CHI-BD: Sistema de Clasificación Basado en Reglas Difusas para problemas de clasificación Big Data

Mikel Elkano[1,2], Mikel Galar[1,2], José Sanz[1,2], y Humberto Bustince[1,2]

[1] Dpto. de Automática y Computación, Universidad Pública de Navarra, Campus Arrosadía s/n, 31006 Pamplona, España,
[2] Institute of Smart Cities, Universidad Pública de Navarra, Campus Arrosadía s/n, 31006 Pamplona, España,
{mikel.elkano, mikel.galar, joseantonio.sanz, bustince}@unavarra.es

Resumen. Los Sistemas de Clasificación Basados en Reglas Difusas (SCBRDs) para problemas Big Data publicados hasta la fecha consisten en aprender de forma concurrente varios clasificadores Chi et al. cuyas bases de reglas se agregan posteriormente. El problema de esta aproximación es que cuando la configuración del cluster varía se obtiene un modelo diferente, reduciéndose la precisión en la clasificación conforme se añaden más nodos. Nuestro objetivo en este trabajo es diseñar un nuevo SCBRD para problemas de clasificación Big Data (CHI-BD) que sea capaz de generar exactamente el mismo modelo que obtendríamos si fuera posible ejecutar el algoritmo Chi et al. original con esta cantidad de datos, independientemente de la configuración del cluster.

Palabras Clave: Sistemas de Clasificación Basados en Reglas Difusas, Big Data, Apache Hadoop, MapReduce

1 Introducción

Entre los problemas que nos encontramos en entornos Big Data [1] está la *clasificación*, donde un algoritmo trata de extraer conocimiento de un conjunto de datos para predecir futuros patrones. Una las técnicas más populares para afrontar este tipo de tareas son los Sistemas de Clasificación Basados en Reglas Difusas (SCBRDs). Estos sistemas generan un modelo formado por varias reglas compuestas por etiquetas lingüísticas interpretables [6]. Sin embargo, los SCBRDs presentan serios problemas de escalabilidad cuando se enfrentan a conjuntos de datos de gran tamaño. Por consiguiente, este tipo de técnicas deben adaptarse para trabajar en entornos Big Data. Entre las aproximaciones más empleadas se encuentra el uso de la computación distribuida mediante frameworks como *Apache Hadoop* [11], donde el conjunto de datos original se divide en múltiples subconjuntos que son distribuidos en una serie de nodos (máquinas). De esta forma, cada nodo procesa únicamente la porción de los datos que almacena (fase *Map*), para que finalmente los resultados parciales generados en todos los nodos se agreguen y se obtenga el resultado final (fase *Reduce*). Las fases Map y Reduce son llevadas a cabo por el *Mapper* y el *Reducer*, respectivamente.

Las aproximaciones publicadas hasta la fecha que proponen SCBRDs capaces de afrontar problemas Big Data [9, 10] consisten en el aprendizaje de varios clasificadores Chi et al. [2] cuyas bases de reglas son agregadas al final, empleando Apache Hadoop para el despliegue del sistema distribuido. La primera aproximación de un SCBRD

para Big Data (Chi-FRBCS-BigDataCS) fue presentada por López et al. en [9]. Sin embargo, la escalabilidad de este tipo de aproximaciones es cuestionable en términos de precisión, debido a que las bases de reglas generadas en cada Mapper se aprenden empleando únicamente el subconjunto de ejemplos asociados al mismo. Por consiguiente, los pesos de las reglas presentan una elevada dependencia respecto a la proporción y la distribución de las clases en cada subconjunto. De esta forma, la calidad de los pesos de las reglas (y consecuentemente el rendimiento de este método) se ve afectado por el incremento del número de Mappers, el cual es necesario cuando se afrontan conjuntos de datos cada vez más grandes.

En este trabajo, tratamos de resolver los problemas existentes en las anteriores aproximaciones diseñando un nuevo SCBRD distribuido para problemas de clasificación Big Data. Nuestro objetivo principal es proporcionar un algoritmo que genere exactamente el mismo modelo que obtendríamos si pudiéramos ejecutar el algoritmo de Chi et al. original en el problema Big Data en cuestión, independientemente del número de nodos (Mappers) empleados para el aprendizaje. Para ello, aprovechamos la idoneidad del método de Chi et al. para el paradigma MapReduce con el objetivo de calcular los pesos de las reglas empleando el conjunto de entrenamiento al completo.

Para evaluar el rendimiento de nuestra propuesta, hemos llevado a cabo un estudio experimental con 20 conjuntos de datos binarios obtenidos de 8 problemas de clasificación multi-clase disponibles en el repositorio UCI [8]. Concretamente, estudiaremos los resultados obtenidos por medio de nuestro nuevo algoritmo en términos de tiempo de ejecución y precisión en comparación con los obtenidos por Chi-FRBCS-BigDataCS [9].

La estructura del trabajo es la siguiente. En la Secciones 2 y 3 repasamos los conceptos básicos de Apache Hadoop y mostramos una breve descripción del método Chi-FRBCS-BigDataCS, respectivamente. La Sección 4 presenta el nuevo SCBRD distribuido para problemas de clasificación Big Data (CHI-BD). En la Sección 5 mostramos la descripción del marco experimental y el análisis de los resultados obtenidos. Finalmente, la Sección 6 presenta las conclusiones del trabajo.

2 Apache Hadoop

Apache Hadoop es uno de los frameworks más empleados en entornos Big Data. Este framework proporciona una implementación libre del algoritmo *MapReduce* [4], compuesto por las siguientes fases.

1. *Fase Map*: los datos de entrada se dividen primero en múltiples fragmentos lógicos asociados a diferentes bloques físicos. De esta forma, cada fragmento será procesado por una sola unidad de procesamiento llamada *Mapper*. Cada nodo puede ejecutar múltiples Mappers de manera concurrente. Durante el procesamiento llevado a cabo en el Mapper, los datos de entrada se transforman en pares clave-valor <*k*, *v*>que son procesados en la función *map()* (definida por el usuario), la cual se invoca para cada par. El resultado de esta función es otro par clave-valor <*k*', *v*'>que forma parte de los denominados *datos intermedios*. Finalmente, los datos intermedios son preparados para ser enviados a la siguiente fase mediante los siguientes pasos:

   (a) *Sorting and Merging*: las salidas se ordenan respecto a las claves y se genera una lista para cada clave que contiene todos sus valores (<*k*', list (*v*')>).
   (b) *Partitioning*: se asigna un Reducer a cada clave.
   (c) *Shuffle*: los datos intermedios se copian a los Reducers.
2. *Fase Reduce*: el Reducer se encarga de agregar las salidas de los Mappers. Para ello, primero se ordenan todos los pares clave-valor recibidos respecto a las claves. Una vez finalizan todos los Mappers y todos los pares clave-valor han sido ordenados, se invoca la función *reduce()* (definida por el usuario) para cada clave (*k*') y se agregan todos sus valores (list (*v*')). Finalmente, el Reducer devuelve el resultado final (*v*") para cada clave.

Estas dos fases forman lo que se denomina un MapReduce *job*.

3 Chi-FRBCS-BigDataCS

Una de las herramientas más populares para resolver problemas de clasificación son los SCBRDs. Estas técnicas proporcionan un modelo expresado mediante reglas que contienen etiquetas lingüísticas interpretables por el ser humano [6].

Para generar la base de reglas, se aplica un algoritmo de aprendizaje empleando un conjunto de entrenamiento $\mathscr{D}_T$ compuesto por P ejemplos etiquetados $x_p = (x_{p1}, \ldots, x_{pn})$ con $p \in \{1, \ldots, P\}$, donde x_{pi} es el valor del i-ésimo atributo ($i \in \{1, 2, \ldots, n\}$) del ejemplo p-ésimo. Cada ejemplo pertenece a una clase $y_p \in \mathbb{C} = \{C_1, C_2, ..., C_m\}$, donde m es el número de clases del problema. La estructura de las reglas empleadas en el algoritmo de Chi et al. [2] es la siguiente:

$$\text{Regla } R_j: \text{ Si } x_1 \text{ es } A_{j1} \text{ y } \ldots \text{ y } x_n \text{ es } A_{jn} \text{ entonces Clase} = C_j \text{ con } RW_j \qquad (1)$$

donde R_j es la etiqueta de la regla j-ésima, $x = (x_1, \ldots, x_n)$ es un vector n-dimensional que representa el ejemplo, A_{ji} es una etiqueta lingüística modelada por una función de pertenencia triangular, C_j es la etiqueta de la clase y RW_j es el peso de la regla. En el caso de conjuntos de datos imbalanceados [5] (como es el caso de este trabajo), calculamos el peso de la regla utilizando el *Penalized Cost-Sensitive Certainty Factor* (PCF-CS) [9], que se trata de una modificación del *Penalized Certainty Factor* (PFC) [7]:

$$RW_j = PCF\text{-}CS = \frac{\sum_{x_p \in ClassC_j} \mu_{A_j}(x_p) \cdot Cs(y_p) - \sum_{x_p \notin ClassC_j} \mu_{A_j}(x_p) \cdot Cs(y_p)}{\sum_{p=1}^{P} \mu_{A_j}(x_p) \cdot Cs(y_p)} \qquad (2)$$

donde $Cs(y_p)$ es el coste asociado a la clase y_p. Los costes se definen como $Cs(min) = IR$ y $Cs(maj) = 1$, siendo min y maj la clase minoritaria y mayoritaria, respectivamente, y IR es el ratio de imbalanceo definido como P_{maj}/P_{min}, donde P_{maj} y P_{min} son el número de ejemplos pertenecientes a la clase mayoritaria y minoritaria, respectivamente.

A la hora de generar las reglas, Chi emplea el siguiente algoritmo.

1. *Construcción de las etiquetas lingüísticas*. Se crean los conjuntos difusos (etiquetas lingüísticas) con la misma forma triangular manteniéndolos distribuidos de forma equitativa a lo largo del rango de valores.
2. *Generación de una regla por cada ejemplo*. Se genera una nueva regla para cada ejemplo x_p de la siguiente manera.
   (a) Se calculan los grados de pertenencia de cada valor x_{pi} a todos los conjuntos difusos de la variable i.
   (b) Se selecciona para cada variable el conjunto difuso con mayor grado de pertenencia.
   (c) Se determina la parte antecedente mediante la intersección de los conjuntos difusos seleccionados. La parte consecuente la determina la etiqueta de la clase del ejemplo (y_p).
   (d) Se calcula el peso de la regla siguiendo la Ecuación (2).

Como podemos apreciar, es posible que obtengamos reglas con los mismos antecedentes pero diferentes consecuentes. En tal caso, se selecciona únicamente la regla de mayor peso. Aquellas reglas que tengan peso negativo son eliminadas de la base de reglas.

La primera aproximación para poder aplicar este algoritmo en problemas de clasificación Big Data fue presentada en [9], centrándose en problemas binarios imbalanceados. Para ello, los autores proponen el uso del framework Apache Hadoop. Este método está compuesto por dos fases:

1. *Generación de las bases de reglas*: en cada Mapper se aprende un clasificador Chi independiente considerando únicamente los ejemplos asociados a cada Mapper. Por consiguiente, las reglas obtenidas en cada clasificador se generan empleando un subconjunto del conjunto de entrenamiento. Una vez ha terminado la fase de aprendizaje de todos los clasificadores, se obtienen tantas bases de reglas como Mappers (clasificadores) se hayan ejecutado.
2. *Agregación de las bases de reglas*: todas las bases de reglas generadas en la fase anterior se agregan en un único Reducer, obteniendo la base de reglas definitiva. Si hay reglas duplicadas, se selecciona aquella con mayor peso.

4 CHI-BD: diseñando una nueva solución MapReduce para problemas de clasificación Big Data

En este trabajo proponemos un nuevo SCBRD distribuido para problemas de clasificación Big Data, haciendo uso del algoritmo de Chi et al. y del paradigma MapReduce. Nuestro método está compuesto por tres fases, en donde cada una representa un MapReduce *job*.

1. *Generación de reglas (sin pesos).* Se obtiene la base de reglas preliminar generando una nueva regla por cada ejemplo de entrenamiento. En esta fase no se consideran los pesos de las reglas, obteniendo únicamente la parte antecedente y consecuente de las mismas. Para ello, la función *map*() genera una nueva regla en forma de par (clave, valor) para cada ejemplo, donde la clave es la parte antecedente de la regla y el valor es el consecuente (clase) de la misma. La función *reduce*() del Reducer se encarga de crear una lista de todos los consecuentes de las reglas que comparten la misma parte antecedente (clave).

2. *Búsqueda de los subconjuntos de antecedentes más frecuentes.* Con el objetivo de evitar posibles cálculos repetidos a la hora de calcular los pesos de las reglas en la siguiente fase, se realiza una búsqueda de los subconjuntos de antecedentes que más se aparecen en la base de reglas. Para ello, la función *map*() divide la parte antecedente de las reglas obtenidas en la fase anterior en un número determinado de subconjuntos contiguos de antecedentes (definido por el usuario) y genera un par (clave, valor) para cada subconjunto, donde la clave es un par que contiene la lista de antecedentes y el identificador del subconjunto y el valor contiene un 1 (representando una ocurrencia). Posteriormente, la función *reduce*() del Reducer suma las ocurrencias de cada subconjunto y devuelve un par (clave, valor) para los subconjuntos con un mínimo número de ocurrencias (definido por el usuario), donde la clave es un par que contiene la lista de antecedentes y el identificador del subconjunto y el valor es el número de ocurrencias del mismo.

3. *Cálculo de los pesos de las reglas.* En esta fase se calculan los pesos de las reglas obtenidas en la primera fase y se obtiene la base de reglas final. Para ello, cada Mapper calcula los grados de emparejamiento de todas las reglas con el subconjunto de ejemplos de entrenamiento asociado a ese Mapper y almacena la suma de los grados de emparejamiento de cada regla para cada clase del problema en una matriz, donde las filas son las reglas y las columnas son las clases. Para agilizar este proceso, antes de llevar a cabo el cálculo del grado de emparejamiento de un ejemplo dado con cada una de las reglas, se realiza un pre-cálculo del grado de emparejamiento de dicho ejemplo con los subconjuntos de antecedentes más frecuentes obtenidos en la fase anterior. De esta forma, cuando una regla contenga un subconjunto de antecedentes marcado como frecuente, no será necesario calcular el grado de emparejamiento correspondiente a dicho subconjunto, ya que habrá sido pre-calculado previamente. Una vez obtenidas las sumas de los grados de emparejamiento para cada clase, el Mapper devuelve un par (clave, valor) para cada regla, donde la clave es la regla en cuestión y el valor es una lista que contiene las sumas de los grados de emparejamiento de esa regla para cada clase. Finalmente, la función *reduce*() del Reducer calcula para cada regla la suma total de las sumas parciales de los grados de emparejamiento obtenidas por el Mapper y calcula los pesos de las reglas aplicando la Ecuación (2). Cabe destacar que el número máximo de reglas procesadas en cada Reducer es configurable.

Como podemos apreciar, el cálculo de los pesos de las reglas se lleva a cabo considerando todos los ejemplos del conjunto de entrenamiento, al contrario que en el resto de aproximaciones basadas en Chi et al. y MapReduce (Sección 3). Por consiguiente, la base de reglas obtenida será siempre la misma independientemente del número de Mappers/Reducers empleados para la ejecución del algoritmo, recuperando el modelo que obtendríamos si pudiéramos ejecutar el método de Chi et al. original en Big Data. De esta forma, en este trabajo presentamos una solución para construir un modelo global empleando el conjunto de entrenamiento al completo, en vez de agregar múltiples modelos locales generados a partir de subconjuntos del conjunto original.

5 Estudio experimental

En este estudio queremos comprobar la calidad de nuestra propuesta analizando los resultados obtenidos en términos de precisión y tiempo de ejecución y comparándolos respecto a los obtenidos por la aproximación presentada en [9].

La Sección 5.1 describe los conjuntos de datos, los parámetros, y los test estadísticos empleados en el estudio. En la Sección 5.2 analizamos los resultados obtenidos.

5.1 Marco experimental

Para el desarrollo del estudio experimental, hemos considerado 8 conjuntos de datos diferentes del repositorio UCI [8]. Con el objetivo de obtener más conjuntos de datos y de trabajar con el mismo marco experimental que en [9], hemos seleccionado 20 problemas de clasificación binarios diferentes convirtiendo los conjuntos de datos originales multi-clase en múltiples problemas binarios *One-vs-Rest*. Para ello, en cada conjunto de datos original seleccionamos una clase positiva y consideramos el resto como la clase negativa. La Tabla 1 muestra la descripción de los conjuntos de datos indicando el número de ejemplos (#Ejemplos), número de ejemplos de la clase mayoritaria y minoritaria (P_{maj}:P_{min}), y el número de atributos (#Atributos) reales (R), enteros (E), nominales (N), y totales (T).

Tabla 1: Descripción de los conjuntos de datos (datasets).

Dataset	#Ejemplos	(P_{maj}:P_{min})	#Atributos			
			R	E	N	Total
Census	142521	(134359:8162)	1	12	28	41
Cov_1	581012	(369172:211840)	10	0	44	54
Cov_2	581012	(297711:283301)	10	0	44	54
Cov_3	581012	(545258:35754)	10	0	44	54
Cov_7	581012	(560502:20510)	10	0	44	54
Far_Fat	100968	(58852:42116)	5	0	24	29
Far_Inc	100968	(85896:15072)	5	0	24	29
Far_No	100968	(80961:20007)	5	0	24	29
Far_Nin	100968	(87078:13890)	5	0	24	29
Higgs	11000000	(5829123:5170877)	28	0	0	28
Kdd_dos	4898431	(3883370:1015061)	26	0	15	41
Kdd_nor	4898431	(3925650:972781)	26	0	15	41
Kdd_prb	4898431	(4857329:41102)	26	0	15	41
Kdd_r2l	4898431	(4897305:1126)	26	0	15	41
Pok_0	1025009	(513701:511308)	0	10	0	10
Pok_1	1025009	(591912:433097)	0	10	0	10
Pok_2	1025009	(976181:48828)	0	10	0	10
Pok_3	1025009	(1003375:21634)	0	10	0	10
Skin	245057	(194198:50859)	0	3	0	3
Susy	5000000	(2712173:2287827)	18	0	0	18

Todos los experimentos han sido realizados aplicando una validación cruzada de 5 particiones. De esta manera, dividimos aleatoriamente el conjunto de datos en 5 particiones, cada una conteniendo el 20% de los ejemplos, y empleamos una combinación de cuatro de ellas (80%) para entrenar el sistema y el resto para testearlo. Por tanto, el resultado de cada conjunto de datos se calcula como la media de las 5 particiones.

Respecto a los parámetros considerados para la ejecución de los métodos, en ambos casos (Chi-FRBCS-BigDataCS y CHI-BD) hemos empleado 3 etiquetas lingüísticas

por variable y hemos aplicado el PCF-CS (Ecuación 2 en la Sección 3) para realizar el cálculo de los pesos de las reglas y el método de razonamiento difuso *winning rule* [3] para clasificar nuevos ejemplos. En cuanto a CHI-BD, hemos empleado la siguiente configuración:

- N° de subconjuntos de antecedentes por regla: 4
- Mínimo n° de ocurrencias para marcar un subconjunto de antecedentes como frecuente: 10
- Máximo n° de reglas procesadas en un Reducer: 400000

Para evaluar la precisión en la clasificación de ambos métodos hemos considerado dos métricas frecuentemente utilizadas en conjuntos de datos imbalanceados [5]: el área bajo la curva ROC (AUC) y la media geométrica (GM). Respecto al análisis estadístico, aplicamos el test de Wilcoxon [12] para realizar comparaciones por pares entre ambos métodos.

En cuanto a la infraestructura usada para los experimentos, todos los métodos han sido ejecutados en un cluster de 8 nodos conectados a una Red de área Local Ethernet a 1Gb/s. La mitad de estos nodos están compuestos por 2 procesadores Intel Xeon E5-2620 a 2.4 GHz (3.2 GHz con Turbo Boost) con 12 núcleos virtuales en cada uno (de los cuales 6 son físicos). Tres de los nodos restantes están equipados con 2 procesadores Intel Xeon E5-2620 a 2.1 GHz con el mismo número de núcleos que los anteriores. El último nodo es el master, compuesto por un procesador Intel Xeon E5-2609 con 4 núcleos físicos a 2.4 GHz. Todos los nodos esclavos están equipados con 32GB de RAM, mientras que el maestro trabaja con 8GB de RAM. Con respecto al almacenamiento, todos los nodos emplean discos duros con velocidades de lectura/escritura de 128 MB/s. Todo el cluster funciona sobre CentOS 6.5 y Apache Hadoop 2.6.0. Esta configuración ofrece 42 containers de YARN concurrentes, donde cada uno puede ser un Mapper, un Reducer, o el Application Master.

5.2 Resultados experimentales

La Tabla 2 muestra la GM y AUC obtenido por CHI-BD y Chi-FRBCS-BigDataCS cuando empleamos 32, 64, 128, y 256 Mappers para cada uno de los 20 conjuntos de datos. Dado que la precisión de CHI-BD es la misma para cualquier número de Mappers, en este caso solo se muestra una columna. El mejor resultado para cada conjunto de datos se muestra en negrita, mientras que el mejor resultado obtenido entre los diferentes números de Mappers aparece subrayado.

Como podemos observar, la precisión en el caso de Chi-FRBCS-BigDataCS se ve claramente afectada por el número de Mappers, ya que ésta se reduce conforme se añaden más Mappers. De acuerdo a la Tabla 2, la GM y la AUC caen de media un 13% y un 6% respecto a CHI-BD, respectivamente. La razón de este comportamiento es que el hecho de que los pesos de las reglas se calculen empleando un subconjunto del conjunto de entrenamiento hace que los pesos sean muy dependientes de la distribución y de la proporción de las clases en cada subconjunto.

Para comprobar si realmente existen diferencias estadísticas entre la precisión de CHI-BD y Chi-FRBCS-BigDataCS aplicamos el test de Wilcoxon, cuyos resultados

Mikel Elkano et al.

Tabla 2: GM y AUC en test para cada método.

(a) GM

Dataset	CHI-BD	Chi-FRBCS-BigDataCS			
		32	64	128	256
Census	**.5231**	.4030	.4058	.4113	.4115
Cov_1	**.7531**	.7528	.7523	.7470	.7350
Cov_2	.7291	**.7296**	.7278	.7264	.7246
Cov_3	**.9565**	.9551	.9456	.9312	.9210
Cov_7	**.9281**	.9089	.8886	.8587	.8322
Far_Fat	**.5871**	**.5871**	**.5871**	.5870	.5870
Far_Inc	**.6919**	.5572	.5549	.5561	.5595
Far_No	**.8675**	.8336	.8338	.8272	.8274
Far_Nin	**.7139**	.5307	.5408	.5397	.5426
Higgs	**.5847**	.5772	.5736	.5691	.5637
Kdd_dos	**.9991**	**.9991**	**.9991**	**.9991**	**.9991**
Kdd_nor	**.9992**	**.9992**	**.9992**	**.9992**	**.9992**
Kdd_prb	**.9924**	.9911	.9910	.9900	.9860
Kdd_r2l	**.9840**	.9251	.8769	.8094	.4232
Pok_0	**.6336**	.6183	.6082	.5973	.5868
Pok_1	**.5848**	.5616	.5483	.5301	.5081
Pok_2	**.6703**	.3713	.2302	.1256	.1119
Pok_3	**.7387**	.2720	.1247	.0720	.0936
Skin	**.9597**	.9595	.9590	.9590	.9588
Susy	**.5524**	.5477	.5459	.5424	.5399
AVG.	**.7725**	.7040	.6846	.6689	.6456

(b) AUC

Dataset	CHI-BD	Chi-FRBCS-BigDataCS			
		32	64	128	256
Census	**.6220**	.5757	.5768	.5791	.5787
Cov_1	**.7532**	.7530	.7528	.7483	.7392
Cov_2	.7373	**.7379**	.7365	.7353	.7325
Cov_3	**.9572**	.9553	.9456	.9316	.9220
Cov_7	**.9285**	.9109	.8926	.8669	.8451
Far_Fat	**.6723**	.6722	**.6723**	.6721	**.6723**
Far_Inc	**.7123**	.6370	.6352	.6358	.6377
Far_No	**.8728**	.8438	.8440	.8383	.8386
Far_Nin	**.7283**	.6195	.6243	.6235	.6249
Higgs	**.5848**	.5776	.5741	.5699	.5650
Kdd_dos	**.9991**	**.9991**	**.9991**	**.9991**	**.9991**
Kdd_nor	**.9992**	**.9992**	**.9992**	**.9992**	**.9992**
Kdd_prb	**.9925**	.9911	.9910	.9900	.9861
Kdd_r2l	**.9841**	.9279	.8844	.8280	.5924
Pok_0	**.6360**	.6206	.6098	.5981	.5875
Pok_1	**.5859**	.5658	.5564	.5475	.5400
Pok_2	**.6709**	.5473	.5197	.5063	.5050
Pok_3	**.7388**	.5302	.5066	.5023	.5038
Skin	**.9605**	.9604	.9599	.9598	.9596
Susy	**.6242**	.6210	.6195	.6173	.6153
AVG.	**.7880**	.7523	.7450	.7374	.7222

se muestran en la Tabla 3 y donde se muestran las victorias (W), empates (T), y las derrotas (L) de CHI-BD frente a Chi-FRBCS-BigDataCS junto con el p-valor. En base a estos resultados, podemos ver que nuestra propuesta mejora estadísticamente a Chi-FRBCS-BigDataCS con un nivel de confianza del 99% (p-valor < 0.01) en todos los casos.

Respecto a los tiempos de ejecución, la Tabla 4 muestra los tiempos de ejecución de ambos métodos cuando se emplean diferentes números de Mappers. Por un lado, se muestra el tiempo total de ejecución cuando se emplea el número máximo de Mappers que puede ejecutar nuestro cluster en paralelo (32). Por otro lado, cuando se trata de números mayores que 32, se muestra el tiempo de ejecución medio de los Mappers. La razón de no incluir los tiempos totales en este caso es que algunos Mappers no se ejecutan en paralelo, ya que se sobrepasa el límite de Mappers concurrentes de nuestro cluster. Los tiempos de ejecución de los Reducers no se muestran debido a que son despreciables en comparación a los de los Mappers. Por otra parte, los tiempos en Higgs no se han incluido en la media debido a la gran diferencia existente en tamaño respecto al resto de datasets, por lo que se muestra separado al resto.

Como podemos observar en la Tabla 4, CHI-BD es considerablemente más rápido que Chi-FRBCS-BigDataCS, exceptuando los casos en los que los conjuntos de datos son demasiado pequeños como para que el sobrecoste de ejecutar tres fases MapReduce sea menor que el tiempo de procesamiento del algoritmo.

CHI-BD: SCBRD para problemas de clasificación Big Data

Tabla 3: Test de Wilcoxon para comparar CHI-BD y Chi-FRBCS-BigDataCS.

CHI-BD vs.	GM		AUC	
Chi-FRBCS-BigDataCS	W/T/L	p-valor	W/T/L	p-valor
32 mappers	16/3/1	0.0002	17/2/1	0.0002
64 mappers	17/3/0	0.0001	17/3/0	0.0001
128 mappers	18/2/0	0.0001	18/2/0	0.0001
256 mappers	18/2/0	0.0001	17/3/0	0.0001

Tabla 4: Tiempo de ejecución (mm:ss) para CHI-BD y Chi-FRBCS-BigDataCS.

Dataset	Total		Mappers							
	CHI-BD	Chi-FRBCS-BigDataCS	CHI-BD				Chi-FRBCS-BigDataCS			
	32	32	32	64	128	256	32	64	128	256
Census	01:36	00:26	00:14	00:07	00:04	00:03	00:00	00:00	00:00	00:00
Cov_1	01:16	01:08	00:08	00:04	00:02	00:01	00:33	00:08	00:02	00:00
Cov_2	01:15	01:10	00:08	00:04	00:02	00:01	00:33	00:08	00:02	00:00
Cov_3	01:14	01:09	00:08	00:04	00:02	00:01	00:33	00:08	00:02	00:00
Cov_7	01:15	01:10	00:08	00:04	00:02	00:01	00:33	00:08	00:02	00:00
Far_Fat	01:21	00:24	00:06	00:04	00:02	00:01	00:00	00:00	00:00	00:00
Far_Inc	01:20	00:24	00:06	00:04	00:02	00:01	00:00	00:00	00:00	00:00
Far_No	01:22	00:23	00:06	00:04	00:02	00:01	00:00	00:00	00:00	00:00
Far_Nin	01:21	00:24	00:06	00:04	00:02	00:01	00:00	00:00	00:00	00:00
Kdd_dos	01:18	72:42	00:21	00:10	00:05	00:03	58:35	10:55	02:21	00:29
Kdd_nor	01:16	71:57	00:21	00:10	00:05	00:03	56:50	10:55	02:21	00:29
Kdd_prb	01:17	73:22	00:21	00:10	00:05	00:03	58:55	10:49	02:21	00:29
Kdd_r2l	01:15	73:32	00:21	00:10	00:05	00:03	58:56	10:49	02:21	00:29
Pok_0	01:47	00:58	00:27	00:14	00:07	00:03	00:25	00:06	00:01	00:00
Pok_1	01:50	00:59	00:28	00:14	00:07	00:04	00:25	00:06	00:01	00:00
Pok_2	01:49	00:57	00:27	00:14	00:07	00:04	00:26	00:06	00:01	00:00
Pok_3	01:46	00:59	00:27	00:13	00:07	00:03	00:25	00:06	00:01	00:00
Skin	00:53	00:22	00:00	00:00	00:00	00:00	00:01	00:00	00:00	00:00
Susy	01:43	33:39	00:37	00:18	00:09	00:05	26:49	05:46	01:14	00:16
AVG.	**01:24**	**17:41**	**00:15**	**00:08**	**00:04**	**00:02**	**13:53**	**02:38**	**00:34**	**00:06**
Higgs	160:58	251:55	119:45	60:54	31:07	15:24	223:58	48:11	10:15	01:52

6 Conclusiones

En este trabajo hemos presentado un nuevo SCBRD distribuido para problemas de clasificación Big Data que hace uso del algoritmo de Chi et al. y del paradigma MapReduce. Este método es capaz de proporcionar el mismo modelo (base de reglas) independientemente de la configuración de cluster (número de nodos, Mappers, Reducers, etc.) empleada para su ejecución, recuperando el modelo que obtendríamos con el algoritmo de Chi et al. original si pudiéramos ejecutarlo en problemas Big Data. Ésto ha sido posible gracias a que los pesos de las reglas se calculan considerando todos los ejemplos del conjunto de entrenamiento, al contrario que en aproximaciones publicadas anteriormente, como es el caso de Chi-FRBCS-BigDataCS. En Chi-FRBCS-BigDataCS los pesos de las reglas se calculan tomando subconjuntos del conjunto de entrenamiento, provocando que la calidad de los pesos (y por consiguiente la precisión) se reduzca cuando se añaden más Mappers. El estudio experimental llevado a cabo muestra que el rendimiento mostrado por CHI-BD mejora al obtenido por Chi-FRBCS-BigDataCS en términos de precisión y tiempo de ejecución.

Agradecimientos. Este trabajo ha sido financiado parcialmente por el Ministerio de Ciencia y Tecnología de España con el proyecto TIN2013-40765-P y por la red TIN2014-56381-REDT.

Referencias

1. Arthur, L.: What is big data? Forbes (2013), http://www.forbes.com/sites/lisaarthur/2013/08/15/what-is-big-data/
2. Chi, Z., Yan, H., Pham, T.: Fuzzy algorithms with applications to image processing and pattern recognition. World Scientific (1996)
3. Cordón, O., del Jesus, M., Herrera, F.: A proposal on reasoning methods in fuzzy rule-based classification systems. International Journal of Approximate Reasoning 20(1), 21–45 (1999)
4. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. Commun. ACM 51(1), 107–113 (2008)
5. Galar, M., Fernández, A., Barrenechea, E., Bustince, H., Herrera, F.: A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches. IEE Transactions on Systems, Man, and Cybernetics 42(4), 463–484 (2011)
6. Ishibuchi, H., Nakashima, T., Nii, M.: Classification and modeling with linguistic information granules: Advanced approaches to linguistic Data Mining. Springer-Verlag (2004)
7. Ishibuchi, H., Yamamoto, T.: Rule weight specification in fuzzy rule-based classification systems. IEEE Transactions on Fuzzy Systems 13(4), 428–435 (2005)
8. Lichman, M.: UCI machine learning repository (2013), http://archive.ics.uci.edu/ml
9. López, V., del Río, S., Benítez, M., Herrera, F.: Cost-sensitive linguistic fuzzy rule based classification systems under the mapreduce framework for imbalanced big data. Fuzzy Sets and Systems 258(0), 5 – 38 (2015)
10. del Río, S., López, V., Benítez, J.M., Herrera, F.: A mapreduce approach to address big data classification problems based on the fusion of linguistic fuzzy rules. Int. J. Computational Intelligence Systems 8(3), 422–437 (2015)
11. White, T.: Hadoop: The Definitive Guide. O'Reilly Media, Inc. (2009)
12. Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics 1(6), 80–83 (1945)