Enfoque MapReduce para el democratizado de métodos de selección de instancias

Alejandro González-Rogel, Álvar Arnaiz-González, Carlos López-Nozal, and José F. Diez-Pastor

Universidad de Burgos, España
gonzalezrogelalejandro@gmail.com,{alvarag,clopezno,jfdpastor}@ubu.es

Resumen El rápido crecimiento de las bases de datos actuales exige de métodos eficientes capaces de procesar este tipo de conjuntos de datos masivos. Si se analizan en detalle los algoritmos de preprocesamiento y los procesos de aprendizaje (clasificación, regresión, agrupamiento...), suelen presentar dificultades cuando se ejecutan sobre conjuntos de datos con un gran número de instancias.
Por ello, a la hora de enfrentarse a este tipo de problemas, la primera dificultad que se aprecia es la de encontrar una plataforma que sea capaz de soportar tal volumen de datos. La solución más intuitiva a ello es la paralelización. Si bien las arquitecturas paralelas de cómputo no son en absoluto novedosas, recientemente han surgido diversas metodologías y *frameworks* de trabajo que facilitan la tarea y liberan al programador de la tediosa gestión de planificación.
En este artículo se plantea la paralelización, siguiendo el paradigma de *MapReduce*, del algoritmo de selección de instancias *Democratic Instance Selection*, un método lineal de selección de instancias. Para evaluar las ventajas de esta implementación, se ha realizado un estudio de su eficiencia de procesamiento analizando los tiempos de ejecución con diferentes configuraciones: desde 4 hasta 512 nodos.

Keywords: big data, MapReduce, selección de instancias, democratic instance selection, clasificación

1 Introducción

En la actualidad, la gran cantidad de datos recopilados en diversos entornos hacen que las técnicas y herramientas que tradicionalmente se han venido utilizando en el ámbito de la minería de datos sean inaplicables. Con este problema, se ha acuñado un nuevo término: *big data*. Laney, a principios de siglo, describió las oportunidades y dificultades que aparecen a medida que aumenta el volumen, variedad y velocidad de los datos [11]. En este modelo, el volumen se refiere a la masiva recolección y generación de datos; variedad a los diversos formatos: semi-estructurados o no estructurados tales como vídeo, audio, web así como a los datos estructurados tradicionales. Por último, la velocidad pone de manifiesto que los datos deben ser tratados y procesados de manera ágil, para

© Ediciones Universidad de Salamanca

que su utilización y explotación tenga valor comercial [5]. Para la explotación de estos conjuntos de datos masivos nuevas metodologías han ido surgiendo en las últimas décadas, siendo MapReduce uno de los paradigmas de computación paralela más populares a día de hoy gracias a su robustez y transparencia de cara a la gestión de recursos [7].

Una aproximación directa al problema de grandes volúmenes de información es la reducción de los mismos mediante técnicas de preprocesado. Para disminuir el tamaño, un método comúnmente empleado es la selección de instancias[1], que consiste en escoger un subconjunto de ejemplos de la muestra completa que sea capaz de mantener la capacidad predictiva original [9]. El problema que surge en este punto, es que la mayoría de métodos de selección de instancias existentes tienen una complejidad de orden cuadrática o mayor.

En este artículo se presenta la adaptación del algoritmo *Democratic Instance Selection* (DIS) [9] al modelo *MapReduce* mediante el *framework* Apache Spark [14]. La organización del mismo es la siguiente: la Sección 2 presenta una introducción a las técnicas de selección de instancias, explicando en detalle el *Democratic Instance Selection*, la Sección 3 introduce el concepto de *MapReduce* y su aplicación concreta sobre DIS, la Sección 4 detalla los experimentos relacionados con la ejecución paralela del algoritmo que nos ocupa y, por último, la Sección 5 resume las conclusiones y líneas de trabajo futuras.

2 Selección de Instancias

Las técnicas de selección de instancias parten de una sencilla idea, seleccionar del conjunto de datos original un subconjunto de instancias que sea capaz de mantener, o incluso mejorar, la capacidad predictiva del conjunto original. Los beneficios de la reducción de tamaño provoca una reducción en el tiempo de clasificación (en los algoritmos perezosos) o en el tiempo de entrenamiento (en el caso de los algoritmos «entusiastas»[2]).

Desde su aparición, múltiples son los algoritmos que han ido surgiendo en la literatura, para el lector interesado recomendamos el trabajo de S. Garcia et al. [8]. No obstante, la mayor parte de estos algoritmos sufren de un problema que los hace difícilmente aplicables a conjuntos de datos masivos: su complejidad. La complejidad computacional de la mayoría de los citados métodos es cuadrática o mayor. Recientes algoritmos han sido desarrollados para superar esta serie de inconvenientes [1,4,6,9], entre ellos hemos destacado el *Democratic Instance Selection* [9] que es analizado en detalle en la siguiente sección.

[1] Selección de instancias, prototipos y ejemplos son términos utilizados de manera indistinta en el presente artículo.

[2] Traducciones de lazy-learning y eager-learning respectivamente, la diferencia entre ambos es que los primeros en la etapa de entrenamiento simplemente almacenan el conjunto de datos.

© Ediciones Universidad de Salamanca

2.1 Democratic Instance Selection

El algoritmo *Democratic Instance Selection* o DIS [9] se basa en la idea clásica «divide y vencerás». Su ejecución consiste en la realización de un número de rondas r donde se aplicará un algoritmo de selección de instancias sobre diferentes particiones del conjunto original. Para cada ronda el proceso consiste en dividir el conjunto original en un número de subconjuntos disjuntos (de aproximadamente el mismo tamaño). Sobre cada subconjunto se aplica un algoritmo de selección de instancias clásico de manera independiente. Las instancias seleccionadas por el algoritmo para ser eliminadas recibirán un voto. Entonces una nueva ronda comienza y el proceso se repite. Después de un número predefinido de rondas, las instancias que están por encima de un determinado umbral son eliminadas.

La mayor ventaja del DIS es la reducción del tiempo de ejecución gracias a su complejidad computacional lineal. Otro punto a su favor es la facilidad de adaptar el algoritmo en entornos paralelos, porque la ejecución de cada algoritmo sobre cada subconjunto seleccionado es independiente de los demás. Cabe destacar que, como el número de instancias de cada subconjunto es parametrizable, se puede elegir la carga que procesamiento de modo que sea acorde a las posibilidades del sistema empleado.

El pseudocódigo 1 presenta el algoritmo, el cual se divide en dos partes que serán analizadas en detalle a continuación:

- Particionado y votación: el conjunto de entrada es dividido en subconjuntos disjuntos y sobre cada uno de ellos se aplica el algoritmo deseado, esto se repite tantas veces como rondas hayan sido indicadas. Por cada algoritmo ejecutado, se almacenan los votos para su posterior procesado.
- Selección de instancias: el umbral de votación es calculado y son eliminadas todas aquellas instancias que lo hayan superado.

Algoritmo 1: Democratic Instance Selection (DIS)

Input: Conjunto de entrenamiento $T = \{(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_n, \mathbf{y}_n)\}$, tamaño de los subconjuntos s, número de rondas r
Output: Subconjunto seleccionado $S \subset T$

```
for k = 1 hasta r do
    Dividir las instancias en n_s conjuntos disjuntos t_i: ⋃_i t_i = T de tamaño s
    for j = 1 hasta n_s do
        Aplicar el algoritmo de selección de instancias a t_j
        Almacenar los votos de las instancias borradas de t_j
Obtener el umbral de votos, v, para borrar las instancias S = T
Borrar de S todas las instancias con un número de votos ⩾ v
return S
```

© Ediciones Universidad de Salamanca

Particionando el conjunto Una etapa importante del método es el particionado del conjunto inicial T en subconjuntos más pequeños t_i, quienes comprenden la totalidad del conjunto $\bigcup_i t_i = T$. El tamaño de los subconjuntos es elegido por el usuario y de él dependerá en buena parte el tiempo de ejecución; por lo que se debe procurar que cada subconjunto tenga, aproximadamente, el mismo número de instancias.

Para cada ronda del algoritmo se necesitan distintos subconjuntos. El método más simple para realizar el particionado sería una selección aleatoria en el cual cada instancia fuese asignada a uno u otro subconjunto de manera aleatoria. Este método presenta un problema principal, ya que las vecindades no son mantenidas de una ronda a la siguiente. Este es el motivo por el que los autores presentaron otra alternativa basada en la teoría del Grand Tour [2]. Por motivos de espacio no ha sido incluida una explicación detallada del método, aquellos lectores interesados pueden consultar la Sección 2.1.1 del artículo original [9].

Determinando el umbral de votos Otra decisión importante en este método es la selección del umbral de votos para borrar las instancias del conjunto solución. Los experimentos realizados por García-Osorio et al. en [9] demuestran que este valor depende del conjunto analizado. Por ello no es posible aplicar un valor preestablecido para cualquier conjunto. Por otra parte, se necesita un método que seleccione un valor directamente del conjunto inicial en tiempo de ejecución. Lo más sencillo, sería realizar una validación cruzada de los datos, aunque sería tremendamente costoso en tiempo.

El método elegido es mucho más ligero y estima el mejor valor del número de votos a partir del conjunto inicial. En la elección del umbral se tienen en cuenta dos criterios: el error de entrenamiento ϵ_t y los requerimientos de memoria o almacenamiento m. Ambos valores deben ser minimizados. Se define así un criterio, $f(v)$, el cual es la combinación de ambos tal y como se muestra en la siguiente ecuación:

$$f(v) = \alpha\epsilon_t(v) + (1-\alpha)m(v)$$

Donde m es el porcentaje de instancias preservadas por el algoritmo, ϵ_t es el error de entrenamiento y α es un parámetro en el intervalo $[0, 1]$ que representa la importancia relativa de cada uno de los dos valores anteriores. Los valores de la variable α puede ser seleccionado por el usuario en función de lo que desee minimizar. Para evitar el problema de tiempo en el cálculo del error, éste se estima utilizando un pequeño porcentaje del conjunto de entrenamiento completo, el cual puede ser según los autores: un $10\,\%$ para conjuntos grandes y un $0,1\,\%$ para conjuntos masivos.

El proceso para obtener el umbral es el siguiente: se almacenan los votos recibidos por cada instancia tras las r rondas realizadas. El umbral para saber si se debe eliminar una instancia será un valor $v \in [1, r]$. El valor de v se calcula minimizando el valor de la función $f(v)$. Una vez calculado, se eliminarán todas aquellas instancias que tengan el mismo número de votos o mayor que el umbral v.

© Ediciones Universidad de Salamanca

3 MapReduce

MapReduce surge como un paradigma de programación que aborda el problema del tratamiento de grandes conjuntos de datos desde la perspectiva de la computación paralela. Su funcionamiento se basa en el uso de pares ⟨clave, valor⟩ y en la división de los problemas en dos fases: *Map* y *Reduce*. La etapa *Map* opera sobre los pares ⟨clave, valor⟩ iniciales para conseguir otros valores ⟨clave, valor⟩ intermedios. Posteriormente, estos datos serán combinados según su clave en la fase Reduce para emitir un resultado final [7].

La tendencia a trabajar cada vez con un mayor número de datos, su alta tolerancia a fallos y transparencia de cara a la gestión de recursos han hecho que este paradigma se haya convertido en uno de los más utilizados en los últimos años [13].

Apache Hadoop y Apache Spark son dos de los *frameworks* más utilizados actualmente para el procesamiento masivo en entornos de *big data*. Ambos utilizan una arquitectura maestro-esclavo donde un único nodo es el encargado de gestionar un número variable de nodos esclavos (trabajadores). No obstante, Spark está diseñado para agilizar las operaciones iterativas mediante un uso intensivo de memoria en vez de disco. Esta característica le hace más indicado para las tareas de minería de datos y de aprendizaje automático, donde es capaz de multiplicar el rendimiento de Hadoop por 10 ó 100 veces [14].

3.1 Aplicación MapReduce en el algoritmo DIS

Para la adaptación del algoritmo *Democratic Instance Selection* a *big data*, se ha utilizado la técnica de paralelización *MapReduce*.

En lo que hace referencia a la estructura de los datos, usaremos pares ⟨votos, inst⟩ para operar a lo largo de las fases *Map* y *Reduce*. El valor ≪inst≫ corresponde a una instancia concreta, mientras que el parámetro ≪votos≫ hace referencia al número de veces que la instancia a la que acompaña no ha sido seleccionada durante la fase de votaciones. Este último valor será inicializado a cero para todas las instancias antes de comenzar el proceso.

En cuanto a la estructura del algoritmo, podemos ver dos ejemplos en la figura 1 y en el código 1. En la figura 1, se presenta un escenario de una posible ejecución donde se realizan 10 votaciones sobre n instancias y se calcula un umbral de 3 votos para realizar la operación *Reduce*. En ambos casos, podemos diferenciar las dos fases características del modelo *MapReduce*:

– **Fase Map:** Realiza las rondas de votaciones, actualizando el valor de la clave ≪votos≫ de nuestros pares en cada iteración (Pasos del 1 al 5 en el pseudocódigo 1 y línea 7 en el código 1). Previo a cada votación, se requiere de una operación *shuffle* que reorganiza los datos y los distribuye a lo largo de n_s particiones. Cada una de ellas es uno de los conjuntos disjuntos descritos en la línea 4 del pseudocódigo 1. En la implementación actual esta distribución de los datos es aleatoria, por lo que el valor de la clave (número de votos) de cada instancia no tiene influencia. Instancias con diferente clave pueden pertenecer

© Ediciones Universidad de Salamanca

o no a un mismo conjunto. El algoritmo de selección de instancias utilizado para realizar las votaciones ha sido el algoritmo *Condensed Nearest Neighbor* (CNN) [10], y es aplicado de manera independiente en cada partición.

– **Fase Reduce:** Comprende el cálculo del umbral de votos y la selección de aquellas instancias cuyos votos se encuentran por debajo de dicho umbral (Pasos del 6 al 7 en el pseudocódigo 1 y líneas de la 10 a la 12 del código 1). El propio cálculo del umbral requiere de la aplicación de algún algoritmo de clasificación. Si la clasificación se realizase con un algoritmo secuencial, se convertiría en un cuello de botella. Este es el motivo por el que se ha decidido utilizar la implementación paralela del algoritmo k-NN [13]. Tras determinar el umbral, todas aquellas instancias con un número de votos menor son seleccionadas para el conjunto editado.

```
1  override def instSelection(
2    originalData: RDD[LabeledPoint]): RDD[LabeledPoint] = {
3    // Añadimos una clave a todas las instancias
4    val dataAndVotes = originalData.map(inst => (0, inst))
5    // Operación Map
6    // Realizar votaciones
7    val ratedData = doVoting(dataAndVotes)
8    // Operación Reduce
9    // Cálculo del umbral de votos y selección de resultado
10   val (indBestCrit, bestCrit) =
         lookForBestCriterion(ratedData)
11   ratedData.filter(tuple => tuple._1 < indBestCrit)
12     .map(tuple => tuple._2)
13 }
```

Código 1. Código de ejecución de MR-DIS.

4 Configuración Experimental

En este trabajo se ha estudiado la escalabilidad del algoritmo *Democratic Instance Selection* utilizando el paradigma *MapReduce* sobre Spark. Para validar que el algoritmo funciona correctamente en términos de precisión (% de aciertos), se ha utilizado el algoritmo del vecino más cercano 1-NN (en su versión paralelizable para Spark [13]).

La implementación del DIS en Spark 1.6.1 ha sido realizada en Scala y se encuentra públicamente accesible en BitBucket [3].

4.1 Marco experimental

El estudio experimental se centra en verificar la *linealidad* del algoritmo implementado. Para ello se ha ejecutado con un número diverso de *trabajadores*: 4,

[3] Autor: Alejandro González-Rogel, `https://bitbucket.org/agr00095/tfg-alg.-seleccion-instancias-spark`.

© Ediciones Universidad de Salamanca

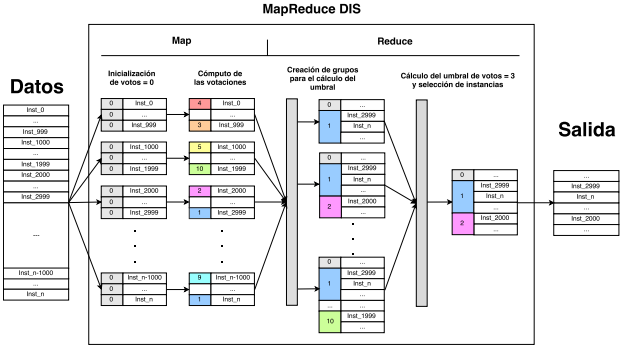

Figura 1. Ejemplo de ejecución del algoritmo DIS bajo el paradigma MapReduce.

8, 16, 32, 64, 128, 256 y 512. Para evaluar que el número de nodos no afecta al funcionamiento general del mismo se han extraído las siguientes medidas en una validación cruzada de 10 *folds*:

- Precisión media: Porcentaje de aciertos calculado sobre un clasificador 1-NN entrenado con el subconjunto seleccionado por el algoritmo MR-DIS sobre 9 de los *folds* y, como conjunto de test, la partición restante.
- Compresión media: Porcentaje de instancias del conjunto original presentes en el subconjunto seleccionado.

Para el algoritmo MR-DIS se han utilizado los siguientes parámetros: número de votaciones 10, número de instancias por partición 1000 y un porcentaje del 1 % del conjunto de datos original para el cálculo del error (durante la fase para determinar el umbral de votos [4]).

El conjunto de datos utilizado del repositorio UCI [12] ha sido Susy. Es un problema de clasificación binaria y está formado por 18 atributos. El conjunto original contiene 5,000,000 instancias pero en este estudio se ha escogido una partición aleatoria del 10 % del conjunto original, es decir 500,000 instancias. La razón por la que no se ha escogido el conjunto completo, es poder conseguir resultados de ejecución con un número pequeño de trabajadores en un tiempo aceptable. La estrategia de aprendizaje utilizada ha sido validación cruzada con diez grupos sin repetición. El porcentaje de aciertos del vecino más cercano sobre el conjunto de datos sin filtrar es del 68 %.

[4] En [9] el porcentaje de instancias usado para estimar el error en conjuntos masivos era del 0.1 %, pero el uso de una implementación paralela del 1-NN permite aumentar este porcentaje, mejorando la precisión de la estimación.

© Ediciones Universidad de Salamanca

Para realizar la experimentación se ha utilizado la distribución Apache Spark 1.6.1 sobre un clúster con 73 nodos Intel Xeon. Este cluster está configurado con un sistema de gestión de colas de trabajo PBS (*Portable Batch System*), por lo que para la ejecución de Spark se ha utilizado un script de lanzamiento de la Universidad de Ohio [3].

En los experimentos, cada trabajador o unidad ejecutora (*executor*) dispone de un core y 1024 MB de RAM. Las colecciones de datos utilizadas por Spark (RDDs) fueron almacenadas bajo el nivel de persistencia `MEMORY_ONLY` del *framework*. Esto implica que los datos son almacenados sin serializar en memoria y, en caso de no haber espacio, algunas particiones de estas estructuras pueden ser eliminadas y recalculadas cuando vuelvan a ser necesarias.

La tabla 1 muestra los resultados de la ejecución sobre el conjunto de datos Susy. El tiempo de ejecución se muestra, adicionalmente, en la figura 2. Para visualizar la tendencia lineal esperada del algoritmo se ha seleccionado como línea base el tiempo de la ejecución con 4 trabajadores. Este tiempo es divido a la mitad a medida que se duplica su número. Se observa como, a medida que aumenta el número de trabajadores, el tiempo disminuye a la mitad hasta llegar a un límite en 32. A partir de ahí la reducción de tiempo es inferior a la que cabría esperar. Esto se debe a que el conjunto de datos Susy no es lo suficientemente grande como para poder dar trabajo a todos los nodos disponibles y parte de la potencia de la paralelización no se está aprovechando. Recordar que el subconjunto de Susy utilizado tiene 500,000 instancias y cada partición contiene aproximadamente 1,000. El número máximo de trabajadores con el que se ha probado es 512, puesto que el tiempo deja de disminuir y comienza a aumentar, debido a la sobrecarga de comunicaciones entre los nodos.

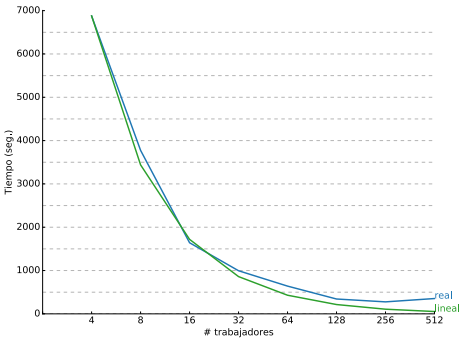

Figura 2. Tiempo de ejecución del conjunto de datos Susy. Se muestra el tiempo de filtrado así como la progresión lineal tomando como línea base 4 trabajadores.

© Ediciones Universidad de Salamanca

Cabe mencionar que las ligeras variaciones en reducción y precisión que pueden apreciarse en la tabla 1, se deben al indeterminismo que genera una ejecución paralela. Esto es, no se conoce el orden en el que se ejecutan las operaciones y, ésto, puede influir en la fase de particionado y votación.

Tabla 1. Resultados de la experimentación con Susy.

Nº de trabajadores	Reducción (%)	Precisión (%)	T. filtrado (s)
4	21.79	66.58	6877.92
8	21.80	66.61	3774.22
16	21.81	66.56	1641.45
32	21.81	66.52	995.02
64	21.80	66.52	640.44
128	21.80	66.53	342.74
256	21.78	66.61	276.70
512	21.79	66.56	353.64

5 Conclusiones y Líneas Futuras

En este trabajo se ha presentado una implementación del algoritmo de selección de instancias *Democratic Instance Selection* en Spark siguiendo el paradigma de *MapReduce*. La complejidad lineal en tiempo de ejecución del DIS, así como la facilidad para paralelizar dicho algoritmo, han sido las motivaciones necesarias para llevar a cabo un estudio de rendimiento.

La etapa de desarrollo ha demostrado la capacidad de Spark para distribuir el trabajo en diversas máquinas sin necesidad de realizar ese trabajo de manera manual por el programador. El trabajo del desarrollador es implementar el algoritmo siguiendo el paradigma *MapReduce* y, a partir de ahí, Spark asigna la carga de trabajo distribuyendo las tareas. No obstante, los problemas que plantea la opacidad de Spark en la depuración y la solución de errores es más que notable.

En el aspecto técnico, se quiere destacar la importancia de una implementación nativa de Spark. La imposibilidad de la instalación en el clúster donde se realizaron los experimentos, obligó el utilizar Spark sobre Torque mediante un script de lanzamiento de la Universidad de Ohio. Pese a que Spark es agnóstico y permite realizar esta tarea según sus especificaciones, la realidad es que no funciona de un modo fiable y la pérdida de comunicación entre nodos fue un problema serio durante la realización de la experimentación.

Pese a que en esta primera versión sólo se ha implementado y probado un algoritmo de selección de instancias, el proyecto inmediato en el que estamos trabajando es la adición de nuevos algoritmos, más rápidos y eficientes: ICF, DROP, LSBo...

© Ediciones Universidad de Salamanca

Por otro lado, el método de particionado aleatorio no es el más inteligente. La adaptación del *Grand Tour* a su ejecución bajo el paradigma *MapReduce* es una de las líneas de investigación que planteamos para el futuro.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Economía y Competitividad, proyecto TIN2015-67534-P.

Referencias

1. F. Angiulli and G. Folino. Distributed Nearest Neighbor-Based Condensation of Very Large Data Sets. *IEEE Transactions on Knowledge and Data Engineering*, 19(12):1593–1606, Dec 2007.
2. Daniel Asimov. The grand tour: A tool for viewing multidimensional data. *SIAM J. Sci. Stat. Comput.*, 6(1):128–143, 1985.
3. Troy Baer, Paul Peltz, Junqi Yin, and Edmon Begoli. Integrating apache spark into PBS-Based HPC environments. In *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*, page 34. ACM, 2015.
4. José Ramón Cano, Francisco Herrera, and Manuel Lozano. Stratification for scaling up evolutionary prototype selection. *Pattern Recognition Letters*, 26(7):953 – 963, 2005.
5. Min Chen, Shiwen Mao, and Yunhao Liu. Big Data: A Survey. *Mobile Networks and Applications*, 19(2):171–209, 2014.
6. Aida de Haro-García and Nicolás García-Pedrajas. A divide-and-conquer recursive approach for scaling up instance selection algorithms. *Data Mining and Knowledge Discovery*, 18(3):392–418, 2009.
7. Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51(1):107–113, January 2008.
8. S. Garcia, J. Derrac, J.R. Cano, and F. Herrera. Prototype Selection for Nearest Neighbor Classification: Taxonomy and Empirical Study. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(3):417–435, March 2012.
9. César García-Osorio, Aida de Haro-García, and Nicolás García-Pedrajas. Democratic instance selection: A linear complexity instance selection algorithm based on classifier ensemble concepts. *Artificial Intelligence*, 174(5–6):410 – 441, 2010.
10. P. Hart. The condensed nearest neighbor rule (corresp.). *Information Theory, IEEE Transactions on*, 14(3):515 – 516, may 1968.
11. Doug Laney. 3-d data management: controlling data volume, velocity and variety. Technical report, META Group Research Note, Februrary 2001.
12. M. Lichman. UCI Machine Learning Repository. *University of California, Irvine, School of Information and Computer Sciences*, 2013. `http://archive.ics.uci.edu/ml`.
13. Jesús Maillo, Isaac Triguero, and Francisco Herrera. Un enfoque MapReduce del algoritmo k-vecinos más cercanos para Big Data. In José Miguel Puerta, José A. Gámez, Bernabé Dorronsoro, Edurne Barrenechea, Alicia Troncoso, Bruno Baruque, and Mikel Galar, editors, *Actas de la XVI Conferencia de la Asociación Española para la Inteligencia Artificial, CAEPIA 2015*, pages 969–978, Nov. 2015.
14. Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster Computing with Working Sets. *HotCloud*, 10:10–10, 2010.

© Ediciones Universidad de Salamanca