Optimización de recurso para el tratamiento de grandes volúmenes de datos

Mónica Santibáñez[1], Rosa María Valdovinos[2], Erendira Rendón[3], Roberto Alejo[4] y J. Raymundo Marcial-Romero[2]

[1]Centro Universitario UAEM Texcoco, Jardín Zumpango s/n Fracc. El Tejocote, México
[2]Universidad Autónoma del Estado de México. Facultad de Ingeniería, Cerro de Coatepec s/n Toluca, México
[3]Instituto Tecnológico de Toluca, Av. Tecnológico s/n Fracc. La Virgen, Metepec, México
[4]Tecnológico de Estudios Superiores de Jocotitlán, Carretera a Toluca Atlacomulco Km. 44.8 s/n Jocotitlan, México
{monicass_isc@hotmail.com, li_rmvr@hotmail.com, erendon@ittoluca.edu.mx, ralejoll@hotmail.com, jrmarcialr@uaemex.mx

Resumen El crecimiento exponencial de los datos en los últimos años ha generado la búsqueda de nuevas soluciones que permitan su tratamiento, con los recursos disponibles de forma rápida y precisa. En minería de datos esta situación es fundamental por utilizar grandes volúmenes de datos en sus procesos. En este artículo se propone una metodología centrada en la administración dinámica de la memoria RAM que permite realizar el aprendizaje *on-line* de patrones. Con la utilización de métodos de agrupamiento y clasificación con la regla de los *k*-vecinos más cercanos, se valida la propuesta sobre 3 conjuntos de datos, los cuales muestran la competitividad de la metodología respecto a una situación donde el conjunto de datos fuera manipulado *off-line*, con una reducción significativa en tiempo de procesamiento.

Palabras clave: aprendizaje *online*, escalabilidad, minería de datos, *clustering*.

1. Introducción

La última década ha representado, para la tecnología, una de las épocas con mayor avance, con lo que el crecimiento de los datos y el flujo de información ha sido un tema que cada vez cobra mayor interés, ya que el tratamiento de los datos se traduce en la adquisición de conocimiento. En consecuencia, el incremento exponencial de éstos, demanda que el almacenamiento y manejo de los mismos ofrezcan una respuesta en tiempo real de forma dinámica. Esta situación ha propiciado la búsqueda de nuevas soluciones que permitan el tratamiento de grandes volúmenes de datos, considerando los recursos disponibles.

Al respecto, la minería de datos es una de las áreas de la Inteligencia Artificial que se ha preocupado por proporcionar estrategias en el tratamiento de grandes volúmenes de datos para su procesamiento y adquisición de conocimiento [1]. En específico, la etapa de Reconocimiento de Patrones (RP), centrada en el reconocimiento y clasificación de objetos definidos por sus atributos, ofrece una alternativa en la adquisición de conocimiento a partir de un conjunto de datos (CD). En la aplicación del RP pueden distinguirse dos aproximaciones: aprendizaje *on-line* y aprendizaje *off-line* [2]. Por un lado, el aprendizaje *on-line* contempla el aprendizaje dinámico de patrones, utilizando para ello CD actualizados, en tanto que en el aprendizaje *off-line* se realiza un aprendizaje estático que no facilita la integración de nuevos ejemplos al conjunto de entrenamiento (CE), donde cualquier caso nuevo debe ser integrado, por un experto en el área, en un nuevo CE e iniciar nuevamente el proceso de entrenamiento.

Un aspecto importante a considerar es que muchos de los datos que se generan en tiempo real, carecen de cualquier descripción que indique de forma clara su clasificación. Para esto, las propuestas existentes se orientan a la utilización de algoritmos de agrupamiento o *clustering*, con los cuales es posible generar grupos a partir de datos sin ninguna descripción, cuyos miembros sean lo más parecidos entre sí y lo más diferentes a los de otros grupos, haciendo posible de esta forma encontrar una relación entre los datos [3].

Para que esta propuesta sea viable con grandes volúmenes de datos, es fundamental administrar de forma adecuada los recursos computacionales disponibles. El problema principal en este sentido es que en ocasiones el CD es más grande que el tamaño de memoria disponible para su procesamiento, por lo que es preciso encontrar una forma de tratarlos sin alterar su precisión. Algunas estrategias se orientan a la escalabilidad de algoritmos [4] y otras a la optimización de la memoria [5]. La primera estrategia busca mantener el adecuado funcionamiento del algoritmo de minería de datos al particionar el CD, en tanto que la segunda, busca el procesamiento de los datos atendiendo a los recursos disponibles del equipo en uso.

En el presente artículo se propone una estrategia que combina ambos enfoques, por un lado realizar la administración de la memoria RAM para trabajar con CD de gran tamaño, sobre los cuales se escalan algoritmos de agrupamiento para llevar a cabo el aprendizaje *on-line* con igual o mejor precisión que si se estuviese trabajando con el CD completo.

La estructura del artículo es la siguiente: en la sección 2 se presenta la fundamentación teórica del aprendizaje *on-line* y se describe el algoritmo de agrupamiento utilizado, en tanto que la sección 3 muestra el funcionamiento y administración de la memoria RAM en Java. La estrategia de solución se describe en la sección 4 y el análisis experimental en la sección 5. Por último se incluye una sección para las conclusiones y las líneas abiertas de estudio.

2. Aprendizaje *Online*

Actualmente, el proceso de minería de datos aplicado a grandes volúmenes de datos para obtener conocimiento requiere de técnicas capaces de soportar el procesa-

miento y análisis en tiempo real de datos; es el aprendizaje on-line o incremental, quien cubre estas necesidades, pues con su implementación se procesan flujos continuos de datos, con la capacidad de dar una respuesta en todo momento [6]. Esta aproximación resulta ventajosa principalmente cuando se requiere mantener un conocimiento del clasificador actualizado y dinámico, no obstante, la complejidad de implementación es una tarea no trivial de resolver. En términos generales el proceso de aprendizaje on-line consiste en procesar uno a uno los patrones que son ingresados [7]. De acuerdo a [8], el aprendizaje on-line supera notablemente al aprendizaje off-line (*batch*), pues a través del uso de una escala para medir el tiempo de cálculo, demuestra que el rendimiento sufre una variación significativa en varios casos.

El aprendizaje *on-line* ha sido aplicado en diversas áreas y tareas en las que la generación y el análisis de datos continuos es de gran importancia. Un caso particular es la utilización en áreas donde el procesamiento constante de cada dato aporta al sistema la información para mantenerse actualizado y poder brindar respuesta en tiempo real, como el monitoreo de redes, los sistemas de telecomunicación, los flujos generados por los clics de los clientes, los mercados de valores, entre otras tareas [9].

Dado que muchos de los datos generados en tiempo real no cuentan con ninguna descripción, es necesario contar con algoritmos que permitan identificar algún comportamiento en ellos. Los algoritmos que brindan esta posibilidad son los de agrupamiento o *clustering*. La idea general de estos métodos es separar los patrones en grupos con características similares, utilizando para ello medidas de similitud o disimilitud que aseguren la mayor semejanza entre los patrones que forman un grupo y lo más diferente a los contenidos en otros grupos [10].

En su aplicación se pueden distinguir dos enfoques: el *hard clustering*, en el cual se generan grupos sin solapamiento y que a su vez se divide en algoritmos de partición, que generan un determinado número de grupos dependiendo de la función de densidad y los algoritmos jerárquicos, que generan agrupamientos internos a uno existente, y el *soft clustering*, que trabaja con el solapamiento de grupos. Estos algoritmos están basados en los métodos difusos, las redes neuronales artificiales y los algoritmos genéticos.

En esta propuesta se emplea el algoritmo de *clustering* Batchelor y Wilkins, correspondiente al *hard clustering*, también conocido como algoritmo de máxima distancia. En su funcionamiento requiere del parámetro θ, el cual, de acuerdo a la literatura, debe estar en el rango $0 \leq \theta \leq 1$, e indica la distancia media entre los grupos existentes. A partir del parámetro θ se calcula un umbral de distancia que permite decidir cuándo se crean nuevos grupos [11].

El algoritmo consiste en asignar arbitrariamente uno de los patrones como centro del primer grupo y utilizando la distancia, en este caso Euclídea, seleccionar el patrón más alejado y asignarlo como centro del segundo grupo. A partir de los dos grupos creados se crea una tabla que contiene los pares (patrón, centro más cercano), entre ellos se elige el patrón más alejado de los grupos existentes y se calcula el umbral de distancia, este umbral es comparado con la distancia del patrón a su centro más cercano, si el umbral es superado se crea un nuevo grupo, si no concluye este paso y se continua con la agrupación libre, donde los patrones que no fueron seleccionados

como centro de grupo de forma iterativa son asignados al centro más cercano, recalculando de esta forma el centroide, hasta que se concluye con todos los patrones [11].

3. Manejo de memoria RAM con Java

El manejo de memoria permite adaptar el recurso de RAM a las necesidades que se tengan, por ejemplo, es posible tener acceso a los datos almacenados en la RAM mediante acceso directo a través de una dirección de memoria y realizar algunas acciones como la lectura y escritura [12]. En el caso de las aplicaciones, la memoria RAM a utilizar es designada de antemano, en el caso de Java (el lenguaje de programación que se utiliza para desarrollar la propuesta), interactúa con la RAM a través de la JVM (JVM por sus siglas en inglés, Java Virtual Machine), siendo esta la responsable de que Java sea multiplataforma. En la división de memoria de la JVM se maneja el espacio que será asignado tanto a los datos permanentes, como a datos que son dinámicos, es decir, aquellos necesarios sólo durante un periodo de la ejecución (Fig. 1) [13].

Fig. 1. Estructura de la memoria JVM.

En la estructura de la memoria de la JVM, la memoria *Heap* se crea al iniciar la máquina virtual de Java, esta se carga inicialmente con espacio de 64 MB, los cuales pueden aumentar o contraerse, de acuerdo a las necesidades de la aplicación. Dentro de esta área de memoria se implementa el administrador automático de almacenamiento (*automatic storage management*) [14] que mantiene un historial de los objetos y al momento de no ser referenciados libera el espacio con el uso del recolector de basura o *Garbage Collector*.

El área de métodos se crea con el inicio de la máquina virtual y puede también ser de tamaño fijo o dinámico, de acuerdo a los requerimientos en tiempo de ejecución, aunque también implementa un límite máximo, en este caso la memoria tampoco necesita ser contigua. Java no provee mayor información sobre la memoria, ya que estos datos se dejan a discreción del implementador, pero dentro de la información que si puede ser consultada, se encuentran las instrucciones para manipular el tamaño de las dos áreas de memoria [15]. Al respecto, una de las propuestas más recientes es la de Choi quien propone un algoritmo escalable de memoria externa, que permite tratar con el problema *MaxRs* de las bases de datos espaciales y los algoritmos *in-memory*, así mismo propone un algoritmo para el problema *MaxCRS* [5].

4. Herramientas y métodos

En esta sección se describen las herramientas y métodos utilizados en la propuesta, donde se busca realizar una recuperación eficiente de patrones en forma on-line y la aplicación de escalabilidad al algoritmo de agrupamiento.

4.1. Administración de memoria

Para identificar la capacidad de memoria RAM del equipo en uso, y a partir de ésta determinar el tamaño que los subconjuntos han de tener para ser procesados de forma consecutiva, se utiliza la librería Libsigar [16], que implementa las funciones necesarias para conocer el tamaño total de memoria en MB, así como el número de procesadores con los que cuenta el equipo, cantidad de espacio libre de memoria, que es manejada en KB, entre otros datos del sistema. Al trabajar con Java, puede modificarse el tamaño por default asignado a la memoria, para aumentar la capacidad y evitar desbordamientos.

Una vez identificado el recurso disponible, se destina un 10% para la ejecución de la aplicación. De acuerdo a este espacio disponible se asigna el 80% a la memoria volátil de la JVM y el 20% a la memoria permanente. Esto se debe a que en la memoria volátil se almacenan las variables dinámicas, que serán utilizadas por un "periodo", mientras que la memoria permanente almacena aquellos datos que estarán presentes durante toda la ejecución del programa, entre ellas se encuentran las librerías, clases, interfaces, entre otras.

Para fines de experimentación con los conjuntos de datos obtenidos, el tamaño fijado en la simulación para la memoria, se realiza a partir del tamaño del conjunto de datos en Bytes. Si el tamaño del conjunto en Bytes no es un número entero entonces este se redondea al siguiente entero, ya que para calcular el número de Bytes cargados en memoria se manejan sólo cantidades enteras. Enseguida la cantidad obtenida como tamaño del conjunto de datos es dividida entre el número de conjuntos con los que se desea realizar la prueba, a esta cantidad se suma el número de patrones existentes en el conjunto, ya que a cada línea hay que agregar el espacio que ocupa el carácter de salto de línea. De esta forma es posible especificar en los experimentos el tamaño total de RAM.

Para el caso particular de los experimentos aquí realizados se utilizó una computadora Dell Insipiron 1545 con 3 GB en RAM, procesador Intel® Core™ 2 Duo T6600 a 2.20 GHz, Disco Duro de 250 GB, con sistema operativo Windows 7 de 64 bits.

4.2. Conjuntos de datos

Los conjuntos de datos fueron tomados del repositorio de la SIPU (por sus siglas en inglés *Speech and Image Processing Unit*) de la Universidad del Este de Finlandia [17]. La Tabla 1 muestra las características de los CD utilizados. La primera columna corresponde a un conjunto artificial de formas arbitrarias, D31, en la segunda y terce-

ra columna se muestra una descripción de los CD reales que corresponden a ubicaciones de sitios de interés generados por usuarios.

Table 1. Conjunto de ubicaciones de usuarios

Característica	D31	Joensuu	MOPSI
No. de vectores	3100	6014	8589
No. de *clusters*	31	-	-
Dimensiones	2	2	2
Tamaño de archivo	49.5KB	110KB	155KB

Pese al tamaño de los CD, no fue posible conseguir uno que de forma real superara los recursos disponibles de RAM. Por esta situación, se introdujo a la aplicación un valor que simula que el tamaño de la memoria es menor al real, para poder trabajar con la división del conjunto de datos de acuerdo a la capacidad de RAM.

Cada uno de los CD se divide de acuerdo al tamaño asignado a la memoria volátil de la JVM. Por lo tanto se obtienen n subconjuntos de un tamaño no mayor al especificado. Cada uno de estos subconjuntos es procesado de forma iterativa hasta concluir con todo el CD. Los subconjuntos obtenidos para todos los conjuntos de datos fueron tres y los tamaños correspondientes son: D31=19.5312KB, Joensuu=48.4131KB y MOPSI=52.2461KB. Con los subconjuntos obtenidos se aplicó el algoritmo heurístico incremental para encontrar los grupos de datos que comparten características similares. Como ya se explicó en la sección 2, el algoritmo requiere del parámetro libre θ. Para fines de esta experimentación, los valores de los parámetros libres son: θ = 0.1, 0.2, 0.3, 0.5, 0.7, 0.9.

5. Resultados y discusión

Para fines de validación, a cada subconjunto obtenido se aplicó el método de validación cruzada [18] con 2 repeticiones, utilizando un 80% de los patrones para entrenamiento y el 20% restante para el conjunto de prueba, en tanto que para analizar los resultados se utilizó la precisión general, expresada por la siguiente ecuación (1).

$$PG = \frac{\sum_e x}{p}. \quad (1)$$

donde e = patrones correctamente clasificados y p = total de patrones presentados para su clasificación.

5.1. Clasificación

La Figura 2 muestra los resultados obtenidos al aplicar el algoritmo Batchelor y Wilkins, seleccionado por ser un algoritmo que no requiere conocer de antemano el número de grupos que deben ser encontrados, además de contar sólo con un parámetro libre que permite calcular de forma dinámica el umbral para saber si debe crear un nuevo grupo o no. Como valor de referencia, se incluyeron los resultados obtenidos

con el CD original, es decir ejecutando el algoritmo de agrupamiento sobre el CD cargado en su totalidad en la RAM. El algoritmo de clasificación base utilizado es la regla del vecino más cercano en su modalidad k-NN [19], siendo k=3 para todos los conjuntos de forma completa y on-line, ya que este es el valor que obtuvo la precisión más alta procesando el conjunto real con mas instancias de forma completa.

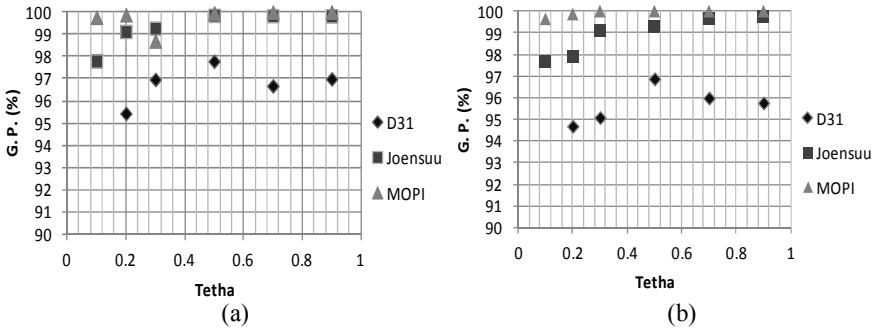

Fig. 2. Precisión general obtenida al procesar el conjunto con el método de *clustering* y la regla k-NN, para cada uno de los conjuntos de datos. (a) precisión para el conjunto completo, (b) la precisión de forma *on-line*.

En ambas gráficas de la Figura 2, se puede observar la precisión general obtenida para cada valor de θ, donde el eje X representa el valor del parámetro θ, y el eje Y el valor de la precisión general, finalmente, la serie indica el conjunto de datos procesado. En la Figura 2a puede observarse la PG obtenida para los conjuntos de datos completos, y en la Figura 2b el resultado del procesamiento por subconjuntos.

En estos resultados es posible observar varios aspectos, primeramente las precisiones obtenidas tanto con la metodología de procesamiento on-line, como las obtenidas utilizando el CD completo, sin importar el valor que tome θ son muy parecidas. En el caso del conjunto D31 procesado de forma completa se tiene una precisión general de 95.09%, mientras que al realizar el proceso on-line es de 93.43%. Para el conjunto Joensuu la precisión que se obtiene con el conjunto completo es en promedio del 99.28, y on-line es del 98.90%. Por último, el procesamiento on-line supera con 99.91% la precisión obtenida con el CD completo, que es de 99.72%.

Por otro lado, respecto al rendimiento del algoritmo de *clustering*, es posible observar que los valores de θ que mejor rendimiento ofrecen al procesar el CD completo son de 0.5 a 0.9, en tanto que para el aprendizaje on-line en su mayoría es de 0.7. Un caso particular son los resultados obtenidos con el CD MOPSI, pues con cualquier valor de θ ofrece una precisión entre 99% y 100%.

Finalmente, cuando se asigna el valor de $\theta = 0.1$, la precisión es menor con respecto a los demás valores. Esta situación puede observarse con claridad en el conjunto D31, cuya precisión general para dicho valor oscila entre 71 y 86%.

5.2. Tiempo de procesamiento

El tiempo requerido en el procesamiento de los CD es un factor importante a analizar, ya que la velocidad en procesamiento es uno de los objetivos que, adicional a la precisión, son perseguidos en minería de datos. En este sentido, el análisis se orienta a validar la propuesta de realizar la escalabilidad del algoritmo de *clustering* al realizar el aprendizaje on-line, respecto a la utilización del CD en su totalidad cargado en memoria.

La comparativa de los resultados de tiempo del tratamiento de los conjuntos se observan en la Figura 3. Esta figura muestra dos gráficas en las que en el eje de las X se incluyen los valores de θ, en tanto que en el eje de las Y se tiene el tiempo requerido para el procesamiento de cada uno de los CD utilizados. Este tiempo es medido en segundos.

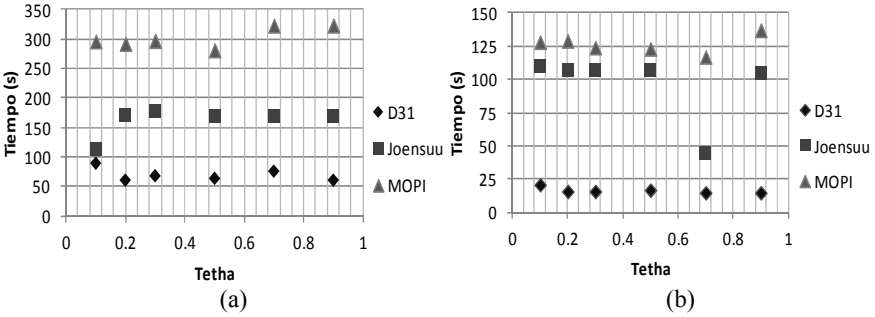

Fig. 3. Tiempo de procesamiento obtenido con cada uno de los CD. (a) Muestra la gráfica para el CD completo, (b) El tiempo por aprendizaje *on-line*.

En estos resultados es interesante observar que al realizar el procesamiento de los CD con aprendizaje on-line, el tiempo es significativamente menor respecto a cuándo se procesa el CD completo en memoria. Esta situación es sumamente ventajosa ya que al realizar la administración de la memoria se tiene un ahorro hasta del 50% con un rendimiento similar a si el CD hubiese estado residente en su totalidad en la RAM (Fig. 2).

Respecto a cada uno de los CD, es posible observar que el CD D31 es 22% más rápido de forma on-line que cuando se procesa el CD de forma completa. El conjunto Joensuu se procesa en promedio un 38% más rápido que de forma completa y el conjunto MOPSI procesado de forma on-line tuvo una disminución promedio del 58% de tiempo en contraposición a su tratamiento de forma completa.

El resultado de tiempo de procesamiento está estrechamente ligado a los parámetros libres del algoritmo de *clustering*. En este sentido, es posible notar que no siempre el valor de θ que arroja la mejor precisión procesa también el conjunto en menor tiempo. Cuando se procesa el CD completo, excepto con el conjunto Joensuu la mejor precisión y el menor tiempo son obtenidos con diferente valor de θ (0.5 y 0.2 respectivamente). Esta misma situación se presenta al realizar el procesamiento on-line, el CD que proporciona un mejor rendimiento con el mismo valor de θ en precisión y

tiempo, es MOPSI ($\theta = 0.3$). No obstante, estas diferencias son poco significativas ya que la precisión obtenida en su equivalente con el CD completo es muy similar en todos los casos. Por último, para todos los conjuntos de datos la diferencia entre los tiempos con respecto al mejor, oscila entre 5 y 15 segundos.

6. Conclusiones

En este artículo se propone la administración de la memoria para permitir la realización del aprendizaje on-line de patrones, dos aspectos fueron analizados, el rendimiento del clasificador y el tiempo de preprocesamiento. Los resultados obtenidos, muestran que, tanto en la simulación del aprendizaje on-line, como la utilización del CD residente en su totalidad en memoria, la precisión es muy similar. No obstante, la reducción en el tiempo requerido por el aprendizaje on-line, es significativamente inferior, siendo en la mayoría de los casos hasta en un 50%.

Esta situación podría estar ocasionada por la cantidad de patrones a procesar en cada momento, en el aprendizaje *on-line* es menor. Esto tiene que ver no sólo con la memoria, sino también con la cantidad de procesos que realiza el procesador. Por tanto, la administración de memoria y la adaptación de los grandes CD hacen que puedan ser procesados en menor de tiempo, sin perder precisión y aprovechando al máximo el recurso computacional. Respecto a los valores de θ se pudo observar que los mejores resultados se obtienen cuando su valor oscila entre 0.3 y 0.9.

Las líneas abiertas de estudio se orientan a ampliar la experimentación con otros CD, buscando la adquisición de CD que superen de manera real el recurso computacional disponible. Además de analizar la conveniencia de utilizar los agrupamientos realizados de forma *on-line* para construir *ensembles* de clasificadores.

Agradecimientos. Este trabajo fue realizado gracias al apoyo recibido del proyecto de la UAEM Generación y conteo de cubiertas de aristas en graficas acíclicas.

Referencias

1. Hernández O., J., Ramírez Q., M. J., Ferri R., C.: Introducción a la Minería de Datos. Person Educación, Madrid (2004)
2. Alppaydin, E.: *Introduction to Machine Learning* (2ª ed.). The MIT Press, Cambridge, MA (2010)
3. Aggarwal, C.: Framework for Clustering Massive-Domain Data Streams. En: IEEE 25th International Conference on Data Engineering, pp. 102-113. IEEE Press, New York (2009)
4. Khalilian, M., Mustapha, N.: Data Stream Clustering: Challenges and Issues. En: Proceedings of the International MultiConference of Engineers and Computer Scientists 1 (2010)
5. Choi, D.W., Chung, CW., Tao, Y.: A Scalable Algorithm for Maximizing Range Sum in Spatial databases. Proceedings of the VLDB Endowment, 5th edn. (2012)
6. Ferrer T., F. J., Aguilar R., J. S.: Aprendizaje Incremental de Reglas en Data Streams. Actas del III Taller Nacional de Minería de Datos y Aprendizaje, TAMIDA, pp. 261-270. Thomson, Sevilla (2005)

7. Mathieu, C., Sankur, O.: Online correlation clustering. Symposium on Theoretical Aspects of Computer Science, 573-584 (2010)
8. Mairal, J., Bach, F., Ponce, J., Sapiro, G.: Online Dictionary Learning for Sparse Coding. Proceedings of the 26th International Conference on Machine Learning. (2009)
9. Beringer, J., Hullermeyer E.: Online clustering of data streams. Technical Report 31, Department of Mathematics and Computer Science, Philipps-University Marburg, Germany (2003)
10. Marques de Sá P., J.: Pattern Recognition, Concepts, Methods and Applications. Springer, Alemania (2001)
11. Murty, M. N., & Devi, V. S.: Pattern recognition: An algorithmic approach. Springer (2012)
12. Cortijo, B. F. J.: Técnicas no supervisadas: Métodos de agrupamiento. Consultado en septiembre de 2012. Apuntes disponibles en la web de la asignatura [http://www-etsi2.ugr.es:8080/depar/ccia/RF0708/material.htm]
13. Lindhom, T., Yellin, F., Bracha, G., Buckley, A.: The Java Virtual Machine Specification. Java SE 7 Edition. Oracle America, Estados Unidos (2011)
14. Menchaca, M. R., García C. F. (2000). Arquitectura de la Máquina Virtual Java. Revista Digital Universitaria 1(2). Consultado en septiembre 2012, http://www.revista.unam.mx/vol.1/num2/art4/
15. Sun, Microsystems (2006). Memory Management in the Java HotSpot Virtual Machine. Sun Microsystems Inc.
16. Hyperic HQ, http://support.hyperic.com/display/SIGAR/Home
17. Speech and Image Processing Unit, Clustering datasets, http://cs.joensuu.fi/sipu/datasets/
18. Theodoridis, S., Koutroumbas K.: Pattern Recognition (4ª ed.). Elseiver Academic Press, Estados Unidos (2006)
19. Dasarathy, B.V.: Nearest Neighbor Norms: NN Pattern Clasification Techniques. IEEE Computer Society Press, Los Alamos, CA, (1991)