EDGAR-MR: un algoritmo evolutivo distribuido escalable para la obtención de reglas de clasificación

Miguel Ángel Rodríguez
Departamento de Tecnologías de la Información
Universidad de Huelva
Huelva, España

Antonio Peregrín
Centro de Estudios Avanzados en Física,
Matemáticas y Computación
Universidad de Huelva
Huelva, España

Resumen. *El modelo evolutivo para la obtención de reglas de clasificación, EDGAR, cuenta entre sus especificidades con diferentes estrategias para resolver el problema del aprendizaje distribuido de reglas cuando el conjunto de entrenamiento está dividido en diferentes subconjuntos. Sin embargo, el modelo encuentra limitaciones prácticas a su escalabilidad con un número creciente de particiones de datos. Las actuales tecnologías derivadas de los principios de MapReduce en cambio, pueden ayudar a resolver dicha limitación facilitando una implementación más escalable del modelo evolutivo distribuido EDGAR, manteniendo aspectos de su esencia. Es este trabajo se desarrolla una primera aproximación práctica a dicho modelo.*

Palabras clave: Big Data, Algoritmos Genéticos Distribuidos, EDGAR, MapReduce, Clasificación, Programación Distribuida.

I. Introduccion

Actualmente, enormes cantidades de datos son recogidas y almacenadas en bases de datos. El fin último de la Minería de Datos es obtener información de interés en forma de modelos precisos y entendibles de estas bases de datos. Dentro de esta disciplina, los sistemas de clasificación basados en reglas son un instrumento tradicional de gran utilidad. La extracción de reglas de un conjunto de datos depende en gran medida de la topología de los datos y del volumen de éstos. Adicionalmente, a medida que el volumen de datos crece, su manejo es más complejo, el tiempo necesario para su tratamiento se incrementa de manera exponencial, y además crece la dificultad de aprendizaje del algoritmo.

Los algoritmos evolutivos han demostrado una gran capacidad como recurso para extraer conocimiento, siendo robustos al ruido y a otras características inherentes a los datos; sin embargo, suele ser difícil escalar un algoritmo evolutivo eficientemente, debido al cálculo reiterativo de la bondad de los individuos que implica la evaluación de un conjunto de reglas sobre el conjunto de datos.

En este sentido, el algoritmo evolutivo distribuido para obtener reglas de clasificación, EDGAR [1], propone un sistema de distribución de poblaciones y repartición de datos que permite escalar en varias magnitudes el número de instancias que es capaz de manejar eficientemente un algoritmo evolutivo de sus características. No sólo eso, EDGAR también emplea otras estrategias dirigidas al manejo de conjuntos de datos con clases no balanceadas sin preprocesamiento específico. Sin embargo, existen límites inherentes a la arquitectura que utiliza, las cuales se pueden confirmar en la práctica.

En los últimos años, la aparición del paradigma *MapReduce* [2] y el sistema de almacenamiento HDFS [3], ha impulsado relevantes mejoras en la escalabilidad. No obstante, no todos los algoritmos son directamente aptos para implementarse directamente de forma equivalente, y por tanto, en tales casos es necesario establecer estrategias que conserven la calidad de los resultados y mantengan el tiempo de ejecución a niveles competitivos en relación a otras alternativas distribuidas.

Este trabajo propone un algoritmo evolutivo escalable para la obtención de reglas de clasificación basado en los principios de EDGAR, implementado sobre el paradigma *MapReduce*, y realiza una experimentación preliminar para comprobar la validez los conceptos propuestos.

La organización de este trabajo es la siguiente: en la primera Sección se introduce el esquema general del algoritmo evolutivo EDGAR. La Sección 2.1 se dedica a exponer las alternativas de implementación del modelo distribuido sobre *MapReduce*. La Sección 3 se centra específicamente en la adaptación de las estrategias para conjuntos de datos no balanceados. Finalmente, la Sección 4 analiza los resultados de precisión, calidad y escalabilidad sobre una experimentación preliminar con conjuntos de datos de diversas características.

II. Antecedentes

En esta Sección se repasan, para el aprendizaje de clasificadores, en primer lugar, las principales referencias en el área de los algoritmos genéticos distribuidos, y en segundo lugar, las propuestas que emplean *MapReduce*.

A. Modelos evolutivos distribuidos para el aprendizaje de clasificadores

Una de las primeras referencias reseñables en este ámbito es REGAL [4]. Éste propone una división de datos en nodos que contienen algoritmos genéticos (AGs) para el aprendizaje de reglas. Posteriormente refina dichas reglas asignando cada una de ellas y sus datos asociados a diferentes nodos. REGAL-TC [5] es una propuesta que mejora la precisión de REGAL a través de un ponderado de contra-ejemplos y otras estrategias. Now-GNet [6] por su parte, propone una mejora sobre el mismo esquema utilizando un buffer de reglas a evaluar que separa las

funciones de evaluación del individuo y del AG, lo cual permite un mayor grado de paralelización. Estos tres algoritmos tienen en común la presencia de un proceso supervisor síncrono que en base a los resultados parciales de los nodos redistribuye datos y reglas a los nodos subordinados.

EDGAR utiliza un modelo mixto de nodos *aprendedores* independientes, que emplean un AG local (AGL), con un subconjunto de los datos y supervisor central. El sistema es asíncrono, utilizando una copia del conjunto global de datos para evaluar las reglas generadas por los AGLs y generar un clasificador final basado en la cobertura y calidad de ellas sobre el conjunto de datos de entrenamiento (Figura 1). El AGL utiliza una codificación de tipo *un individuo = una regla*, dentro del paradigma GCCL; la población representa un conjunto redundante de reglas y la solución del mismo es un clasificador compuesto por un subconjunto de las mismas que clasifica al conjunto de los datos asignados.

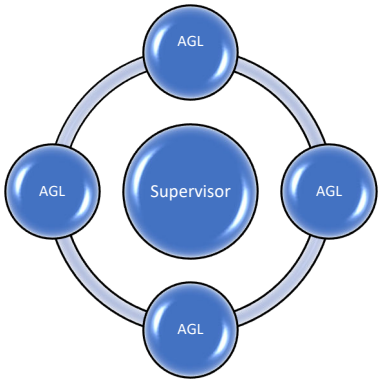

Fig. 1. Modelo distribuido de EDGAR: poblaciones de algoritmos geneticos y datos con particiones en nodos y nodo central con conjunto completo

La regla se representa como una cadena binaria donde cada posible valor de un atributo está asociado a un bit. Esta representación puede tener varios valores activos en cada atributo, consiguiendo un lenguaje de descripción de conceptos compacto, pero por otra parte requiere que los conjuntos de datos continuos sean discretizados previamente. La clase también está representada en el cromosoma con un único valor a la vez (Figura 2).

c_1			c_2		c_3			Clase	
v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
1	0	1	0	0	1	0	0	0	1

si c_1 en (v_1,v_3) y c_3 en (v_1) entonces clase es v_2

Fig. 2. Ejemplo de representación de regla en EDGAR a formato binario

Las mejores reglas, por calidad y cobertura, se intercambian periódicamente entre nodos para favorecer la cooperación entre los AGLs de modo similar al modelo de AGs distribuidos basado en Islas [9] y se envian a un nodo central que hace las veces de supervisor. Este nodo dispone de una copia del conjunto global de datos que utiliza para reevaluar las reglas recibidas y generar un clasificador final basado en la cobertura y calidad de ellas sobre el conjunto completo de datos de entrenamiento. El supervisor manda una señal de terminación a los nodos AGL cuando el clasificador generado no mejora durante un periodo de tiempo y genera el clasificador como una lista ordenada en base al valor de las reglas sobre el conjunto completo de datos.

B. *MapReduce en el ámbito del aprendizaje de clasificadores*

MapReduce [1] permite un modelo de diseño paralelo basado en la división de datos. Este modelo se basa en la existencia de un conjunto de datos distribuidos que permiten una alta escalabilidad de datos y la codificación de tareas sobre estos basados fundamentalmente en dos operaciones *Map* y *Reduce*.

Centrándonos en el aprendizaje de modelos de clasificación, hay técnicas que se prestan mejor que otras para este paradigma porque no necesitan de un acceso repetido a los datos. En este sentido, la generación de clasificadores con *Random Forest* [7] implementado bajo *MapReduce* realiza la evaluación de múltiples árboles de decisión eficientemente. Un ejemplo reciente de este uso puede verse en [8].

Otro algoritmo directamente compatible con esta arquitectura es el de generación de reglas basadas en ejemplos como semillas que posteriormente formarán el clasificador. En [10] se generan reglas en la fase *Map* en base a un conjunto de etiquetas lingüísticas que son evaluadas en la fase de construcción y posteriormente seleccionadas en la fase *Reduce* para generar el conjunto de reglas final teniendo en cuenta el coste de la clasificación en caso de existir clases no balanceadas.

III. El Modelo EDGAR-MR

En este apartado proponemos una versión del modelo EDGAR sobre el paradigma *MapReduce*. Esta versión, a la que denominamos EDGAR-MR, aprovecha las mejoras en eficiencia, robustez y facilidad de ejecución con conjuntos de datos de alta cardinalidad que proporciona *MapReduce* manteniendo un modelo de calidad y precisión en conjuntos de datos complejos.

Como se ha indicado anteriormente, EDGAR lleva a cabo el aprendizaje de reglas empleando un conjunto de AGLs, en lo que cada uno de ellos trabaja sobre una fracción del conjunto completo de datos. Este modelo de datos distribuido es parcialmente compatible con el paradigma *MapReduce*, ya que el aprendizaje de reglas está asociado a las particiones de datos, y la ulterior extracción del clasificador final es una tarea independiente. Sin embargo, EDGAR emplea mecanismos cuya adaptación al modelo *MapReduce*, por los principios de éste, no pueden llevarse cabo, como son por ejemplo aquellos relacionados con la cooperación entre AGLs basado en el intercambio de reglas.

En el resto de esta Sección pues, se va a describir el diseño concreto empleado para el modelo EDGAR-MR.

Las fases principales de EDGAR-MR (fig. 3) son:

- Inicial, carga de datos: Preparación de conjuntos de datos para iniciar el proceso distribuido.
- Fase de aprendizaje de reglas: Es llevada a cabo por los AGLs implementados en los *Map*, y en ella, cada uno aprende un clasificador local basado en su partición de datos.
- Fase de agregación de reglas: Las reglas provenientes de los *Map* son procesadas en nodos *Reduce* para generar una única regla que agrega los casos positivos y negativos encontrados en los *Map*.
- Fase de generación del clasificador: el clasificador se genera como una lista ordenada de reglas por mayor cobertura y calidad.
- Fase de *test*: se clasifican los conjuntos de *test* con el clasificador generado para tener una medida de precisión del mismo.

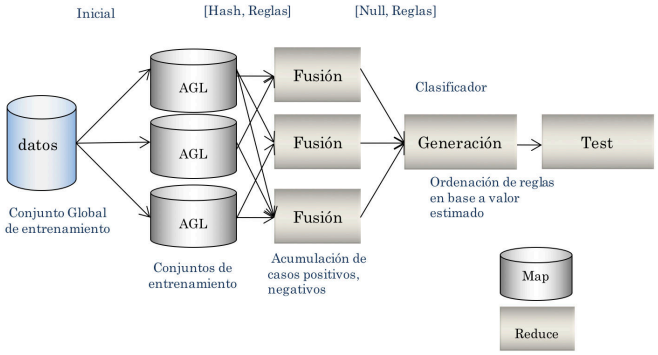

Fig. 3. Esquema *MapReduce* de las fases de aprendizaje, agregación, generacion de clasificador y *test*

A continuación se describe en detalle en cada una de las fases :

A. *Inicial, carga de datos*

El proceso de aprendizaje requiere una partición de los datos en los nodos *Map* en bloques HDFS independientes que son replicados y transferidos a otras máquinas para ser procesados por cada tarea *Map* independiente. Esta partición es realizada mediante un muestreo aleatorio de cada clase teniendo en cuenta equilibrar las clases en los AGL en caso de existir desbalanceo (figura 2). Otro paso de esta preparación inicial consiste en la recodificación en el formato de cromosoma binario que será utilizado por los AGL en los *Map*.

B. *Fase de aprendizaje de reglas (Map)*

Cada AGL genera un conjunto de de reglas que cubre completamente el conjunto de los datos en el nodo. La salida del *Map* son conjuntos (*Key, Value*), donde *Value* corresponde a la regla concatenada con el número de casos positivos y negativos que cubre y *Key* contiene un valor generado mediante una función hash para repartir la carga sobre los nodos *Reduce*. Esta función utiliza una codificación sobre los antecedentes de la regla para reunir en el mismo *Reduce* las reglas con los mismos antecedentes. La función hash codifica en un número en base 10 la codificación binaria de la clase. Por ejemplo, en un conjunto de datos con 3 atributos de tres valores cada uno y dos clases, EDGAR usa 11 bits para su codificación; por tanto el rango de la clave se encuentra entre 20 y 211. Si se implementa con 2 nodos *Reduce*. Las reglas con clave entre 0 y 210 serán asignados al primero y las claves entre 210 y 211 se asignan al segundo. El pseudocódigo 1 muestra el detalle de este *Map*.

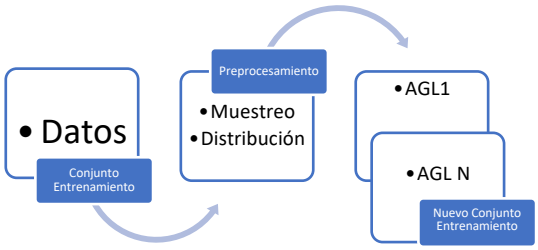

Fig. 4. Esquema de rebalanceo inicial de clases en fase de carga inicial

```
Input (Key, Value)
Key, Valor no utilizados, Map inicial.
AGL.generarClasificador(ConjuntoDatosLocal)
Para cada Regla en el clasificador
        Key' : Valor decimal (Regla)
Valor: (Regla, casos positivos, casos negativos)
Emit (Key', Valor)
Fin Para
```

Pseudocódigo 1. Fase de aprendizaje de reglas (*Map*)

C. *Fase de agregación de reglas (Reduce)*

Este proceso combina los conjuntos de reglas recibidos de cada *Map*, fusionando las reglas con los mismos antecedentes. El proceso de fusión es como sigue: acumula los casos positivos y negativos de las reglas con los mismos antecedentes para cada clase y crea una nueva regla fusionada. La nueva regla tendrá la clase de la regla con mayor valor ponderado de cobertura y calidad, denominado Π (1). Éste criterio minimiza la longitud de la regla y cobertura de casos negativos y maximiza la cobertura de casos positivos.

$$\Pi = (1+ longitud^{-1})^{-CasosNeg} * \text{CasosPos} \qquad (1)$$

```
Input (Key, Values)
        Key es una clave hash calculada en base a la regla
        Values: las reglas generadas por los AGLs en los Map
ArrayList auxReglas= new ArrayList();
Para cada Regla en values
        Si (Regla not in auxReglas) auxReglas.add(Regla)
        Sino
            AuxReglas.get(Regla).AcumulaCasos(Regla)
        Fin Si
Fin Para
Para cada Regla en auxReglas
        Emit (null, auxReglas)
Fin Para
```

Pseudocódigo 2. *Reduce*, aprendizaje. Fusión de reglas

La salida del proceso *Reduce* es un conjunto de reglas con información necesaria para el proceso de generación del clasificador: casos positivos, casos negativos (pseudocódigo 2).

D. *Fase de generación del clasificador:*

El clasificador producido EDGAR originalmente, es generado mediante la evaluación de las reglas sobre el conjunto completo de datos de entrenamiento y la posterior ordenación en una lista ordenada donde cada regla elimina las instancias cubiertas por las predecesoras.

Sin embargo, en la propuesta de este trabajo, la agregación de reglas aprendidas sobre modelos locales puede no tener la misma validez sobre el conjunto completo de datos, por ello se han implementado dos variantes dependiendo del método de cálculo del valor de la regla:

- Estimado: toma como valor de ordenación para la generación del clasificador los casos positivos y negativos calculados por la fase de agregación de reglas.
- Evaluación global: efectúa una re-evaluación de las reglas generadas en la fase de agregación de reglas mediante otro ciclo *MapReduce* con el conjunto de datos preexistente en cada *Map* (pseudocódigo 3). La salida de los *Map* será acumulada en un *Reduce* que tendrá como resultado una lista de reglas con valores positivos y negativos relativos al conjunto completo de datos (fig. 5).

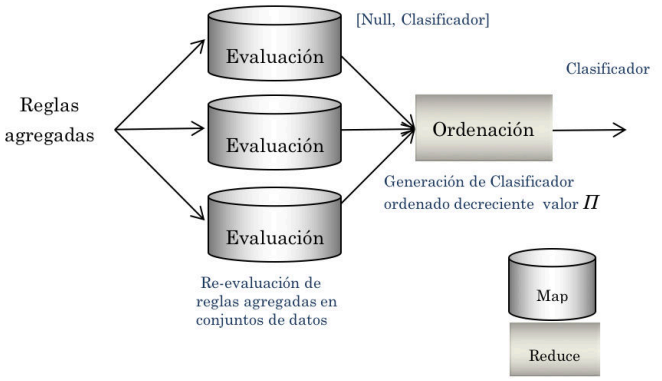

Fig. 5. Esquema *MapReduce* de la re-evaluación de reglas en variante de evaluación global en la fase de generación del clasificador

Finalmente se genera un clasificador como una lista ordenada de reglas en base al valor decreciente de Π (1).

```
Input (Key, Value)
Value es conjunto de reglas generado por el Reduce de agregación de reglas
ArrayList auxReglas = new ArrayList()
auxReglas, auxpi = GenerarClasificadorGredy(Values, Datos de test Local)
posición=0;
Para cada Regla en auxReglas
  posición ++
  Valor = Regla + Auxpi + casos positivos y negativos + posición
Fin Para
```

Pseudocódigo 3. Map de fase de evaluación global

E. *Fase de test*

Habitualmente se siguen esquemas de aprendizaje que reservan un 10% o un 20% del tamaño del conjunto de datos para *test*, en alguna de las configuraciones propuestas para minimizar la fractura de datos en el proceso de aprendizaje. En conjuntos de datos grandes, la medida de la precisión del clasificador generado implica en si misma la asignación de grupos de datos siguiendo un esquema de distribuido de datos y por tanto la ejecución de un nuevo ciclo *MapReduce*.

Inicialmente se carga en los nodos el conjunto de *test*. El *Map* se encarga de aplicar el clasificador generado sobre cada ejemplo generando como salida una línea para cada instancia con el valor real y el predicho (Fig. 6). El proceso *Reduce* acumula los aciertos y fallos en cada ejemplo respecto de la clase predicha generando un informe que puede ser utilizado para realizar distintas variantes de cálculo de la precisión sobre clasificadores.

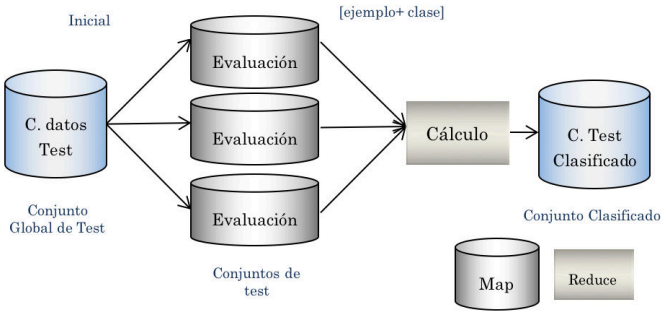

Fig. 6. Cálculo de la precisión en fase *test*

IV. ESTUDIO EXPERIMENTAL

Esta sección muestra una experimentación tentativa sobre un conjunto de *datasets* representativos para valorar la validez de la propuesta y un algoritmo de referencia no distribuido. Se estimará la calidad del clasificador generado en base al la interpretabilidad del mismo (número de reglas) y precisión (media geométrica). Asimismo se mostrará la escalabilidad del mismo según el rendimiento alcanzado respecto del número de *Maps*.

A. *Entorno de Experimentación*

El entorno de experimentación se basa en un cluster Hadoop compuesto por 13 nodos. Cada nodo es una máquina virtual sobre un servidor basado en memoria común y multiprocesador: 192GB de memoria RAM, 40 núcleos con *hyperthreading* y sistema operativo CentOS 7 de 64 bits. Todo el desarrollo se ha realizado en Java. Las características de las máquinas virtuales son:

Nodo maestro:

- Sistema operativo CentOS 7 64 bits, y 8 núcleos.
- 16 GB de memoria RAM.
- 20 GB de almacenamiento en disco configurado en modo dinámico.

Nodos esclavos:

- Sistema operativo CentOS 7 64 bits, y 4 núcleos.
- 12 GB de memoria RAM.
- 20 GB de almacenamiento en disco configurado en modo dinámico.

Respecto de los conjuntos de prueba se han seleccionado *datasets* balanceados y no balanceados con distintas cardinalidades y dimensionalidades. Éstos se encuentran disponibles en los repositorios UCI [12] y KEEL [13]. Los conjunto de datos con atributos continuos han sido discretizados para EDGAR-MR usando Chi2-Merge [14]. Las tablas muestran valores medios de ejecuciones con 10 semillas y 5x2 particiones de datos.

TABLA I. CARACTERÍSTICAS DE LOS CONJUNTO DE DATOS

Nombre	Instancias	Atributos	IR
Mushroom	8124	23	0.93
Page-blocks0	5472	10	8.77
Segment0	2308	19	6.01
Yeast3	1484	8	8.11

Los algoritmos genéticos utilizados como aprendedores en los *Map* (AGLs) siguen la estructura de operadores de EDGAR con la siguiente configuración:

- Probabilidad de Mutación = 0.01
- Tamaño de población = 40
- Número de iteraciones de parada = 500

B. *Análisis*

En la tabla II se observa que la precisión en *test* con media geométrica de las dos modalidades de aprendizaje, con la opción de balanceo de clases a los nodos mediante redistribución aleatoria de las instancias, sin balanceo y con aplicación del método de balaceo SMOTE [11] respecto del algoritmo de referencia C4.5 para una distribución de 12 nodos. El acrónimo NA (*No Aplica*) en la columna del conjunto de datos Mushroom refleja el hecho de que no se aplica el método de rebalanceo por tratarse de un dataset balanceado en origen. Para facilitar la visualización de los datos se ha sombreado levemente la fila correspondiente a la variante EDGAR-MR Global en las tablas II a IV.

En relación de la comparación entre las dos variantes, EDGAR-MR Estimado y Edgar-MR Global, se aprecia que la precisión de la variante con evaluación global alcanza en general las mismas cotas de precisión que el estimado. Acerca de la eficacia del método de rebalanceo, se observa que si bien la aplicación previa de balanceado con SMOTE, permite que EDGAR-MR alcance mejores niveles de precisión (muy apreciable en la precisión alcanzada sobre *Yeast3* para la variante global), el método de redistribución puede ser una alternativa válida, que para algunos conjunto de datos, puede llegar a superar al estándar de facto en el preprocesamiento para desbalanceadas, SMOTE. Se puede observar este caso en las precisiones alcanzadas sobre *Page-bloks0*, que son superadas por la variante de rebalanceo mediante la distribución balanceada de clases a los nodos (redistribución).

Mushroom es típicamente un conjunto de datos robusto frente al particionamiento de datos para su aprendizaje, o dicho de otra forma no tiene un gran riesgo de fractura de datos por la división de los mismos previa al proceso de aprendizaje. Sin embargo, se observa una diferencia apreciable entre las variantes estimada y global en el número de reglas sobre este conjunto de datos, aunque la precisión sea similar en ambos. Esto puede ser debido a la representación del mismo concepto en diferentes Maps por reglas equivalentes, que en la variante global elimina, eligiendo sólo la mejor representante del concepto y eliminando las reglas parcial o totalmente redundantes.

TABLA II. PRECISIÓN COMPARADA EN MEDIA GEOMÉTRICA

Algoritmo	Balanceo	Mushroom	Page-blocks0	Segment	Yeast3
EDGAR- MR Estimado	Sin Banlanceo	99,8	80.84	96.55	88.97
	SMOTE	NA	83.96	97.85	88.00
	Redistribución	NA	86.39	97.02	91.01
EDGAR- MR Global	Sin Balanceo	99,9	76.08	97.23	52.07
	SMOTE	NA	81.62	98.59	87.15
	Redistribución	NA	84.96	97.14	91.30
C4.5	Sin	100	97.33	99.18	94.14
	SMOTE	NA	94.33	99.22	93.53

Relativo a la comparación con otros algoritmos de clasificación, se comprueba que el algoritmo alcanza niveles próximos a al algoritmo de referencia C4.5 , sin llegar a superarlo. Probablemente el uso de conjunto de datos continuos no sea el más apropiado para un algoritmo de representación discreta y la discretización pueda afectar a la calidad del mismo.

En cuanto al número de reglas se puede observar en la tabla III, que el número de reglas generada por la variante global es claramente inferior al de la variante estimada, resultando en un clasificador más compacto. El número de reglas no se ve afectado de manera general por el método de desbalanceo, aunque se aprecia una ligera mejora del numero de reglas en el balanceo por redistribución frente a los otros.

TABLA III. NÚMERO DE REGLAS DEL CLASIFICADOR

Algoritmo	Balanceo	Mushroom	Page-blocks0	Segment	Yeast3
EDGAR-MR Estimado	Sin Balanceo	20	530	316	240
	SMOTE	NA	559	320	220
	Replica	NA	510	285	230
EDGAR-MR Global	Sin Balanceo	15	142	46	52
	SMOTE	NA	150	43	56
	Réplica	NA	144	45	49

Finalmente , en cuanto a la escalabilidad, se puede observar en la tabla IV, que existe una mejora o *speed-up* (ratio tiempo proceso original–paralelo) que dista mucho del ideal, 1/#nodos. Los tiempos indican minutos por cada configuración de nodos.

En el detalle de las ejecuciones observamos que la mayor parte del tiempo de proceso (en torno a un 70%) se consume en las tareas propias de preparación de datos, codificación en el formato de instancias y aquellos procesos en los reduce, y que en esta experimentación es un proceso único: agregación y fusión de reglas, generación de clasificador global y generación de los valores de *test* con el clasificador, siendo este tiempo similar en todas las configuraciones.

TABLA IV. TIEMPOS MEDIOS POR EJECUCIÓN

Algoritmo	#Nodos	Mushroom	Page-blocks0	Segment	Yeast3
EDGAR-MR Estimado	4	5,30	7,34	5,30	4,23
	8	4,45	6,68	4,45	3,81
	12	3,87	6,23	3,87	3,24
EDGAR-MR Global	4	5,81	7,71	5,81	4,65
	8	5,03	6,75	5,03	4,28
	12	4,34	6,34	4,34	4,02

V. CONCLUSIONES

EDGAR propuso un método eficiente para trabajar sobre conjuntos de datos particionados mediante la colaboración de algoritmos genéticos y un clasificador compacto basado en una lista ordenada validada sobre un conjunto de datos global. Sin embargo, con un número de particiones elevado disminuye la influencia de la comunicación en tiempo de aprendizaje. Asimismo la validación central, aunque de menor entidad algorítmica, supone un cuello de botella directamente ligado a la capacidad de la memoria del nodo central para albergar el conjunto completo de datos. Esta propuesta mejora la escalabilidad del modelo mediante un proceso de evaluación escalable mientras conserva el algoritmo genético de aprendizaje del original. Por otro lado no implementa la comunicación entre nodos incoherente con la implementación *MapReduce* seguida e ineficiente con un numero creciente de particiones.

Este trabajo presenta una arquitectura novedosa para la implementación de algoritmos evolutivos de clasificación basados en reglas, en el que el antecedente de la regla aprendida en distintas particiones forma una clave que permitirá su posterior fusión. Se propone asimismo un proceso de reevaluación que aprovecha los datos ya particionados para mejorar la calidad del clasificador.

Los resultados muestran una escalabilidad creciente, aunque mejorable, con el uso de un mayor numero de fases *Reduce* en las etapas de agregación. La comparación con un algoritmo de referencia parecen mostrar que la discretización afecta a la precisión del clasificador, siendo más adecuado para conjunto de datos nominales, por lo que en futuros trabajos se planteará la posibilidad de trabajar con datos continuos en los nodos para mejorar su precisión.

AGRADECIMIENTOS

Este trabajo ha sido financiado por el Ministerio de Educación y Ciencia dentro del Proyecto TIN2017-89517-P.

REFERENCIAS

[1] Rodríguez M., Escalante D. M., Peregrín A. (2011). Efficient distributed genetic algorithm for rule extraction. Applied soft computing, 11(1), 733-743.

[2] Dean J., Ghemawat S. (2008): MapReduce: Simplified Data Processing on Large Clusters. Commun ACM 51, 107–113.

[3] White T.(2009): Hadoop: The Definitive Guide. 1st ed.Sebastopol, CA: O'Reilly .

[4] Giordana A., Saitta L. (1994). Learning disjunctive concepts by means of genetic algorithms. Proceedings of the International Conference on Machine Learning, 96-104.

[5] Lopez L. I., Bardallo J. M., De Vega M. A., Peregrin A. (2011). REGAL-TC: a distributed genetic algorithm for concept learning based on REGAL and the treatment of counterexamples. Soft Computing, 15(7), 1389-1403.

[6] Anglano C., Botta M. (2002). NOW G-Net: learning classification programs on networks of workstations. IEEE Transactions on Evolutionary Computation, 6(5), 463-480.

[7] Svetnik V., Liaw A., Tong C., Culberson J. C., Sheridan, R. P., Feuston, B. P. (2003). Random forest: a classification and regression tool for compound classification and QSAR modeling. Journal of chemical information and computer sciences, 43(6), 1947-1958.

[8] Río S., López S., Benítez J.M., Herrera F. (2014): On The Use of MapReduce for Imbalanced Big Data using Random Forest. Information Sciences 285 112-137.

[9] Cantú-Paz E. (1998). A survey of parallel genetic algorithms. Calculateurs paralleles, reseaux et systems repartis, 10(2), 141-171.

[10] Peralta D., del Río S., Ramírez-Gallego S., Triguero I., Benitez J. M., Herrera F. (2015). Evolutionary feature selection for big data classification: A MapReduce approach. Mathematical Problems in Engineering.

[11] Chawla N. V., Bowyer K. W., Hall L. O., Kegelmeyer W. P. (2002). SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research, 16, 321-357.

[12] Asuncion A., Newman D. (2007). UCI machine learning repository.

[13] Alcalá-Fdez J., Fernández A., Luengo J., Derrac J., García S., Sánchez L., Herrera F. (2011). Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework. Journal of Multiple-Valued Logic & Soft Computing, 17.

[14] Liu H., Setiono R. (1995). Chi2: Feature selection and discretization of numeric attributes. In Tools with artificial intelligence, 1995. proceedings., seventh international conference on IEEE, 388-391.