Un enfoque MapReduce del algoritmo k-vecinos más cercanos para Big Data

Jesús Maillo[1], Isaac Triguero[2,3], and Francisco Herrera[1]

[1] Dept. de Ciencias de la Computación e Inteligencia Artificial, CITIC-UGR. Universidad de Granada. 18071, Granada, España
jesusmh@correo.ugr.es, herrera@decsai.ugr.es
[2] Department of Respiratory Medicine. Ghent University, 9000, Ghent, Belgium.
[3] Data Mining and Modelling for Biomedicine group, VIB Inflammation Research Center, 9052, Ghent, Belgium
Isaac.Triguero@irc.vib-UGent.be

Resumen El clasificador de los k-vecinos más cercanos es uno de los métodos más conocidos en minería de datos debido a su eficacia y sencillez. Sin embargo, por su funcionamiento, éste no puede abordar problemas con un elevado número de muestras si el tiempo de ejecución se considera relevante. En la actualidad, la clasificación de grandes cantidades de datos se está convirtiendo en una tarea indispensable. Este problema se conoce con el nombre de *big data*, en el que las técnicas estándar de minería de datos no pueden hacer frente al volumen de datos. En esta contribución se propone un enfoque basado en MapReduce para el clasificador de los k-vecinos más cercanos. Este modelo nos permite clasificar al mismo tiempo una gran cantidad de casos desconocidos (ejemplos de prueba) sobre un gran conjunto de datos (entrenamiento). Para ello, la fase de *map* determinará los k-vecinos más cercanos en las distintas particiones de los datos. Después, la fase de *reduce* calculará los vecinos definitivos de la lista obtenida en la fase de *map*. Este modelo permite escalar con conjuntos de datos de tamaño arbitrario, simplemente añadiendo más nodos de cómputo en caso de ser necesario. Además, esta implementación obtiene la tasa de clasificación exacta al clasificador original.

Palabras clave: K-vecinos más cercanos, clasificación, MapReduce, aprendizaje perezoso, big data

1. Introducción

La clasificación de grandes conjuntos de datos (*big data*) se está convirtiendo en una tarea esencial en muchos ámbitos como la biomedicina, las redes sociales, el marketing, etc. Los avances recientes en la recopilación de datos traen consigo un aumento inexorable de los datos a manejar. El volumen, diversidad y complejidad de dichos datos dificulta los procesos de análisis y extracción del conocimiento [1], así, los modelos de minería de datos estándar necesitan ser re-diseñados para funcionar adecuadamente.

El algoritmo de los k-vecinos más vercanos (*k-Nearest Neighbor*, k-NN) [2] pertenece a la familia del aprendizaje perezoso, donde no se necesita fase de

entrenamiento. Los datos de partida se encuentran almacenados, y los datos de entrada se clasifican buscando las k muestras más cercanas. Para determinar cómo de cerca están dos ejemplos pueden utilizarse distintas distancias o medidas de similitud. Este cálculo debe realizarse para todos los ejemplos de entrada contra todo el conjunto de datos de entrenamiento. Por lo tanto, el tiempo de respuesta se ve comprometido cuando se aplica en el contexto de *big data*.

Las recientes tecnologías basadas en la nube ofrecen un entorno ideal para dar solución a este problema. El esquema MapReduce [3] destaca como un paradigma de programación simple y robusto con la capacidad de afrontar grandes conjuntos de datos en un grupo de nodos de cómputo (*cluster*). Su aplicación es muy apropiada en la minería de datos gracias a su mecanismo de tolerancia a fallos y su facilidad de uso [4], a diferencia de otros esquemas de paralelización como *Message Passing Interface* [5]. En los últimos años, diversas técnicas de minería de datos se han adaptado satisfactoriamente mediante el uso de este paradigma, como se muestra en [6], [7]. Algunos trabajos relacionados utilizan MapReduce para búsquedas similares a k-NN. Por ejemplo, en [8] y [9] los autores utilizan consultas kNN-join dentro de un proceso MapReduce.

En este trabajo se presenta un nuevo algoritmo paralelo de k-NN basado en MapReduce para clasificación en *big data*. La fase *map* consiste en desplegar el cómputo de similitud entre los ejemplos de prueba y el conjunto de entrenamiento a través de un *cluster* de nodos de computación. Como resultado de cada *map*, los k vecinos más cercanos, junto con su valor de distancia son enviados a la etapa de *reduce*. La fase *reduce* determinará cuáles son los k vecinos más cercanos finales de la lista proporcionada por los *maps*. En este escrito, se denota este enfoque como MR-kNN. Para probar el rendimiento de este modelo, los experimentos realizados se han llevado a cabo en un conjunto de datos con hasta un millón de datos. El estudio experimental incluye un análisis de la precisión, tiempo de ejecución y ganancia de tiempo o aceleración (Speed up). Por otro lado, se estudiarán distintos valores de k y número de procesos *map*.

El documento está estructurado de la siguiente manera. La Sección 2 proporciona información básica sobre el algoritmo k-NN y MapReduce. La Sección 3 expone la propuesta. La Sección 4 define el estudio experimental y presenta los resultados. Por último, la Sección 5 concluye el trabajo.

2. Preliminares

En esta sección se repasa algunos componentes preliminares utilizados. La Sección 2.1 presenta el algoritmo k-NN así como sus debilidades para tratar con la clasificación de grandes conjuntos de datos. La Sección 2.2 describe el paradigma MapReduce y la implementación utilizada en este trabajo.

2.1. k-NN y sus debilidades frente a problemas big data

El algoritmo k-NN es un método no paramétrico que se puede utilizar tanto para tareas de clasificación como de regresión. En esta sección se define k-NN, sus

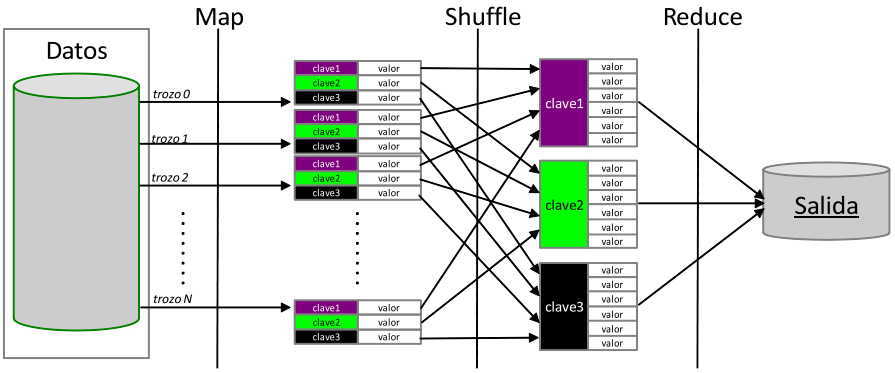

Figura 1: Flujo de trabajo del paradigma MapReduce

tendencias actuales y los inconvenientes para trabajar con *big data*. Formalmente, el k-NN se podría definir como:

Sea CE un conjunto de datos de entrenamiento y CP un conjunto de prueba, que están formados por n y t muestras respectivamente. Cada muestra $\mathbf{x}_p$ es una tupla ($\mathbf{x}_{p1}$, $\mathbf{x}_{p2}$, ..., $\mathbf{x}_{pD}$, ω), donde $\mathbf{x}_{pf}$ es el valor de la f-ésima característica de la p-ésima muestra. Esta muestra pertenece a la clase ω, dada por $\mathbf{x}_p^{\omega}$, y un espacio de dimensión D. Para el conjunto CE la clase ω es conocida, mientras que para el conjunto CP es desconocida. Por cada muestra $\mathbf{x}_{prueba}$ contenida CP, el modelo k-NN busca las k muestras más cercanas en el conjunto CE. Para hacer esto, calcula la distancia entre x_{prueba} y todos los ejemplos de CE. La distancia Euclidea suele ser la más utilizada. Las k muestras ($vecino_1$, $vecino_2$, ..., $vecino_k$) que van a decidir la clasificación son elegidas por su distancia más pequeña entre todo el conjunto CE. Con estos vecinos se calcula la clase predicha por mayoría de voto. El valor k influye en el rendimiento y tolerancia a ruido de la técnica.

A pesar de los prometedores resultados mostrados por k-NN en una amplia variedad de problemas, este carece de escalabilidad para trabajar con grandes conjuntos CE. Los principales problemas son:

- Tiempo de ejecución: La complejidad del algoritmo de k-NN es $O((n \cdot D))$, donde n es el número de instancias y D el número de características.
- Memoria consumida: Para un cálculo veloz de las distancias, el modelo de k-NN necesita almacenar los datos de entrenamiento en memoria. Cuando CE es demasiado grande, podría superar fácilmente la memoria RAM disponible.

Estos inconvenientes motivan el uso de tecnologías *big data* para distribuir el procesamiento de k-NN sobre un *cluster* de nodos. Este trabajo, se centrará en la reducción del tiempo de ejecución de acuerdo al número de datos.

2.2. MapReduce

MapReduce es un paradigma de programación paralela muy popular [3] desarrollado para procesar y/o generar conjuntos de datos grandes que no se ajustan a la capacidad de memoria física. Caracterizado por su transparencia para los programadores, se basa en la programación funcional y trabaja en dos pasos principales: la fase *map* y la fase de *reduce*. Cada fase tiene una pareja clave-valor ($< clave; valor >$) como entrada y como salida. Lo único que el programador debe desarrollar son estas fases. La fase *map* toma cada pareja $< clave; valor >$ y genera un conjunto intermedio de pares $< clave; valor >$. Entonces, MapReduce fusiona todos los valores asociados con la misma clave en una lista (etapa *shuffle*). La fase *reduce* toma esta lista como entrada y produce los valores finales.

La Figura 1 presenta un esquema de flujo del paradigma MapReduce. En una aplicación MapReduce, en primer lugar, todas las funciones *map* se ejecutan de forma independiente y paralela. Mientras tanto, las operaciones *reduce* esperan hasta que la fase *map* ha terminado. Entonces, estos procesan las diferentes claves al mismo tiempo y de forma independiente. Cabe destacar que las entradas y salidas de un proceso MapReduce se almacenan en un sistema de archivos distribuido que es accesible desde cualquier otra máquina del *cluster* utilizado.

Dependiendo de la arquitectura del *cluster* disponible, pueden utilizarse distintas implementaciones de MapReduce. En este trabajo se utiliza Hadoop[4] por su rendimiento, ser código abierto y su facilidad de instalación y sistema distribuido de archivos (*Hadoop Distributed File System*, HDFS).

Desde el punto de vista del programador, la implementación de MapReduce de Hadoop divide el ciclo de vida de las tareas *map/reduce* como: (1) configuración, (2) operación en sí (*map* o *reduce*) y (3) *cleanup*. El método *cleanup* se puede utilizar para limpiar cualquier recurso que pueda haber asignado.

Un *cluster* Hadoop sigue una arquitectura maestro-esclavo, donde un nodo maestro gestiona un número arbitrario de nodos esclavos. HDFS replica los ficheros de datos en múltiples nodos de almacenamiento, para acceder a estos simultáneamente. Hadoop proporciona un mecanismo de tolerancia a fallos, de modo que si un nodo falla, Hadoop reinicia automáticamente la tarea encargada a dicho nodo en otro activo. En cuanto a la minería de datos, los proyectos como Apache Mahout[5] y la biblioteca MLlib de Apache Spark[6] recopilan algoritmos distribuidos y escalables de aprendizaje automático implementados con MapReduce y otras extensiones como MapReduce iterativo.

3. MR-kNN: Una implementación MapReduce para k-NN

En esta sección se explica cómo paralelizar el algoritmo k-NN sobre MapReduce. El cómputo se organiza en dos operaciones principales: la fase *map* y la

[4] "Apache Hadoop Project" 2015 [Online] Available: http://hadoop.apache.org/
[5] "Apache Mahout Project," 2015 [Online] Available: http://mahout.apache.org/
[6] "Apache spark Project," 2015 [Online] Available: http://spark.apache.org/

fase *reduce*. La fase *map* calculará los k vecinos más cercanos de cada ejemplo del conjunto CP en las diferentes particiones de CE, guardando su distancia y clase. La etapa *reduce* procesará las distancias de los k candidatos de cada *map* y creará la lista definitiva de vecinos. Posteriormente, se calcula la clase predicha con el procedimiento de mayoría de votos. En las Secciones 3.1 y 3.2 se detallan las fases *map* y *reduce*, presentando un esquema del modelo paralelo.

3.1. Fase map

Sean CE un conjunto de entrenamiento y CP un conjunto de prueba de tamaño arbitrario almacenados en HDFS como ficheros independientes. La fase *map* divide el conjunto CE en un número subconjunto de particiones. En Hadoop, los ficheros están formados por h bloques HDFS accesibles desde cualquier máquina del *cluster*. Sea m el número de procesos de *map*, cada tarea *map* $(Map_1, Map_2, \ldots, Map_m)$ creará un CE_j, donde $1 \leq j \leq m$, con la muestra de cada partición en el que el archivo de conjunto de entrenamiento es dividido. Este proceso de partición se realiza secuencialmente, de modo que, el proceso Map_j corresponde a la partición de datos j del bloque HDFS h/m. Como resultado, cada *map* analiza aproximadamente el mismo número de muestras de entrenamiento.

La división de los datos en varios subconjuntos, y su análisis de forma individual, se adapta a la filosofía de paralelización de MapReduce mejor que a otras por dos razones: En primer lugar, cada subconjunto es procesado de forma independiente, por lo que no necesita ningún tipo de intercambio de datos entre los nodos. En segundo lugar, el coste computacional de cada trozo podría ser tan alto que se requiere de un mecanismo de tolerancia a fallos.

Dado que el objetivo es obtener una implementación exacta del algoritmo k-NN, el conjunto CP no se dividirá para tener acceso a todas las muestras de prueba en todos los *maps*. Así, cuando cada *map* ha formado su correspondiente conjunto CE_j se calcula la distancia de cada $\mathbf{x}_p$ contra las muestras de CE_j. Se guarda la distancia y la etiqueta de clase de los vecinos más cercanos para cada ejemplo de prueba. Como resultado se obtiene una matriz CD_j de parejas $< clase, distancia >$ con dimensión $n \cdot k$. Por lo tanto, en la fila i, está la distancia y la clase de los k vecinos más cercanos. Es de destacar que cada fila será ordenada de forma ascendente con respecto a la distancia al vecino.

Para evitar problemas de restricción de la memoria, el fichero de ejemplos es leído linea por línea. Así se evita cargar en memoria el archivo de ejemplos de prueba completo. Funciones primitivas de Hadoop nos permiten hacerlo sin perder eficiencia. El Algoritmo 1 expone el pseudo-código de la función *map*. Cuando finaliza cada *map*, los resultados de su cálculos son enviados a un solo proceso *reduce*. La *clave* de salida de cada *map* es el identificador de valor de cada proceso *map*.

Algorithm 1 Función Map

Require: CP; k
1: Formar CE_j con las instancias de la partición j.
2: **for** $i = 0$ **hasta** $i <$ tamaño(CP) **do**
3: Calcular k-NN ($\mathbf{x}_{prueba,i}$,CE_j,k)
4: **for** $n = 0$ **hasta** $n < k$ **do**
5: $CD_j(i,n) =< Clase(vecino_n), Distancia(vecino_n) >_i$
6: **end for**
7: **end for**
8: $clave = idMapper$
9: EMIT($< clave, CD_j >$)

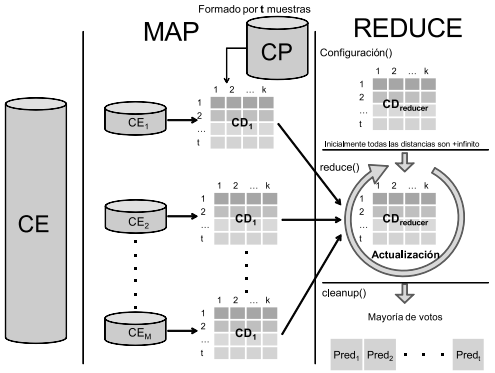

Figura 2: Diagrama de flujo del algoritmo MR-kNN

3.2. Fase reduce

El objetivo de la fase *reduce* consiste en seleccionar entre los vecinos propuestos en cada *map* como más cercanos, cuales son los más cercanos para el conjunto completo CE. Teniendo en cuenta que nuestro objetivo es diseñar un modelo que pueda escalar para conjuntos de entrenamiento de tamaño arbitrario independientemente del número de vecino, se implementó cuidadosamente esta operación utilizando la configuración, *reduce* y *cleanup* de *reduce*. (Introducido en la Sección 2.2). El Algoritmo 2 describe el funcionamiento *reduce* y el Algoritmo 3 la fase *cleanup*. Una explicación detallada es la siguiente:

- **Configuración:** además de la lectura de los parámetros del objeto de configuración de Hadoop, esta operación asignará un matriz clase-distancia de tamaño fijo $CD_{reducer}$ (tamaño$(CP) \cdot k - vecinos$). Como requisito, esta función conocerá el tamaño del CP, pero no necesita leer el conjunto. Esta matriz se inicializa con valores aleatorios para la clase y positivo infinito para las distancias. Esta operación solo se realiza la primera vez que se ejecuta *reduce*. El proceso *reduce* comienza a recibir datos conforme los procesos *map* van finalizando.
- **Reduce:** cuando la fase *map* acaba, la matriz de clases y distancias $CD_{reducer}$ se actualiza comparando con los valores actuales de distancia y los que llegan

de cada Map_j, es decir, CD_j. Debido a que las matrices resultantes de los *maps* vienen ordenados, el proceso de actualización es más rápido. Consiste en fusionar ambas listas ordenadas para tener los k más cercanos (complejidad $O(k)$). Así, para cada ejemplo de prueba $\mathbf{x}_{prueba}$, se compara cada valor de la distancia de los vecinos uno a uno, comenzando por el vecino más cercano. Si la distancia es menos que el valor actual, la clase y la distancia de esta posición de la matriz se actualiza con los valores correspondientes, de lo contrario se continúa iterando (instrucciones 3-6 del Algoritmo 2). A diferencia de la operación de configuración, esta se ejecuta una vez por cada clave existente. Por lo tanto, podría ser interpretado como un procedimiento iterativo al que se agregan los resultados proporcionados por los *map*.

Algorithm 2 Función Reduce

Require: tamaño(TS), k, CD_j. Puesta en marcha de la configuración.
1: **for** $i = 0$ **hasta** $i <$ tamaño(TS) **do**
2: Actualizar $CD_{reducer}(i)$ con $CD_j(i)$
3: cont=0
4: **for** $n = 0$ **hasta** $n < k$ **do**
5: **if** $CD_j(i, cont).Distancia < CD_{reducer}(i, n).Distancia$ **then**
6: $CD_{reducer}(i, n) = CD_j(i, cont)$
7: cont++
8: **end if**
9: **end for**
10: **end for**

- **Cleanup:** tras la fase *reduce*, $CD_{reducer}$ contendrá la lista definitiva de los vecinos (clase y distancia) de todos los ejemplos de CP. Entonces, se determina la clase predicha mediante votación, como suele hacerlo k-NN. Como resultado, las clases predichas para todo el conjunto CP se proporcionan como la salida final de la etapa *reduce*. La *clave* se establece como el número de instancia (instrucción 3 en el Algoritmo 3).

Algorithm 3 Función Cleanup del Reduce

Require: tamaño(TS), k. Función reduce finalizada
1: **for** $i = 0$ **to** $i <$ tamaño(TS) **hasta: do**
2: $PredClase_i = VotoMayoria(Clases(CD_{reduce}))$
3: $clave = i$
4: Salida: ($< clave, PredClase_i >$)
5: **end for**

MR-kNN sólo utiliza un único *reduce*. El uso de un solo *reduce* se debe a reducir la sobrecarga de MapReduce relacionada con la red. También nos permite obtener un único archivo de salida. Como resumen, la Figura 2 representa un diagrama de flujo que contiene los pasos principales del modelo propuesto.

4. Estudio experimental

En esta sección se presentan todas las cuestiones planteadas en el estudio experimental. La Sección 4.1 determina el marco de los experimentos y la Sección 4.2 expone y analiza los resultados obtenidos.

4.1. Marco experimental

Se tendrán en cuenta las siguientes medidas para evaluar el rendimiento de la técnica propuesta:

- *Precisión:* contabiliza el número de clasificaciones correctas en relación con el número total de muestras.
- *Tiempo de ejecución:* se anotará el tiempo dedicado por MR-kNN en cada *map* y en el *reduce*, así como el tiempo global para la clasificación del conjunto completo de prueba CP. El tiempo total incluye los cálculos realizados por el framework MapReduce (comunicaciones).
- *Speed up:* el cociente entre el tiempo de ejecución paralelo y el tiempo de la versión secuencial.

El estudio experimental se centra en el análisis del efecto del número de *maps* (16, 32, 64, 128 y 256) y el número de vecinos (1, 3, 5 y 7) en del modelo propuesto MR-kNN. El conjunto de datos utilizado es PokerHand, tomado del repositorio UCI [10]. Contiene un total de 1.025.010 datos, 10 características y 10 clases diferentes. Se seguirá el esquema de validación cruzada en 5 particiones.

Los experimentos se han llevado a cabo en un *cluster* formado por dieciséis nodos: uno maestro y quince de cómputo. Cada uno de los nodos de cómputo tiene un procesador Intel Core i7 4930, 6 núcleos por procesador a 3.4 Ghz y 64 GB de memoria RAM. La tarjeta de red es Ethernet 1 Gbps. En términos de software, se ha utilizado la distribución de código abierto de Cloudera de Apache Hadoop (Hadoop 2.5.0-cdh5.3.2). Debe tenerse en cuenta que el número de *maps* que pueden ejecutarse en paralelo está configurado a 128, por lo que los experimentos con 256 *maps* no pueden obtener un *speed up* lineal.

4.2. Resultados obtenidos y análisis

En esta sección se presenta y analiza los resultados obtenidos en el estudio experimental. En primer lugar, se centra en los resultados de la versión secuencial de k-NN como nuestra referencia base. El Cuadro 1 recoge la precisión promedio (PrePrueba) en las particiones de prueba y el tiempo de ejecución (en segundos) de la versión estándar de k-NN para distintas cantidades de vecinos. La Figura 3 presenta el *speed up* frente al número de *maps* para k igual a 1, 3, 5 y 7.

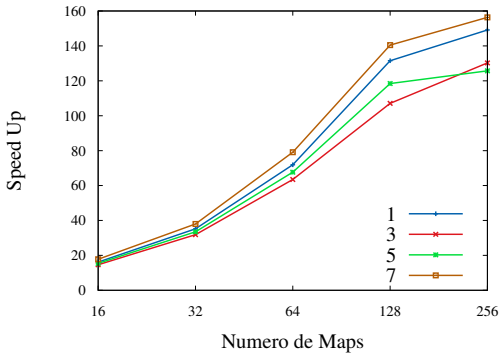

Figura 3: Speed up

Vecinos	PrePrueba	TieEje(s)
1	0,5019	105.475,0060
3	0,4959	105.507,8470
5	0,5280	105.677,1990
7	0,5386	107.735,0380

Cuadro 1: k-NN secuencial

El Cuadro 2 resume los resultados obtenidos por el enfoque propuesto. Muestra, para cada número de vecino y número de *maps*, el tiempo medio empleado para la fase *map* (TieMedMap), el tiempo medio consumido por el *reduce* (TieMedRed), el tiempo medio total (TieMedTotal), la precisión media obtenida (PrePrueba) y el *speed up* (SpeedUp).

Cuadro 2: Resultados obtenidos por el algoritmo MR-kNN

#Vecinos	#Maps	TieMedMap	TieMedRed	TieMedTotal	PrePrueba	SpeedUp
1	256	356,3501	9,5156	709,4164	0,5019	149,1014
	128	646,9981	5,2700	804,4560	0,5019	131,4864
	64	1.294,7913	3,1796	1.470,9524	0,5019	71,9092
	32	2.684,3909	2,1978	3.003,3630	0,5019	35,2189
	16	5.932,6652	1,6526	6.559,5666	0,5019	16,1253
3	256	351,6059	20,3856	735,8026	0,4959	130,2356
	128	646,4311	11,0798	894,9308	0,4959	107,0783
	64	1.294,3999	6,3078	1.509,5010	0,4959	63,4830
	32	2.684,9437	3,9628	3.007,3770	0,4959	31,8642
	16	5.957,9582	2,4302	6.547,3316	0,4959	14,6361
5	256	371,1801	33,9358	793,4166	0,5280	125,7262
	128	647,6104	17,9874	842,3042	0,5280	118,4291
	64	1.294,6693	10,0790	1.474,5290	0,5280	67,6509
	32	2.679,7405	5,6878	2.977,1442	0,5280	33,5064
	16	5.925,3548	3,5698	6.467,3044	0,5280	15,4242
7	256	388,0878	49,3472	746,2892	0,5386	156,3386
	128	647,0609	23,6258	830,7192	0,5386	140,4492
	64	1.295,0035	12,8888	1.475,3190	0,5386	79,0838
	32	2.689,1899	7,3508	3.069,3328	0,5386	38,0128
	16	5.932,6652	1,6526	6.559,5666	0,5386	17,7868

A partir de estos Cuadros y Figuras se destacan varios factores:

- Como se puede observar en el Cuadro 1, el tiempo de ejecución de k-NN secuencial es bastante elevado. Sin embargo, utilizando el enfoque propuesto, se muestra una gran reducción de tiempo de ejecución cuando aumenta el número de *maps*. La propuesta siempre proporciona la misma precisión que el modelo original k-NN, independientemente del número de *maps*. Un mayor número de *maps* implica un cálculo ligeramente mayor en la fase *reduce* puesto que agrega más resultados llegados desde los *maps*.
- En el problema considerado, valores más altos de k obtienen mejores resultados en precisión. En términos de tiempo de ejecución, valores de k más grandes aumentan ligeramente el tiempo de cálculo en ambas versiones. Esto se traduce a grandes matrices CD_j. Sin embargo, debido a la codificación utilizada, no da lugar a grandes tiempos de ejecución.
- De acuerdo con la Figura 3, se ha logrado un *speed up* linear en todos los casos, salvo para 256 *maps*. Como se ha indicado, este caso supera el número máximo de tareas *map* que pueden ejecutarse simultáneamente. Por otra

parte, existe alguna ganancia superior a la lineal que podría deberse a problemas de memoria en la versión secuencial.

5. Conclusiones

En esta contribución se ha propuesto un enfoque MapReduce para permitir el uso del algoritmo k-NN en problemas *big data*. Sin el uso de esta paralelización, k-NN estaría limitado a conjuntos de datos medios, especialmente cuando el tiempo de ejecución es relevante. El esquema propuesto es una paralelización exacta del k-NN, de modo que, la precisión sigue siendo la misma y la eficiencia se ha mejorado en gran medida. Los experimentos realizados han demostrado la reducción de tiempo de cálculo alcanzado en comparación con la versión secuencial. Como trabajo futuro, se plantea llevar a cabo más experimentos así como el uso de las tecnologías más recientes como Spark para acelerar aún más el cómputo mediante el uso de operaciones ajenas a la filosofía MapReduce. Para mejorar la escalabilidad en el conjunto de test, se realizará un nuevo diseño que permita la utilización de varios *reduces*.

Agradecimientos Este trabajo se ha sustentado por los Proyetos de Investigación TIN2014-57251-P y P11-TIC-7765. J. Maillo disfruta de una beca de iniciación a la investigación de la Universidad de Granada. I. Triguero tiene una beca postdoctoral BOF de la Universidad de Gante.

Referencias

1. M. Minelli, M. Chambers, and A.Dhiraj, *Big Data, Big Analytics: Emerging Business Intelligence and Analytic Trends for Today's Businesses (Wiley CIO)*, 1st ed. Wiley Publishing, 2013.
2. T. M. Cover and P. E. Hart, *"Nearest neighbor pattern classification" IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21-27, 1967.
3. J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communicacionts of the ACM*, vol. 51, no. 1, pp. 107-113, Jan. 2008.
4. A. Fernández, S. Río, V. López, A. Bawakid, M. del Jesús, J. Benítez, and F. Herrera, "Big data with cloud computing: An insight on the computing environment, mapreduce and programming frameworks," WIREs *Data Mining and Knowledge Discovery*, vol. 4, no. 5, pp. 380-409, 2014.
5. W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: portable parallel programming with the message-passing interface.* MIT press, 1999, vol. 1.
6. A. Srinivasan, T. Faruquie, and S. Joshi, "Data and task parallelism in ILP using mapreduce," *Machine Learning*, vol. 86, no. 1, pp. 141-168, 2012.
7. I. Triguero, D. Peralta, J. Bacardit, S. García, and F.Herrera, "MRPR: A mapreduce solution for prototype reduction in big data classification," *Neurocomputing*, vol. 150, Part A, no. 0, pp. 331-345, 2015.
8. T. Yokoyama, Y. Ishikawa, and Y. Suzuki, "Processing all k-nearest neighbor queries in hadoop," in *Web-Age Information Management*, ser. Lecture Notes in Computer Science, H. Gao, L. Lim, W. Wang, C. Li, and L. Chen, Eds. Springer Berlin Heidelberd, 2012, vol. 7418, pp. 346-351.
9. C. Zhang, F. Li, and J. Jestes, "Efficient parallel knn joins for large data in mapreduce," in *Proceedings of the 15th International Conference on Extending Database Technology*, ser. EDBT '12. New York, NY, USA: ACM, 2012, pp. 38-49.
10. A. Frank and A. Asuncion, "UCI machine learning repository." 2010. [Online]. Available: http://archive.ics.uci.edu/ml