Un modelo difuso lingüístico adaptativo para regresión con selección de reglas en Big Data

Antonio Ángel Márquez Ana María Roldán
Departamento de Tecnologías de la Información
Universidad de Huelva
Huelva – España
amarquez, amroldan@dti.uhu.es

Francisco Alfredo Márquez Antonio Peregrín
C.E. Avanzados en Física, Matemáticas y Computación
Universidad de Huelva
Huelva – España
alfredo.marquez, peregrin@dti.uhu.es

***Abstract*—En el diseño de sistemas basados en reglas difusas para regresión, son bien conocidos los modelos que permiten obtener la base de reglas a partir de conjuntos de datos por cubrimiento, así como los que posteriormente permiten mejorar la precisión del sistema empleando, por ejemplo, un ajuste del operador de defuzzificación adaptativo que, además, permita obtener un conjunto de reglas más compacto y con mejor nivel de cooperación. En este trabajo se lleva a cabo este proceso en entornos de *Big Data*, es decir, adaptando los algoritmos a esta tecnología para que se puedan utilizar partiendo de conjuntos de ejemplos que no serían manejables sin disponer de una infraestructura escalable.**

Keywords—regresión difusa; bases de reglas; defuzzificación adaptativa; Big Data; MapReduce.

I. Introducción

En el área de los *Sistemas Basados en Reglas Difusas* en general, y para modelado y control en particular, existe una considerable cantidad de trabajos que emplean algoritmos evolutivos para aprender y/o ajustar elementos del sistema, que van por ejemplo desde los de la Base de Conocimiento (BC) como el aprendizaje de la Base de Reglas (BR), o ajuste de la Base de Datos (BD), hasta incluso los operadores, como en [1], [2]. A todo este ámbito se le llamó el de los *Sistemas Difusos Evolutivos* [3], [4], [5], y en él se han desarrollado pues, gran cantidad de recursos muy útiles desde hace más de dos décadas. Estos algoritmos utilizan conjuntos de ejemplos desde los que aprender o evaluar los distintos citados elementos.

La posibilidad de emplear conjuntos de ejemplos más grandes con algunas de estas técnicas ha sido tratada a veces por distintos autores: cuando los conjuntos de ejemplos son muy grandes, las técnicas utilizadas pueden necesitar modificaciones o re-implementaciones para que sigan siendo aptas. Tradicionalmente, las mejoras en este sentido [5] se han orientado a la reducción de datos, al uso de aproximaciones o estimaciones, a la introducción de heurísticas en los algoritmos para mejorar su eficiencia, y en algunos pocos casos, al uso de computación distribuida. Sin embargo, es con la llegada de los recursos para afrontar el tratamiento de grandes conjuntos de datos en entornos de Big Data, cuando se dispone de herramientas para hacer verdaderamente escalables algunas de estas técnicas tradicionales.

Este trabajo trata precisamente del uso de recursos computacionales distribuidos del ámbito del *Big Data*, para emplear una técnica tradicional, como es el ajuste de la defuzzificación adaptativa en regresión y control difusos, cuando el conjunto de ejemplos es grande, o simplemente cuando el aprovechamiento de estos recursos nos permite realizar el mismo trabajo en tiempos menores, lo cual por ejemplo con el uso de algoritmos evolutivos, puede ser una interesante mejora.

II. Conocimientos Previos

En esta Sección se van a repasar brevemente dos elementos que empleamos en nuestra propuesta: la defuzzificación adaptativa, y la tecnología *MapReduce*.

A. Defuzzificación Adaptativa Evolutiva

La defuzzificación adaptativa es un recurso tradicional y bien conocido que permite adaptar el comportamiento del mecanismo de obtención de una salida específica desde la aportación de cada regla disparada en la fase de inferencia en regresión difusa, consiguiendo mejorar la precisión del modelo resultante, pues cada regla participa en dicho proceso de una forma específica y diferente al resto [1],[2].

Es posible encontrar múltiples propuestas en la literatura para conseguir regular o adaptar el mecanismo de defuzzificación mediante el uso de parámetros. Sin embargo, lo más frecuente suele ser utilizar la expresión (1), dados sus buenos resultados, su sencillez computacional y facilidad de implementación [2],

$$y_0 = \frac{\sum_i^N h_i \cdot \alpha_i \,.\, CG_i}{\sum_i^N h_i \,.\, \alpha_i}, \qquad (1)$$

donde h_i es el *matching degree* o grado de emparejamiento, α_i es el parámetro que se aplica con la regla i-esima, R_i, con $i=1$ *hasta* N, y CG_i es el Centro de Gravedad de la figura obtenida de la inferencia con la regla R_i. Como se puede ver, se trata de la expresión de un método de defuzzificación que actúa convirtiendo la aportación de cada regla en un número, y posteriormente calculando una media ponderada, es decir, del llamado *Modo-B*. El parámetro α_i pues, tomando valores en el intervalo [0,1], permite a su vez ponderar o reducir el efecto del

grado de emparejamiento, h_i, como si de un peso, w_i, se tratase [6].

Para conseguir el ajuste de los N citados α_i parámetros, es decir, reducir la diferencia entre la salida del sistema difuso y el conjunto de ejemplos, se podrían emplear diferentes métodos. En este sentido, el uso de algoritmos evolutivos suele ser una buena opción. La codificación empleada en este caso es de tipo real, pues como se ha indicado, el valor de los parámetros α_i se encuentra entre 0 y 1.

Obsérvese que este mecanismo de ajuste paramétrico no sólo permite que la aportación de cada regla sea regulada sino que a través de él, se consigue mejorar el nivel de cooperación entre las reglas de la BC [1]. Incluso llevado al extremo, la defuzzificación adaptativa puede emplearse como mecanismo de selección y reducción de reglas: si una regla tiene un valor 0 para su parámetro, dicha regla no participa o deja de ser considerada. El hecho de disponer de un mecanismo que permita eliminar reglas es particularmente interesante cuando dichas reglas provienen de un proceso de aprendizaje que no las ha valorado en su conjunto, como suelen ser los métodos basados en cubrimiento de ejemplos, donde cada regla se obtiene con criterios de calidad individual.

Cabe mencionar también en este sentido, que en [7] se propone un mecanismo para facilitar la eliminación de reglas con aportación baja, basado en eliminar aquellas reglas cuyo parámetro desciende por debajo de cierto umbral (es decir, no sólo cuando el valor es exactamente 0). En el citado trabajo, también se elimina el parámetro de las reglas cuyo valor está cercano a 1, suponiendo por tanto una reducción de la carga computacional y reduciendo la complejidad de la BC, pues se reduce el número de reglas, y el número de reglas con parámetro asociado.

B. *Tecnologías para Big Data*

En los últimos años se han desarrollado un conjunto de tecnologías para el almacenamiento, gestión y tratamiento de grandes conjuntos de datos de forma distribuida y escalable. A estas tecnologías se les suele llamar genéricamente, tecnologías para *Big Data* [8] y los tres pilares en los que se apoyan estas tecnologías actualmente son:

- Sistemas de archivos distribuidos, donde los grandes archivos de datos se almacenan divididos entre varios servidores, tales como el popular *Apache Hadoop Distributed File System* (HDFS) [10].
- Paradigmas de programación tales como *MapReduce* [11] o *Pregel* [12], que permiten implementar procesos de computación en *clusters* de computadoras.
- Entornos para *clusters* de computación como *Apache Hadoop* [10] o *Apache Spark* [13] que nos permiten organizar y gestionar grupos de computadoras tanto como sistemas de almacenamiento (a través de sistemas de archivos distribuidos) como estructuras de procesamiento de datos (implementando modelos de programación distribuida) eficientemente.

El paradigma de programación *MapReduce* lo introdujo Google en 2004 [11]. Una de sus mejor conocidas y empleadas implementaciones en código abierto es *Apache Hadoop*, la cual, implementa características como el almacenamiento y procesamiento escalables, los cuales se pueden usar de forma relativamente sencilla, tienen elevada tolerancia a fallos, alta disponibilidad, redundancia de datos automática, etc. A nivel de procesamiento, *Hadoop* está pensado para emplearse procesando datos en una sola pasada, esto es, no es eficiente implementando múltiples pasadas de procesamiento sobre los datos, así como tampoco está pensando para procesamiento de datos interactivamente. *Apache Spark* [13] sin embargo, sí resuelve estas situaciones.

Apache Spark es igualmente un entorno para la programación distribuida de código abierto, ideado como un paso adelante en cuanto a flexibilidad y eficiencia. No sólo permite utilizar el modelo de computación *MapReduce*, si bien considerablemente más rápido [13] que *Hadoop*, sino también *Pregel* [12] y el suyo propio. Pero una de las ventajas más interesantes de *Spark* es que permite implementar de forma eficiente el procesamiento de datos iterativo o multi-pasada, debido al uso particular de la memoria RAM que implementa, a través de una abstracción de memoria distribuida conocida como RDD (*Resilent Distributed Dataset*) [13], que le permite reducir el acceso a disco intermedio mejorándose así dramáticamente su rendimiento. Los RDDs son conjuntos de datos que se ubican físicamente troceados entre distintos servidores del *cluster*, y que pueden ser procesados en paralelo. Los programadores emplean estos RDDs a través de transformaciones, que los procesan produciendo otros RDDs, o bien mediante acciones, que les aplican un procesamiento que ofrece como salida un valor o cálculo. La tolerancia a fallos se obtiene precisamente gracias a los citados RDDs, ya que estos pueden ser reconstruidos automáticamente si por alguna razón, se perdiesen durante su procesamiento.

Los programas implementados para ser ejecutados en *Spark* se componen de una única aplicación *driver* que se ejecuta en el nodo maestro, y un conjunto de tareas que corren en paralelo en los *ejecutores* sobre los nodos trabajadores del *cluster*, devolviendo sus resultados al *driver*.

Cabe destacar que *Spark* también permite realizar trabajos de computación distribuida de forma interactiva, es decir, instrucción a instrucción desde intérprete de comandos, y que puede utilizar HDFS como sistema de archivos distribuido.

El mecanismo que se propone en este trabajo utiliza *Spark* y el modelo de computación *MapReduce*.

III. Modelo con Post-Procesamiento para el Ajuste de la Defuzzificación y Selección de Reglas

En esta Sección se propone un modelo distribuido escalable implementado en el entorno *Spark*, que consta de dos etapas: en una primera etapa se obtiene la BR basándonos en la versión *MapReduce* de un método archiconocido para la obtención de reglas por cubrimiento de ejemplos, y una segunda en la que también de modo distribuido y escalable, se ajustarán mediante un algoritmo evolutivo los parámetros de la defuzzificación, empleando umbrales para seleccionar el subconjunto de reglas que mejor cooperan.

A. *Obtención Escalable de la BR desde Conjuntos de Datos*

El llamado "Método de Wang y Mendel" dirigido por datos [14] (WM en adelante) es un algoritmo extraordinariamente conocido para obtener BRs partiendo de un conjunto de ejemplos. En un trabajo previo [15], propusimos una versión adaptada del mismo basada en el uso de *MapReduce*, que llamamos *WM-Escalable*. Los resultados que produce son idénticos a los que produciría la metodología secuencial original de WM, y resumimos a continuación:

Los conjuntos de datos de ejemplos son troceados y repartidos entre los servidores del *cluster* para ser procesados de forma separada. En este caso, para obtener la BR usando el método de WM, sólo se necesita un esquema de una sola pasada, es decir, una sola fase *Map* y a continuación una fase *Reduce*. Los elementos son:

- Programa *Driver*: El programa *driver* ejecutado en el nodo maestro, en primer lugar, crea la BD utilizando un número predefinido de funciones de pertenencia distribuidas uniformemente en el universo de discurso de cada variable. En segundo lugar, el conjunto de entrenamiento se divide en *n* particiones disjuntas con el mismo número de instancias, y cada una de estas particiones se distribuye entre los nodos trabajadores del *cluster* junto con la citada BD.
- Función *Map*: Las funciones *Map* consisten en la aplicación de la metodología original de WM por parte de los procesos *ejecutores* sobre nodos trabajadores, cada uno en su subconjunto de ejemplos, que quedan así cubiertos por alguna de las reglas generadas. El método de WM consigue esto en primer lugar cubriendo cada ejemplo con una regla difusa que utiliza las etiquetas que tienen el mayor grado de pertenencia para cada variable, y además le asigna un valor que es el grado de emparejamiento de esa regla con ese ejemplo, el cual será empleado en una segunda fase para reducir el conjunto de reglas, es decir, para no tener tantas reglas como ejemplos sino sólo un subconjunto combinado de las reglas más representativas. La salida de estos procesos *Map* se produce en términos de pares *clave-valor* (siendo recibida por el *driver*), donde la **clave** son *las etiquetas de los antecedentes de las reglas halladas*, y el **valor** asociado son *el consecuente de las reglas encontradas, junto con el grado de emparejamiento de las reglas*.
- Función *Reduce*: La tarea de las funciones *Reduce* en este esquema es la de recibir las distintas reglas producidas por las funciones *Map*, (RB_i), y agruparlas para construir la BR definitiva. Para ello, debe eliminar las reglas repetidas, es decir, las que tienen los mismos antecedentes y el mismo consecuente, y resolver las reglas contradictorias, es decir, las que con los mismos antecedentes tienen distinto consecuente, lo cual se lleva a cabo eligiendo la que viene acompañada por el mayor grado de emparejamiento. En términos del diseño *clave-valor* de *MapReduce*, la función *Reduce* recibe una lista de valores intermedios compuesta por pares agrupados por claves (**clave**: *etiquetas de la parte antecedente de una regla*, y **lista de valores**: *consecuentes para esa parte antecedente de una regla y sus grados de emparejamiento*). La BR final está pues compuesta por el conjunto de reglas obtenidas por las funciones *Reduce*.

B. *Aprendizaje Escalable Evolutivo de los parámetros para la Defuzzificación Adaptativa con Selección de Reglas*

En esta Subsección se describe la parte original de este trabajo, donde se propone un método evolutivo para aprender o ajustar los valores de la defuzzificación adaptativa utilizando conjuntos de ejemplos muy grandes de forma eficiente, y especialmente apropiada al uso de BRs aprendidas mediante métodos de cubrimiento de ejemplos, ya que incorpora un mecanismo de selección de reglas, tal como se introdujo previamente en la Sección II, al que llamaremos *Defuzzificación Adaptativa Evolutiva con Selección de Reglas Escalable* (abreviado como *E-DAESR*). La originalidad en este trabajo está, no en el diseño evolutivo de defuzzificación adaptativa [2], sino en la escalabilidad de la propuesta. No repetiremos pues los detalles ya descritos previamente sino que nos centraremos en el diseño *MapReduce* del mecanismo, el cual, es equivalente exactamente al modelo secuencial original [2].

1) *Algoritmo Evolutivo*

Previamente, se va a describir el esquema evolutivo empleado, ya que de su detalle depende la posterior descripción del mecanismo para la escalabilidad de la propuesta. Se basa en el modelo CHC [16]. Sus componentes son:

Esquema de codificación y población incial

El esquema de codificación empleado es el de [1], que consiste en un cromosoma de tipo real con N genes, α_i, que representan cada uno al parámetro asociado a cada regla R_i de la RB, y cada uno de ellos toma valores en el intervalo [0,1].

$$C = (\alpha_1, \ldots, \alpha_N) \mid \alpha_i \in \{0, 1\}$$

La población inicial se inicializa con todos los genes de cada cromosoma elegidos aleatoriamente en el intervalo anteriormente citado, excepto los de uno, cuyos genes se fijan a 1, lo que equivale al método estándar conocido como *CG ponderado por el grado de emparejamiento*.

Evaluación

La evaluación de los cromosomas consiste en tasar el modelo difuso con los parámetros de ese cromosoma. Dado que lo que se pretende resolver con esta propuesta es precisamente el uso de conjuntos de ejemplos muy grandes, este proceso es el que se realiza de forma distribuida por los *ejecutores* de *Spark*, como se describirá posteriormente. Para maximizar la precisión, minimizamos el Error Cuadrático Medio (MSE) cuya expresión es (2), donde $S[i]$ denota a los diferentes modelos difusos probados. Para ello, se utiliza un conjunto de evaluación compuesto por P pares de datos numéricos $Z_k = (x_k, y_k)$, $k=1,..,P$, siendo x_k los valores de las variables de entrada, e y_k los valores correspondientes de las variables de salida.

$$MSE(S[i]) = \frac{\frac{1}{2}\sum_{k=1}^{N} (y_k - S[i](x_k))^2}{N} \qquad (2)$$

Destacamos que, para llevar a cabo la selección de reglas mediante umbrales como en [7] se tiene en cuenta sólo y

simplemente cuando se evalúa el sistema difuso, es decir, los valores de los parámetros en los cromosomas se mantienen, pero por debajo de cierto umbral (se ha utilizado el valor 0.1), no se usa la regla, al igual que por encima de cierto umbral (se ha usado 0.9), se considera que el peso es 1.

Operador de cruce y criterio de rearranque

Empleamos el operador de cruce BLX-α [17], con $\alpha = 0.5$, y umbral inicial *L/4*. Cuando el umbral es menor que 0, se rearranca, construyéndose la población aleatoriamente excepto para el mejor de los padres de la población anterior.

2) Esquema MapReduce para la Defuzz. Adaptativa

A continuación se describe el diseño MapReduce realizado para construir el modelo *E-DAESR,* ilustrado en la Figura 1.

- Programa *Driver*: El programa *driver*, ejecutado en el nodo maestro, es el que se ocupa de llevar acabo la mayor parte del algoritmo evolutivo que aprende los parámetros de la defuzzificación asociados a cada una de las reglas R_i de la BR, cediendo el proceso más pesado, la evaluación con el conjunto de ejemplos, a la función *Map*, que se ejecuta de forma distribuida en el *cluster* en cada iteración del algoritmo evolutivo. El programa *driver* al inicio, divide y distribuye el conjunto de ejemplos en tantas unidades diferentes como elementos de procesamiento tenga el *cluster*. Asimismo, también distribuye una copia de la BC y la población completa de cromosomas a evaluar.
- Función *Map*: La evaluación de la población de cromosomas del algoritmo evolutivo es llevada a cabo por parte de los procesos *ejecutores* dentro de los nodos *trabajadores* del *cluster*, con su fracción del conjunto de ejemplos, obteniéndose por tanto el MSE para ese subconjunto. La función *Map* produce una lista de pares *clave-valor* intermedios que son devueltos al proceso *driver* tras su ejecución, de modo que la **clave** son *los cromosomas (parámetros asociados con cada regla)* y el **valor** es *el valor de evaluación (MSE) obtenido por la BR con los parámetros para la defuzzificación que lleva cada cromosoma.*
- Función *Reduce*: También son realizadas por los *ejecutores*: una o varias funciones *Reduce* reciben los resultados de los *Map*s y los combinan para tener el resultado completo de la evaluación para cada cromosoma. En términos de pares *clave-valor*, las funciones *Reduce* reciben una lista de valores intermedios agregados por la clave, de modo que la **clave** *es el cromosoma (los valores de los parámetros para cada regla)*, y como **lista de valores**, una *lista de la evaluación de cada cromosoma en cada partición de datos*, y ofrecen una lista de cromosomas con su valor de evaluación con el conjunto de ejemplos completo. La evaluación obtenida por todos los cromosomas es enviada al programa driver para que continúe el proceso evolutivo.

IV. Estudio Experimental

En esta Sección se describe el estudio experimental llevado a cabo para comprobar el funcionamiento del modelo propuesto, tanto en cuanto a precisión como a escalabilidad.

En primer lugar, se hará referencia a los conjuntos de datos empleados, posteriormente se describirá la configuración seleccionada para el estudio experimental y los test estadísticos, y finalmente se mostrarán los resultados y su análisis, así como el estudio de escalabilidad.

Se van a utilizar 10 problemas con diferente nivel de dificultad, por ejemplo, con distinto número de variables e instancias, etc. Provienen del repositorio KEEL [18], y se muestran en la Tabla I junto con sus características.

A. Configuración de los Experimentos

En cuanto la configuración de los experimentos, se ha utilizado validación cruzada de orden 5, es decir, se han hecho 5

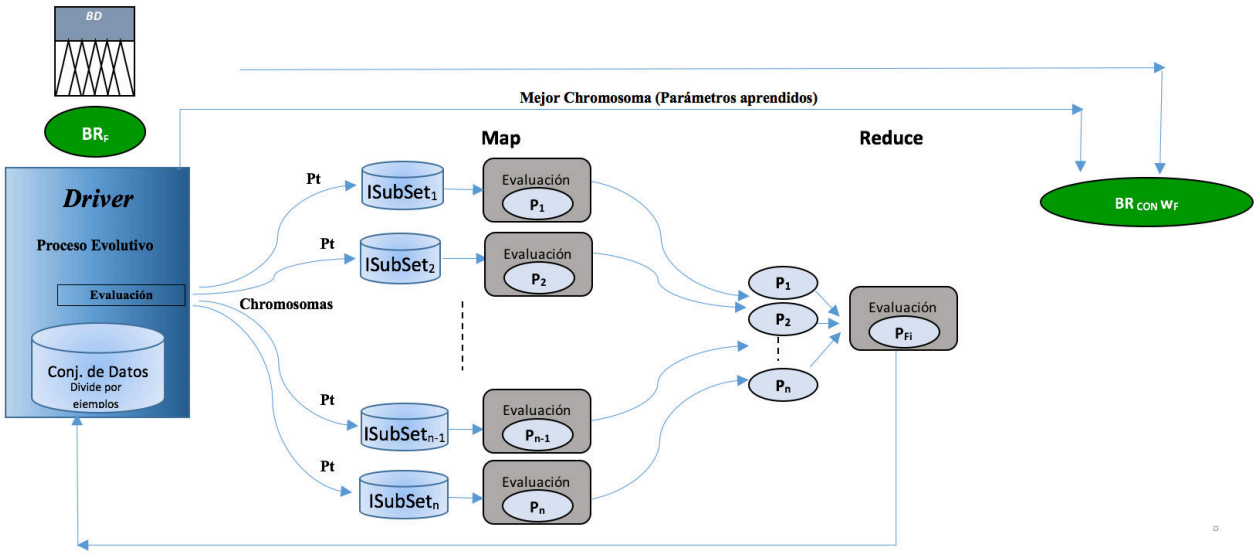

Fig. 1. Esquema de E-DAESR

TABLE I. CONJUNTOS DE DATOS UTILIZADOS, DISPONIBLES EN EL REPOSITORIO KEEL (HTTP://SCI²S.UGR.ES/KEEL/DATASETS.PHP)

Problem	Abbreviation	Instances	Variables
Delta-elv	DELV	9517	6
California	CAL	20640	8
Mv	MV	40768	10
House	HOU	22768	16
Elevators	ELV	16599	18
Compactiv	CA	8192	21
Pole	POL	14998	26
Puma32	PUM	8192	32
Airelons	AIL	13750	40
Tic	TIC	9822	85

particiones aleatorias de los conjuntos de datos, cada una con un 20%, y la combinación de 4 de ellos, es decir, el 80%, se han usado como conjunto de entrenamiento, mientras que los restantes son el conjunto de prueba.

En cuanto a los sistemas difusos, las BCs utilizadas emplean 3 etiquetas lingüísticas triangulares en todos los casos. El operador utilizado como conjunción y como inferencia para ambos casos es la t-norma del mínimo. El método de defuzzificación empleado para la obtención de la BR, por tanto, no adaptativo, es el *CG ponderado por el grado de emparejamiento*, mientras que el adaptativo de la propuesta es el de la expresión (1) (los cuales son equivalentes cuando el parámetro de cada regla tiene el valor 1).

En cuanto al algoritmo evolutivo, cuyo modelo se ha descrito en la Sección III, se ha ejecutado 30 veces: 6 semillas para el generador de números aleatorios, y 5 particiones para cada conjunto de datos o problema. La longitud de población utilizada es 61 cromosomas; la probabilidad de cruce se ha fijado a 1, y el número máximo de evaluaciones configurado ha sido 100000.

Para realizar la experimentación hemos utilizado un *cluster virtual* formado por 17 servidores con 4 núcleos y 8 GB de RAM cada uno. Uno de ellos se ocupa de ejecutar exclusivamente el programa *driver*, es decir, actúa de nodo maestro, y los 16 restantes como trabajadores para los procesos *ejecutores*.

Para realizar los estudios de escalabilidad se ha optado por hacer las dos siguientes configuraciones del *cluster*:

- Una configuración simple, con solo 2 núcleos, para poder medir el tiempo necesario por un servidor básico.
- Una configuración en modo *cluster* con el total de los 16 servidores, tanto con 16 como con 32 núcleos.

Se han realizado test estadísticos [19] [20], para comparar los resultados, empleando un *signed-rank test* de Wilcoxon.

B. Resultados Obtenidos y Análisis

La Tabla II muestra los resultados obtenidos por el modelo propuesto con defuzzificación adaptativa escalable, *E-DAESR,* en comparación con el modelo que no la utiliza, ambos con las mismas BRs obtenidas por el método de *WM-Escalable* [15]. En particular, para cada modelo, se muestra el número de reglas obtenidas, y los MSE en entrenamiento y prueba. Observando esta información, analizamos que:

- El número de reglas que se obtienen en general empleando el método propuesto *E-DAESR* es considerablemente inferior al que se obtiene empleando el método de cubrimiento (*WM-Escalable*), por tanto, se consiguen modelos más compactos. La excepción son precisamente los problemas que por su complejidad, presentan mayor número de reglas: quizás el espacio de búsqueda es "demasiado grande" para la configuración del algoritmo evolutivo básica empleada, por lo que habría que probar otros ajustes que hiciesen la búsqueda más efectiva.
- La precisión en general es de nuevo considerablemente mejor en el modelo propuesto. Sin embargo, se observa que en algunos problemas dicha reducción es escasa, coincidiendo con los problemas que en el párrafo anterior destacábamos con una inferior reducción en el número de reglas: se hace conveniente probar configuraciones diferentes.

Las Tablas III y IV muestran los resultados de los test de Wilcoxon empleados para confirmar que la reducción del número de reglas y la mejora en la precisión son significativas.

TABLA II. WM-ESCALABLE VS. E-DAESR (LOS VALORES DE MSE EN ESTA TABLA DEBEN MULTIPLICARSE POR 10^{-6}. 10^{8}. 10^{9}.10^{-6}. 10^{-8} Y 10^{-4} PARA DELV. CAL. HOU. ELV. AIL Y TIC RESPECTIVAMENTE).

Datasets	WM-Escalable			E-DAESR		
	#R	MSE_{tra}	MSE_{tst}	*#R*	MSE_{tra}	MSE_{tst}
DELV	129.4	2.372718	2.375347	**47.1**	1.602017	**1.628900**
CAL	123.4	5.319310	5.335769	**41.2**	3.215826	**3.225589**
MV	3677.8	13.795066	13.998455	**3126.4**	6.281729	**6.669283**
HOU	756.4	16.445739	16.724978	**281.8**	9.706077	**10.455755**
ELV	508.8	16.446985	16.494120	**108.3**	9.829008	**10.052612**
CA	424.8	40.383642	40.956170	**139.0**	5.345838	**6.636041**
POL	1087.0	399.773700	401.554456	**559.1**	174.488994	**181.557838**
PUM	6553.6	0.000250	0.000589	**6500.1**	0.000242	**0.000587**
AIL	1072.0	3.450536	3.496296	**470.3**	1.893806	**2.004145**
TIC	5802.4	151.609692	500.967975	**5707.9**	126.557568	**499.817874**

TABLA III. WILCOXON TEST PARA COMPARAR LA PRECISIÓN DEL MODELO WM-ESCALABLE CON EL MODELO E-DAESR. (R+ SE CORRESPONDE CON LA SUMA DE LOS RANKINGS PARA EL E-DAESR Y R- AL WM-ESCALABLE)

Comparación	R+	R-	p-value
E-DAESR vs. WM-Escalable	0	55	0.0019532

TABLA IV. WILCOXON TEST PARA COMPARAR EL #R DEL MODELO WM-ESCALABLE CON EL DEL MODELO E-DAESR. (R+ SE CORRESPONDE CON LA SUMA DE LOS RANKINGS PARA EL E-DAESR Y R- AL WM-ESCALABLE)

Comparación	R+	R-	p-value
E-DAESR vs. WM-Escalable	0	55	0.0019532

C. Escalabilidad

Uno de los elementos más interesantes de emplear un enfoque escalable es precisamente, que escale correctamente, es decir, que aumentando el número de elementos de proceso en el *cluster,* le permita reducir los tiempos de ejecución.

Como se comentó en la descripción del estudio experimental, se ha empleado una configuración en el *cluster* con 2 núcleos como equipo básico, y posteriormente con 16 y 32. Los resultados se muestran en la Figura 2: se observa como un mayor número de elementos de cómputo, se refleja en el

speed-up (ganancia del tiempo empleado por 32 y 16 núcleos frente al empleado con sólo 2) si bien no es lineal debido a la carga computacional derivada del *framework*.

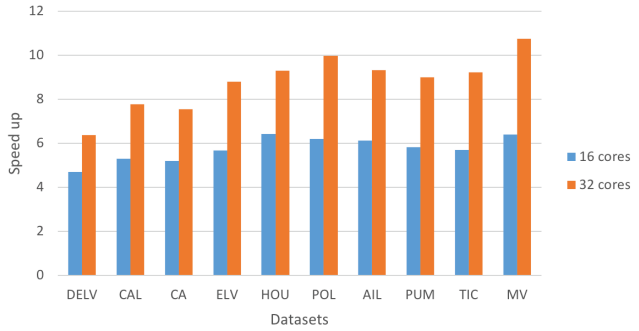

Fig. 2. Escalabilidad obtenida para cada conjunto de datos

V. Conclusiones

El objetivo planteado en este trabajo era estudiar la adaptación de la metodología del uso de defuzzificación adaptativa con selección de reglas mediante umbrales, a entornos escalables, es decir, la adaptación para su uso en *frameworks* de *Big Data*, y al modelo de computación *MapReduce* implementado en *Apache Spark*.

El resultado de este estudio preliminar es positivo, es decir, se ha conseguido un modelo equivalente (sus resultados son idénticos al modelo secuencial del que proviene) que funciona correctamente, si bien también se observa la necesidad de estudiar algunos casos, con un perfil muy claro (mayor número de reglas) en los que seguramente por falta de una configuración específica del algoritmo evolutivo, la calidad de los resultados no es tan buena, y por tanto, este particular será materia de estudio futuro.

Como se ha visto, en este trabajo se parte del uso del clásico método de WM para aprender las RBs. Este método es muy sencillo e incluso podría decirse que "*muy malo*" en cuanto a precisión y compacidad de la BR aprendida comparativamente con otros métodos, algunos de ellos evolutivos, aparecidos en la literatura durante varias décadas. Sin embargo, queremos hacer notar que lo utilizamos intencionadamente porque el modelo evolutivo empleado posteriormente precisamente permite corregir el principal defecto de las reglas aprendidas siguiendo el método de WM, que es el de la falta de cooperación entre ellas, lo cual es fundamental en regresión difusa. El modelo evolutivo empleado no sólo *gradúa* la contribución de cada regla a través del parámetro o peso, sino que busca la cooperación seleccionado las reglas más apropiadas de las encontradas por el método de WM, produciéndose así una BR final de considerable mayor calidad, más precisa y compacta, sin necesidad de usar un modelo más complejo y de mejor calidad en la primera fase. De cualquier modo, y queda pendiente para un futuro, realizar un estudio sobre cómo afecta la calidad del modelo inicial previo al post-procesamiento evolutivo distribuido propuesto empleando otras alternativas al método de WM, aunque *a priori*, por lo expuesto anteriormente, creemos que será escasa.

Agradecimientos

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

Referencias

[1] J. Alcala-Fdez, F. Herrera, F.A. Márquez, and A. Peregrín, "Increasing fuzzy rules cooperation based on evolutionary adaptive inference systems", Int J Intell Syst. 2007;22(9), pp. 1035–1064.

[2] F.A. Márquez, A. Peregrín, and F. Herrera, "Cooperative evolutionary learning of linguistic fuzzy rules and parametric aggregation connectors for Mamdani fuzzy system", IEEE Trans on Fuzzy Sys. 2007;15(6), pp.1168–1178.

[3] F. Herrera, "Genetic fuzzy systems: taxonomy, current research trends and prospects",. Evol Intell. 2008;1(1), pp.27–46.

[4] M. Fazzolari, R. Alcalá, Y. Nojima, H. Ishibuchi, and F. Herrera, "A review of the application of multi-objective evolutionary systems: Current status and further directions", IEEE Trans Fuzzy Syst. 2013; 21(1), pp.45–65.

[5] A. Fernández, V. López, M.J. del Jesus, and F. Herrera, "Revisiting evolutionary fuzzy systems: Taxonomy, applications, new trends and challenges", Knowl Based Syst. 2015;80, pp.109–121.

[6] J.S. Cho, and D.J. Park, "Novel fuzzy logic control based on weighting of partially inconsistent rules using neural network", J Intel Fuzzy Syst. 2000;8, pp.99–100.

[7] A. Márquez, F.A. Márquez, and A. Peregrín, "A Mechanism to Improve the Interpretability of Linguistic Fuzzy Systems with Adaptive Defuzzification based on the use of a Multi-objective Evolutionary Algorithm", Int. J. of Comp. Intell. Syst., 2012; 5 (2), pp. 297–321.

[8] D. Laney, "3D data management: controlling data volume, velocity and variety", META Group Research Note 6. 2001; 70.

[9] A. Fernández, S. del Río, V. López, A. Bawakid, M.J. del Jesus, J.M. Benítez, and F. Herrera, "Big Data with Cloud Computing: an insight on the computing environment, MapReduce, and programming frameworks", Wiley Interdiscip. Rev. Data Mining Knowl. Discovery. 2014;4(5), pp.380–409.

[10] T. White, "Hadoop: The Definitive Guide", O'Reilly Media, Inc.; 2012.

[11] J. Dean, and S. Ghemawat, "MapReduce: a flexible data processing tool", Commun ACM. 2010; 53(1), pp.72–77.

[12] G. Malewicz, M.H. Austern, A.J. Bik, J.C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing", In Proceedings of the ACM SIGMOD International Conference on Management of Data. 2010, pp. 135–146.

[13] M. Zaharia, R.S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, et al., "Apache spark: a unified engine for big data processing", Commun ACM. 2016; 59(11), pp.56–65.

[14] L. Wang and J. Mendel, "Generating fuzzy rules by learning from examples", IEEE Trans Syst, Man, Cybern. 1992;22(6), pp.1414–1427.

[15] A.A. Márquez , F.A. Márquez, and A. Peregrín, "A scalable evolutionary linguistic fuzzy system with adaptive defuzzification in big data", In Proceedings of the IEEE International Conference on Fuzzy System (FUZZ-IEEE). 2017; pp.1–6.

[16] L.J. Eshelman, "The CHC adaptive search algorithm: how to safe search when engaging in nontraditional genetic recombination", in G.J.E. Rawlings (Ed.), Foundations of genetic algorithms. 1991, pp.265–283.

[17] F. Herrera, M. Lozano, and A. Sánchez, "A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study", Int J Intell Syst. 2003;18, pp.309–338.

[18] J. Alcala-Fdez, L. Sánchez, S. García, M.J. del Jesus, S. Ventura, J. Garrell, et al., "Keel: a software tool to assess evolutionary algorithms for data mining problems", Soft Comput. 2009;13(3), pp.307–318.

[19] J. Demšar, "Statistical comparisons of classifiers over multiple data sets", J Mach Learn Res. 2006;7, pp.1–30.

[20] S. García and F. Herrera, "An extension on statistical comparisons of classifiers over multiple data sets for all pairwise comparisons", J Mach Learn Res. 2008;9, pp.2579–2596.