D3CAS: un Algoritmo de Clustering para el Procesamiento de Flujos de Datos en Spark

Roberto Molina [1,2] y Waldo Hasperué [1,3]

[1] Instituto de Investigación en Informática (III-LIDI), Facultad de Informática – Universidad Nacional de La Plata
[2] Becario CIN-EVC
[3] Investigador Asociado - Comisión de Investigaciones Científicas (CIC)
rpmolina94@gmail.com, whasperue@lidi.info.unlp.edu.ar

Abstract. En este trabajo se presenta una prueba de concepto de un algoritmo de clustering basado en densidad, denominado D3CAS, el cual fue implementado para ser ejecutado bajo el framework Spark Streaming y que permite el procesamiento de flujos de datos. La principal característica del algoritmo presentado es que es dinámico, es decir selecciona automáticamente el número de clusters del flujo de datos. El algoritmo fue probado datasets de CLUTO, midiendo la calidad de los clusters obtenidos. Los resultados, obtenidos en un ambiente virtualizado, fueron comparados con otro algoritmo de clustering (CluStream), demostrando que D3CAS arroja mejores resultados.

Keywords: Clustering, Spark, Streaming processing.

1 Introducción

Últimamente el avance tecnológico ha conseguido que las organizaciones y empresas puedan generar y almacenar grandes cantidades de datos. Cuando estos datos se generan de manera continua y rápida en el tiempo, siendo imposible almacenarlos en su totalidad, se los denominan 'Flujos de datos' (Data streams).

El procesamiento y análisis de flujos de datos es una característica que se encuentra presente en muchas aplicaciones y sistemas de software de la actualidad. Debido al crecimiento de internet, la automatización de sistemas, el aumento de la conectividad social y el avance en la tecnología, las aplicaciones generan flujos de datos potencialmente infinitos, volátiles y continuos, lo que requiere un procesamiento en tiempo real, simple y rápido.

En muchos casos, estos grandes volúmenes de datos pueden ser minados para obtener información interesante y relevante en una amplia variedad de aplicaciones, como lo son la de detección de intrusiones en la red, flujos de transacciones, registros telefónicos, redes sociales, monitoreo de sensores, internet de las cosas (IoT), en aplicaciones médicas, monitoreo del clima, entre otras.

Una tarea comúnmente realizada sobre flujos de datos es el clustering, la cual consiste en dividir o agrupar la información de tal manera que los datos en cada grupo sean similares y los datos entre grupos sean disímiles. Con flujos de datos potencialmente infinitos, la tarea de clustering es un desafío atractivo ya que los

algoritmos de clustering tradicionales suelen iterar sobre el conjunto de datos más de una vez, es por ello que las restricciones de tiempo de ejecución, uso de la memoria y no poder contar con el conjunto de datos completo deben considerarse cuidadosamente en el contexto del análisis de los flujos de datos. Además, por lo general, la distribución de los datos del flujo cambian continuamente, fenómeno conocido como Concept-drift [1]. Por ello resulta de interés contar con algoritmos de clustering dinámicos, donde la cantidad de clusters en un momento dado dependerá de la distribución de los datos del flujo [2] [3].

En este trabajo proponemos un algoritmo de clustering dinámico para el procesamiento de flujos de datos con funcionamiento distribuido denominado D3CAS (Distributed Dynamic Density based Clustering Algorithm for dataStreams).

El resto del paper se organiza de la siguiente manera. En la sección 2 se lleva a cabo la presentación del modelo de procesamiento de flujos de datos, el estado del arte de las técnicas de clustering sobre flujos de datos y un resumen de las características principales del framework Spark Streaming. En la sección 3, se presenta el algoritmo propuesto. En la sección 4 se presentan los experimentos realizados junto a la comparación con una de las técnicas más importantes dentro de esta área. Finalmente en la sección 5 se presentan las conclusiones y trabajos a futuro.

2 Clustering sobre Flujos de Datos en el Entorno Spark

Spark [4] [5] es un framework distribuido cuya finalidad es ejecutar de manera eficiente operaciones sobre grandes conjuntos de datos, como así también el procesamiento de flujos de datos. Una de las características principales de Spark es la velocidad de ejecución, debido a que tiene la capacidad de ejecutar operaciones sobre los datos en la memoria principal. Spark Streaming es un componente de Spark que permite el procesamiento de flujos de datos, como lo pueden ser datos generados por servidores web o colas de mensajes publicadas por usuarios de un servicio web.

Spark Streaming utiliza una arquitectura denominada "micro-batch", donde el procesamiento del flujo de datos se trata como una serie continua de computaciones sobre pequeños batch de datos, los cuales se crean a intervalos de tiempo regulares. Cada batch de entrada forma un RDD y se procesa mediante operaciones específicas de la API de Spark para generar otros RDD transformados o resultados finales que luego pueden enviarse a sistemas externos.

2.1 El Modelo de Flujos de Datos

En el modelo de flujo de datos, los datos de entrada con los que se van a operar no están permanentemente disponibles para el acceso aleatorio desde la memoria o para recuperar desde un disco, sino que llegan como una o más secuencias continuas de datos temporales [3]. Los flujos de datos son potencialmente ilimitados, lo que lleva a que los datos sean procesado de manera 'on-the-fly', es decir, una vez recibido cierto elemento, este es procesado es ese mismo momento [2] [6].

Existen dos enfoques al momento de procesar un flujo de datos. El aprendizaje incremental donde el modelo evoluciona incrementalmente para adaptarse a los

cambios que presentan los datos entrantes en el flujo [7] y el aprendizaje en dos fases, también conocido como *online-offline learning*, donde la idea básica es tener una primera fase (*online*) la cual va generando un resumen de los datos que llegan en tiempo real. En la segunda fase (*offline*), el proceso de minería se realiza utilizando las síntesis generadas en la etapa online [8].

Como los flujos son potencialmente infinitos, sólo es posible procesar una parte de su totalidad. A la porción de datos sobre la cual se va a trabajar se la define como ventana temporal: $W[i,j] = (x_i, x_{i+1}, x_{i+2}, \ldots, x_j)$, donde i y j son puntos en el tiempo con la propiedad de que $i < j$. En la literatura es posible encontrar diferentes tipos de ventanas de tiempo: *landmark window* [9], *sliding window* [9], *fading window* [10], y *tilted time window* [8], cada una con sus ventajas y desventajas.

El problema de agrupamiento sobre flujos de datos, o *data stream clustering*, es un proceso que divide los datos de forma continua teniendo en cuenta las restricciones de memoria y tiempo. En la literatura, la mayoría de los algoritmos de agrupamiento de flujo de datos usan un esquema de dos fases. Muchos de los algoritmos de agrupamientos de flujo de datos existentes modifican los métodos tradicionales de clustering para usar el enfoque de dos fases propuesto en [8]. Por ejemplo, DenStream [10] es una extensión del algoritmo DBSCAN [11], StreamKM++ [12] es una extensión de k-means++ [13] y StrAP [14] es una extensión de AP.

3 D3CAS

El algoritmo propuesto está basado en las estructuras de datos (micro-clusters) y la metodología de procesamiento online-offline usadas por CluStream [8] y por DenStream [10]. Un micro-cluster, es la estructura de datos que representa y resume a un conjunto de datos proveniente del flujo. Un micro-cluster es una tupla de elementos donde principalmente se almacena: el número total de datos del micro-cluster, la suma lineal y cuadrática de los elementos del conjunto de datos. La metodología de procesamiento online-offline propone separar el proceso de clustering en dos etapas, la etapa online se encarga de generar los micro-clusters y la etapa offline de utilizar los micro-clusters para realizar la tarea de clustering.

El objetivo es implementar un algoritmo que funcione en una arquitectura distribuida, específicamente, bajo el modelo Spark Streaming, lo cual representa la característica diferencial más importante con respecto a las técnicas mencionadas, ya que ambas están diseñadas para trabajar en un ambiente no distribuido.

Para realizar el tratamiento del flujo de datos, se utiliza el modelo de ventana *fadding window* [10], el cual posee la ventaja de aumentar la importancia a los datos más recientes, lo cual facilita la detección temprana de cambios en la distribución de los datos. Para ello, cada dato debe almacenar un atributo que determine su importancia en el tiempo. Este valor es calculado por medio de una función de envejecimiento, en la que el valor de cada dato disminuye exponencialmente a medida que pasa el tiempo (Ec. 1).

$$f(dt) = 2^{-\lambda.dt} \quad \textbf{(1)}$$

donde $\lambda > 0$ y *dt* es la diferencia entre el timestamp de llegada del dato y un instante de tiempo mayor al de llegada, generalmente representado por el instante de tiempo actual. Como puede observarse en la ecuación (1) cuanto mayor sea el valor de λ, menor será la importancia de un dato en comparación con los datos más recientes.

En D3CAS se utiliza el enfoque *online-offline* para el tratamiento del flujo. La etapa online se ejecuta paralelamente en los *workers* de Spark debido a que la tarea de generar los micro-clusters se ajusta al tipo de procesamiento que realizan las operaciones *map* de Spark, mientras que la etapa offline se ejecuta en el nodo *master*, ya que para llevar a cabo la tarea de detección de agrupaciones necesita conocer todos los micro-clusters generados.

3.1 Etapa Online

En esta etapa se utiliza la técnica de micro-clusters, la cual consiste en reducir el volumen de datos a un modelo más pequeño y representativo. Un micro-cluster en un instante de tiempo *t* para un conjunto *P* de *n* puntos de *d* dimensiones $p_1, ..., p_n$ y sus respectivos timestamps $T_1, ..., T_n$, se define como una 7-tupla $MC = \{ LS, SS, w, n, d, t_c, t_m \}$ donde *LS* representa la suma lineal ponderada del grupo de datos, *SS* representa la suma cuadrática ponderada del grupo de datos, *w* representa el peso o importancia del micro-cluster en el tiempo, *n* representa la cantidad de puntos en el conjunto *P*, *d* representa la dimensionalidad de los puntos en el conjunto *P*, t_c representa el timestamp del instante de tiempo cuando fue creado el micro-cluster y t_m representa el timestamp de la última actualización del micro-cluster.

Ante la llegada de un nuevo dato p_i de la partición de datos P_k, se buscan todos los puntos q_j cuya distancia a p_i sea menor a *e* (parámetro del algoritmo que representa el radio del micro-cluster). Si no se encontró ningún punto q_i, entonces se crea un micro-cluster para representar el punto p_i. Si se encontró un conjunto de puntos cercanos a p_i entonces, sea el conjunto *Q* que se encuentra formado por p_i y por el conjunto de puntos cercanos a p_i, se crea un micro-cluster de la siguiente forma: $MC(LS(Q), SS(Q), t_q, cant(Q), cant(Q), timestamp(Q), timestamp(Q))$.

Luego de este proceso, los nodos *workers*, se encargan de enviar el conjunto de micro-clusters generados al nodo *master*, quien será el encargado de realizar el proceso de actualización y llevar a cabo el proceso de detección de clusters.

3.2 Etapa Offline

En esta etapa el proceso de clustering se lleva a cabo mediante la utilización de una técnica basada en densidad, particularmente DBSCAN [11], la cual permite detectar agrupaciones sin especificar cuantas formar, proporcionando la posibilidad de detectar agrupaciones con formas arbitrarias y detectar ruido o outliers presentes en los datos.

Para este proceso se utilizan los centros de todos los micro-clusters generados en la etapa previa cuyo resultado final se ve afectado por dos parámetros pertenecientes a DBSCAN: *eps* y *minpoints*. *eps* representa el radio de un punto *i* en donde buscar sus

puntos vecinos y *minPoints* representa la cantidad mínima de puntos vecinos en el radio *eps* que tiene que tener el punto *i* para considerarse un cluster.

En esta etapa además se lleva a cabo el proceso de actualización del peso de los micro-clusters, ya que recién en esta etapa se tienen todos los micro-clusters del flujo, debido a que los *workers* solo conocen sus respectivos micro-clusters.

La actualización de los micro-clusters se realiza periódicamente y se actualizan los atributos de peso *w* y los atributos *LS* y *SS* que también están influenciados por la importancia del tiempo. Sea *t* el instante de tiempo actual, t_m el timestamp de última modificación del micro-cluster y *dt* la diferencia entre *t* y t_m, entonces cada micro-cluster actualiza sus atributos con las ecuaciones (2) (3) y (4).

$$w = 2^{-\lambda.dt}.w \quad (2)$$

$$LS = 2^{-\lambda.dt}.LS \quad (3)$$

$$SS = 2^{-\lambda.dt}.SS \quad (4)$$

Como puede observarse, dada la característica decreciente de la función de envejecimiento utilizada, los pesos de los micro-clusters van disminuyendo gradualmente. Luego del proceso de actualización del peso, se lleva a cabo el proceso de eliminación de los micro-clusters que se consideran viejos según su peso. Un micro-cluster se considera viejo y debe ser descartado si su peso *w* es menor a un umbral μ, siendo $0 < \mu < 1$.

4 Resultados

Se realizaron dos experimentos. En una primera instancia se probó con flujos de datos, los cuales forman clusters esféricos y bien separados entre sí, utilizando tres datasets generados por el script GENERATEDATA [15] [16], los cuales en el texto son nombrados como 100K10C, 100K40C y 100K14C. El segundo experimento consistió en la detección de clusters en datasets con agrupaciones irregulares.

Para todas las ejecuciones se utilizó el mismo juego de parámetros. En la componente online: $e = 2$ y en la componente offline: *eps*=3.0 y *minPoints* = 4.0, los cuales se corresponden con los parámetros originales de DBSCAN [11]. Para controlar la ventana deslizante utilizada se utilizaron los parámetros $\lambda = 0.25$ y $\mu = 10$ qué son los parámetros que se utilizan generalmente en este tipo de ventanas [10].

Las comparaciones se realizaron contra los resultados arrojados por CluStream [8] utilizando la implementación sobre Apache Spark creado por “HUAWEI Noah's Ark Lab” [17] que se encuentra dentro de su proyecto de minería de flujos de datos StreamDM [18]. Todos los experimentos se realizaron en un ambiente local, en una computadora con un procesador Intel i5 3210M y 8GB de RAM.

4.1 Datasets con clusters esféricos

Una de las características de D3CAS es la detección de clusters de forma dinámica. Para comprobar su correcto funcionamiento se realizó un primer experimento sobre un flujo de datos donde, en distintos instantes en el tiempo, el flujo posee diferentes cantidades de agrupaciones, es decir, con el correr del tiempo, los datos del flujo entrante conforman nuevos clusters.

Para esta prueba se utilizó el dataset denominado 100K10C, el cual está compuesto de 100.000 elementos que se encuentran agrupados en 10 clusters. Se evaluaron los resultados parciales al formarse 4 y 7 clusters y al finalizar la ejecución. En cada evaluación se midió el índice silhouette de cada cluster para evaluar la calidad de los grupos formados.

La tabla 1 muestra los valores de los índices silhouettes correspondientes luego de cada evaluación. Como era de esperarse, ambos algoritmos dan muy buenos resultados, lo cual indica que los clusters detectados por D3CAS son correctos.

Tabla 1. Valores del índice silhouette para las evaluaciones de 4, 7 y 10 clusters.

Número de cluster	Evaluación 1 (4 clusters)		Evaluación 2 (7 clusters)		Evaluación 3 (10 clusters)	
	D3CAS	CluStream	D3CAS	CluStream	D3CAS	CluStream
1	0.95	0.96	0.97	0.95	0.98	0.98
2	0.95	0.96	0.96	0.97	0.98	0.96
3	0.95	0.96	0.95	0.97	0.94	0.98
4	0.95	0.96	0.95	0.95	0.98	0.97
5			0.94	0.97	0.98	0.98
6			0.94	0.96	0.97	0.99
7			0.95	0.98	0.99	0.95
8					0.95	0.97
9					0.94	0.95
10					0.95	0.96

Al elevar el número de clusters por arriba de 10 (usando el dataset 100K40C) es posible observar que D3CAS ofrece mejores resultados con respecto a CluStream. En la figura 1 se puede observar que CluStream confunde clusters distintos agrupándolos en un único cluster, lo que queda evidenciado en la figura 2 donde se observa para CluStream un valor de silhouette promedio de 0.65 mientras que para D3CAS un valor promedio de 0.95. Para un número de clusters alto, D3CAS los detecta correctamente mientras que CluStream no.

4.2 Datasets con Formas Arbitrarias

En este experimento se realiza una comparación utilizando flujos de datos compuestos por datos que forman clusters con formas arbitrarias y aleatorias como por ejemplo se puede encontrar en flujos de datos de monitoreo de tráficos de redes [10] [11].

Para estas pruebas, se utilizan datasets conocidos para este tipo de análisis como los que ofrece el kit de herramientas “CLUTO” [19], desarrollado por el departamento

de Computer Science de la universidad de Minnesota. Los datasets de CLUTO que se van a utilizar son los siguientes:

- cluto-t4-8k: 8.000 datos, 6 clusters irregulares con ruido.
- cluto-t8-8k: 8.000 datos, 8 clusters irregulares con ruido.
- cluto-t7-10k: 10.000 datos, 9 clusters irregulares con ruido.

Para la validación de estas pruebas se utiliza el índice de pureza (Ec. 5).

$$pur = \frac{\sum_{i=1}^{K} \frac{|C_i^d|}{|C_i|}}{K} \tag{5}$$

donde K es el número total de clusters, $|C_i^d|$ representa la cantidad de elementos agrupados en el cluster i por el algoritmo de clustering y $|C_i|$ representa la cantidad total de elementos etiquetados para el cluster i. Por lo tanto, la pureza mide la cantidad de elementos que fueron correctamente agrupados según la etiqueta real del dataset, característica que poseen los datasets de CLUTO.

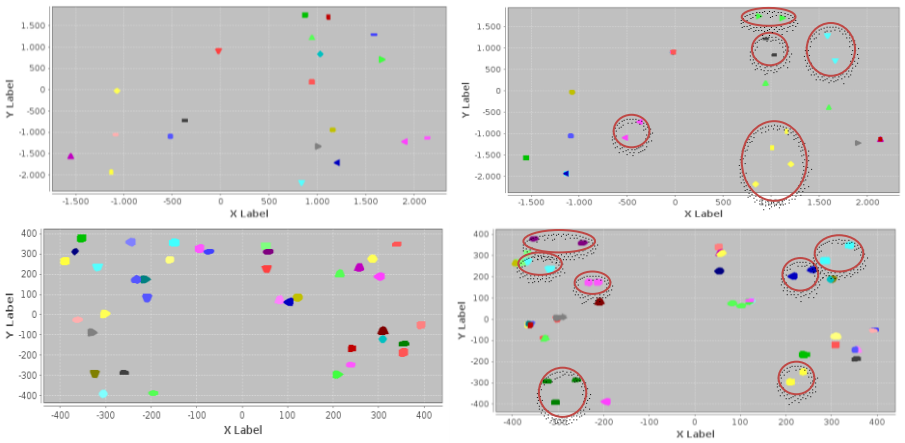

Fig. 1. Clusters hallados al finalizar la ejecución. En la columna de la izquierda los resultado de D3CAS y en la columna de la derecha los de CluStream. La fila de arriba son los resultados con 21 clusters y la fila de abajo corresponde a los resultados luego de presentar los 40 clusters. Los clusters mal agrupados están marcados con un óvalo.

Tabla 2. Valores de pureza para los datasets de CLUTO.

Dataset	D3CAS	CluStream
cluto-t4-8k	90 %	47 %
cluto-t8-8k	93 %	39 %
cluto-t7-10k	92 %	27 %

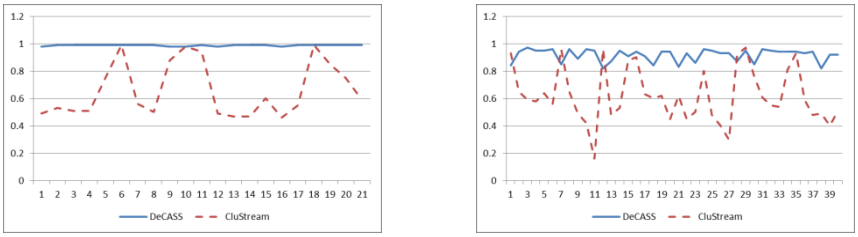

Fig. 2. Índices silouette conseguidos en los clusters hallados por D3CAS y CluStream. A la izquierda el resultado con de 21 clusters y a la derecha el de 40 clusters.

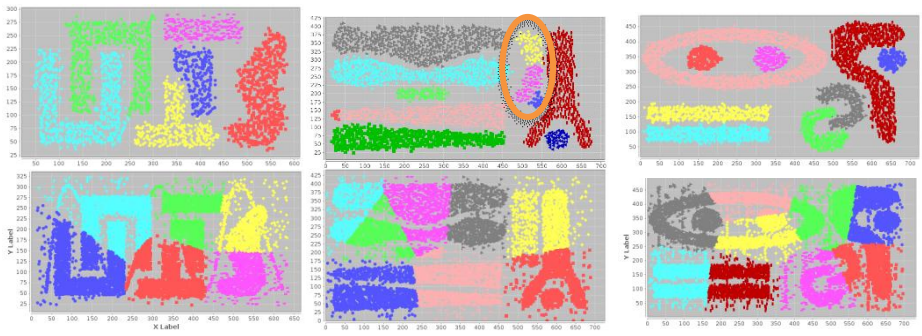

Fig. 3. Resultados hallados por D3CAS (gráficos de la fila superior) y CluStream (gráficos de la fila inferior) para distintos datasets (de izquierda a derecha: cluto-t4-8k, cluto-t8-8k y cluto-t7-10k). La mala agrupación de D3CAS aparece remarcada con un óvalo.

Como se puede ver en la figura 3, D3CAS detecta correctamente los clusters excepto en el caso de cluto-t8-8k, que uno de los grupos los separa en más de un cluster. Se puede observar también que CluStream no consigue detectar correctamente ninguno de los grupos. Todo esto queda reflejado en los resultados de pureza obtenidos para los dos algoritmos que se puede apreciar en la tabla 2. Por lo tanto, es posible afirmar que el D3CAS es capaz de detectar clusters con formas arbitrarias o irregulares y además obtiene mejores resultados que CluStream.

Otra característica de suma importancia para destacar es que D3CAS puede detectar y filtrar los datos que representan ruido, como se aprecia en la figura 3, en estos casos los puntos alejados de los clusters formados no forman parte de ellos. Esta característica no es satisfecha por CluStream, la cual, toma los datos que representa ruido como parte de los clusters.

4.3 Variando el parámetro *e*

En la implementación de D3CAS la etapa online es la encargada de procesar los datos que llegan al flujo y generar los micro-clusters que se utilizaran en la etapa offline. La cantidad de estos micro-clusters depende del parámetro *e*, el cual determina el radio del micro-cluster que representa a los datos originales que se encuentran dentro de su área, por lo que se desprende que a mayor radio menor cantidad de micro-clusters.

Como último experimento se propuso evaluar, para un mismo dataset, cual es la calidad lograda dependiendo del radio del micro-cluster. Para esta prueba se utiliza el dataset denominado 100K14C que posee 100.000 datos distribuidos entre 14 clusters con el objetivo de consumir el flujo con distintos valores de *e* (0.2, 0.25, 0.3, 0.35, 0.5, 1.0, 2.0, 4.0) a fin de generar distintos cantidades de micro-clusters y analizar la variación en los resultados.

Como se ve en la figura 4, utilizando 500 micro-clusters (*e* = 4.0) se obtiene un índice de silhouette de 0.81, lo cual prueba que se está detectando correctamente los distintos grupos, teniendo en cuenta además que 500 micro-clusters representa el 0,5% del dataset completo. Con 25.000 micro-clusters (*e* = 2.0), (25% del dataset), se observa un coeficiente de 0.86, marcando una diferencia de 0.05 con respecto a la prueba de 500 micro-clusters. Esto demuestra que una representación más específica con más micro-clusters no genera una diferencia significativa entre los resultados obtenidos.

Por último, es importante remarcar que es más conveniente tener la menor cantidad de micro-clusters posible para reducir el tiempo de ejecución de la etapa offline ya que, al ser la encargada de generar los modelos, tiene que iterar sobre los micro-clusters generados repetidas veces.

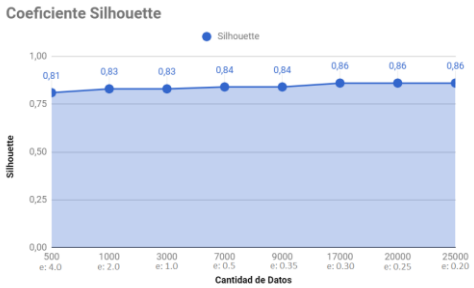

Fig. 4. Índice silhouette hallados por D3CAS para diferentes cantidades de micro-clusters.

5 Conclusión y Trabajo a Futuro

En este trabajo presentamos D3CAS, una prueba de concepto de un algoritmo de clustering dinámico basado en densidad para el procesamiento de flujos de datos que puede ser ejecutado en el framework Spark Streaming.

Los experimentos fueron llevados a cabo en un ambiente virtualizado no distribuido, midiendo únicamente la calidad de los resultados obtenidos sin medir eficiencia en los recursos necesarios durante la ejecución del algoritmo.

Los resultados obtenidos en los experimentos y comparaciones con otro algoritmo de clustering distribuido son alentadores, logrando una buena calidad tanto para los datasets de clusters esféricos como para datasets de formas arbitrarias.

Como trabajo futuro se propone la ejecución en un ambiente distribuido que permitan medir y mejorar el rendimiento del algoritmo presentado, con el fin de realizar comparaciones tanto a nivel de resultados como también a nivel de consumo de recursos como memoria, overhead de comunicación, consumo energético, etc.

Otro aspecto a estudiar es el de ejecutar pruebas en un entorno real, consumiendo un flujo de datos real, como por ejemplo, los flujos de datos que brindan los servicios de Twitter o de redes de sensores.

References

[1] Geoffrey I. Webb, Loong Kuan Lee, Bart Goethals, Franc¸ois Petitjean. *Understanding Concept Drift*. Computer Science. arXiv:1704.00362 (2017)

[2] Aggarwal C.C. *Data Streams: An Overview and Scientific Applications*. In: Gaber M. (eds) Scientific Data Mining and Knowledge Discovery. Springer, Berlin, Heidelberg (2009)

[3] Brian Babcock Shivnath Babu Mayur Datar Rajeev Motwani Jennifer Widom, *Models and Issues in Data Stream Systems*. Department of Computer Science, Stanford University Stanford, CA 94305 (2002)

[4]. Apache Spark. http://spark.apache.org/ . Accedido en 07/2018.

[5]. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S., Stoica, I. *Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing*. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, pp. 1–14. USENIX Association (2012)

[6] Gama J, Rodrigues P. *An overview on mining data streams*. Studies in computational intelligence, Vol. 206 pp 29–45. Springer, Berlin. (2009)

[7] Gepperth, A., Hammer, B.. *Incremental learning algorithms and applications*. European Symposium on Artificial Neural Networks (ESANN), Bruges, Belgium (2016).

[8] Charu C Aggarwal, Jiawei Han, Jianyong Wang, and Philip S Yu. *A framework for clustering evolving data streams*. In Proceedings of the 29th international conference on Very large data bases-Volume 29, pages 81–92. VLDB Endowment, (2003)

[9] Zhang P, Zhu X, Shi Y, Wu X *An aggregate ensemble for mining concept drifting data streams with noise*, Lecture notes in computer science, 5476 pp 1021–1029. Springer, Berlin. (2009)

[10] Cao F, Ester M, Qian W, Zhou A. *Density-based clustering over an evolving data stream with noise*. In: Proceedings of the SIAM international conference on data mining, pp. 328–339. (2006)

[11] Ester, M., Kriegel, H.-P., Sander, J., Xu, X. *A density-based algorithm for discovering clusters 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 Pages 226-231 (1996)

[12] Ackermann, M. R., Märtens, M., Raupach, C., Swierkot, K., Lammersen, C., Sohler, C. *StreamKM++: A clustering algorithm for data streams*. ACM J Exp Algorithmics; 17(1):173–187. (2012)

[13] Arthur, D. and Vassilvitskii, S. *k-means++: The Advantages of Careful Seeding*. Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms Pages 1027-1035 (2007)

[14] Zhang X, Furtlehner C, Sebag M. *Data streaming with affinity propagation*. In: ECML/PKDD (2). Berlin: Springer Berlin Heidelberg. p. 628–43. (2008)

[15] GENERATEDATA. https://github.com/FakenMC/generateData. Accedido en 07/2018.

[16] Fachada, N., Figueiredo, M.A.T., Lopes, V.V., Martins, R.C., Rosa, A.C., *Spectrometric differentiation of yeast strains using minimum volume increase and minimum direction change clustering criteria*, Pattern Recognition Letters, vol. 45, pp. 55-61 (2014)

[17] HUAWEI Noah's Ark Lab. http://www.noahlab.com.hk. Accedido en 07/2018.

[18] StreamDM. https://github.com/huawei-noah/streamDM. Accedido en 07/2018.

[19] CLUTO. http://www.cs.umn.edu/~cluto. Accedido en 07/2018.