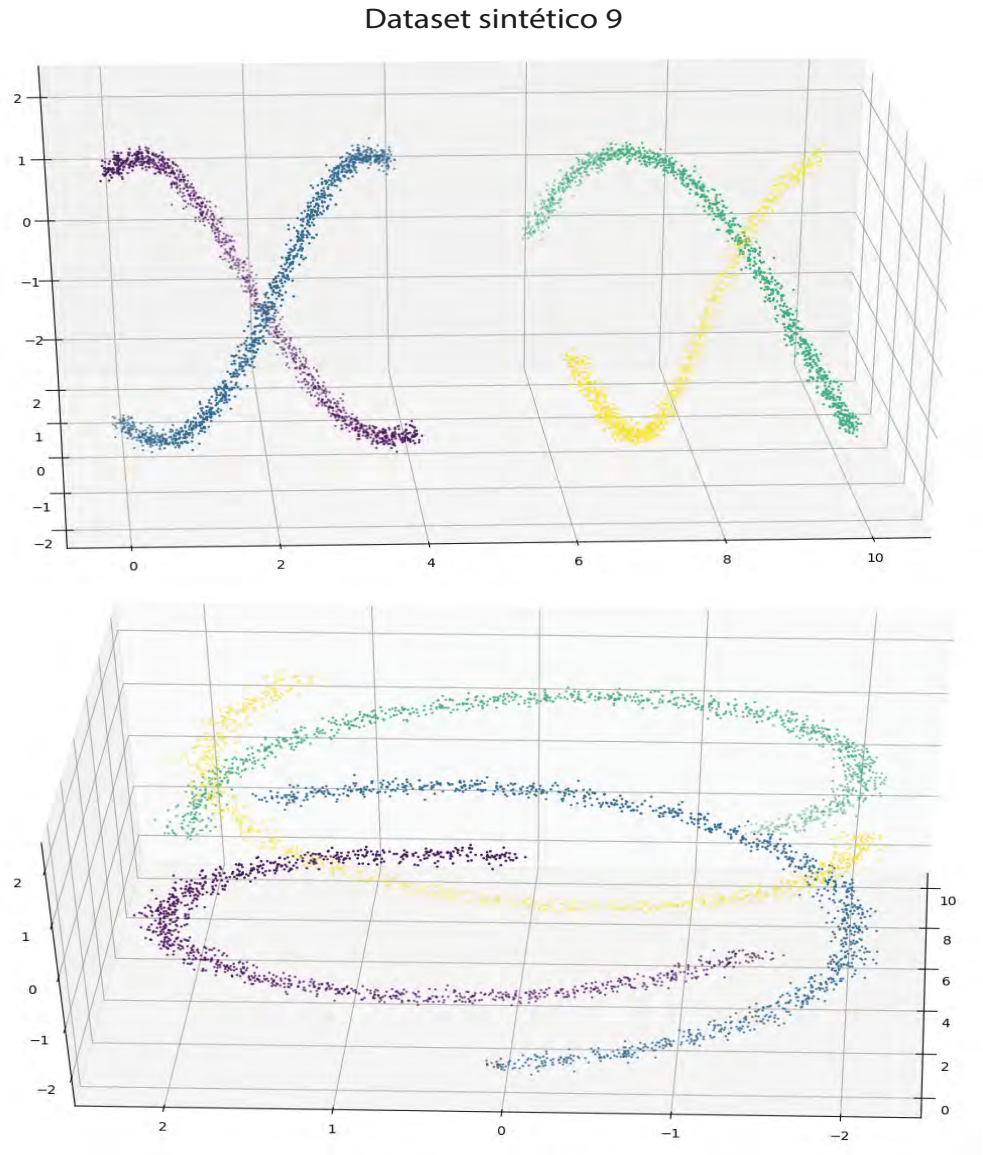

Figura 3.11: Representación del Dataset artificial 9.

Fuente: *Elaboración propia.*

3.6. Experimentos

Con el fin de contrastar la información obtenida de los experimentos del algoritmo propuesto, se utilizó 3 algoritmo de *Data Streaming Clustering*: *DenStream* este algoritmo requiere de parámetros (Cao et al., 2006), *ClusTree* este algoritmo no requiere de parámetros (Kranen et al., 2009) y *CluStream* este algoritmo no requiere de parámetros (Aggarwal et al., 2003). Cada uno de estos algoritmos devolverá el conjunto de etiquetas de los *clusters* de la siguiente manera. Sea n el número de puntos a etiquetar, la salida de los algoritmos será una secuencia de n elementos $a_1, a_2, ..., a_n$ siendo a_i un número entero que va entre 0 hasta $l-1$ representando la etiqueta que el algoritmo asigno a ese elemento. El número l no necesariamente representa el número de *clusters* reales en cada conjunto de datos, debido a que los algoritmos no tienen la

Tabla 3.2: Información de los dataset artificiales planteados para la evaluación del algoritmo.

Data 6	Espacio	Puntos por cluster	μ	Σ
	$\mathbb{R}^2$	Cluster 1 500 puntos	(1,2)	$\begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$
		Cluster 2 1000 puntos	(10,2)	$\begin{bmatrix} 3 & 0.2 \\ 0.2 & 6 \end{bmatrix}$
		Cluster 3 2000 puntos	(25,2)	$\begin{bmatrix} 6 & 0.2 \\ 0.2 & 12 \end{bmatrix}$
		Cluster 4 3000 puntos	(15,-15)	$\begin{bmatrix} 25 & 0.2 \\ 0.2 & 5 \end{bmatrix}$
Data 7	Espacio	Puntos por cluster	μ	Σ
	$\mathbb{R}^2$	Cluster 1 1600 puntos	-	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
		Cluster 2 1600 puntos	-	
		Cluster 3 1600 puntos	-	
		Cluster 4 1600 puntos	-	
		Cluster 5 1600 puntos	-	
		Cluster 6 1600 puntos	-	
Data 8	Espacio	Puntos por cluster	μ	Σ
	$\mathbb{R}^2$	Cluster 1 1000 puntos	(2,2)	$\begin{bmatrix} 0.2 & 0 \\ 0 & 0.2 \end{bmatrix}$ $\begin{bmatrix} 1.2 & 0 \\ 0 & 1.2 \end{bmatrix}$
		Cluster 2 1000 puntos	(6,6)	
Data 9	Espacio	Puntos por Cluster	μ	Σ
	$\mathbb{R}^3$	Cluster 1 1120 puntos	-	$\begin{bmatrix} 0.008 & 0 & 0 \\ 0 & 0.008 & 0 \\ 0 & 0 & 0.008 \end{bmatrix}$
		Cluster 2 1120 puntos	-	
		Cluster 3 1120 puntos	-	
		Cluster 4 1120 puntos	-	

Fuente: *Elaboración propia.*

información de cuántos *cluster* van a ser formados, por lo tanto, crean tantos *clusters* como determina que sean necesarios, además este número no será necesariamente el mismo en el resultado de los algoritmos. Una vez obtenida la salida del algoritmo se comparará con la secuencia de n elementos $b_1, b_2, ..., b_n$ siendo b_i un número entero que va desde 0 hasta $k-1$, k es el número real de *clusters*. Estas dos secuencias son evaluadas utilizando la métrica de evaluación *V-measure-score* la cual devuelve valores entre 0 y 1, 0 siendo el peor resultado y 1 siendo un etiquetado perfecto.

El algoritmo *SetClust* propuesto fue implementado utilizando el lenguaje de programación *Python* (versión 3.7) y la librería de computación científica *Numpy* (version 1.17) sobre un procesador Intel Core i7-9750H 2.60 GHz. Utilizando una memoria RAM de 16 GB y Windows 10. Los algoritmo con los que se realizó la comparación fueron utilizados mediante la siguiente plataforma para *Machine Learning* para flujos de datos *Massive Online Analysis MOA* (Bifet et al., 2010). Los archivos utilizados se encuentran en formato (.csv) y formato (.arff), el primero formato se usó para la lectura de los datos para el algoritmo *SetClust*, la segunda para los algoritmos de prueba. El código fuente del algoritmo *SetClust* propuesto está en la siguiente dirección (Campos

& León, 2019).

Teniendo nueve conjuntos de datos artificial, la primera tarea realizada fue ejecutar los algoritmos: *SetClust*, *DenStream*, *Clustream* y *ClusTree* sobre todos los conjuntos de datos. Se realizo 50 ejecuciones de cada algoritmo sobre cada conjunto de datos. Los resultados de estos experimentos se muestran en la Tabla 3.3, los resultados comprenderán el máximo, la mediana y la media de los resultados. Los mejores resultados de cada conjunto de datos se encuentran resaltados.

Tabla 3.3: Resultados de los experimentos sobre los conjuntos de datos utilizados.

Datasets					
Dataset		ClusTree	CluStream	DenStream	SetClust
Dataset S. 1	Máximo	0.810	0.810	0.800	**0.983**
	Mediana	0.800	0.800	0.740	**0.958**
	Media	0.800	0.800	0.739	**0.952**
Dataset S. 2	Máximo	**1.000**	**1.000**	**1.000**	**1.000**
	Mediana	**1.000**	**1.000**	**1.000**	**1.000**
	Media	**1.000**	**1.000**	**1.000**	**1.000**
Dataset S. 3	Máximo	0.810	0.820	0.750	**0.935**
	Mediana	0.81	0.810	0.700	**0.900**
	Media	0.807	0.812	0.702	**0.898**
Dataset S. 4	Máximo	0.820	0.840	0.810	**0.941**
	Mediana	0.820	0.830	0.790	**0.913**
	Media	0.819	0.830	0.785	**0.911**
Dataset S. 5	Máximo	0.840	0.850	0.840	**0.952**
	Mediana	0.830	0.840	0.790	**0.926**
	Media	0.830	0.839	0.793	**0.925**
Dataset S. 6	Máximo	0.720	0.720	0.880	**0.993**
	Mediana	0.680	0.510	0.800	**0.720**
	Media	0.682	0.559	0.741	**0.889**
Dataset S. 7	Máximo	0.710	0.730	0.950	**1.000**
	Mediana	0.700	0.710	0.920	**0.997**
	Media	0.696	0.715	0.903	**0.957**
Dataset S. 8	Máximo	0.560	0.500	0.950	**0.971**
	Mediana	0.500	0.500	0.875	**0.967**
	Media	0.505	0.500	0.873	**0.928**
Dataset S. 9	Máximo	0.370	0.370	0.850	**0.941**
	Mediana	0.280	0.270	0.700	**0.843**
	Media	0.287	0.297	0.701	**0.836**

Fuente: *Elaboración propia.*

Los análisis estadísticos fueron realizados de acuerdo a (Demšar, 2006), ver también ((García & Herrera, 2008) y (Derrac et al., 2011)), sobre los resultados de los algoritmos, utilizando una muestra de 50 elementos, como se muestra a continuación.

1. El test de Friedman fue realizado para tener una conclusión sobre la hipótesis nula (aplicada a los algoritmos elegidos).

   $$H_0 : \textit{Todos los algoritmos tienen el mismo desempeño.}$$

   Rechazar esta hipótesis implica que existe por lo menos un algoritmo que es diferente.

2. Una vez rechazada la hipótesis nula según el test de Friedman, entonces el test de Wilcoxon es realizado para obtener una conclusión con respecto a la nueva hipótesis nula (aplicada a los algoritmos elegidos), el error acumulado por este test es corregido según el método de Holm.

H_0 : *Un algoritmo de control tiene el mismo desempeño con respecto a cualquier otro algoritmo.*

Rechazar esta hipótesis implica que el algoritmo de control es superior a los demás.

En ambas pruebas se utilizó un nivel de significancia de $\alpha = 0.05$ por defecto. Dados los resultados en el *Dataset* artificial 2, este conjunto de datos no se utilizó en el análisis estadístico.

Tabla 3.4: Resultados del test de Friedman y los resultados del test de wilcoxon corregidos según el método de holm.

Dataset	Control	Algoritmo	Rank	p-value	p-value corregido
Dataset S. 1	**SetClust (Friedman: 1.80E-31)**	**DenStream**	1	7.56E-10	2.27E-09
		ClusTree	2	7.56E-10	2.27E-09
		CluStream	3	7.56E-10	2.27E-09
Dataset S. 3	**SetClust (Friedman: 1.81E-31)**	**DenStream**	1	7.56E-10	2.27E-09
		ClusTree	2	7.56e-10	2.27E-09
		CluStream	3	7.56E-10	2.27E-09
Dataset S. 4	**SetClust (Friedman: 4.75E-32)**	**DenStream**	1	7.56E-10	2.27E-09
		ClusTree	2	7.56E-10	2.27E-09
		CluStream	3	7.56E-10	2.27E-09
Dataset S. 5	**SetClust (Friedman: 2.58E-30)**	**DenStream**	1	7.56E-10	2.27E-09
		ClusTree	2	7.56E-10	2.27E-09
		CluStream	3	7.56E-10	2.27E-09
Dataset S. 6	**SetClust (Friedman: 1.60E-18)**	**DenStream**	1	6.48E-07	1.94E-06
		ClusTree	2	1.13E-08	1.94E-06
		CluStream	3	1.56E-09	1.94E-06
Dataset S. 7	**SetClust (Friedman: 9.82E-27)**	**DenStream**	1	1.97E-07	5.90E-07
		ClusTree	2	1.57E-07	5.90E-07
		CluStream	3	5.65E-08	5.90E-07
Dataset S. 8	**SetClust (Friedman: 3.10E-30)**	**DenStream**	1	3.32E-06	9.96E-06
		ClusTree	2	7.00E-10	9.96E-06
		CluStream	3	6.87E-10	9.96E-06
Dataset S. 9	**SetClust (Friedman: 9.57E-29)**	**DenStream**	1	1.38E-09	4.15E-09
		ClusTree	2	7.56E-10	4.15E-09
		CluStream	3	7.56E-10	4.15E-09

Fuente: *Elaboración propia.*

Capítulo 4

Discusión

4.1. Discusión de los resultados

Como se indicó anteriormente en la ejecución de los experimentos, para cada *dataset* se ejecutó 50 pruebas por algoritmo, esto resulta en un total de 200 pruebas por conjunto de datos. Los resultados de estos experimentos fueron resumidos de tal manera que se puedan observar los valores máximos (la mejor precisión de los resultados), los valores medios y los valores de las medianas. Esta información puede ser consultada en la Tabla 3.3, sobre estos resultados podemos observar lo siguiente.

Podemos observar que los resultados de los test de Friedman para todos los *datasets*, menos el *dataset* 2, indican valores menores a nuestro valor $\alpha = 0.05$, esto indica que se rechaza para todos los conjuntos de datos la suposición inicial, que indica que todos los algoritmos tienen el mismo desempeño. Además de esto, los resultados del test de Wilcoxon utilizando *SetClust* como algoritmo de control, indican que valores nuevamente menores al valor α, esto muestra que se rechaza la segunda suposición, que indica que un algoritmo de control tiene el mismo desempeño con respecto a cualquier otro algoritmo. Véase la Tabla 3.4.

En los experimentos de *dataset* sintético 1, los mejores resultados son obtenidos por el algoritmo propuesto *SetClust*, que requiere de solo dos parámetros que son fácilmente ajustados para cada *dataset*, podemos intuir que esto es debido a la forma de algunos *clusters* y la presencia de *outliers*. Los algoritmos *CluStream* y *Clustree* son algoritmos que no requieren de parámetros, tienen un desempeño parecido ya que estos tratan de encontrar el número de *clusters* y a partir de esto crear zonas circulares de pertenencia lo cual no es apropiado para la forma de estos *clusters* y su dispersión. Mientras el algoritmo *DenStream* es un algoritmo basado en densidades que requiere de una gran cantidad de parámetros lo que dificulta encontrar un conjunto de parámetros que funcione de manera eficiente.

En los resultados de los experimentos del *dataset* sintético 2 podemos observar que todos los algoritmos tienen una precisión perfecta, esto es debido a que los *clusters*

están claramente separados en un alto espacio dimensional, no poseen *outliers* y tiene una forma regular ,ver Figura 3.8. Este *dataset* fue escogido para comprobar el desempeño del algoritmo propuesto en altas dimensiones, podemos observar que es claramente capaz.

En los resultados de los *datasets* sintéticos 3, 4 y 5 observamos un patrón en el comportamiento de los algoritmos, mientras mayor la cantidad de datos, mejores son sus resultados. Los algoritmos *CluStream* y *ClusTree* tiene un desempeño estándar, tienen la ventaja de al no necesitar parámetros, esto vuelve su ejecución más rápida al no tener que realizar la búsqueda de parámetros, sin embargo, no les permite poder ajustarse correctamente a los datos. El algoritmo *DenStream* no tiene los resultados esperados debido a la cercanía de los *clusters* y sus *outliers*, esto hace interpretar al algoritmo que todos son parte de un solo *clusters* más grande. Los mejores resultados son obtenidos por el algoritmo propuesto *SetClust* debido que con los parámetros adecuado puede ignorar a los *outliers* que podrían afectar al resultado.

Podemos observar que en los resultados del *dataset* sintético 6, los algoritmos *CluStream* y *ClusTree* siguen teniendo un comportamiento aceptable, aunque la presencia de *outliers* y el desequilibrio del tamaño y dispersión de los *clusters* reduce su desempeño. El algoritmo *DenStream* muestra un mejor desempeño que los otros dos algoritmos de control al poder ajustarse mejor a estas variaciones. El algoritmo propuesto *Setclust* sigue teniendo el mejor desempeño ya que la unión de *micro-clusters* le permite crear regiones de diferentes tamaños y adaptar las formas y las dispersiones deseadas.

El *dataset* sintético 7 es un *dataset* que representa un gran desafío para algoritmos basados en centroides o que requieran de *clusters* con formas regulares, este *dataset* posee mayor densidad de puntos en la zona "central" del *cluster*. Observamos que los algoritmos *CluStream* y *ClusTree* tiene un desempeño medio debido a la concentración de puntos cerca de la posición que estos reconocen como centro de los *cluster*, sin embargo, el resto de puntos son no son claramente definidos, ver Figura 4.1. Por otra parte, el algoritmo *DenStream* reconoce los sectores de alta concentración de puntos las cuales define como el *micro-clusters* luego une estas áreas de gran densidad como un solo *cluster*, por estas razones este algoritmo muestra una clara mejora sobre los otros algoritmos de comparación. El algoritmo propuesto *SetClust* nuevamente obtiene el mejor desempeño, incluso llegando a tener un máximo desempeño perfecto, la forma de estos *cluster* puede ser ajustada gracias a la unión de *micro-clusters* y sus formas. Los resultados del *dataset* sintético 8 vemos que los algoritmos *CluStream* y *CusTree* tiene un bajo desempeño, esto es debido a que a pesar que los algoritmos encuentran claramente el centro de los dos *clusters* la cantidad de puntos que están fueran de esta zona (*outliers*) es considerable, y al no poder controlar el tamaño de esta zona muchos puntos quedan si ser clasificados. En cambio, modificando los parámetros del algoritmo *DenStream* podemos considerar estas zonas de baja densidad de puntos como admitidos para ser parte del *cluster* principal. El algoritmo *SetClust* posee los mejores resultados al poder considerar estos puntos alejados como *micro-clusters* y al estar cerca de otros *micro-clusters* más cercanos al centro pueden enlazarse y formar un solo *cluster*.

Para concluir observamos los resultados del *dataset* sintético 9, este *dataset*, de

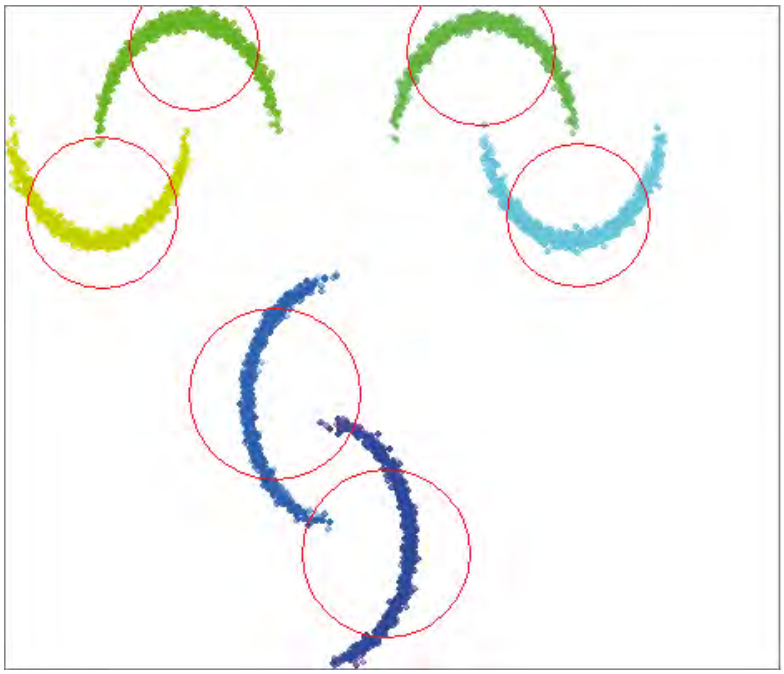

Figura 4.1: Experimento del dataset sintético 7 con el algoritmo *CluStream*

Fuente: *Elaboración propia.*

igual forma que el *dataset* sintético 7, resulta un desafío para los algoritmos basados en *centroides* o en centros, ya que su forma alargada y de densidad constante a lo largo del *cluster* (ver Figura 3.10 y Figura 3.11) dificulta encontrar un centro el cual considerar *cluster*. Los algoritmos *CluStream* y *ClusTree* muestran uno desempeño bajo debido a la conformación del *cluster*. En cambio, en algoritmo *DenStream* nuevamente obtiene un buen desempeño al poder reconocer las zonas de alta densidad y poder unirlas bajo un mismo *cluster*. De igual manera al algoritmo propuesto *Setclust* posee un buen desempeño al poder reconocer *micro-clusters* en cada sección del *cluster* y poder unirlos debido a su cercanía, sin embargo, la forma complicada de este *cluster* y su alta densidad no le permite conseguir un resultado perfecto.

4.2. Discusión del algoritmo

Si bien la idea principal del algoritmo propuesto se basó en el uso del algoritmo de clustering *DBSCAN*, podemos observar que se hizo un redefinición del funcionamiento del mismo, ya que los parámetros utilizados en el algoritmo *DBSCAN* son estáticos lo que indica que no se adaptan a la densidad de los clusters mientras estos se van formando, además, este algoritmo se ejecuta sobre los datos en bruto lo que indica que la complejidad computacional de este algoritmo está basado en n que como se vio anteriormente puede ser muy grande.

Por lo tanto, al redefinir las reglas de funcionamiento del algoritmo de *clustering* *DBSCAN* se logra proponer un nuevo algoritmo que mejorar las deficiencias que tiene

a la hora de manejar gran cantidad de datos, ya que de esta manera no se depende de n, siendo este el número total de puntos en las estructuras, en cambio ahora se define la complejidad del algoritmo en función a la variable r, siendo el número total de *micro-clusters*, que es un valor que depende de la cantidad de clusters existentes y a las complejidad de la forma de estos. La relación entre n y r es una relación sublineal o incluso constante ya que r solo depende de la complejidad del dataset y no de la cantidad de datos a procesar.

Sabiendo que los *micro-clusters* tiene formas pseudo elípticas, podemos intuir que mientras más simples sean las formas de los *cluster* a los que se vaya a aplicar este algoritmo menos *micro-cluster* formara este, debido a que con pocos *micro-cluster* se puede amoldar a las formas de los *clusters*. Pero mientras más complejas sean estas formas requerirá más *micro-clusters* para adaptarse a su forma, lo que tiene una implicación directa en el tiempo computacional.

Conclusiones

1. Se logró proponer un algoritmo para el problema *Data Streaming Clustering* para conjuntos amorfos y con *outliers* que posee una alta precisión y versatilidad, además de tener un costo computacional aceptable para este problema, $O(r^2 d)$, ya que se pudo intercambiar la variable n número de puntos procesados hasta el momento a la variable r que es el número de *micro-cluster* en la estructura. Esto fue logrado utilizando la idea básica del algoritmo *DBSCAN* y la unión de este con las estructuras de resumen de información y la estructura que mantiene toda la información del *cluster, Union Find and Disjoint sets*.

   Se comprobó estadísticamente que el algoritmo propuesto *SetClust* alcanza una precisión superior a los algoritmos con los que fue comparado *CluStream*, *ClusTree* y *DenStream* en un conjunto de *datasets* que poseen todas las características requeridas para este problema, con un nivel de significancia del 5 %, valor estándar para la comunidad científica.

2. Gracias a la correcta exploración e investigación sobre las técnicas utilizada en el problema *Data Streaming Clustering* se pudo dar con el algoritmo *DBSCAN* el cual forma parte fundamental el algoritmo propuesto *SetClust*. Además, se pudo elegir el uso de la estructura de resumen de datos *feature vector*.

3. Una vez establecida y adaptada la estructura para el resumen de datos *feature vector*, es creada una macro estructura de resumen y enlazamiento de la información utilizando la estructura de enlazamiento de información *Union Find and Disjoint sets*. Podemos observar mediante los experimentos que el desempeño de esta estructura es correcto pudiendo almacenar la información requerida y permitiendo realizar consultas con un tiempo computacional casi constante.

4. Teniendo como base el algoritmo *DBSCAN* se pudo desarrollar un nuevo algoritmo pudiendo conservar su capacidad de generar *cluster* con formas irregulares y mejorando su tiempo computacional al funcionar ahora con *micro-clusters*, esto no solo cambio el receptor (puntos) de la acción del algoritmo, sino también, sus parámetros y sus condiciones de pertenencia.

5. Conociendo el problema *Data Streaming Clustering* se pudo proponer un conjunto de *datasets* sintéticos que cumplían con los requerimientos para justificar las propiedades del algoritmo. Por lo tanto, fueron planteados conjuntos de datos con alta presencia de *outliers* y solapamiento de *clusters*, conjuntos de datos con *clusters* desbalanceados y *outliers* y por último *clusters* con formas irregulares con densidades de puntos variables. Estos *dataset* fueron correctamente

probados con los algoritmos de comparación *CluStream*, *Clustree*, *DenStream* y el algoritmo propuesto *SetClust*.

6. Una vez realizando los experimentos requeridos, podemos determinar que el algoritmo planteado *SetClust* cumple con las características definidas en su proposición. Aún más, demuestra tener un mejor desempeño que los algoritmos con los que fue comparado en todos los *dataset* utilizados, *dataset* sintéticos utilizados en el ámbito científico para hacer estudios y los *dataset* sintéticos propuesto que resaltando las características necesarias para demostrar su capacidad.

Trabajos futuros

1. En muchas situaciones los conjuntos de datos pueden tener como inconveniente el ruido, esto no fue tomado en cuenta para la construcción del algoritmo. Si bien la presencia de ruido no afectaría demasiado la precisión del *clustering*, debido a que usaría ese ruido como posibles *clusters* en formación, si afectaría al costo computacional aumentando de gran manera la variable r que es el número de *micro-clusters*. En trabajos futuros se planea implementar el descarte de *clusters* y *micro-clusters* según una función de envejecimiento.

2. Otra característica que no se tomó en cuenta sobre este problema son los conjuntos de datos en evolución. Esto implica que un *cluster* no solo se forma y tiene una posición constante en el espacio, sino que, mientras avanza el tiempo los *clusters* cambian de posición forma o distribución. Esto puede ser solucionado implementando un envejecimiento de *clusters*, logrando implementar una eliminación de una relación en la estructura *Union Find*.

3. Estudiar mas a profundidad los hiperpárametros utilizados en el algoritmo σ_{ini}, c y f. Conocer el comportamiento de estos podrá a encontrar los mejores hiperpárametros para un dataset. Además, determinar la relación que existe entre c y σ_{ini}.

4. Utilizar técnicas más avanzadas para la generación de *datasets* más complejos con el fin de evaluar los límites del algoritmo propuesto.

5. Realizar modificaciones al algoritmo para utilizar otras estructuras de resumen de información como *prototype arrays* o *coreset trees*.

Bibliografía

Ackermann, M. R., Märtens, M., Raupach, C., Swierkot, K., Lammersen, C. & Sohler, C. (2012). StreamKM++: A clustering algorithm for data streams. *Journal of Experimental Algorithmics (JEA)*, *17*, Article No. 2.4. https://doi.org/10.1145/2133803.2184450

Ackermann, W. (1928). Zum Hilbertschen Aufbau der reellen Zahlen. *Mathematische Annalen*, *99*(1), 118-133. https://doi.org/10.1007/BF01459088

Aggarwal, C. C., Han, J., Wang, J. & Yu, P. S. (2003). A Framework for Clustering Evolving Data Streams. En *Proceedings of the 29th International Conference on Very Large Data Bases*.

Andrew, R. & Hirschberg, J. (2007). V-Measure: A Conditional Entropy-Based External Cluster Evaluation Measure. En *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*.

Bifet, A., Holmes, G., Kirkby, R. & Pfahringer, B. (2010). MOA: Massive Online Analysis. *J. Mach. Learn. Res.*, *11*, 1601-1604. http://portal.acm.org/citation.cfm?id=1859903

Campos, I. & León, J. (2019). Setclust. https://doi.org/10.5281/zenodo.3235837

Campos, I., León, J. & Campos, F. (2020). An Efficient Set-Based Algorithm for Variable Streaming Clustering (J. A. Lossio-Ventura, N. Condori-Fernandez & J. C. Valverde-Rebaza, Eds.). En J. A. Lossio-Ventura, N. Condori-Fernandez & J. C. Valverde-Rebaza (Eds.), *Information Management and Big Data*. https://doi.org/https://doi.org/10.1007/978-3-030-46140-9_9

Cao, F., Estert, M., Qian, W. & Zhou, A. (2006). Density-Based Clustering over an Evolving Data Stream with Noise. *2006 SIAM International Conference on Data Mining*, 328-339. https://doi.org/{10.1137/1.9781611972764.29}

Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2009). *Introduction to Algorithms, 3rd Edition*. MIT Press. https://dl.acm.org/doi/book/10.5555/1614191

Demšar, J. (2006). Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research*, *7*, 1-30.

Derrac, J., García, S., Molina, D. & Herreraa, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, *1*, 3-18.

Dunford, N. & Schwartz, J. (1971). *Linear Operators: Spectral operators*. Wiley-Interscience. https://books.google.com.pe/books?id=PGNHtAEACAAJ

Ester, Kriegel, H.-P., Sander, J. & Xu, X. (1996). A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. *KDD'96 Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, *3*, 226-231.

Estivill-Castro, V. (2002). Why so many clustering algorithms. *ACM SIGKDD Explorations Newsletter*, *4*, 65-75. https://doi.org/10.1145/568574.568575

Fränti, P. & Sieranoja, S. (2018). K-means properties on six clustering benchmark datasets. http://cs.uef.fi/sipu/datasets/

Galil, Z. & Italiano, G. F. (1991). Data Structures and Algorithms for Disjoint Set Union Problems. *ACM Comput. Surv.*, *23*(3), 319-344. http://doi.acm.org/10.1145/116873.116878

Galler, B. A. & Fisher, M. J. (1964). An Improved Equivalence Algorithm. *Commun. ACM*, *7*(5), 301-303. https://doi.org/10.1145/364099.364331

Gama, J. (2010). *Knowledge Discovery from Data Streams*. Chapman & Hall.

García, S. & Herrera, F. (2008). An Extension on "Statistical Comparisons of Classifiers over Multiple Data Sets"for all Pairwise Comparisons. *Journal of Machine Learning Research*, *9*, 2677-2694.

Guha, S., Meyerson, A., Mishra, N., Motwani, R. & O'Callaghan, L. (2003). Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, *15*, Issue 3.

Guha, S., Mishra, N., Motwani, R. & O'Callaghan, L. (2000). Clustering data streams. *Foundations of Computer Science, Annual IEEE Symposium*, 359-366.

Hernández Sampieri, R., Fernandez Collado, C. & Baptista Lucio, P. (2014). *Metodología de la Investigación*. Mc Graw Hill.

Jiang, Y., Bi, A., Xia, K., Xue, J. & Qian, P. (2020). Exemplar-based data stream clustering toward Internet of Things. *The Journal of Supercomputing*, *76*, 2929-2957. https://doi.org/10.1007/s11227-019-03080-5

Kärkkäinen, I. & Fränti, P. (2002). *Dynamic local search algorithm for the clustering problem* (inf. téc. A-2002-6). Department of Computer Science, University of Joensuu. Joensuu, Finland.

Kranen, P., Assent, I., Baldauf, C. & Seidl, T. (2009). Self-Adaptive Anytime Stream Clustering. En *Ninth IEEE International Conference on Data Mining (ICDM '09)*. https://doi.org/10.1109/ICDM.2009.47

Kriegel, H.-P., Kröger, P., Sander, J. & Zimek, A. (2011). Density-based clustering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *1*. https://doi.org/https://doi.org/10.1002/widm.30

León, J., Chullo, B., Enciso, L. & Soncco, J. L. (2020). A multiobjective optimization algorithm for center-based clustering [The proceedings of CLEI 2019, the XLV Latin American Computing Conference]. *Electronic Notes in Theoretical Computer Science*.

Lloyd, S. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory*, *28*, 129-137. https://doi.org/10.1109/TIT.1982.1056489

Miller, Z., Dickinson, B., Deitrick, W., Hu, W. & Wang, A. H. (2014). Twitter spammer detection using data stream clustering. *Information Sciences*, *260*, 64-73. https://doi.org/https://doi.org/10.1016/j.ins.2013.11.016

Russell, Norvig, S. J. & Peter. (2004). *Inteligencia artificial :un enfoque moderno*. México Pearson Prentice Hall200xviii.

Shah, R., Krishnaswamy, S. & Gaber, M. (2005). Resource-aware very fast K-Means for ubiquitous data stream mining. *Proceedings of the Second International Workshop on Knowledge Discovery in Data Streams*.

Silva, J. A., Faria, E. R., Barros, R. C., Hruschka, E. R., de Carvalho, A. C. P. L. F. & Gama, J. (2013). Data stream clustering: A survey. *ACM Computing Surveys (CSUR)*, *46*, Article No. 13. https://doi.org/10.1145/2522968.2522981

Tarjan, R. E. & van Leeuwen, J. (1984). Worst-Case Analysis of Set Union Algorithms. *J. ACM*, *31*(2), 245-281. https://doi.org/10.1145/62.2160

Zhang, T., Ramakrishnan, R. & Livny, M. (1996). BIRCH: an efficient data clustering method for very large databases. *SIGMOD '96 Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, *25*, 103-114. https://doi.org/10.1145/235968.233324

Apéndice A:

Publicación

Campos I., León J., Campos F. (2020) An Efficient Set-Based Algorithm for Variable Streaming Clustering. In: Lossio-Ventura J., Condori-Fernandez N., Valverde-Rebaza J. (eds) Information Management and Big Data. SIMBig 2019. Communications in Computer and Information Science, vol 1070. Springer, Cham.

An Efficient Set-Based Algorithm for Variable Streaming Clustering

Isaac Campos(✉), Jared León, and Fernando Campos

Department of Informatics, Universidad Nacional de San Antonio Abad del Cusco, Cusco, Peru
{isaac.campos,jared.leon,fernando.campos}@unsaac.edu.pe

Abstract. In this paper, a new algorithm for Data Streaming clustering is proposed, namely the SetClust algorithm. The Data Streaming clustering model focuses on making clustering of the data while it arrives, being useful in many practical applications. The proposed algorithm, unlike other streaming clustering algorithms, is designed to handle cases when there is no available a priori information about the number of clusters to be formed, having as a second objective to discover the best number of clusters needed to represent the points. The SetClust algorithm is based on structures for disjoint-set operations, making the concept of a cluster to be the union of multiple well-formed sets to allow the algorithm to recognize non-spherical patterns even in high dimensional points. This yields to quadratic running time on the number of formed sets. The algorithm itself can be interpreted as an efficient data structure for streaming clustering. Results of the experiments show that the proposed algorithm is highly suitable for clustering quality on well-spread data points.

Keywords: Data streaming clustering · Variable clustering · Disjoint-set operations · Data structures for clustering

1 Introduction

Clustering is probably the most important unsupervised machine learning problem. While there are several models of clustering, the vast majority of them require working with the entire set of points to be clustered [1,10]. A particular clustering model called *Data Stream Clustering* allows streaming algorithms to cluster points while they arrive. However, the Data Stream Clustering problem focuses on finding clusters given a priori knowledge of the number of clusters [13].

The proposed algorithm (SetClust) tries to solve a very similar problem, the Variable Streaming Clustering. The problem, as in the Data Stream model, handles the case where the data arrives one point at a time; but in this case, there is no a priori knowledge of the number of clusters to be formed. Because of this, the algorithm not only tries to make clustering of the data but also tries to discover the correct number of clusters *online*, i.e., the number of clusters

J. A. Lossio-Ventura et al. (Eds.): SIMBig 2019, CCIS 1070, pp. 89–96, 2020.
https://doi.org/10.1007/978-3-030-46140-9_9

predicted by the algorithm can considerably change depending on the structure of the data. The algorithm tries to keep the number of predicted clusters as small as possible at any moment.

The experimental part was conducted using synthetic datasets from [7,8,11] aiming to evaluate the performance on well spread clusters and some other closer clusters. The experimental procedure consisted of evaluating the performance of the proposed algorithm and 3 other Data Stream Clustering algorithms on the datasets.

The paper is organized as follows: Sect. 2 briefly mentions the tool needed to construct the algorithm and some related work on the topic. Then, the SetClust Algorithm is described in Sect. 3. Section 4 explains how the experiments were performed and the achieved results. Section 5 offers a brief discussion on the algorithm and the relation hyperparameters have with the data. Finally, the conclusions are presented in Sect. 6.

2 Background

Below, we mention the data structure needed to develop the SetClust algorithm, then the evaluation metric used for testing clustering quality, and finally we mention 3 algorithms that are later used in the experimental comparisons.

Union-Find Disjoint Sets Operations. This data structure [9] allows keeping track of sets of elements storing minimal information about them. The structure allows two kinds of operations: (1) Make a query to know whether two elements are in the same set, and (2) Join two sets into one single set (see [6] for more).

V-measure. This metric is an entropy-based measure which explicitly measures how successfully the criteria of homogeneity and completeness have been satisfied. V-measure is computed as the harmonic mean of distinct homogeneity and completeness scores (see [14] for more on this).

Clustream. This Data Streaming Clustering Algorithm is based on micro-cluster structures that store information about a streaming. The algorithm has two phases: an online phase in which the streams are processed and an offline phase in which the clusters are created using k-means (see [2] for more on this).

ClusTree. This Data Streaming Clustering Algorithm is a self-adaptive index structure that keeps the summarized information about the stream. Also, this algorithm adds aging to the data to eliminate unnecessary information (see [12] for more on this).

DenStream. This Data Streaming Clustering Algorithm is based on densities, the reason why it can take arbitrary cluster shapes during the execution. This algorithm can easily deal with outliers (see [5] for more on this).

3 The SetClust Algorithm

From now on, the terms "cluster" and "set" will be used independently: we use the term set to denote the minimum unit of grouped information, and the term cluster to denote a collection of one or more sets joined together by the same label.

At any time, each set is represented by the following information: (1) label of the set, (2) arithmetic mean of the points that belong to the set, (3) standard deviation of the points, (4) sum of element-wise squared points and (5) the number of points. Also, the disjoint-set structure mentioned in Sect. 2 is built over all the sets. Of course, for each point that arrives, we also must store the label of the set it belongs to. With this information, we define three operations that represent the action to be taken depending on the situation: element aggregation, set linking, and set absorption. The three operations are described below.

Element Aggregation. The purpose of the element aggregation operation is to add the information of a point to a set. This is achieved in the following way: given a point $p \in \mathbb{R}^d$, find the set with the nearest mean $\mu_i \in \mathbb{R}^d$ having a standard deviation $\sigma_i \in \mathbb{R}^d$. Then, we take the number $c \in \mathbb{R}^+$ as hyperparameter. The point p is added to the set if the following inequation holds:

$$\|(p - \mu_i)./(\sigma_i \cdot c)\|^2 < 1 \tag{1}$$

Where ./ is the element-wise division. The intuition behind this condition is that a point is to be added to the set if it lies inside c standard deviations of an ellipsoid with center μ_i. The extension of the ellipsoid in each dimension is given by σ. Given this condition, there are two possible scenarios. First, the point does not belong to the set. In this case, a new set is created having a unique element: the single point. The mean of the set is the point itself, and the standard deviation is initially fixed to σ_{ini} being this last a hyperparameter. At every moment, the standard deviation of each set is at least σ_{ini}. Then, the information about this point is added to the disjoint-set structure. Second, the point belongs to the set. In this case, the information of the set should be updated, i.e., update the mean, standard deviation, and the number of elements[1]. Both calculations have a running time complexity of $O(d)$, being d the dimension the points lie in. The overall complexity of the element aggregation operation is $O(rd)$, being r the number of current formed sets.

Set Absorption. The set absorption operation is responsible for converting two close sets into a single one when discovering that one can be included inside another. The operation goes as follows: given two sets with means and standard deviations μ_i, σ_i and μ_j, σ_j respectively and the hyperparameter c, the two sets must become a single one if one set, when extended c times by its standard deviation is included into the other set extended c times by its standard deviation

[1] These operations should be done online without traversing through all the elements.

in *all* its components. Formally, the set absorption must be performed if the following inequation holds:

$$|\mu_i - \mu_j| + c \times \sigma_j <^d c \times \sigma_i. \tag{2}$$

Where $<^d$ represents the minor operator on *every* component (see Fig. 1 for a one dimensional example).

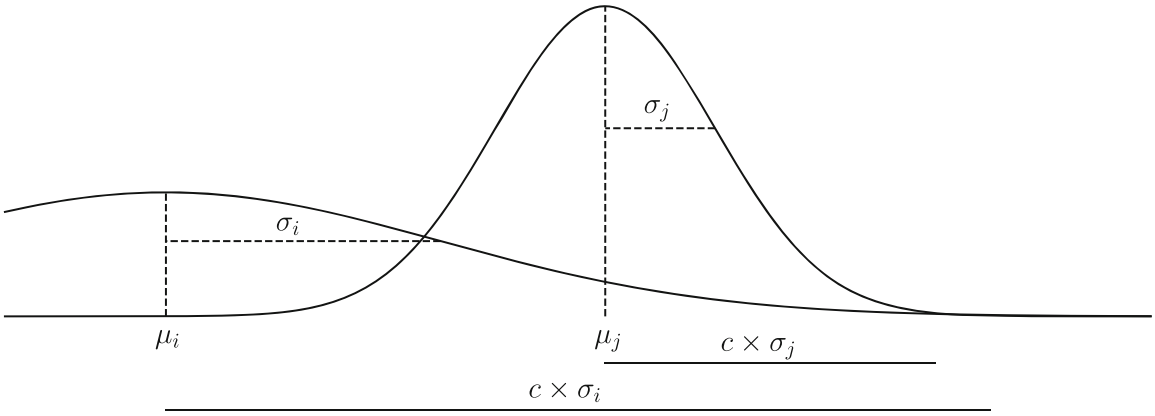

Fig. 1. One dimensional example of two sets satisfying the absorption condition. Set with mean μ_j is inside the acceptance border of set with mean μ_i.

When the absorption condition is met, we proceed to merge both sets recalculating its information to form a new set[2]. The absorption operation is done until no two sets meet the absorption condition. It is possible that after joining two sets the need for making more absorption operations increase since the number of elements that belong to the last modified set also change. The absorption of a set has a running time complexity of $O(r^2 d)$. Depending on the implementation, we might need to traverse all sets once for every verification of the condition[3]. This operation has a good effect on the algorithm since every time it is used, the number of sets decreases by one.

Set Linking. The goal of the set linking operation is to determine whether two sets are close enough to be considered a single *cluster*. Given two sets with means μ_i, μ_j and standard deviations σ_i, σ_j respectively, we say that both sets are linked if the following inequation holds:

$$\|(\mu_i - \mu_j)./(\sigma_i \cdot c)\|^2 < 1 \tag{3}$$

[2] We need to calculate the new mean, standard deviation and the rest of the information in constant time.

[3] If we take advantage of the fact that only the last formed set can make instability, we can achieve an overall worst-case running time complexity of $O(rd)$.

Here, we use the previously defined hyperparameter c. Intuitively, the condition tests whether the mean of a set satisfies the element aggregation condition for another set.

Once determined if two sets will be linked, a *union* operation of the disjoint-set structure is performed on their labels, meaning that both sets belong to a single cluster. Doing the operation in this way, we allow the algorithm to link sets that create complex cluster forms, not just spherical.

The set linking operation is performed after the set absorption operation because if two sets satisfy the absorption condition, then they satisfy the linking condition with high probability. We restrict linking sets that can be absorbed by any other to reduce the number of clusters. Depending on the implementation, this operation can be performed in almost linear time in the number of sets[4].

It is important to clarify that, whenever a new point arrives, the algorithm tries to perform the three operations (element aggregation, set absorption, and set linking in that order) to the whole structure. The three operations are independently executed one after the other. As mentioned before, the worst-case running time complexity of the operations are: $O(rd)$, $O(rd)$ (at most $O(r)$ times), and $O(\alpha(r)rd)$ respectively. If we assume $\alpha(r)$ to be a small constant, the overall complexity of the algorithm when adding a point is $O(r^2d)$.

When two or more sets are linked, they are considered to belong to the same cluster. This property was specifically chosen for the algorithm as it has an advantage: linked sets can form non-spherical cluster shapes. It is quite easy to see that when two or more sets are joined into a single one, the shape of the final set is "pseudo elliptical". However, when two of these sets link each other, then the resulting shape can be more sophisticated. The linking process can continue until exotic shapes are formed. This last behavior helps the algorithm to achieve better cluster quality since it can make more suitable cluster shapes of the data.

At any moment, we can know if two elements belong to the same cluster with the help of the disjoint-set structure. Also, at any moment, the algorithm can return a list of the labels of the points seen so far. The algorithm can also return the entire structure containing all the information used completely to define the sets and clusters to continue doing clustering if necessary. A high-level pseudocode of the SetClust algorithm is shown in Algorithm 1.

Whenever a new point arrives, an element aggregation operation is performed. This can yield into creating a new set or adding the new point to a previously existing set. In any case, the modified or newly created set is taken as a possible candidate for the absorption operation. If the absorption condition is met, then the absorption operation is performed. The process of verifying the absorption condition and performing the absorption operation continues until no absorption operation can be done. Then, the same set is taken as a candidate

[4] As in the previous case, if we take advantage of the fact that only the last formed set can make instability, we can perform this operation in worst-case running time complexity of $O(\alpha(r)rd)$, where α is the inverse Ackerman function. For any practical situation, the function is never greater than 4.

for the linking operation, checking for the corresponding condition as in the previous case. If the condition is met, then the set linking operation is performed only once.

Algorithm 1: The SetClust Algorithm

```
while point p arrives do
    Element Agregation(p)
    Let A be the set where p belongs
    while an absorption can be performed with A do
        Set Absorption(A)
    if a link can be performed with A then
        Set Linking(A)
```

4 Experiments

For the experiments, six synthetic datasets were selected from [7,8,11]. Each dataset contains several points in some specified dimension together with its correct label. Relevant information about the datasets is shown in Table 1.

Table 1. Information about the synthetic datasets used for the experiments

Dataset	Dimension	Number of elements	Number of clusters
Dim512	512	1024	16
Dim1024	1024	1024	16
A1	2	3000	20
A2	2	5250	35
A3	2	7500	50
S1	2	5000	15

The first two datasets contain several Gaussian clusters. Clusters are well separated. The next three are incrementally included; i.e., A1 is included in A2 and A2 is included in A3. These datasets have low dimension and are not so spread as the previous two. The reason for including this type of datasets is to test the performance of the algorithm in an outlier scenario. The last dataset, S1, contain some non-spherical (amorphous) cluster shapes. The purpose of this dataset is to test how algorithms handle this kind of scenarios.

The experimental procedure was performed as follows: for every dataset and algorithm, we performed a grid search to find good hyperparameters. The evaluation of the clustering quality was performed using the V-measure metric [14],

which gives a score between −1 and 1. Table 2 shows the results of this procedure. Each numerical value of the table contains the best score seen by the current algorithm on the current dataset. The best score achieved on every dataset (every column) is highlighted in bold.

Table 2. Results of the experiments

	Dim512	Dim1024	A1	A2	A3	S1
SetClust	**1.00**	**1.00**	**0.97**	**0.96**	**0.97**	**0.97**
Clustream	**1.00**	**1.00**	0.79	0.82	0.83	0.80
ClusTree	**1.00**	**1.00**	0.75	0.74	0.56	0.80
DenStream	**1.00**	**1.00**	0.70	0.66	0.61	0.73

As showed in the table, the first two datasets were easy for all the algorithms due to its spread nature. The next four datasets were more challenging because of the complex nature of clusters shapes and the addition of some outliers present on them. In those datasets, the proposed algorithm clearly shows superior performance, showing the highest score on all cases. Numerical experiments were conducted using the Python programming language (Python 3.XX) on a 2.50 GHz Intel Core i7-4710HQ with 12 GB of RAM and running Windows 10 version 1809 as operating system. The source code of the SetClust algorithm is available in [4]. For the other algorithms, we used the `MOA` framework for Big Data stream mining [3].

5 Discussion

The selection of hyperparameters σ_{ini} and c can have a deep interpretation when looking at the structure of the dataset used. The parameter σ_{ini} can be interpreted as a first approximation to the standard deviation that a cluster will have, for that, it might be a good practice to choose its value looking at the "separation degree" of the dataset. The parameter c can be seen similarly: its value can be interpreted as a guess of how spread the clusters will be during the streaming. In the same way, looking at the structure of the dataset can help to make a good choice for its value.

The Clustream and ClusTree algorithms make use of the correct number of clusters to be formed. The DenStream algorithm, despite not needing this information, requires tuning a great number of other hyperparameters. The proposed algorithm has exactly two hyperparameters, which makes it suitable for a streaming scenario with semi-unknown information.

6 Conclusions

In this paper, a new algorithm for variable data streaming clustering was proposed, the SetClust algorithm. The algorithm has a disjoint-set operations background and can be used as a data structure with online query support.

A maximum clustering quality was achieved when the clusters are well spread. This case is an optimal scenario for the SetClust algorithm (as well as for the other algorithms). The experiments also show that, for the last four datasets, the proposed algorithm finds better clusters than the other tested algorithms.

In spite of the theoretical efficient based formulation of the algorithm, there are lots of implementation details that give much freedom when implementing it. As future work, we will focus on adapting better data structures to handle redundant information so as to make a more efficient implementation of the algorithm. Also, we plan to create a more sophisticated version of SetClust incorporating aging to the data. We also plan to perform running time comparisons between the SetClust algorithm and other Data Streaming Clustering algorithms.

References

1. Aggarwal, C.C., Reddy, C.: Data Clustering: Algorithms and Applications, 1st edn. Chapman & Hall/CRC (2013)
2. Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for clustering evolving data streams. In: Proceedings of the 29th International Conference on Very Large Data Bases, VLDB 2003, vol. 29, pp. 81–92. VLDB Endowment (2003)
3. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: MOA: massive online analysis. J. Mach. Learn. Res. **11**, 1601–1604 (2010)
4. Campos, I., Leon, J.: Setclust. Zenodo, July 2019. https://doi.org/10.5281/zenodo.3270842
5. Cao, F., Ester, M., Qian, W., Zhou, A.: Density-based clustering over an evolving data stream with noise. In: Conference on Data Mining (SIAM 2006), pp. 328–339 (2006)
6. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press, Cambridge (2009)
7. Fränti, P., Sieranoja, S.: K-means properties on six clustering benchmark datasets. Appl. Intell. **48**(12), 4743–4759 (2018). https://doi.org/10.1007/s10489-018-1238-7
8. Fränti, P., Virmajoki, O.: Iterative shrinking method for clustering problems. Pattern Recognit. **39**(5), 761–775 (2006)
9. Galler, B.A., Fisher, M.J.: An improved equivalence algorithm. Commun. ACM **7**(5), 301–303 (1964)
10. Jain, A., Dubes, R.: Algorithms for Clustering Data. Prentice-Hall Inc., Upper Saddle River (1988)
11. Karkkainen, I., Franti, P.: Dynamic local search for clustering with unknown number of clusters. In: Object Recognition Supported by User Interaction for Service, vol. 2, pp. 240–243 (2002)
12. Kranen, P., Assent, I., Baldauf, C., Seidl, T.: Self-adaptive anytime stream clustering. In: 2009 Ninth IEEE International Conference on Data Mining, pp. 249–258 (2009)
13. Muthukrishnan, S.: Data streams: algorithms and applications. Found. Trends Theor. Comput. Sci. **1**(2), 117–236 (2005)
14. Rosenberg, A., Hirschberg, J.: V-measure: a conditional entropy-based external cluster evaluation measure. In: Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pp. 410–420. EMNLP-CoNLL (2007)