II Workshop en Big Data y Análisis de Datos Escalable (II BigDADE)

AMPARO ALONSO BETANZOS
MARÍA JOSÉ DEL JESUS
FRANCISCO HERRERA

II Workshop en Big Data y Análisis de Datos Escalable (II BigDADE)

SESIÓN 1

Una primera aproximación para la extracción de patrones emergentes en flujos continuos de datos

A.M. Garcia-Vico, C.J. Carmona, P. González, M.J. del Jesus
Departamento de Informática, Data Science and Computational Intelligence
Universidad de Jaén, Jaén, España
{agvico|ccarmona|pglez|mjjesus}@ujaen.es

***Resumen*—A día de hoy, la cantidad de información proveniente de fuentes que emiten constantemente datos es inmensa, por lo que se hace necesario la extracción automática de conocimiento para la mejora de servicios en la vida cotidiana de las personas. La búsqueda de patrones emergentes permite la descripción de las características discriminativas entre clases o la descripción de tendencias emergentes en los datos. En este trabajo se presenta un nuevo enfoque basado en un sistema evolutivo difuso para la extracción de patrones emergentes en flujos continuos de datos. Los resultados del estudio experimental muestran unos resultados prometedores para la extracción de este tipo de conocimiento en el ámbito de la minería de flujo de datos.**

***Index Terms*—Descubrimiento de reglas descriptivas supervisadas, minería de patrones emergentes, algoritmos evolutivos multi-objetivo, minería de flujo de datos.**

I. Introducción

Vivimos en la era de la información. A día de hoy, el desarrollo en las tecnologías de la información y la comunicación ha permitido un aumento exponencial en la cantidad de datos generados. Esto se debe principalmente al abaratamiento de los sistemas de almacenamiento y sensores generadores de datos [1]. Toda esta cantidad de datos contiene conocimiento muy relevante para las empresas para poder mejorar sus servicios [3] lo que ha propiciado el desarrollo en los últimos años de técnicas de extracción de conocimiento en estos enormes volúmenes de información heterogénea en lo que se conoce como *Big Data* [2]. Sin embargo, existen ámbitos de trabajo, como por ejemplo en gestión de energía [4], donde los datos muy antiguos son completamente irrelevantes. En este caso, un análisis continuo de la información conforme los datos van llegando es más interesante. A este tipo de minería de datos se le conoce como minería de flujo de datos [5].

La minería de flujo de datos tiene en cuenta varios factores que hacen que la extracción de conocimiento en este tipo de datos sea un desafío en comparación con la minería de datos tradicional, como por ejemplo la actualización continua del modelo de aprendizaje o la necesidad de desechar información antigua [6], [5]. Además, muchos sensores y fuentes de datos poseen una tasa de refresco muy elevada (del orden de Khz) que implican además un aprendizaje lo más rápido posible [7].

La minería de patrones emergentes (EPM) [8], [9] es una tarea de minería de datos encuadrada dentro del marco de tareas denominado "descubrimiento de reglas descriptivas mediante aprendizaje supervisado" (SDRD) [10]. El principal objetivo de la tarea es la extracción de patrones descriptivos cuyo soporte varíe significativamente de un conjunto de datos (o clase) a otro. Esto quiere decir que EPM se encuentra a medio camino entre las inducciones descriptiva y predictiva ya que se pretende describir relaciones entre los datos utilizando para ello aprendizaje supervisado. Las principales finalidades de esta tarea son la descripción de las características discriminativas entre clases o la descripción de tendencias emergentes. No obstante, a pesar de las claras capacidades descriptivas de la tarea, esta se ha utilizado ampliamente en la literatura como un clasificador, aplicándose con éxito en campos como la Química [11], [12], Bioinformática [13], [14] o Medicina [15], [16], entre otros [17]. No obstante, un nuevo enfoque basado en en el uso de sistemas difuso evolutivos (EFSs) [18] ha sido desarrollado recientemente con propuestas capaces de extraer conocimiento con un buen balance entre la capacidad descriptiva de las reglas y su fiabilidad [19], [9], [20].

En este trabajo se presenta una primera propuesta para la extracción de patrones emergentes de calidad en entornos de minería de flujo de datos denominado SE2P (*Stream Extraction of Emerging Patterns*). Este algoritmo se basa principalmente en el empleo de dos fases: una fase *online* en la que se almacena la información hasta obtener un bloque de datos con un tamaño determinado y una fase *offline* basada en un sistema difuso evolutivo (EFS) multi-objetivo capaz de extraer conocimiento de las características discriminativas entre clases con un buen balance entre capacidad descriptiva y fiabilidad sobre dicho bloque de datos.

El trabajo se estructura de la siguiente manera: en la Sección II se introduce el problema y, en concreto, la definición y características de EPM y de la minería de flujo de datos a lo largo de la literatura. A continuación, en la Sección III se presenta el enfoque de extracción de conocimiento propuesto y en la Sección IV se muestra un estudio experimental para la validación de la calidad del conocimiento extraído. Por último, se presentan las conclusiones extraídas junto a los posibles trabajos futuros.

II. Preliminares

En esta sección se revisan los conceptos básicos referentes a EPM y la minería de flujo de datos. En primer lugar se presenta la definición de EPM así como sus objetivos principales. A continuación, se define la minería de flujo de datos junto a una breve descripción de los diferentes enfoques utilizados. Por último, se presentan las principales medidas de calidad

empleadas en EPM y cómo son empleadas en SE2P para la minería de flujo de datos.

II-A. *Minería de patrones emergentes*

La minería de patrones emergentes fue definida por Dong y Li [8], [9] como:

> "Sea un patrón X cualquiera, y sea $\rho > 1$ un valor de umbral, X se denominará como emergente si y solo si su índice de crecimiento entre dos conjuntos de datos D_1 y D_2 es mayor que ρ."

Este índice de crecimiento (GR) es definido con una función representada en la Ecuación 1.

$$GR(X) = \begin{cases} 0, & \text{Si } Sop_1(X) = Sop_2(X) = 0, \\ \infty, & \text{Si } Sop_1(X) = 0 \wedge Sop_2(X) \neq 0, \\ \frac{Sop_2(X)}{Sop_1(X)}, & \text{en otro caso} \end{cases} \quad (1)$$

donde $Sop_i(X)$ es el soporte del patrón X en el conjunto de datos i.

Los propósitos principales para los que fue definida la tarea son:

- La descripción de las diferencias características entre clases o conjuntos de datos.
- La descripción de tendencias emergentes.
- La detección de diferencias entre múltiples variables.

En concreto, nuestra propuesta se centrará en el primer objetivo, es decir, lo que buscamos son diferencias características entre las clases de un flujo de datos a lo largo del tiempo. Habitualmente, estos patrones se presentan al experto en forma de reglas con el siguiente formato [21]:

$$R : Cond \rightarrow Clase \quad (2)$$

donde $Cond$ es un conjunto de características, normalmente en forma de pares atributo-valor y $Clase$ es el valor de la variable objetivo o de interés.

La principal dificultad que posee la extracción de estos patrones se encuentra en la misma definición de patrón emergente. El GR se define en función de un ratio entre soportes, lo cual propicia que el espacio de los patrones emergentes no sea convexo [22]. Esto quiere decir que patrones más específicos, y por tanto, con menor soporte, puedan poseer valores de GR más elevados que aquellos cuyos soportes sean más altos. Es por esta razón que se han definido a lo largo de la literatura diferentes tipos de patrones emergentes cuyas restricciones permiten una mayor facilidad de extracción, como los patrones *Jumping* o los patrones emergentes χ^2, entre otros [23], [24], [25], [9]. Asimismo, se han desarrollado diferentes técnicas algorítmicas para la extracción de estos patrones encuadradas en cuatro categorías diferentes [9]. A pesar del amplio desarrollo de la tarea, la gran mayoría de algoritmos han sido desarrollados como clasificadores, ignorando las cualidades descriptivas de la tarea. Sin embargo, en los últimos años se ha desarrollado un enfoque basado en EFSs, donde destaca el algoritmo MOEA-EFEP [20], cuyos resultados poseen un buen balance entre las cualidades descriptivas y la fiabilidad de las reglas.

II-B. *Minería de flujo de datos*

Un flujo de datos se define como una secuencia ordenada y potencialmente infinita de ejemplos que llegan al sistema a lo largo del tiempo a una velocidad que puede ser variable [26]. Esta definición tan simple de un flujo de datos trae consigo una gran variedad de diferencias respecto a la minería de datos tradicional, donde se destaca [5]:

- No se puede almacenar toda la información en memoria. Al ser potencialmente de tamaño infinito hay que buscar estrategias para procesar y descartar los datos.
- Todos los datos no se encuentran disponibles en el momento del aprendizaje. Los datos van llegando a lo largo del tiempo y es el modelo el que tiene que aprender conforme los datos lleguen, es decir, el modelo debe de adaptarse a lo largo del tiempo.
- Como el flujo es generado por una fuente a lo largo del tiempo, el fenómeno que subyace en la generación de dichos datos suele cambiar con mayor o menor frecuencia. A este hecho se le denomina en la literatura como cambio de concepto [27]. Esto quiere decir que el modelo aprendido con los datos en un instante t, si se produce un cambio de concepto, no será válido para los datos en el instante $t + x$.
- Esta velocidad de llegada es, normalmente, alta respecto a la capacidad de procesamiento que se posee. Por lo tanto, se busca que el algoritmo de aprendizaje sea capaz de aprender lo más rápido posible con el fin de evitar el encolamiento de las instancias.

Teniendo en cuenta estas características, las instancias que llegan al sistema se pueden procesar de dos formas diferentes [5]:

- Online. Las instancias llegan una a una y son procesadas por el algoritmo de aprendizaje tan pronto estén disponibles.
- Por bloques. Las instancias son almacenadas hasta obtener un bloque de datos de un tamaño predeterminado y donde todo el bloque es procesado a la vez por el algoritmo de aprendizaje.

Una vez se ha elegido la metodología de procesamiento de los datos, es necesario determinar la estrategia de aprendizaje del método. Entre las técnicas más utilizadas para el aprendizaje en minería de flujo de datos destacan [28]:

- Detectores de cambio de concepto [29]. Son métodos externos al algoritmo de aprendizaje que calculan diferentes propiedades del flujo de datos para detectar cambios. Normalmente poseen dos niveles: un nivel de alerta en donde el cambio empieza a ocurrir, donde solo se aprende usando los datos más recientes; y un nivel de alarma indicando un cambio severo donde el clasificador se reemplaza.
- Ventanas deslizantes [30]. Se almacenan en un *buffer* de memoria las instancias más recientes, deshaciéndose de aquellas más antiguas que no caben en dicho *buffer*. Esto permite al método aprender únicamente de las instancias más recientes en el flujo.

- Ensemble [31]. En donde se utilizan diferentes clasificadores que permiten hacer un seguimiento del flujo de datos. Existen actualmente dos estrategias principales: aprender un nuevo método y añadirlo al ensemble, descartando métodos antiguos si es necesario, o bien mediante un esquema de pesos en los diferentes clasificadores en función de su rendimiento.

II-C. Medidas de calidad

En minería de datos tradicional, la calidad de un modelo se determina mediante diferentes mecanismos como por ejemplo la validación cruzada [32]. Sin embargo, muchos de estos mecanismos requieren del uso del conjunto de datos completo para poder realizar una evaluación correcta. Esto es imposible en minería de flujo de datos porque no poseemos el conjunto de datos al completo. Por lo que es necesario el uso de técnicas diferentes para la evaluación de los modelos.

Uno de los esquemas de evaluación más utilizado es el denominado *test-then-train* [33]. En este esquema, cuando una instancia o bloque de datos llega al sistema sirve para evaluar el modelo actual o hacer una predicción sobre la instancia concreta o el bloque de datos. Una vez se realiza este proceso, dicha instancia o bloque pasa a entrenar el modelo actual.

En EPM, la medida más importante es el GR, pues es la métrica que define la tarea. No obstante, es necesario determinar diferentes aspectos tales como generalidad, interés o fiabilidad [9], claves para la correcta extracción de reglas interpretables y precisas. La tarea fue concebida inicialmente para el análisis de problemas entre dos clases o conjuntos de datos. Sin embargo, la tarea puede ser fácilmente extendida a problemas multiclase mediante estrategias como *One vs All* (OVA) [34] donde la clase positiva es la que se encuentra descrita en el consecuente de la regla y la negativa el resto de clases del problema.

Cuadro I
MATRIZ DE CONFUSIÓN PARA UNA REGLA EN EPM.

	Clase predicha	
Clase real	Positive	Negative
Positive	$p = tp$	$\overline{p} = fn$
Negative	$n = fp$	$\overline{n} = tn$

En la Tabla I se puede observar la matriz de confusión, donde: p representa el número de ejemplos correctamente cubiertos, $\overline{p}$ el número de ejemplos de la clase no cubiertos, n el número de ejemplos cubiertos incorrectamente, y $\overline{n}$ el número de ejemplo correctamente no cubiertos.

Las medidas de calidad más utilizadas en EPM son las descritas a continuación [9]:

- Growth Rate (GR). Definida en la Ecuación 1, mide el poder discriminativo de una regla.
- Confianza (Conf). Se define como el ratio de la capacidad predictiva de la regla para la clase positiva [35].

$$Conf(R) = \frac{p}{p+n} \tag{3}$$

- Atipicidad (Atip). Esta medida híbrida muestra el balance existente entre generalidad y ganancia de precisión de la regla [21].

$$Atip(R) = \frac{p+n}{P+N}\left(\frac{p}{p+n} - \frac{P}{P+N}\right) \tag{4}$$

  El dominio de esta medida tiene una dependencia directa con el porcentaje de la clase a medir, por lo tanto, para realizar comparaciones es necesario normalizar esta medida. Esta normalización se ha llevado a cabo de la siguiente manera:

$$Atip_Norm(R) = \frac{Atip(R) - \left(\frac{P}{T}\left(0 - \frac{P}{T}\right)\right)}{\left(\frac{P}{T}\left(1 - \frac{P}{T}\right)\right) - \left(\frac{P}{T}\left(0 - \frac{P}{T}\right)\right)} \tag{5}$$

- Tasa de falsos positivos (FPR). Mide el porcentaje de ejemplos incorrectamente cubiertos respecto al total de ejemplos de la clase negativa. Esta medida debe ser minimizada para la obtención de reglas precisas [36].

$$FPR(R) = \frac{n}{N} \tag{6}$$

- Tasa de verdaderos positivos (TPR). Mide el porcentaje de ejemplos correctamente cubiertos respecto al número total de ejemplos de la clase positiva [37].

$$TPR(R) = \frac{p}{P} \tag{7}$$

- Número de reglas. Mide la cantidad de reglas extraídas.
- Número de variables. Mide el número medio de variables que se obtienen en el conjunto de reglas.

III. SE2P: STREAM EXTRACTION OF EMERGING PATTERNS

Esta sección presenta la propuesta algorítmica para la extracción de patrones emergentes descriptivos para minería de flujo de datos llamado SE2P.

SE2P utiliza un enfoque basado en dos fases online/offline utilizando para ello el esquema de procesamiento de instancias basado en bloques. Esto implica que la fase online en un primer lugar agrupará instancias provenientes del stream hasta que se alcance el número de instancias determinado por el tamaño de bloque. Cuando hay un bloque de datos disponible, la fase offline ejecutará el núcleo de SE2P, el cual es un EFS multi-objetivo basado en el enfoque NSGA-II [38] para la extracción de un conjunto de reglas que representen las características discriminativas de las diferentes clases que se encuentren en dicho bloque. Es importante destacar que la estrategia de aprendizaje de SE2P se basa en una modificación del esquema de ventanas deslizantes donde se almacenan los modelos de reglas obtenidos previamente más una función de evaluación en el proceso evolutivo que permite el uso de esta estructura con el objetivo de adaptarse a los cambios de concepto.

Este algoritmo evolutivo utiliza una codificación "cromosoma = regla" donde se representa tanto el antecedente como el consecuente de la regla, lo que permite la extracción de reglas de todas las posibles clases en una única ejecución. La única representación disponible en este método es la forma

normal disyuntiva (DNF), ya que se ha demostrado que se obtienen mejores resultados que con otras representaciones [39]. La Figura 1 presenta una regla DNF, representada en el genotipo por un vector binario. Es importante destacar que para variables numéricas se emplea el número de conjuntos difusos correspondientes a etiquetas lingüísticas que se definen por funciones de pertenencia triangulares uniformes.

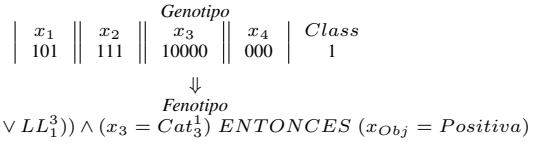


Figura 1. Representación de una regla DNF en EvAEP.

III-A. Operador de inicialización

SE2P utiliza un operador de inicialización sesgada basado en el conocimiento previo con el fin de poder actualizarlo con los nuevos datos. Este operador añade el modelo de reglas extraído del bloque de datos anterior a la población inicial (P_0) mientras que el resto de individuos son inicializados de manera completamente aleatoria, pero controlando la inicialización de, como máximo, el 25 % de sus variables en el 75 % de individuos de esta población. Esto permite la obtención de individuos con gran generalidad al proceso evolutivo.

III-B. Operadores genéticos

La población de la siguiente generación es obtenida mediante la aplicación de distintos operadores genéticos: operador de selección por torneo binario [40], un operador de cruce multipunto [41] y un operador de mutación sesgada empleado por primera vez en un algoritmo de descubrimiento de subgrupos en [42].

Asimismo, SE2P utiliza un operador de reinicialización con el fin de evitar el estancamiento de la población en un óptimo local. Esta reinicialización se lleva a cabo mediante la utilización de un operador de token competition [43] con el que se eliminan reglas solapadas. A continuación, el resto de individuos se genera aleatoriamente.

III-C. Función de evaluación

Uno de los aspectos fundamentales en la minería de flujo de datos es la necesidad de adaptar el modelo en función de los datos que van llegando. Como hemos visto anteriormente, el fenómeno subyacente en los datos puede variar a lo largo del tiempo, pudiendo invalidar todo el conocimiento previo. Por lo tanto SE2P utiliza un esquema de ventana deslizante con pesos en donde se almacenan únicamente los n últimos conjuntos de reglas extraídos. Con esta estructura, la función que se utilizará para determinar la calidad de cada objetivo en SE2P para una regla R_i viene dada por la Ecuación 8.

$$Fitness_t^k(R_i) = QM_t^k(R_i) \cdot \left[1 - \sum_{j=t-n}^{t} SW(R_i, j) \cdot 2^{-(t-j)}\right] \quad (8)$$

donde $QM_t^k(R_i)$ es el valor de la medida de calidad usada como objetivo k en el bloque de datos t actual, $SW(R_i, j)$ es una función que devuelve cero si R_i se encuentra en el conjunto de reglas devuelto para el bloque j o uno en caso contrario. El objetivo de esta función de evaluación no es otro que la penilización de aquellas reglas que no definen el fenómeno subyacente en los datos, representado por las reglas extraídas anteriormente. Esto se debe a que el cambio normalmente se produce de manera gradual, por lo que el conocimiento previo sigue siendo relevante hasta que el nuevo prevalece.

III-D. Esquema de funcionamiento

El proceso de ejecución de SE2P es el siguiente: una vez se ha obtenido un bloque de datos, se lanza la fase de aprendizaje offline. En un primer lugar, siguiendo el esquema *test-then-train* se evaluará el modelo de reglas extraído en el bloque anterior. Una vez evaluado el modelo de reglas, dicho bloque de datos pasa a ser un conjunto de entrenamiento. Es importante destacar que, al no existir un modelo de reglas previo, el primer bloque pasa a ser directamente de entrenamiento. Por lo tanto, las evaluaciones se realizan a partir del segundo bloque.

A continuación, el EFS multi-objetivo se ejecuta, comenzando con la aplicación del operador de inicialización sesgada basada en conocimiento previo para generar P_0. Después, el proceso evolutivo da comienzo, ejecutándose durante g generaciones o hasta que un nuevo bloque de datos esté disponible. Dentro de este proceso evolutivo se aplicarán los operadores genéticos, obteniendo una población de descendientes Q_g de igual tamaño que la población actual. A continuación ambas poblaciones se unen en U_g, se evalúan aquellos individuos no evaluados y se obtiene la población de la siguiente generación mediante la ordenación por frentes de dominancia propia del algoritmo NSGA-II. Por último, se analiza el estancamiento de la población, donde se comprueba si la población actual no ha cubierto ejemplos nuevos del bloque durante un 25 % del total de generaciones. El operador de reinicialización se aplicará en caso de que la población se estanque.

Una vez finalizado el proceso evolutivo, el algoritmo aplicará el operador de token competition [43] para eliminar aquellas reglas redundantes y el resultado de este procedimiento será el devuelto al usuario.

IV. Estudio experimental

En este trabajo se presenta un estudio preliminar de una primera propuesta para la extracción de patrones emergentes con un buen balance entre la capacidad descriptiva de las reglas extraídas y la fiabilidad de las mismas dentro de la minería de flujo de datos. En concreto, se abordará el tratamiento de Big Data mediante técnicas de minería de flujos de datos. Para ello se ha simulado un flujo de datos mediante la herramienta de minería de datos MOA [44]. Las características de los conjuntos de datos utilizados como por ejemplo el número de instancias totales, número de variables y número de clases se presentan en la Tabla II.

Cuadro II
Conjuntos de datos utilizados en el estudio experimental.

Nombre	# Instancias	# Variables	# Clases
Air	539395	7	2
Elec	45312	7	2
forest	581012	54	7
Higgs	11000000	28	2
Susy	5000000	18	2

Los parámetros utilizados por SE2P en este estudio experimental se muestran en la Tabla III, los parámetros utilizados son similares a los utilizados por EFSs desarrollados para EPM sobre datos estáticos [20]. Sin embargo, uno de los parámetros más importantes para determinar el rendimiento de SE2P es el tamaño del bloque de datos que se procesará. En este trabajo se van a seleccionar tres tamaños de bloque diferentes: 1500, 2500 y 5000 instancias por bloque con el objetivo de determinar el más apropiado.

Cuadro III
Parámetros utilizados por SE2P en el estudio experimental.

Parámetro	Valor
Etiquetas lingüísticas	3
Número de generaciones máximas	60
Tamaño de población	50
Probabilidad de cruce	0.7
Probabilidad de mutación	0.1
Tamaño de ventana temporal	5

Cuadro IV
Resultados medios obtenidos por SE2P en los flujos de datos analizados.

Nombre	Tam. bloque	n_r	n_v	ATIP	CONF	GR	TPR	FPR	Tiempo ejec. (ms)
Air	1500	2,006	**2,539**	**0,588**	**0,594**	**0,953**	**0,521**	0,346	**400,402**
	2500	**2,000**	2,539	0,584	0,581	0,914	0,498	0,330	473,280
	5000	**2,000**	2,580	0,575	0,541	0,873	0,477	**0,327**	1159,472
Elec	1500	2,000	2,259	0,657	0,688	0,966	0,583	0,269	**433,586**
	2500	**1,941**	**2,088**	0,659	**0,700**	**0,971**	**0,612**	**0,294**	557,235
	5000	2,000	2,875	**0,661**	0,687	0,938	0,583	0,262	581,125
forest	1500	**4,054**	**9,107**	**0,770**	**0,519**	**0,969**	**0,799**	**0,241**	**1692,702**
	2500	4,134	9,578	0,739	0,495	0,948	0,764	0,263	1915,446
	5000	4,252	9,184	0,693	0,437	0,914	0,730	0,307	2616,087
Higgs	1500	2,011	**7,640**	0,524	0,549	0,913	0,475	0,426	**263,109**
	2500	2,013	8,237	0,526	**0,550**	0,960	**0,477**	0,426	337,272
	5000	**2,007**	9,904	**0,526**	0,536	**0,992**	0,433	**0,380**	436,454
Susy	1500	2,006	**7,081**	**0,622**	0,674	**1,000**	**0,516**	**0,272**	**212,349**
	2500	**2,004**	7,454	0,604	**0,692**	**1,000**	0,486	0,277	257,945
	5000	2,005	7,821	0,596	0,681	**1,000**	0,487	0,296	315,184

En la Tabla IV se presentan los resultados medios obtenidos por SE2P a lo largo de los diferentes flujos de datos analizados. Es importante remarcar que, debido al funcionamiento de SE2P, estos resultados medios se han obtenido usando $n_b - 1$ bloques, siendo n_b el número total de bloques analizados. A continuación se muestra un análisis de cada una de las diferentes medidas de calidad analizadas:

- Número de reglas y variables. En general el número de reglas obtenido es muy bajo. Se destaca que el número de instancias por bloque no influye significativamente en estos resultados. Sin embargo, se puede observar que el número de variables es en algunos casos elevado, llevando el modelo extraído a una nivel de complejidad mayor. Sin embargo, en líneas generales, el modelo extraído es simple.
- Atipicidad. Esta medida, que mide el interés de las reglas extraídas, nos indica que en general las reglas obtenidas son interesantes, obteniéndose mejores resultados en bloques de 1500 instancias.
- Confianza. En esta medida se puede observar una amplia variabilidad en los resultados, donde el mejor resultado se obtiene con tamaños de bloque de 2500 en tres de los cinco conjuntos analizados. No obstante, el nivel de confianza es aceptable en los diferentes tamaños de bloque.
- GR. En esta medida se representa el porcentaje de patrones extraídos que son patrones emergentes. Como se puede observar, en general se extraen reglas emergentes y que por tanto poseen altas capacidades discriminativas, independientemente del tamaño de bloque escogido.
- Balance TPR-FPR. En estos resultados se puede observar que la generalidad de las reglas medidas como TPR es bastante elevada. Sin embargo, a pesar de que los niveles de FPR son elevados, la diferencia entre ambas métricas nos permite poder decir que los resultados obtenidos poseen, en general, un buen balance entre la generalidad y la precisión. También es importante destacar que los resultados son mejores a menor tamaño de bloque.
- Tiempo de ejecución. En cuento al tiempo de ejecución del algoritmo se puede observar que, obviamente, el tiempo medio de procesamiento de un bloque es menor en función del tamaño de bloque. Además, se destaca que el tiempo de ejecución es lo suficientemente rápido para el procesamiento de flujos de datos en donde se pueda procesar un bloque de datos cada segundo, lo cual es bastante aceptable.

V. Conclusiones

En este trabajo se ha presentado un estudio preliminar de un primer enfoque la extracción de patrones emergentes para minería de flujo de datos. Este algoritmo se basa en el procesamiento de las instancias por bloques de datos de tamaño fijo en donde se ejecuta un algoritmo evolutivo multiobjetivo por cada bloque con el objetivo de extraer patrones emergentes que describan las características discriminativas de las diferentes clases del problema.

Los resultados del estudio realizado demuestran unos resultados muy prometedores y abren una línea de investigación donde se necesita implementar nuevas técnicas para mejorar los resultados actuales y los tiempos de ejecución a fin de poder procesas flujos de datos más veloces.

Agradecimientos

Este trabajo ha sido subvencionado por el Ministerío de Economía y Competitividad bajo el proyecto TIN2015-68454-R y el contrato predoctoral FPI referencia BES-2016-077738 asociado al mismo (Fondos FEDER).

Referencias

[1] A. Fernández, S. Río, V. López, A. Bawakid, M. del Jesus, J. Benítez, and F. Herrera, "Big Data with Cloud Computing: An Insight on the Computing Environment, MapReduce and Programming Frameworks," *WIREs Data Mining and Knowledge Discovery*, vol. 5, no. 4, pp. 380–409, 2014.

[2] T. Kraska, "Finding the Needle in the Big Data Systems Haystack," *IEEE Internet Computing*, vol. 17, no. 1, pp. 84–86, 2013.

[3] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data mining with big data," *IEEE transactions on knowledge and data engineering*, vol. 26, no. 1, pp. 97–107, 2014.

[4] I. Žliobaitė, M. Pechenizkiy, and J. Gama, "An overview of concept drift applications," in *Big Data Analysis: New Algorithms for a New Society*. Springer, 2016, pp. 91–114.

[5] J. Gama, *Knowledge discovery from data streams*. CRC Press, 2010.

[6] A. Bifet, "Adaptive learning and mining for data streams and frequent patterns," Ph.D. dissertation, Universitat Politècnica de Catalunya, 2009.

[7] D. Han, C. Giraud-Carrier, and S. Li, "Efficient mining of high-speed uncertain data streams," *Applied Intelligence*, vol. 43, no. 4, pp. 773–785, 2015.

[8] G. Z. Dong and J. Y. Li, "Efficient Mining of Emerging Patterns: Discovering Trends and Differences," in *Proc. of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM Press, 1999, pp. 43–52.

[9] A. M. García-Vico, C. J. Carmona, D. Martín, M. García-Borroto, and M. J. del Jesus, "An overview of emerging pattern mining in supervised descriptive rule discovery: Taxonomy, empirical study, trends and prospects," *WIREs: Data Mining and Knowledge Discovery*, vol. 8, no. 1, 2018.

[10] P. Kralj-Novak, N. Lavrac, and G. I. Webb, "Supervised Descriptive Rule Discovery: A Unifying Survey of Constrast Set, Emerging Pateern and Subgroup Mining," *Journal of Machine Learning Research*, vol. 10, pp. 377–403, 2009.

[11] R. Sherhod, V. J. Gillet, T. Hanser, P. N. Judson, and J. D. Vessey, "Toxicological knowledge discovery by mining emerging patterns from toxicity data," *Journal of Chemical Information and Modeling*, vol. 5, no. S-1, p. 9, 2013.

[12] A. Lepailleur, G. Poezevara, and R. Bureau, "Automated detection of structural alerts (chemical fragments) in (eco) toxicology," *Computational and structural biotechnology journal*, vol. 5, no. 6, pp. 1–8, 2013.

[13] M. Piao, H. G. Lee, G. Y. Sohn, G. Pok, and K. H. Ryu, "Emerging patterns based methodology for prediction of patients with myocardial ischemia," in *Proc. of the 6th International Conference on Fuzzy Systems and Knowledge Discovery*. IEEE, 2009, pp. 174–178.

[14] G. Tzanis, I. Kavakiotis, and I. P. Vlahavas, "Polya-iep: A data mining method for the effective prediction of polyadenylation sites," *Expert Systems with Applications*, vol. 38, no. 10, pp. 12 398–12 408, 2011.

[15] P. W. Angriyasa, Z. Rustam, and W. Sadewo, "Non-invasive intracranial pressure classification using strong jumping emerging patterns," in *Proc. of the 2011 International Conference on Advanced Computer Science and Information System (ICACSIS)*. IEEE, 2011, pp. 377–380.

[16] Y. Yu, K. Yan, X. Zhu, and G. Wang, "Detecting of PIU Behaviors Based on Discovered Generators and Emerging Patterns from Computer-Mediated Interaction Events," in *Proc. of the 15th International Conference on Web-Age Information Management*, ser. LNCS, vol. 8485. Elsevier, 2014, pp. 277–293.

[17] G. Li, R. Law, H. Q. Vu, J. Rong, and X. R. Zhao, "Identifying emerging hotel preferences using emerging pattern mining technique," *Tourism management*, vol. 46, pp. 311–321, 2015.

[18] F. Herrera, "Genetic fuzzy systems: taxomony, current research trends and prospects," *Evolutionary Intelligence*, vol. 1, pp. 27–46, 2008.

[19] A. M. García-Vico, J. Montes, J. Aguilera, C. J. Carmona, and M. J. del Jesus, "Analysing Concentrating Photovoltaics Technology through the use of Emerging Pattern Mining," in *Proc. of the 11th International Conference on Soft Computing Models in Industrial and Environmental Applications*. Springer, 2016, pp. 1–8.

[20] A. M. García-Vico, C. J. Carmona, P. González, and M. J. del Jesus, "Moea-efep: Multi-objective evolutionary algorithm for extracting fuzzy emerging patterns," *IEEE Transactions on Fuzzy Systems*, In Press.

[21] C. J. Carmona, M. J. del Jesus, and F. Herrera, "A Unifying Analysis for the Supervised Descriptive Rule Discovery via the Weighted Relative Accuracy," *Knowledge-Based Systems*, vol. 139, pp. 89–100, 2018.

[22] L. Wang, H. Zhao, G. Dong, and J. Li, "On the complexity of finding emerging patterns," in *Proc. of the 28th Annual International Computer Software and Applications Conference*, vol. 2, 2004, pp. 126–129.

[23] G. Z. Dong, J. Y. Li, and X. Zhang, "Discovering jumping emerging patterns and experiments on real datasets," in *Proc. on International Database Conference Heterogeneous and Internet Databases*, 1999, pp. 155–168.

[24] H. Fan and K. Ramamohanarao, "Noise Tolerant Classification by Chi Emerging Patterns," in *Proc. of the 8th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, ser. LNCS, vol. 3056. Springer, 2004, pp. 201–206.

[25] K. Ramamohanarao and H. Fan, "Patterns Based Classifiers," *World Wide Web*, vol. 10, no. 1, pp. 71–83, 2007.

[26] M. M. Gaber, "Advances in data stream mining," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 1, pp. 79–85, 2012.

[27] J. a. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, pp. 44:1–44:37, 2014.

[28] S. Ramírez-Gallego, B. Krawczyk, S. García, M. Wozniak, and F. Herrera, "A survey on data preprocessing for data stream mining: Current status and future directions," *Neurocomputing*, vol. 239, pp. 39–57, 2017.

[29] R. M. Vallim and R. F. De Mello, "Proposal of a new stability concept to detect changes in unsupervised data streams," *Expert Systems with Applications*, vol. 41, no. 16, pp. 7350–7360, 2014.

[30] J. Shan, J. Luo, G. Ni, Z. Wu, and W. Duan, "Cvs: fast cardinality estimation for large-scale data streams over sliding windows," *Neurocomputing*, vol. 194, pp. 107–116, 2016.

[31] B. Krawczyk, L. L. Minku, J. a. Gama, J. Stefanowski, and M. Woźniak, "Ensemble learning for data stream analysis: A survey," *Information Fusion*, vol. 37, pp. 132–156, 2017.

[32] N. Japkowicz and M. Shah, *Evaluating learning algorithms: a classification perspective*. Cambridge University Press, 2011.

[33] A. Wald, *Sequential analysis*. Courier Corporation, 1973.

[34] M. Galar, A. Fernández, E. Barrenechea, H. Bustince, and F. Herrera, "An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes," *Pattern Recognition*, vol. 44, no. 8, pp. 1761 – 1776, 2011.

[35] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From data mining to knowledge discovery: an overview," in *Advances in knowledge discovery and data mining*. AAAI/MIT Press, 1996, pp. 1–34.

[36] D. Gamberger and N. Lavrac, "Expert-Guided Subgroup Discovery: Methodology and Application," *Journal Artificial Intelligence Research*, vol. 17, pp. 501–527, 2002.

[37] W. Kloesgen, "Explora: A Multipattern and Multistrategy Discovery Assistant," in *Advances in Knowledge Discovery and Data Mining*. American Association for Artificial Intelligence, 1996, pp. 249–271.

[38] K. Deb, A. Pratap, S. Agrawal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[39] A. García-Vico, C. J. Carmona, and M. J. del Jesus, "Análisis de diferentes tipos de reglas en sistemas difusos evolutivos para minería de patrones emergentes," in *Proc. of the XII Spanish Conference on Metaheuristics, Evolutive and Bioinspired Algorithms (MAEB 2017)*, 2017, p. 876–885.

[40] B. L. Miller and D. E. Goldberg, "Genetic Algorithms, Tournament Selection, and the Effects of Noise," *Complex System*, vol. 9, pp. 193–212, 1995.

[41] J. H. Holland, "Adaptation in natural and artificial systems," *University of Michigan Press*, 1975.

[42] C. J. Carmona, P. González, M. J. del Jesus, and F. Herrera, "NMEEF-SD: Non-dominated Multi-objective Evolutionary algorithm for Extracting Fuzzy rules in Subgroup Discovery," *IEEE Transactions on Fuzzy Systems*, vol. 18, no. 5, pp. 958–970, 2010.

[43] K. S. Leung, Y. Leung, L. So, and K. F. Yam, "Rule Learning in Expert Systems Using Genetic Algorithm: 1, Concepts," in *Proc. of the 2nd International Conference on Fuzzy Logic and Neural Networks*, K. Jizuka, Ed., 1992, pp. 201–204.

[44] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "MOA: massive online analysis," *Journal of Machine Learning Research*, vol. 11, pp. 1601–1604, 2010. [Online]. Available: https://moa.cms.waikato.ac.nz/

Segmentación de mercado explicable sobre datos de alta dimensión

Carlos Eiras-Franco
Grupo LIDIA. CITIC.
Universidade da Coruña
A Coruña, España
carlos.eiras.franco@udc.es

Bertha Guijarro-Berdiñas
Grupo LIDIA. CITIC.
Universidade da Coruña
A Coruña, España
cibertha@udc.es

Amparo Alonso-Betanzos
Grupo LIDIA. CITIC.
Universidade da Coruña
A Coruña, España
ciamparo@udc.es

Antonio Bahamonde
Universidad de Oviedo
Gijón, España
abahamonde@uniovi.es

***Resumen*—Obtener información relevante de la gran cantidad de datos que se generan en las interacciones de un mercado o, en general, de un conjunto de datos diádicos, es un amplio problema que despierta gran interés académico y en la industria. Por otra parte, la interpretabilidad de los algoritmos de aprendizaje máquina está adquiriendo relevancia hasta el punto de ser requerida legalmente, lo que dispara la necesidad de contar con algoritmos con estas características. En este trabajo proponemos una medida de calidad que tiene en cuenta el nivel de interpretabilidad de un resultado. Presentamos, además, un algoritmo de agrupamiento sobre datos diádicos que ofrece resultados con el nivel de interpretabilidad que desee el usuario y que es capaz de manejar grandes volúmenes de datos. Detallamos experimentos que avalan tanto la precisión de los resultados obtenidos, comparable a métodos tradicionales, como su escalabilidad.**

***Index Terms*—segmentación de mercado, interpretabilidad, explicabilidad, escalabilidad, aprendizaje automático, big data**

I. Introducción

Los datos obtenidos al monitorizar el funcionamiento de un mercado son, en su mayoría, diádicos, es decir, representan la relación entre dos entidades, ya sean estas usuarios y productos, compradores y vendedores u otra pareja de agentes. Más generalmente, los datos diádicos [1] recogen información sobre la interacción de dos entidades de cualquier ámbito y son prevalentes en problemas tan comunes como los sistemas recomendadores [2], lingüística computacional, recuperación de información y aprendizaje de preferencias [3], además de estar presentes en campos más específicos como la corrección automática de exámenes [4].

Un problema tradicional a tratar sobre datos de este tipo consiste en obtener agrupaciones de entidades con un comportamiento similar, dando lugar a un modelado de mayor nivel del entorno estudiado. De esta manera se podría, por ejemplo a partir de los datos de un recomendador de libros, buscar conjuntos de libros que atraen a usuarios similares o conjuntos de usuarios que tienen gustos parecidos respecto a la lectura.

Este trabajo ha sido parcialmente financiado por el Ministerio de Economía y Competitividad (proyectos de investigación TIN 2015-65069-C2, tanto 1-R como 2-R y Red Española de Big Data y Análisis de datos escalable, TIN2016-82013-REDT), por la Xunta de Galicia (GRC2014/035 y ED431G/01) y por Fondos de Desarrollo Regional de la Unión Europea.

El estudio de los mercados con el objetivo de identificar subconjuntos de actores que se comportan como mercados más pequeños y homogéneos es la meta de la segmentación de mercados [5]. La información obtenida a partir de estas agrupaciones que se comportan de manera homogénea permite a las empresas hacer campañas de marketing dirigido a medida de cada grupo, aumentando así su efectividad y disminuyendo costes [6]. Esta información es, por tanto, una ventaja codiciada por las empresas, pero resulta complicada de obtener en la práctica aún cuando se dispone de abundantes datos.

Por otra parte, que los resultados obtenidos de la segmentación de mercados sean comprensibles por un gestor humano es un factor esencial que está atrayendo mucha atención recientemente. El concepto de interpretabilidad se refiere a esta característica, aunque no se trata de un concepto monolítico sino que, de acuerdo a [7], refleja distintas ideas. Idealmente, el modelo obtenido debe:

1. permitir a un supervisor interpretar sus resultados de manera que pueda verificar que los objetivos del modelo están alineados con los deseados (p.e. un sistema de evaluación de crédito no debe tener sesgos raciales o de género),
2. motivar sus predicciones de manera que un supervisor pueda hacer hipótesis de causalidad que luego verificará, descartando meras correlaciones fortuitas o dependientes de un tercer factor,
3. explicar sus salidas de manera que se pueda corroborar cómo de generalizables son. Por ejemplo, en el caso descrito en [8], un sistema de predicción de mortalidad por neumonía asignaba, erróneamente, menor riesgo a los pacientes que también padecían asma. Esto se debía a que estos pacientes habían recibido un tratamiento más agresivo por lo que, en el historial de datos analizados, sus porcentajes de recuperación solían mejorar respecto a otros pacientes; en consecuencia, el modelo automático asignaba un menor riesgo resultado de una mala generalización que posiblemente causaría el resultado contrario al esperado. Al utilizar el modelo derivado para asignar tratamientos, estos pacientes recibirían un tratamiento más leve, puesto que al haber obtenido históricamente mejores resultados, su riesgo estimado de mortalidad

sería menor,
4. ser informativo, es decir, ofrecer al supervisor información novedosa sobre las variables bajo estudio.

Estas características en ocasiones son incluso requeridas por ley, como en el caso del "derecho a la explicación" recogido en el nuevo Reglamento Europeo de Protección de Datos de la Unión Europea[1], de manera que resultan no solo deseables en todo caso sino que también obligatorias en muchos otros.

Los algoritmos de aprendizaje automático ofrecen interpretabilidad generalmente de dos maneras distintas: (1) mediante la transparencia del modelo y/o del algoritmo, que permite al supervisor seguir la "lógica" usada para hacer las predicciones, como ocurre por ejemplo con las reglas de producción, o (2) interpretabilidad *post-hoc*, consistente en justificar las salidas bien ofreciendo casos similares o visualizaciones u otros métodos que ayuden a identificar las características de la entrada que llevan a una predicción. Esta segunda opción es la más común pero, aunque ofrece una explicación de cada caso individual, no es informativa ya que no ofrece al supervisor información general del entorno modelado.

Un problema adicional es que, si bien, el estudio de los mercados se ve potenciado por la inmensa cantidad de datos que se generan y recogen —tanto en entornos online como, en menor medida, en tiendas físicas— y la valiosa información que estos datos encierran, a su vez, su inmenso volumen dificulta la extracción de esta información. Por tanto, se requiere la utilización de algoritmos escalables que puedan procesarlos.

En este trabajo proponemos un nuevo algoritmo de agrupamiento de datos diádicos que se puede utilizar para realizar segmentación de mercado. Este algoritmo busca obtener datos fácilmente interpretables e informativos para un supervisor humano, mejorando así su proceso de toma de decisiones. Además, su implementación en la plataforma escalable Apache Spark [9] permite procesar grandes cantidades de datos para obtener información decisiva en un tiempo práctico. En la Sección II detallamos los trabajos previos de otros autores en este campo; en la Sección III se definen los principales conceptos manejados por el sistema propuesto y en la Sección IV presentamos el nuevo algoritmo. En la Sección V mostramos y comparamos los resultados de aplicar nuestro algoritmo a un conjunto de datos de lectores de noticias. Por último, la Sección VI resume las conclusiones obtenidas y el trabajo futuro que proyectamos realizar.

II. Trabajo relacionado

Las técnicas de segmentación de mercado se pueden dividir atendiendo al origen de los datos que utilizan [10]:

1. *A priori*. Utilizan información de los compradores o de los productos que está disponible antes de su interacción en el mercado. Adolecen de no prestar atención al comportamiento de los actores en el mercado y ofrecer, por tanto, información limitada.
2. *Post-hoc*. Se basan en el estudio de los datos que se generan durante la actividad en el mercado. Es un enfoque más explorado para el que se han utilizado algoritmos de agrupamiento [11], árboles de clasificación y regresión [12], mapas autoorganizativos [13], algoritmos de reducción de la dimensionalidad [14] o algoritmos de co-agrupamiento [15].

Desde el punto de vista de la interpretabilidad, los algoritmos mencionados muestran características dispares. Mientras los resultados de algoritmos evolutivos y de co-agrupamiento no son apropiados si la interpretabilidad es un objetivo, los mapas autoorganizativos, y los algoritmos de reducción de la dimensionalidad y de agrupamiento ofrecen explicaciones *post-hoc* al supervisor. No obstante su capacidad de relacionar de manera sencilla las variables de entrada que describen a cada grupo es limitada, lo que reduce su utilidad para motivar sus predicciones y explicar sus salidas. Los árboles de clasificación y regresión, por otra parte, sí relacionan de manera clara las variables de entrada con las predicciones efectuadas y han sido utilizados con este fin sobre datos no diádicos [16], aunque su tamaño debe ser controlado para ofrecer explicaciones verificables por un supervisor.

III. Definiciones

Los datos diádicos X describen una interacción entre dos entidades $\mathcal{U}$ e $\mathcal{I}$. Cada dato $x \in \mathcal{X}$ tiene la forma (u, i, v) donde $u \in \mathcal{U}$ y $i \in \mathcal{I}$ son los elementos de cada entidad que se relacionan y v es un valor que informa respecto a esa interacción. Para cualesquiera u, i existirá un dato x que describirá su relación (ya sea observado o predicho), por lo cual se puede representar $\mathcal{X}$ como una función $f : (\mathcal{U}, \mathcal{I}) \rightarrow \{-1, +1\}$ que se denomina función de utilidad. Nótese que aunque v puede tomar cualquier valor, para este trabajo hacemos la simplificación de transformarlo a -1 o 1. Obtener una predicción del valor de esta función de utilidad es un problema muy habitual que se intenta resolver sobre estos datos y que ha sido repetidamente resuelto en la literatura mediante métodos como la factorización matricial [2]. Por otra parte, definiremos una agrupación $Cl(\mathcal{U})$ sobre un $\mathcal{U}$ como un conjunto de m grupos disjuntos que abarcan todos los elementos de $\mathcal{U}$. En fórmula:

$$Cl(\mathcal{U}) = \{Clu_1, \ldots, Clu_m\}. \tag{1}$$

Se puede utilizar la homogeneidad de la función de utilidad dentro de cada grupo Clu_k para establecer la idoneidad de la agrupación [17]. Definiremos con este objetivo la proporción p de elementos positivos en un grupo se como

$$p_{kj} = Pr(+1|Clu_k, i_j) = \frac{|\{u \in Clu_k : f(u, i_j) = +1\}|}{|Clu_k|} \tag{2}$$

donde i_j representa el elemento j-ésimo de $\mathcal{I}$.

Diremos que un grupo Clu_k es *coherente* cuando haya gran consenso entre sus valores de f, es decir, que p se acerque a 0 o a 1. Podemos medir esta coherencia, como anunciamos, utilizando la entropía de p.

$$H(p) = -p \log_2(p) - (1 - p) \log_2(1 - p). \tag{3}$$

[1] https://ec.europa.eu/info/law/law-topic/data-protection_en

$H(p_{kj})$ nos permite medir la coherencia de un único grupo Clu_k respecto al elemento i_j. Para extender esta medida a todo el agrupamiento $Cl(\mathcal{U})$ debemos tener en cuenta todos sus grupos y todos los elementos de $\mathcal{I}$. Para conseguir esto, utilizaremos la media ponderada. Formalmente, definimos la *entropía ponderada (EP)* del agrupamiento $Cl(\mathcal{U})$ como

$$EP(Cl(\mathcal{U})) = \sum_{k,j} \frac{|Clu_k|}{|\mathcal{U}||\mathcal{I}|} H(p_{kj}). \quad (4)$$

Con las características de interpretabilidad descritas en la Sección I en mente, en este trabajo nos enfocaremos en la capacidad de los algoritmos para motivar sus predicciones de manera sencilla utilizando las variables de entrada, con el objetivo de que un supervisor pueda formular y verificar hipótesis, así como también de resultar informativo respecto a la situación general del mercado en un instante dado. Debemos, por tanto, incluir en nuestra medida de idoneidad la complejidad de la explicación que requiere cada grupo, que mediremos con el número de variables que lo describen. Definimos, en consecuencia, la *calidad* de un agrupamiento como

$$calidad(Cl(\mathcal{U})) = -EP(Cl(\mathcal{U})) - \lambda \sum_{Clu_k \in Cl(\mathcal{U})} NV(Clu_k). \quad (5)$$

donde NV se refiere al número de variables necesarias para describir Clu_k y λ es un hiperparámetro que permite al supervisor balancear la entropía del agrupamiento con su explicabilidad. Se puede comprobar que existe un balance entre la complejidad de la explicación y la precisión del agrupamiento obtenido, lo cual es esperable intuitivamente: para obtener grupos uniformes generalmente será preciso formar gran número de ellos, lo que obligará a describirlos con más variables; ambas cosas reducen la explicabilidad del modelo. Por el contrario, para obtener un modelo más explicable será necesario obtener un menor número de grupos descritos con menos variables, que consecuentemente tendrán que ser menos uniformes. El hiperparámetro λ permite al supervisor decidir cómo manejar esta disyuntiva.

Cabe destacar que, aunque la *calidad* así descrita es un número negativo, esto no es relevante. El objetivo del algoritmo será maximizar su valor, acercándolo a 0. Esta medida de *calidad* se puede aplicar a agrupamientos construidos con cualquier algoritmo sobre una entidad $\mathcal{U}$ de un conjunto de datos diádicos.

IV. Algoritmo propuesto

En este trabajo proponemos un algoritmo de agrupamiento sobre datos diádicos (*post-hoc*) que obtenga grupos lo más homogéneos posibles y que, al mismo tiempo, se expliquen con el mínimo número posible de variables que describen a las entidades de la díada (*a priori*). De esta manera, en el escenario de la segmentación de mercado, el supervisor puede obtener información sobre cómo se dividen sus usuarios/productos y la relación entre estos grupos para así tomar decisiones informadas en consecuencia. Para ello se construirá un árbol de decisión que describirá una agrupación $Cl(\mathcal{U})$ en la que cada hoja del nodo representa un grupo descrito por las variables que conducen por el árbol hasta esa hoja.

El proceso global consiste en, a partir de la función de utilidad que relaciona a ambas entidades de la díada, realizar una exploración de los posibles árboles de decisión formados sobre las variables de entrada buscando maximizar la *calidad* de la agrupación resultante definida según la Ecuación 5.

Para poder explorar de manera eficiente el espacio de soluciones es necesario realizar ciertas simplificaciones. En primer lugar, para simplificar el cómputo de *calidad* de una agrupación y dado que el número de elementos de $\mathcal{I}$ puede hacer impracticable su cálculo, en lugar de realizar el cómputo de entropía respecto a cada elemento de $\mathcal{I}$, consideraremos una muestra aleatoria representativa de los elementos de $\mathcal{I}$. Una manera de hacer esto es haciendo un agrupamiento previo de $\mathcal{I}$ por procedimientos convencionales y tomando los centroides ci_j como representantes. Una vez calculados estos centroides, la proporción p descrita en la Ecuación 2 se aproximará como

$$p_{kj} = Pr(+1|Clu_k, ci_j) = \frac{|\{u \in Clu_k : f(u, ci_j) = +1\}|}{|Clu_k|} \quad (6)$$

y la Ecuación 4 deberá modificarse para tener en cuenta el tamaño del grupo Cli_j representado por ci_j

$$EP(Cl(\mathcal{U})) = \sum_{k,j} \frac{|Clu_k||Cli_j|}{|\mathcal{U}||\mathcal{I}|} H(p_{kj}). \quad (7)$$

Por otra parte, dado que el problema de explorar todos los posibles árboles de decisión anteriormente descritos es generalmente inabarcable, se hace necesario establecer una estrategia de búsqueda. En primer lugar, para evitar que el árbol de decisión pueda bifurcarse en cualquier posible valor de cada variable de entrada, debe reducirse el número de estos puntos de bifurcación. Se establecerá, por tanto, un máximo de puntos de bifurcación por variable y estos se determinarán mediante discretización en el caso de las variables continuas.

A continuación se construirá el árbol de decisión realizando una búsqueda voraz. Para ello en cada nodo se tomará el candidato más prometedor, que se determinará utilizando una heurística. Por tanto, siendo v_i el i-ésimo punto de corte de la variable V, calcularemos la heurística del agrupamiento $L = \{u \in \mathcal{U} : V(u) < v_i\}, R = \{u \in \mathcal{U} : V(u) > v_i\}$ asociado a v_i con la siguiente fórmula, derivada de manera empírica:

$$\eta(L, R) = -EP(L) - EP(R) - (|L| - |R|)^2 \quad (8)$$

Realizando una búsqueda voraz en la que para cada nodo se tome el punto de bifurcación candidato con mejor heurística y repitiendo el proceso recursivamente para los grupos L y R obtenidos hasta alcanzar un nivel L_{MAX} preestablecido por el usuario, se construirá un único árbol de decisión de L_{MAX} niveles. Para expandir la cantidad de espacio de búsqueda explorado y mejorar así las posibilidades de obtener una buena solución, el algoritmo propuesto explora en cada paso no solo el candidato con mejor valor de la heurística, sino los N mejores, dando lugar a N árboles posibles. Cada uno de

esos árboles, a su vez, darán lugar a otros $2 * N$ árboles al explorar el nivel siguiente, con lo cual el número de árboles explorados aumenta exponencialmente con el L_{MAX}. Deberá, en consecuencia, elegirse un valor bajo tanto para N como para L_{MAX}. El árbol de decisión obtenido será un árbol binario de L_{MAX} niveles y, por tanto, delimitará $2^{L_{MAX}}$ grupos.

Este proceso aparece descrito en el Algoritmo 1.

Datos: $\mathcal{U}, f, L_{MAX}, N$
Resultado: Árbol de decisión que determina el agrupamiento.
funcion CONSTRUYEARBOL($\mathcal{U}, nivel, puntosC, f, L_{MAX}, N$)
  si $nivel >$*NIVEL_MAX* **entonces**
    devolver $\emptyset$
  fin
  $candidatos \leftarrow$ lista ordenada con capacidad N;
  para $(variable, valor) \in puntosC$ **hacer**
1   $izq \leftarrow \{u \in \mathcal{U} : u[variable] < valor\}$;
    $der \leftarrow \{u \in \mathcal{U} : u[variable] > valor\}$;
    si HEURISTICA(izq, der) $> candidatos.minimo$ **entonces**
2     $candidatos.añadir((variable, valor))$;
    fin
  fin
  $mejor \leftarrow \emptyset$;
  para $(variable, valor) \in candidatos$ **hacer**
3   $izq \leftarrow \{u \in \mathcal{U} : u[variable] < valor\}$;
4   $der \leftarrow \{u \in \mathcal{U} : u[variable] > valor\}$;
5   $arbolIzq \leftarrow$ CONSTRUYEARBOL($izq, nivel + 1, puntosCorte$);
6   $arbolDer \leftarrow$ CONSTRUYEARBOL($der, nivel + 1, puntosCorte$);
7   $nuevo \leftarrow (variable, valor, arbolIzq, arbolDer)$
    si EP($nuevo, f$) $>$ EP($mejor, f$) **entonces**
8     $mejor = nuevo$;
    fin
  fin
  devolver $mejor$;
fin
$puntosC \leftarrow$ lista de puntos de corte en todas las variables;
devolver
  CONSTRUYEARBOL($\mathcal{U}, 0, puntosC, f, L_{MAX}, N$);

Algoritmo 1: Algoritmo de construcción del agrupamiento.

Por último, es posible que tras completar la búsqueda un subconjunto de este árbol alcance mayor calidad que el árbol completo. Por tanto, se intentará mejorar la calidad de la agrupación eliminando grupos mediante un proceso de poda. Para ello se recorrerán los nodos desde $L_{MAX} - 1$ hasta la raiz y se eliminarán aquellas bifurcaciones cuyo efecto en la *calidad* general no supere un $umbral$ indicado por el usuario, tal como aparece detallado en el Algoritmo 2.

El algoritmo descrito consta de cómputos que se pueden

Datos: $árbol$, $umbral$, λ
Resultado: Árbol podado.
funcion PODA($árbol$,$umbral$,λ,$nivel$)
1   **si** ESHOJA($árbol$) **entonces**
2   **devolver** $árbol$;
    fin
3   $izq \leftarrow$PODA($árbol.ramaIzq, umbral, nivel + 1$);
4   $der \leftarrow$PODA($árbol.ramaDer, umbral, nivel + 1$);
5   $entropíaDividido \leftarrow \frac{izq.entropía * |izq| + der.entropía * |der|}{|árbol|}$;
6   $\Delta E = entropíaDividido - árbol.entropía$;
7   $\Delta NV = izq.numVars + der.numVars - árbol.numVars$;
    si $-\Delta E - \lambda \Delta NV < umbral$ **entonces**
8     $árbol.ramaIzq \leftarrow \emptyset$;
9     $árbol.ramaDer \leftarrow \emptyset$;
    devolver $árbol$;
fin
devolver PODA($árbol$,$umbral$,λ,0);

Algoritmo 2: Algoritmo de podado del árbol.

realizar en paralelo dado que son independientes. Para aprovechar esta característica se ha implementado en Apache Spark. Valiéndose de la computación distribuida que facilita Spark se aumenta la escalabilidad del algoritmo y se posibilita así el análisis de grandes conjuntos de datos en tiempos razonables.

V. Experimentación

Para comprobar la validez de este algoritmo hemos realizado dos experimentos. En primer lugar hemos calculado un agrupamiento con una entropía ponderada comparable a la obtenida en [17] para una agrupación de 100 grupos obtenidos mediante K-Means. El segundo experimento consiste en comprobar el efecto sobre el tiempo de ejecución de la adición de más nodos de cómputo para el cálculo distribuido, para establecer qué nivel de escalabilidad tiene la implementación en Apache Spark del algoritmo propuesto.

Para ello hemos utilizado un conjunto de datos de la empresa Outbrain, liberado en el contexto de una competición auspiciada por el popular sitio web Kaggle.com. Outbrain es la empresa líder en descubrimiento de contenidos, mediante la sugerencia a lectores de noticias en las que pueden estar interesados. El conjunto de datos[2] recoge las visitas de un grupo de usuarios a las páginas web de multitud de publicaciones periodísticas durante 14 días. De cada visita se registra, además de metadatos como la fecha, la localización o el medio empleado para conectarse, el documento visitado, del cual se conocen ciertos aspectos del contenido, y el resultado de su interacción con los anuncios mostrados en la página que, a su vez, enlazan a otros documentos. Aunque la competición original buscaba predecir qué anuncios tendrían más éxito en una situación dada, el problema que hemos decidido abordar

[2]El conjunto de datos está disponible para descargar en el sitio https://www.kaggle.com/c/outbrain-click-prediction

Cuadro I
Descripción del conjunto de datos

Conjunto	Atributos	Muestras
DIA1	647	1,289,506

Cuadro II
Tiempo de cómputo del un árbol de 3 niveles. Escalabilidad

	Tiempo (H:M:S)		
# cores	12	2x12	4x12
DIA1	3:19:46	2:35:33	1:33:45

es la agrupación de los pares usuario-documento en función de su comportamiento. Para ello hemos determinado las visitas correlativas de un usuario y hemos registrado la preferencia del usuario por un documento ofrecido frente a otros, obteniendo tuplas de preferencias sobre las que aprender la función de utilidad mediante factorización matricial [17]. Dado que los documentos visitados se refieren a temas de actualidad, las visitas registradas varían de temática con el tiempo y, por ello, tiene sentido agrupar los datos en subconjuntos más pequeños que abarquen menos tiempo. En particular, para este trabajo hemos utilizado los datos recogidos durante el primer día, descritos en el Cuadro I. De acuerdo a lo descrito en la Sección IV, hemos agrupado las noticias (conjunto $\mathcal{I}$) en 100 grupos utilizando K-Means sobre los valores de la función de utilidad para cada noticia. El número de candidatos (N) que hemos explorado para cada nodo es 5.

V-A. *Resultados*

Para realizar el primer experimento es necesario determinar un valor de λ y decidir qué nivel de profundidad adquirirá el árbol calculado. En este caso optamos por un árbol de 5 niveles. Al fijar este valor, la agrupación obtenida constará de 32 grupos descritos con 5 variables cada uno, con lo cual su valor NV, necesario para computar la calidad según la Ecuación 5 será de $32 * 5 = 160$. Tal como describimos en la Sección III, el valor λ debe balancear el valor de la entropía ponderada (que oscila entre 0 y 1) con el valor $NV(Cl(\mathcal{U}))$, que en este caso oscilará entre 1 y 160. El valor de λ dependerá de la importancia que dé el supervisor a la interpretabilidad de la solución. En este experimento hemos tomado $\lambda = 0{,}001$.

Tras aplicar el algoritmo descrito al conjunto *DIA1* con los parámetros mencionados. La entropía ponderada de la agrupación es de 0.244 y, en consecuencia, su calidad es de -0.375. Tal como se indica en la Sección IV, este valor de calidad se puede mejorar con el proceso de poda descrito en el Algoritmo 2. Para ello es necesario establecer un umbral de mejora de la calidad que indique qué nodos deberán permanecer sin dividir. Habiendo calculado el árbol previamente, este proceso es lo suficientemente poco costoso como para que se pueda realizar la poda con cientos de umbrales distintos y representar en una gráfica los valores obtenidos, tal como se puede apreciar en la Figura 1. Tomando el valor de umbral (0.003) que obtiene mayor calidad, se obtuvo un árbol que describe 18 grupos, 2 de los cuales se describen usando 3 variables, 8 necesitan 4 variables y los 8 restantes se describen con 5 de las 647 variables. Su valor NV es, por tanto, 78, su entropía ponderada es de 0.2529 y su calidad es -0.3308. El árbol obtenido puede observarse en la Figura 2. Comparativamente, el agrupamiento obtenido en [17] ofrece una menor entropía ponderada (0.21), pero se compone de 100 grupos que precisan las 647 variables para describir sus centroides, lo que se traduce en un valor de calidad muy bajo (-64,91). Si utilizásemos un árbol de clasificación para predecir a qué grupo pertenece cada muestra y de él extrajésemos nuevas descripciones para los grupos, aún en el caso óptimo estas nunca serían menos de 8 variables de media (en consecuencia $NV = 800$), dado que se precisa un árbol binario de más de 8 niveles para discernir entre 100 elementos. Por tanto, la calidad de una agrupación compuesta de 100 grupos nunca será mayor de -1.01. Esto nos permite apreciar la utilidad del algoritmo propuesto para encontrar una agrupación bastante homogénea y con explicabilidad alta.

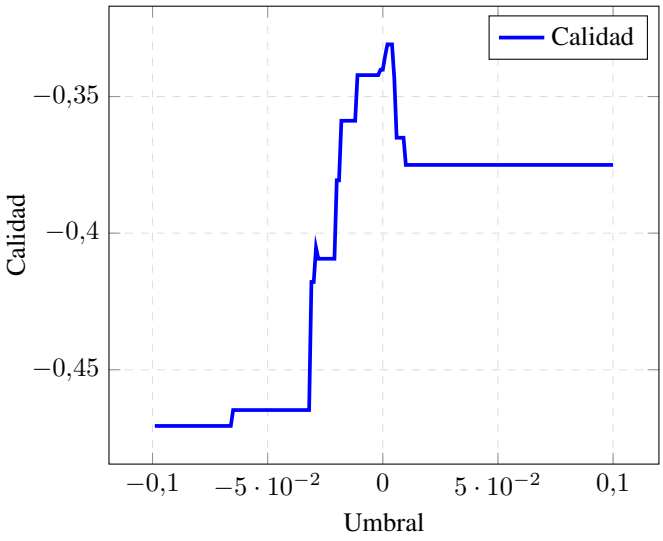


Figura 1. Calidad del agrupamiento vs umbral de bifurcación para el conjunto DIA1 con $\lambda = 0{,}001$.

En segundo lugar, para comprobar la escalabilidad del algoritmo, realizamos el mismo cómputo varias veces variando el número de núcleos de computación involucrados en el cálculo para observar su efecto en el tiempo invertido. El cómputo que realizamos se corresponde a la búsqueda del árbol óptimo de tres niveles. Los tiempos de ejecución aparecen reflejados en el Cuadro II.

Como se puede apreciar, la independencia de los cálculos realizados, que facilita su cómputo en paralelo, y la facilidad de distribución del cálculo que proporciona Apache Spark provocan que la escalabilidad sea buena, correspondiéndose casi linealmente el incremento de nodos de computación con el descenso del tiempo de ejecución.

VI. Conclusiones

En este trabajo hemos reflexionado sobre las características que debe cumplir un algoritmo para ser considerado fácilmente

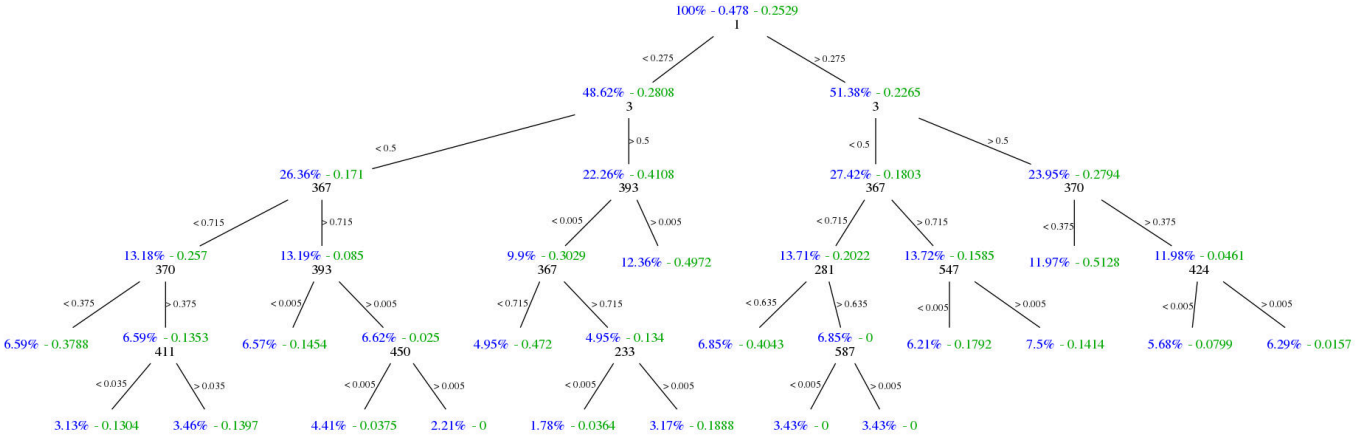


Figura 2. Árbol de decisión que determina el agrupamiento de los usuarios del conjunto de datos DIA1. En cada nodo se listan, respectivamente, la proporción de elementos que representa respecto al total (indicada en color azul), la entropía ponderada de ese nodo (verde) y la variable utilizada para la siguiente bifurcación (negro). Los valores utilizados para las bifurcaciones se indican al lado de las aristas. En la raíz se indica también en azul la entropía ponderada del conjunto entero.

interpretable y, adaptando una medida previamente utilizada en la literatura, hemos desarrollado un algoritmo de agrupamiento sobre datos diádicos que las cumple. El algoritmo propuesto permite, cuando es aplicado a datos referentes a interacciones en un mercado, realizar una segmentación de mercado a gran escala que ofrece a un supervisor información sobre qué está ocurriendo en ese medio en términos de las variables *a priori* que maneja. Asimismo, hemos realizado una implementación del citado algoritmo en la plataforma de computación distribuida Apache Spark que permite su aplicación a grandes volúmenes de datos y hemos realizado experimentos que verifican tanto la validez de este enfoque como la escalabilidad del mismo. Como trabajo futuro queremos adaptar este algoritmo para que pueda operar en situaciones en que el tiempo de cómputo es crucial. Para ello, queremos replantear la estrategia de búsqueda del árbol óptimo para que permita ser interrumpida en cualquier momento y ofrezca un resultado parcial relevante. Igualmente, creemos que la obtención de una heurística con mayor poder predictivo haría que la búsqueda fuese mucho más eficiente.

Agradecimientos

Los autores desean agradecer a la Fundación Pública Galega Centro Tecnolóxico de Supercomputación de Galicia (CESGA) el uso de sus recursos de computación.

Referencias

[1] Thomas Hofmann, Jan Puzicha, and Michael I Jordan. Learning from dyadic data. In *Advances in neural information processing systems*, pages 466–472, 1999.

[2] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8), 2009.

[3] Oscar Luaces, Jorge Díez, Amparo Alonso-Betanzos, Alicia Troncoso, and Antonio Bahamonde. A factorization approach to evaluate open-response assignments in moocs using preference learning on peer assessments. *Knowledge-Based Systems*, 85:322–328, 2015.

[4] Oscar Luaces, Jorge Díez, Amparo Alonso-Betanzos, Alicia Troncoso, and Antonio Bahamonde. Content-based methods in peer assessment of open-response questions to grade students as authors and as graders. *Knowledge-Based Systems*, 117:79–87, 2017.

[5] Philip Kotler and Keith Kohn Cox. *Marketing management and strategy*. Prentice Hall, 1980.

[6] Philip Kotler. *Dirección de mercadotecnia: Análisis, planeación, implementación y control*. Magíster en Administración-Tiempo Parcial 29, ESAN, 2001.

[7] Zachary C Lipton. The mythos of model interpretability. *arXiv preprint arXiv:1606.03490*, 2016.

[8] Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1721–1730. ACM, 2015.

[9] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.

[10] Jiapeng Liu, Xiuwu Liao, Wei Huang, and Xianzhao Liao. Market segmentation: A multiple criteria approach combining preference analysis and segmentation decision. *Omega*, 2018.

[11] Pavel Berkhin. A survey of clustering data mining techniques. In *Grouping multidimensional data*, pages 25–71. Springer, 2006.

[12] Bo Fan and Pengzhu Zhang. Spatially enabled customer segmentation using a data classification method with uncertain predicates. *Decision Support Systems*, 47(4):343–353, 2009.

[13] Melody Y Kiang, Michael Y Hu, and Dorothy M Fisher. An extended self-organizing map network for market segmentation—a telecommunication example. *Decision Support Systems*, 42(1):36–47, 2006.

[14] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[15] Hanhuai Shan and Arindam Banerjee. Bayesian co-clustering. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 530–539. IEEE, 2008.

[16] Jayanta Basak and Raghu Krishnapuram. Interpretable hierarchical clustering by constructing an unsupervised decision tree. *IEEE transactions on knowledge and data engineering*, 17(1):121–132, 2005.

[17] Jorge Díez, Pablo Pérez, Oscar Luaces, and Antonio Bahamonde. Readers segmentation according to their preferences to click promoted links in digital publications. Technical report, Universidad de Oviedo, 2018.

Bagging-RandomMiner - Un Algoritmo en MapReduce para Detección de Anomalías en Big Data

Luis Pereyra*, Diego García-Gil†, Francisco Herrera†, Luis C. González-Gurrola*, Jacinto Carrasco†, Miguel Angel Medina-Pérez‡ y Raúl Monroy‡

*Facultad de Ingeniería, Universidad Autónoma de Chihuahua,
Chihuahua, Chih. 31000, México
Email: {a261804, lcgonzalez}@uach.mx
†Departamento de Ciencias de la Computación e Inteligencia Artificial,
Universidad de Granada, Granada, España, 18071
Email: {djgarcia,herrera,jacintocc}@decsai.ugr.es
‡Tecnológico de Monterrey, Campus Estado de México,
Carretera al Lago de Guadalupe Km. 3.5, Atizapán de Zaragoza, Estado de México, 52926, México.
Email: {migue, raulm}@itesm.mx

***Index Terms*—Detección de Anomalías, Big Data, MapReduce, Bagging-RandomMiner, Apache Spark, Detección de Anomalías**

***Resumen*—La detección de anomalías es una tarea de interés en muchas aplicaciones del mundo real cuyo objetivo es identificar el comportamiento irregular de un proceso basado en el aprendizaje de cómo éste se comporta normalmente. En este trabajo, presentamos una versión MapReduce de una estrategia que ha demostrado ser exitosa para encontrar valores atípicos (anomalías) en conjuntos de datos de tamaño mediano. Esta estrategia, conocida como Bagging-RandomMiner, ahora ha sido extendida para aprovechar las bondades que Apache Spark ofrece y poder demostrar su utilidad en contextos de Big Data. Para evaluar su rendimiento, se ha utilizado el conjunto de datos PRIDE como Benchmark para identificar situaciones de estrés (anomalías) en un grupo de 23 sujetos. En este estudio comparamos Bagging-RandomMiner contra otra versión MapReduce del algoritmo que mejores resultados ha reportado sobre PRIDE, llamado OCKRA. Los experimentos indican que Bagging-RandomMiner supera claramente a OCKRA en al menos 6 % en el score AUC, más aún su tiempo de ejecución es al menos un orden de magnitud menor. Estos resultados soportan la idea de Bagging-RandomMiner como un algoritmo contendiente en tareas de Big Data. En este trabajo hacemos disponible el código de esta versión.**

I. Introducción

Hoy en día el mundo avanza implacablemente hacia la era de Big Data. Todos los días se generan quintillones de datos. La *International Data Corporation* (IDC) predijo en 2014 que para 2020, el universo digital sería diez veces más grande que en 2013[1]. La tecnología actual se ha visto superada por esta cantidad de datos. Esto ha provocado que el concepto de Big Data aparezca como un nuevo paradigma para permitir el almacenamiento y procesamiento de toda esta información. Dentro de los desafíos que presenta Big Data, hay uno de especial relevancia que ha encontrado un buen número de aplicaciones en el mundo real. La **detección de anomalías** se define como la tarea de averiguar cuándo un sistema o proceso está actuando erráticamente [1]. La importancia de esta tarea es tal que la existencia de un enfoque eficaz podría llevar a brindar soluciones a escenarios altamente sensibles, como seguridad en redes de computadoras, la detección de fraude bancario o incluso situaciones de vida o muerte [2], [3], [4].

Recientemente, se ha propuesto un algoritmo llamado Bagging-RandomMiner [5] para identificar anomalías o valores atípicos en conjuntos de datos de tamaño mediano. Motivados por la calidad de los resultados que ha registrado, incluso superando a conocidos competidores en tareas de Data Mining tales como AdaBoost.M1, Random Forrest y TreeBagger, entre otros, y su posible aplicación en escenarios de Big Data, presentamos en este artículo un diseño de MapReduce [6] de este algoritmo en el framework Apache Spark. Para evaluar este algoritmo utilizamos el conjunto de datos de Detección de Riesgos Personales (PRIDE) [7], que se creó para servir como benchmark para la detección de anomalías en el contexto de la detección de situaciones de estrés (o peligro) experimentadas por un grupo de 23 personas. Para contrastar los resultados obtenidos por Bagging-RandomMiner, también implementamos la versión MapReduce del algoritmo que mejores resultados ha reportado sobre PRIDE, conocido como OCKRA [8]. La comparación de ambas estrategias muestra que nuestro diseño supera los resultados del estado del arte en al menos $\sim 6\,\%$ en el score promedio de Área Bajo la Curva (AUC), lo cual ha sido validado por el test Bayesian Signed-Rank test. Así mismo su tiempo de ejecución es al menos un

[1]IDC: The Digital Universe of Opportunities. 2018 [Online] Disponible: http://www.emc.com/infographics/digital-universe-2014.htm

orden de magnitud menor
Ponemos a disposición de la comunidad esta implementación en el repositorio respectivo de Spark[2].

La organización de este trabajo se estructura de la siguiente manera: la Sección II presenta las definiciones básicas de la tarea de detección de anomalías, el conjunto de datos PRIDE y el modelo MapReduce. La Sección III describe el diseño del algoritmo distribuido propuesto. La Sección IV muestra los experimentos llevados a cabo sobre el conjunto de datos PRIDE. Finalmente, la Sección V brinda algunas conclusiones de este trabajo.

II. Antecedentes

II-A. Detección de Anomalías

En la tarea de Detección de Anomalías (también conocida como detección de valores atípicos) [1], a diferencia de la clasificación binaria (o multiclase), el clasificador aprende a reconocer solo una categoría de objeto, la que pertenece a la clase mayoritaria. En general, se supone que la clase mayoritaria representa regularidad, es decir, objetos que se expresan a menudo en el dominio del problema. Suponiendo que el clasificador es competente en esta tarea, es fácil ver que éste discriminará automáticamente un objeto que no parece pertenecer a la clase mayoritaria, es decir, una anomalía.
Teniendo este contexto se puede ver que la Detección de Anomalías es relevante en una gran cantidad de escenarios, por ejemplo, para identificar ataques en la web [2], rastrear errores de software [3] y detección de fraude [4], entre otros. En este estudio nos enfocaremos en la tarea de detección de anomalías usando como Benchmark el conjunto de datos conocido como PRIDE.

II-B. PRIDE y OCKRA

El conjunto de datos PRIDE fue diseñado originalmente por Barrera et al. [7] con el objetivo de ayudar a apoyar tareas de detección de anomalías en un escenario sensible de usuarios que experimentan situaciones estresantes. PRIDE fue creado con la ayuda de 23 usuarios que fueron supervisados durante una semana las 24 horas del día mientras realizaban sus actividades cotidianas. La recopilación de la información se logró a través de un sensor de la empresa Microsoft, llamado *Microsoft Band*. A partir de los datos generados por esta banda, se extrajeron 26 características. La Tabla I presenta las características consideradas en PRIDE para cada usuario. Para recolectar datos asociados a situaciones estresantes (condiciones de anomalía), los 23 usuarios fueron sometidos a diferentes escenarios de estrés que simulaban condiciones peligrosas que podrían enfrentarse en la vida real, por ejemplo, subiendo y bajando escaleras de un edificio, corriendo 16 metros a alta velocidad o una sesión de boxeo para simular una pelea, entre otras actividades [7]. En promedio hay 323,161 muestras por usuario y un total de 7,432,715 muestras para los 23 usuarios.

OCKRA a su vez, es el método que mejores resultados reporta usando la base de datos PRIDE, incluso superando a una Máquina de Vectores de Soporte para una sola clase (OC-SVM) [9]. El algoritmo *K-means One-Class with Randomly-projected features Algorithm* (OCKRA) [8] es un ensamble de 100 clasificadores de una sola clase, basado en múltiples proyecciones del conjunto de datos respecto a subconjuntos de datos aleatorios de características. OCKRA aplica k-means++ [10] a cada subconjunto de características para obtener una colección de centroides que representan la distribución de los datos. Para clasificar una nueva observación, cada clasificador determina una medida de similitud (0 – 1). Finalmente, se promedian las similitudes de cada clasificador y se obtiene la probabilidad de pertenencia a la clase mayoritaria.

Tabla I
Estructura del conjunto de datos PRIDE

Característica			Número
Acelerómetro Giroscópico	**Eje X**	$\bar{x}$	1
		s	2
	Eje Y	$\bar{x}$	3
		s	4
	Eje Z	$\bar{x}$	5
		s	6
Velocidad Angular	**Eje X**	$\bar{x}$	7
		s	8
	Eje Y	$\bar{x}$	9
		s	10
	Eje Z	$\bar{x}$	11
		s	12
Acelerómetro	**Eje X**	$\bar{x}$	13
		s	14
	Eje Y	$\bar{x}$	15
		s	16
	Eje Z	$\bar{x}$	17
		s	18
Ritmo Cardíaco			19
Temperatura de la piel			20
Pasos			21
Velocidad			22
UV			23
Δ Podómetro			24
Δ Distancia			25
Δ Calorías			26

II-C. Modelo MapReduce

MapReduce es un framework introducido por Google en 2003 [6]. Este modelo se compone de dos procedimientos para procesar datos de forma paralela: *Map* y *Reduce*. La operación de *Map* realiza una transformación a los datos de entrada, mientras que el método *Reduce* consiste en una operación de agregación. El flujo de trabajo de un programa MapReduce se compone de cuatro pasos: primero, el nodo maestro divide los datos de entrada y los distribuye en todos los nodos. En segundo lugar, la operación *Map* aplica una transformación a los datos presentes localmente. A continuación, los resultados se redistribuyen en el clúster en función de los pares clave-valor generados en la operación *Map*. Después de que este proceso haya finalizado, todos los pares que pertenecen a una misma clave están en el mismo nodo. Finalmente, los pares clave-valor se procesan en paralelo.
Apache Spark [3] es un framework de código abierto basado en

[2]https://spark-packages.org/package/wuicho-pereyra/Bagging-RandomMiner

[3]Apache Spark Project 2018 [Online] Disponible: https://spark.apache.org/

el modelo MapReduce construido para favorecer la eficiencia, la facilidad de uso y el cálculo en memoria interna [11], [12]. La característica central de Spark son los *Resilient Distributed Datasets* (RDDs) [13]. Los RDDs se pueden describir como una colección distribuida e inmutable de particiones de los datos, distribuidos por el clúster. Los RDDs implementan dos tipos de operaciones: *(i)* transformaciones, que no se evalúan cuando se definen y devuelven un nuevo RDD después de aplicar una función; y *(ii)* acciones, que devuelven un valor y desencadenan todas las transformaciones previas del RDD. Todas las transformaciones se aplican en paralelo a cada partición.

III. Una versión MapReduce de Bagging-RandomMiner

Para una descripción de la versión local del algoritmo Bagging-RandomMiner se sugiere al lector consultar el trabajo [5]. Bajo el modelo de MapReduce se usaron las siguientes funciones de Spark: (*i*) **mapPartitions** (aplica una función a cada partición de un RDD y devuelve un nuevo RDD), (*ii*) **zipWithIndex** (agrega a un RDD sus índices de elementos) y, (*iii*) **join** (devuelve un RDD que contiene todos los pares de elementos con claves coincidentes en éste y otro RDD). Bagging-RandomMiner está compuesto por dos fases: aprendizaje y clasificación. En la fase de aprendizaje, se selecciona de manera aleatoria un porcentaje RS de los datos. A partir de esta muestra, se realiza otro sub muestreo para seleccionar aquellos objetos que representarán la distribución completa de los objetos originales, los cuales son llamados **Objetos Más Representativos de la población (MRO)**. A partir de aquí, se calcula un umbral de decisión (δ) promediando la matriz de distancia de los MRO con la ayuda de la función *mapPartitions*. Esto se hace iterativamente para un número T de clasificadores. En la fase de clasificación, debido a que los RDD se distribuyen aleatoriamente de forma natural, la función *zipWithIndex* se aplica para agregar un índice a cada instancia. Luego, cada instancia de test se compara con los MRO, eligiendo solo el que tenga la distancia más cercana a la instancia de test. Finalmente, la similitud se calcula con la distancia más cercana y el umbral (δ), esto corresponde a la probabilidad del comportamiento genuino. Este proceso se repite hasta que se completen los T clasificadores, y, mediante la función *join*, se almacena el voto de cada clasificador, lo que nos permite promediar un resultado final. La Figura 1 muestra un diagrama del funcionamiento de ambas fases.

III-A. Fase de Aprendizaje

El Algoritmo 1 es la clase principal que orquesta la operación de Bagging-RandomMiner. Para iniciar el procedimiento, es necesario proporcionar el conjunto de entrenamiento ($dataTrain$) y el conjunto de prueba ($dataTest$) en la estructura RDD de tipo $LabeledPoint$, el número de clasificadores (T), el porcentaje de objetos seleccionados para cada clasificador (RS), el porcentaje de objetos más representativos ($MRO_{percent}$) y el tipo de distancia que se aplicará ($disType; 1 \rightarrow$ euclidea, $2 \rightarrow$ manhattan y $3 \rightarrow$ chebyshev). En el paso 5, un clasificador se entrena aplicando el Algoritmo 2, pasando los parámetros de $dataTrain$, RS y $MRO_{percent}$ como argumentos. En el paso 7, se obtiene una predicción del conjunto de test aplicando el Algoritmo 3, pasando como parámetros el conjunto de prueba y el modelo entrenado. Luego, cada vez que se hace una predicción del conjunto de test, la función de $union$ se usa para almacenar las predicciones de cada clasificador (paso 8). Finalmente, en el paso 10, se promedian los resultados del clasificador para obtener la probabilidad de pertenencia al comportamiento genuino de cada instancia de test, y se devuelve en una variable con estructura RDD [índice, probabilidad, clase real].

El Algoritmo 2 es la parte principal de la fase de aprendizaje. Aquí es donde se determinan los MRO y el umbral de decisión para la clasificación de un nuevo objeto. Los parámetros necesarios para comenzar con esta etapa son: $dataTrain$, RS, $MRO_{percent}$ y $disType$. Primero, debemos determinar los MRO del conjunto. Para lograr esto, en el paso 3 elegimos aleatoriamente el porcentaje de RS de los datos que serán usados por el clasificador para representar la distribución de los datos originales. Luego, en el paso 4, el porcentaje de MRO indicado se selecciona aleatoriamente, siendo estos objetos los prototipos que representan el clasificador. Ahora, para determinar el umbral de decisión (δ) es necesario calcular la matriz de distancias. Con la ayuda de la función $mapPartitions$, la matriz de distancias se calcula para cada nodo, es decir, en paralelo, cada nodo es responsable de calcular el promedio de la matriz de distancias con los MRO correspondientes. Al final, cada uno de los promedios se recopila y se calcula el promedio de estos resultados, que es el umbral de decisión δ (pasos 5-15). Finalmente, se crea un modelo y se devuelven los MRO, el umbral calculado δ y el tipo de distancia utilizada (paso 16).

III-B. Fase de predicción

El Algoritmo 3 es responsable de la clasificación de los nuevos objetos. Para lograr esto, necesitamos el $DataTest$ y el modelo entrenado por el Algoritmo 2. El proceso comienza aplicando la función $zipWithIndex$ (paso 3) que asigna un índice a cada instancia para no perder el orden y poder aplicar la función de unión en el algoritmo 1. Después, con la ayuda de la función $mapPartitions$, la distancia de cada objeto de test se calcula con todas las MRO distribuidos en cada nodo. Al final, se elige la distancia más cercana ($dmin$) al objeto de test (paso 10) y la medida de similitud se calcula con la $dmin$ obtenida y el umbral de decisión δ (paso 11). Cuando todos los nodos terminan sus respectivas ejecuciones, el Algoritmo 3 devuelve un RDD [índice, probabilidad] (paso 14).

IV. Resultados Experimentales

Los experimentos se diseñaron utilizando *5-fold cross-validation* y dos versiones de Bagging-RandomMiner, variando el número de clasificadores en el ensamble: 10 y 50, respectivamente. Este último aspecto es interesante de evaluar ya que OCKRA también es un método de ensamble con un número fijo de clasificadores igual a 100. En cuanto a los

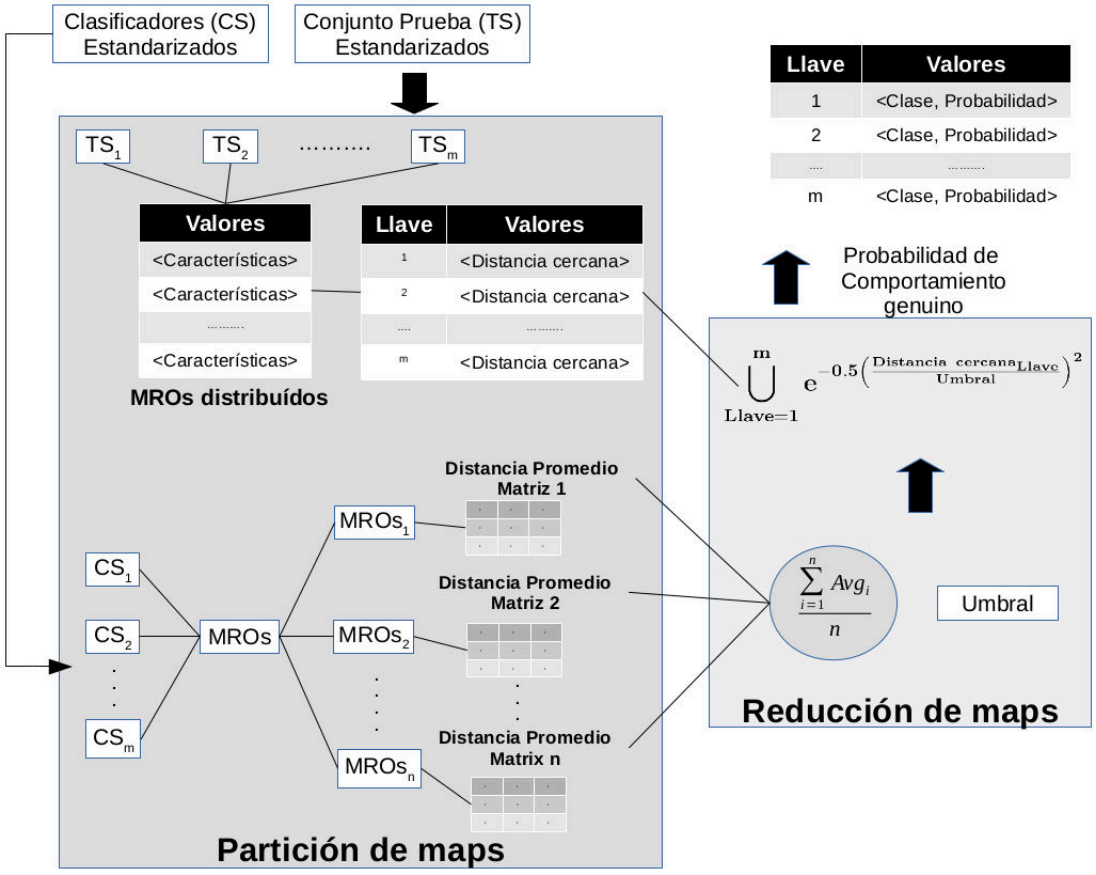


Figura 1. Diagrama de aprendizaje y predicción de Bagging-RandomMiner en MapReduce.

Algoritmo 1 Main Bagging-RandomMiner

Entrada: *datos:* RDD de tipo *LabeledPoint* (características, clase), *prueba:* RDD de tipo LabeledPoint (características, clase) *T:* número de clasificadores, *RS:* porcentaje de muestreo, $MRO_{percent}$: porcentaje de *MROs* y *distType:* tipo de distancia.
Salida: Probabilidad de pertenencia al comportamiento genuino.
for $i = 0...T$ **do** || Itera para crear un modelo de cada clasificador.
    || Se manda llamar el algoritmo 2.
    $Model \leftarrow$ Aprendizaje$(dataTrain, RS, MRO_{percent}, distType)$
    || Se manda llamar el algoritmo 3
    $pred_tmp \leftarrow$ Predicción$(dataTest, Model)$
    $predicciones$.join$(pred_tmp)$ || Con la función *join*, las predicciones se almacenan con la llave índice.
end for
return Promedio$(predicciones)$ || Los resultados de cada modelo son promediados, esto corresponde a la probabilidad del comportamiento genuino

Algoritmo 2 Aprendizaje

Entrada: *dataTrain:* RDD de tipo LabeledPoint (características, clase), *RS:* porcentaje de muestreo, $MRO_{percent}$: porcentaje de *MROs* y *distType:* tipo de distancia.
Salida: El modelo entrenado, un objeto de clase RandomMiner
$dataTrain' \leftarrow$ Clasificador$(dataTrain, RS)$ || Un porcentaje de los datos es seleccionado aleatoriamente.
$MROs \leftarrow$ Seleccionar$(MRO_{percent},\ dataTrain')$ || Un porcentaje de MRO es seleccionado aleatoriamente.
mapPartition $l \in MROs$ || Computar la distancia de cada MRO.
    $Distance \leftarrow 0$ || Se inicializa la distancia.
    for $i = 0$ To $size(l)$ **do** || Itera sobre cada MRO de la partición.
        for $j = i + 1$ To $size(l)$ **do** || Itera sobre cada MRO de la partición.
            || Se calcula la distancia entre cada par de MRO
            $Distance \leftarrow Distance +$ CalcDist$(l(i),\ l(j),\ distType)$
        end for
    end for
    $umbral$.append(Avg$(Distance)$) || El promedio de cada partición.
end mapPartition
$\delta \leftarrow$ Avg$(umbral)$ || Promedio global de las particiones.
return $(MROs,\ \delta,\ distType)$ || Se retorna un modelo con los MRO, el umbral δ y el tipo de distancia.

parámetros de RandomMiner, se usaron la distancia chebyshev y los valores del $1\,\%$ para muestreo de los datos y $1\,\%$ de objetos MRO.

La métrica para evaluar los modelos es el área bajo la curva (AUC) de la tasa de detección positiva (comportamiento genuino) (TP) versus la tasa de detección de falso positivo (comportamiento anormal) (FP). Este indicador de acuerdo con [14] es una excelente medida para evaluar clasificadores

Algoritmo 3 Predicción

1: **Entrada:** *dataTest:* RDD de tipo *LabeledPoint* (características, clase), *Model:* modelo entrenado (*MRO, δ, distType*).
2: **Salida:** Probabilidad de pertenencia al comportamiento genuino.
3: $testIndex \leftarrow$ zipWithIndex$(dataTest)$ ‖ Un índice es añadido a cada instancia.
4: **mapPartition** $l \in testIndex$ ‖ Se evalua por partición el conjunto de prueba (maps)
5: **for** $i = 0$ To $size(l)$ **do** ‖ Itera sobre cada instancia de la partición.
6: **for** $j = 0$ To $size(MROs)$ **do** ‖ Itera sobre cada MRO de la partición.
7: ‖ Se calcula la distancia entre cada instancia de prueba y los MRO.
8: $Distance(j) \leftarrow$ CalcDist$(l(i),\ MRO(j),\ disType)$
9: **end for**
10: $dmin \leftarrow$ min$(Distance)$ ‖ Para cada instancia de prueba, se determina la más cercana a los MRO.
11: $similaridad \leftarrow e^{-0{,}5\left(\frac{dmin}{\delta}\right)^2}$ ‖ Se determina la similaridad de la distancia más cercana.
12: **end for**
13: **end mapPartition**
14: **return** $similaridad$ de tipo RDD ‖ Probabilidad del comportamiento genuino.

de una clase. Finalmente, parar tener mejor evidencia que la ofrecida por test frecuentistas [15], se aplicó la prueba estadística *Bayesian Signed Rank Test* [16] para comparar la métrica AUC de la mejor versión de Bagging-RandomMiner vs OCKRA.

La infraestructura de cómputo (clúster) utilizado para todos los experimentos en este trabajo está compuesto por 14 nodos gestionados por un nodo maestro. Todos los nodos tienen la misma configuración de $hardware$ y $software$. En cuanto al $hardware$, cada nodo tiene 2 procesadores $IntelXeonE5-2620$, 6 núcleos (12 hilos) por procesador, 2 GHz y 64 GB de RAM. La red utilizada es $Infiniband$ de $40Gb/s$. El sistema operativo es Cent OS 6.5, con Apache Spark y MLlib 2.2.0, 336 núcleos (24 núcleos / nodo), 728 GB de RAM (52 GB / nodo).
La Figura 2 muestra el porcentaje del score AUC para cada uno de los 23 sujetos que integran el dataset PRIDE obtenido por RandomMiner (2 configuraciones) y OCKRA. De acuerdo a esta figura se pueden obtener 2 conclusiones: (1) los resultados de RandomMiner mejoran la identificación de anomalías en prácticamente todos los usuarios, y (2) No hay una diferencia marcada entre usar 10 clasificadores o 50 para RandomMiner. Este último punto sugiere que el método es robusto.

Para analizar estadísticamente los resultados se aplicó el test no paramétrico *Bayesian Signed Rank* a un nivel de confianza del $95\,\%$. Este test, a diferencia de los test frecuentistas (t-test), nos permite ser más concluyentes respecto a las diferencias de los algoritmos comparados [15]. En la Figura 3 se muestra a través de coordinadas baricéntricas la distribución de diferencias en el desempeño de los algoritmos en tres regiones: $left$ (OCKRA es mejor que RandomMiner), $rope$ (no hay diferencia) y $right$ (RandomMiner es mejor que OCKRA). Observando esta figura es evidente que la mayoría de los casos se inclina más a RandomMiner que a la región de no diferencia y definitivamente más que a soportar la probabilidad de que OCKRA sea el mejor algoritmo.

La Figura 4 muestra el tiempo de ejecución total para las dos versiones de RandomMiner, 10 y 50 clasificadores con 256 $maps$ para cada ejecución y OCKRA. Ambas versiones de RandomMiner tienen un mejor tiempo de ejecución que OCKRA, ahora, si consideramos que usando 10 clasificadores en RandomMiner es suficiente para obtener resultados competitivos esta diferencia se vuelve sustancial.

Dado que la contribución de este artículo es un nuevo diseño escalado de Bagging-RandomMiner, es importante contextualizar el tiempo de ejecución con respecto a la versión original del mismo algoritmo, llamado Local, esto se puede observar en la Figura 5. La escalabilidad de Bagging-RandomMiner es notable, ya que el tiempo de ejecución del algoritmo por usuario pasó de 14.4 horas a 3.66 minutos en promedio.

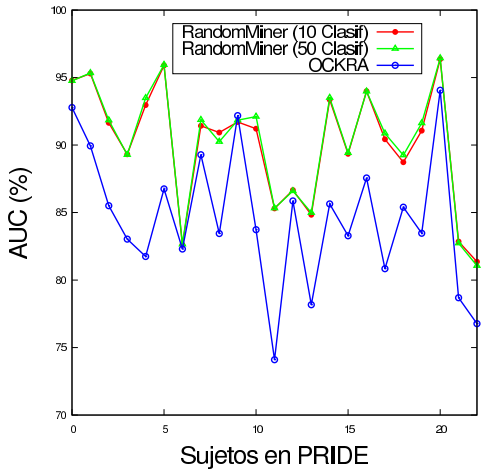


Figura 2. Porcentaje de Área bajo la curva (AUC) para RandomMiner (2 configuraciones) y OCKRA para cada uno de los 23 sujetos en PRIDE

V. Conclusiones

En este trabajo, se ha propuesto un diseño escalable y distribuido del algoritmo Bagging-RandomMiner desde la perspectiva MapReduce, basada en la filosofía de ensambles y aleatoriedad, que permite abordar problemas con alta dimensionalidad en el área de clasificación de una sola clase, para la detección de anomalías y conjuntos de datos altamente desequilibrados. Los resultados sugieren que Bagging-RandomMiner supera a OCKRA, la mejor estrategia para la detección de anomalías en el conjunto de datos PRIDE. Además, el método parece ser robusto con respecto al número de clasificadores que considera, ya que con un número muy pequeño de clasificadores (10) se logra un alto rendimiento y mejora sustancialmente el tiempo de ejecución.

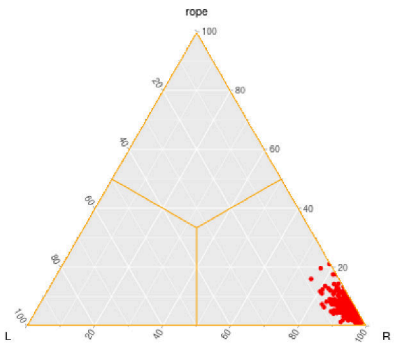


Figura 3. Resultados del Bayesian Signed Rank test sobre los resultados de OCKRA y Bagging-RandomMiner. La aglutinación de datos en la esquina inferior derecha (asociada a RandomMiner) sugiere fuertemente que éste es mejor algoritmo que OCKRA.

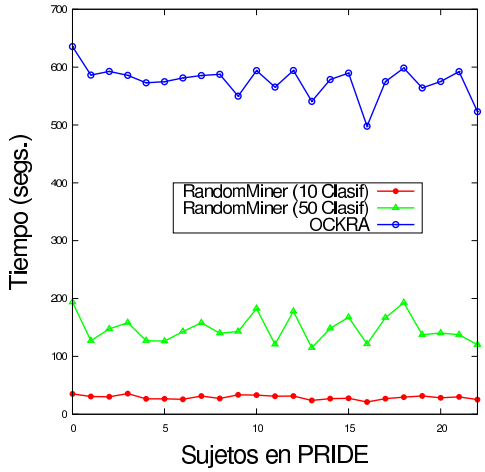


Figura 4. Tiempo de ejecución de los algoritmos RandomMiner (2 configuraciones) y OCKRA para cada uno de los 23 sujetos en PRIDE

Agradecimientos: Se agradece al Consejo Nacional de Ciencia y Tecnología de México (CONACYT) por la beca 620097 para estudios de posgrado otorgada al primer autor.

Referencias

[1] B. Krawczyk and B. Cyganek, "Selecting locally specialised classifiers for one-class classification ensembles," *Pattern Anal. Appl.*, vol. 20, no. 2, pp. 427–439, May 2017. [Online]. Available: https://doi.org/10.1007/s10044-015-0505-z

[2] C. Kruegel and G. Vigna, "Anomaly detection of web-based attacks," in *Proceedings of the 10th ACM Conference on Computer and Communications Security*, ser. CCS '03. New York, NY, USA: ACM, 2003, pp. 251–261. [Online]. Available: http://doi.acm.org/10.1145/948109.948144

[3] S. Hangal and M. S. Lam, "Tracking down software bugs using automatic anomaly detection," in *Proceedings of the 24th International Conference on Software Engineering*, ser. ICSE '02. New York, NY, USA: ACM, 2002, pp. 291–301. [Online]. Available: http://doi.acm.org/10.1145/581339.581377

[4] T. Fawcett and F. Provost, "Adaptive fraud detection," *Data Mining and Knowledge Discovery*, vol. 1, no. 3, pp. 291–316, Sep 1997. [Online]. Available: https://doi.org/10.1023/A:1009700419189

[5] J. B. Camiña, M. A. Medina-Pérez, R. Monroy, O. Loyola-González, L. A. P. Villanueva, and L. C. González-Gurrola, "Bagging-randomminer: a one-class classifier for file access-based masquerade detection," *Machine Vision and Applications*, Jul 2018. [Online]. Available: https://doi.org/10.1007/s00138-018-0957-4

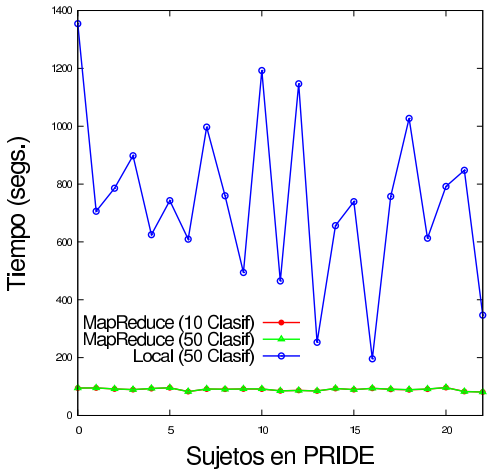


Figura 5. Tiempo de ejecución de los algoritmos RandomMiner en versión MapReduce y local.

[6] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, ser. OSDI'04, 2004, pp. 10–10.

[7] A. Y. Barrera-Animas, L. A. Trejo, S. García, M. A. Medina-Pérez, and R. Monroy, "Online Personal Risk Detection Based on Behavioural and Physiological Patterns," *Information Sciences*, August 2016.

[8] J. Rodríguez, A. Y. Barrera-Animas, L. A. Trejo, M. A. Medina-Pérez, and R. Monroy, "Ensemble of one-class classifiers for personal risk detection based on wearable sensor data," *Sensors*, vol. 16, no. 10, 2016.

[9] L. Trejo and A. Barrera-Animas, "Towards and efficient one-class classifier for mobile devices and wearable sensors on the context of personal risk detection," *Sensors*, vol. 18, no. 9, 2018.

[10] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '07. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035. [Online]. Available: http://dl.acm.org/citation.cfm?id=1283383.1283494

[11] D. García-Gil, S. Ramírez-Gallego, S. García, and F. Herrera, "A comparison on scalability for batch big data processing on Apache Spark and Apache Flink," *Big Data Analytics*, vol. 2, no. 1, p. 11, Mar 2017.

[12] D. García-Gil, S. Ramírez-Gallego, S. García, and F. Herrera, "Principal components analysis random discretization ensemble for big data," *Knowledge-Based Systems*, vol. 150, pp. 166 – 174, 2018.

[13] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 2–2. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228298.2228301

[14] T. Fawcett, "An introduction to roc analysis," *Pattern Recogn. Lett.*, vol. 27, no. 8, pp. 861–874, Jun. 2006. [Online]. Available: http://dx.doi.org/10.1016/j.patrec.2005.10.010

[15] A. Benavoli, G. Corani, J. Demsar, and M. Zaffalon, "Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis," *Journal of Machine Learning Research*, vol. 18, pp. 1–36, 2017.

[16] J. Carrasco, S. García, M. del Mar Rueda, and F. Herrera, "rnpbst: An r package covering non-parametric and bayesian statistical tests," in *Hybrid Artificial Intelligent Systems*, F. J. Martínez de Pisón, R. Urraca, H. Quintián, and E. Corchado, Eds. Cham: Springer International Publishing, 2017, pp. 281–292.

CGLAD: GLAD en problemas de Big Crowdsourced Data

Enrique G. Rodrigo
Departamento de Sistemas Informáticos
Universidad de Castilla-La Mancha
Albacete, España
Enrique.GRodrigo@uclm.es

Juan A. Aledo
Departamento de Matemáticas
Universidad de Castilla-La Mancha
Albacete, España
JuanAngel.Aledo@uclm.es

José A. Gámez
Departamento de Sistemas Informáticos
Universidad de Castilla-La Mancha
Albacete, España
Jose.Gamez@uclm.es

***Resumen*—En este artículo proponemos una mejora del algoritmo GLAD con el fin de mejorar su funcionamiento en problemas con grandes conjuntos de datos, en términos de eficiencia y de precisión del modelo resultante. El algoritmo GLAD permite aprender a partir de datos procedentes de múltiples anotadores, teniendo en cuenta su capacidad y la dificultad de las instancias que se predicen. Sin embargo, debido al número de parámetros del modelo, este no escala bien a grandes cantidades de datos, sobre todo si se requiere que el tiempo de ejecución sea bajo. Nuestra propuesta, que llamamos CGLAD, resuelve en gran medida estos problemas mediante *clustering* a partir de vectores procedentes de factorización de matrices, lo que permite reducir el número de parámetros del modelo y, en general, facilitar el aprendizaje de modelos que siguen la estrategia de GLAD.**

***Index Terms*—aprendizaje automático no estándar, crowdsourcing, múltiples anotadores, débilmente supervisado**

I. Introducción

El algoritmo GLAD [5] permite abordar problemas de aprendizaje automático a partir de múltiples anotadores (por ejemplo, los que proceden de plataformas de *crowdsourcing*) [7]. Este problema se enmarca dentro del aprendizaje automático no estándar [2], el cual aborda el análisis de conjuntos de datos que difieren en ciertas características del aprendizaje de aprendizaje automático tradicional. Concretamente, en el aprendizaje a partir de múltiples anotadores no disponemos de la etiqueta verdadera de los ejemplos, si no que disponemos de varias anotaciones para cada ejemplo procedentes de anotadores de calidad desconocida, por lo que se obtiene un conjunto de datos como el de la Tabla I.

$\mathbf{Y_1}$	$\mathbf{Y_2}$	...	$\mathbf{Y_N}$
0	0	...	1
1	1	...	-
0	-	...	0
1	0	...	1
1	-	...	1
...	...	...	...

Tabla I: Ejemplo de conjunto de anotaciones

Este trabajo ha sido parcialmente financiado por la Agencia Estatal de Investigación (AEI) y el Fondo Europeo de Desarrollo Regional (FEDER, UE) mediante los proyectos TIN2016-77902-C3-1-P y TIN2016-82013-REDT. Enrique G. Rodrigo también ha sido financiado por el MECD mediante la beca FPU15/02281.

Aparte de estas anotaciones es posible obtener más características de las instancias, existiendo algoritmos [4] que permiten hacer uso de estas características a medida que se aprende un modelo. Sin embargo, gran parte de los algoritmos en esta línea trabaja en resolver únicamente el problema de agregación de estas etiquetas [8]. El objetivo último es poder aprender un modelo a partir de estas anotaciones que permita predecir la clase verdadera para cada instancia, que entonces podrá utilizarse en un algoritmo de aprendizaje automático tradicional. El enfoque más sencillo y empleado es el de usar la clase más frecuente como entrada de un algoritmo de aprendizaje automático, método comúnmente conocido como *Majority Voting* cuando tenemos una variable de salida de tipo discreto (en el caso continuo usaríamos la media). Sin embargo, en la actualidad, existen algoritmos más efectivos a la hora de agregar opiniones puesto que permiten tener en cuenta información tal como la experiencia de los anotadores [1] o, incluso, la dificultad de cada ejemplo [5]. Esta información no solo es útil para estimar la verdadera etiqueta, si no que puede resultar interesante en numerosos problemas.

Este tipo de algoritmos es especialmente interesante cuando el tamaño del problema es mayor, ya que es en este caso cuando es más complejo obtener un gran número de anotaciones fiables a partir de expertos (o, en general, de personas en las que podamos confiar) en un determinado problema. Es, por tanto, recomendable, que los algoritmos utilizados en la agregación de etiquetas puedan ser escalables a una gran cantidad de datos, de forma que puedan aplicarse también en estos casos.

En este artículo exponemos algunos problemas relacionados con la escalabilidad del algoritmo GLAD, uno de los algoritmos principales en la agregación de anotaciones cuando uno está interesado no solo en evaluar la calidad de los anotadores sino también en la dificultad de los ejemplos. Por otro lado, proponemos una mejora de este algoritmo, CGLAD, que permite abordar problemas de mayor envergadura y facilita en gran medida el uso de este en todos los contextos, añadiendo una mayor estabilidad frente a ligeros cambios en la configuración del algoritmo. Por último, ofrecemos una serie de comparativas con ambos métodos y exponemos nuestras conclusiones al respecto.

II. El algoritmo GLAD

El algoritmo GLAD [5] permite resolver problemas de aprendizaje con clase binaria usando múltiples anotaciones. A diferencia de otros algoritmos [1], [3], [4], permite estimar tanto la precisión de cada anotador como la dificultad de cada ejemplo. A continuación describiremos en qué consiste este algoritmo, y expondremos sus problemas de escalabilidad, los cuales pretende resolver nuestra propuesta.

II-A. Modelo

El modelo supone que la anotacion depende de tres elementos: la dificultad del ejemplo a anotar, la experiencia del anotador y la verdadera etiqueta. Los dos primeros elementos se modelan de la siguiente forma:

- **Dificultad de cada ejemplo**. Se modela usando un parámetro, $1/\beta_i \in [0, \infty)$, para cada ejemplo i, donde β_i es positivo. Si $1/\beta_i$ se acerca a 0, el ejemplo será más sencillo (anotadores con menos experiencia son capaces de anotarlo correctamente). Al contrario, si se acerca a ∞, el ejemplo sería tan ambiguo que incluso un anotador experimentado tendría solo un 50 % de probabilidad de etiquetar el ejemplo incorrectamente.
- **Experiencia del anotador**. Se modela usando un parámetro, $\alpha_j \in (-\infty, \infty)$, para cada anotador j. Si α_j se acercase a ∞, el anotador tendría tanta experiencia que siempre anotaría correctamente. Si α_j se acercase a $-\infty$ el anotador siempre anotaría incorrectamente, lo que significaría que el anotador es tan bueno como el anterior distinguiendo las clases, pero está invirtiendo la etiqueta, maliciosamente o por un malentendido. Por último, si α_j se acerca a 0, significa que el anotador no puede discriminar entre las clases (podría ser un *spammer*).

El modelo de anotación (generativo), se vale de los dos parámetros anteriores de forma que la probabilidad de que un anotador etiquete una instancia correctamente es

$$c_{ji} = p(y_i^j = y_i | \alpha_j, \beta_i) = \frac{1}{1 + e^{-\alpha_j \beta_i}}$$

donde y_i^j es la anotación del anotador j a la instancia i e y_i es la etiqueta verdadera de la instancia. De esta forma, los anotadores más experimentados (con alto valor de α_j) tienen una mayor probabilidad de etiquetar correctamente. Asimismo, si la dificultad de un ejemplo, $1/\beta_i$, es mayor, la probabilidad de que la etiqueta sea correcta se acerca a 0,5. Esto ocurre también si el valor de α_j se acerca a 0.

II-B. Inferencia

Las variables observadas son las etiquetas generadas por los anotadores. La variables ocultas son las verdaderas etiquetas y los parámetros del modelo de anotador $\boldsymbol{\alpha}$ y dificultad de las instancias $\boldsymbol{\beta}$. Para estimar las variables ocultas se puede utilizar el enfoque *Expectation-Maximization* para obtener estimaciones por máxima verosimilitud:

- **Paso E**: Sea $\mathcal{Y}_i = \{y_i^j\}$ el conjunto de anotaciones para el ejemplo i (no todos los anotadores tienen que anotar una instancia). Para obtener la probabilidad sobre la verdadera etiqueta, y_i, conocidos los valores de $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ y las anotaciones $\mathcal{Y}_i$ podemos utilizar:

$$p(y_i = k | \mathcal{Y}_i, \boldsymbol{\alpha}, \boldsymbol{\beta}) \propto p(y_i = k) \prod_j p_{ji}^k$$

donde

$$\begin{aligned} p_{ji}^k &= p(y_i^j = k | y_i, \alpha_j, \beta_i) \\ &= \begin{cases} (c_{ji})^k \cdot (1 - c_{ji})^{1-k} & \text{si } y_i = 1 \\ (c_{ji})^{1-k} \cdot (1 - c_{ji})^k & \text{si } y_i = 0 \end{cases} \end{aligned}$$

- **Paso M**: Se maximiza la siguiente función usando gradiente descendiente con respecto a los valores de $\boldsymbol{\alpha}$ y $\boldsymbol{\beta}$

$$Q(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \sum_i E[ln(p(y_i = k))] + \sum_{ij} E[ln(p_{ji}^k)],$$

donde E es la esperanza con respecto a las estimaciones procedentes del anterior paso E.
A partir de esta se obtienen los siguientes gradientes para cada parámetro

$$\frac{\partial Q}{\partial \alpha_j} = \sum_i (p_{ji}^1 \, y_i^j + p_{ji}^0 \, (1 - y_i^j) - \sigma(\alpha_j \beta_i)) \beta_i,$$

$$\frac{\partial Q}{\partial \beta_i} = \sum_j (p_{ji}^1 \, y_i^j + p_{ji}^0 \, (1 - y_i^j) - \sigma(\alpha_j \beta_i)) \alpha_j,$$

donde $\sigma(x) = 1/(1 + e^{-x})$.

II-C. Pseudocódigo

A partir del modelado anterior podemos resumir el algoritmo mediante el pseudocódigo en el Algoritmo 1.

Algoritmo 1 Algoritmo GLAD sin optimización

1: **Paso E inicial**: Agregación por mayoría $\rightarrow y_i$
2: **for** $i = 0$ hasta converger **do**
3: **Paso M**: Optimizar mediante gradiente descendiente usando las siguientes fórmulas:

$$\frac{\partial Q}{\partial \alpha_j} = \sum_i (p_{ji}^1 \, y_i^j + p_{ji}^0 \, (1 - y_i^j) - \sigma(\alpha_j \beta_i)) \beta_i,$$

$$\frac{\partial Q}{\partial \beta_i} = \sum_j (p_{ji}^1 \, y_i^j + p_{ji}^0 \, (1 - y_i^j) - \sigma(\alpha_j \beta_i)) \alpha_j,$$

4: **Paso E**: Estimar la probabilidad de la clase para cada instancia.

$$p(y_i = k | \mathcal{Y}_i, \boldsymbol{\alpha}, \boldsymbol{\beta}) \propto p(y_i = k) \prod_j p(y_i^j = k | y_i, \alpha_j, \beta_i)$$

5: **end for**

El algoritmo comienza realizando una inicialización de las verdaderas etiquetas a la clase más frecuente presente en las anotaciones para cada ejemplo. Tras ello realiza varias iteraciones del algoritmo EM, estimando los parámetros del modelo en el paso M y volviendo a estimar la verdadera etiqueta en el paso E. Este algoritmo se detiene cuando se logra la convergencia de la función de verosimilitud o cuando se llega a un número máximo de iteraciones.

II-D. Problemas de escalabilidad

Al intentar utilizar este algoritmo en problemas con un tamaño considerable descubrimos que el modelo pierde precisión en comparación con problemas similares de menor envergadura. Para comprobarlo, hemos realizado diferentes pruebas con el algoritmo variando el tamaño de los conjuntos de datos así como la tasa de aprendizaje del algoritmo de gradiente descendiente. Esta tasa gobierna en gran medida la capacidad del algoritmo para converger hacia una solución, en especial si mantenemos constantes el umbral y el número máximo de iteraciones del algoritmo de gradiente descendiente. Para las pruebas se han utilizado conjuntos de datos simulados con las siguientes características:

- **Tamaño del conjunto**. Se ha generado una serie de conjuntos de datos con los siguientes números de instancias[1]: 5000, 10000, 20000, 40000, 80000, 160000, 320000, 640000, 1280000, 2560000.
- **Anotaciones**. Para cada uno de los conjuntos anteriores se generan 10 anotaciones realizadas por anotadores simulados usando una distribución de probabilidad discreta. Usamos las distribuciones de la Figura 1, generando 6 anotadores con una precisión alta, 2 anotadores aleatorios y 2 anotadores adversarios.

Clase	Negativa	Positiva
Negativa	0.8	0.2
Positiva	0.1	0.9

(a) Precisión alta

Clase	Negativa	Positiva
Negativa	0.5	0.5
Positiva	0.5	0.5

(b) Aleatorio

Clase	Negativa	Positiva
Negativa	0.2	0.8
Positiva	0.8	0.2

(c) Adversario

Figura 1: Tipos de anotadores generados

Para cada conjunto de datos recogemos la precisión obtenida (al tener la información de la clase verdadera podemos evaluar directamente sobre ella) para cada conjunto de datos y 3 valores de la tasa de aprendizaje: 0.1, 0.01, 0.001. Resumimos los resultados en la Figura 2. Podemos observar que al aumentar el tamaño del conjunto de datos, la precisión del método parece disminuir. Variando la tasa de aprendizaje podemos conseguir resolver algunos casos más (aumentando el tiempo de aprendizaje).

Creemos que la razón fundamental de este problema es el aumento de parámetros que se estiman. En cuanto al modelo de anotador no hay problema, ya que en los conjuntos no varía el número de anotadores. Sin embargo, al aumentar el número de instancias del problema, aumenta de forma lineal el número de parámetros del modelo de dificultad (uno por instancia). Para tamaños de problema pequeños, como puede verse en la Figura 2, el algoritmo de gradiente descendiente converge, pero a medida que aumentamos el número de parámetros la convergencia es más compleja. Parece necesario, por tanto, reducir el número de parámetros del modelo de dificultad para conseguir un aprendizaje más estable y, sobre todo, para poder utilizar el algoritmo en contextos de grandes volúmenes de datos.

[1]Los datasets están disponibles en `.csv` y `.parquet` en el siguiente enlace: http://bit.ly/caepia2018-cglad

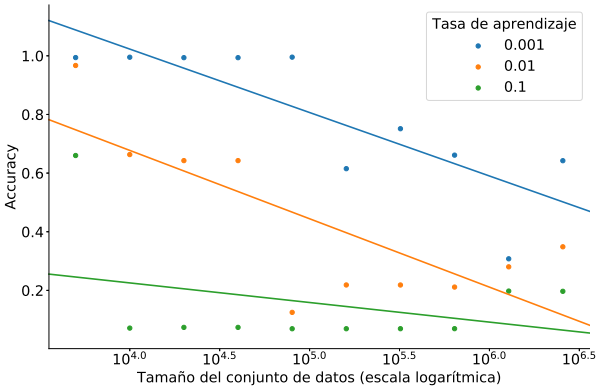


Figura 2: Problemas de escalabilidad del algoritmo GLAD.

III. CGLAD

En este artículo proponemos una mejora al algoritmo GLAD que permite reducir el número de parámetros necesarios para aprender el modelo. Para ello, añadimos un paso previo al algoritmo EM que reduce los parámetros mediante *clustering*.

III-A. Modelo

El modelo usa la misma estructura que GLAD, pero calculando dificultades de *clusters* de instancias en vez de instancias concretas.

- **Dificultad de cada *cluster***. Se modela usando un parámetro, $1/\beta_t \in [0, \infty)$, para cada *cluster* k, donde β_t es positivo. Si $1/\beta_t$ se acerca a 0, los ejemplos que pertenecen al *cluster* t serán más sencillos, mientras que serán más ambiguos si se acerca a ∞.
- **Experiencia del anotador**. Se modela usando un parámetro, $\alpha_j \in (-\infty, \infty)$, para cada anotador j. El significado de este parámetro es el mismo que en el caso de GLAD.

El modelo de anotación es equivalente al caso de GLAD, pero utiliza el parámetro de dificultad del *cluster*. Si denotamos como $\phi(i)$ la aplicación que mapea cada ejemplo i a uno de los *clusters*, el modelo de anotación sería el siguiente:

$$c_{ji} = p(y_i^j = y_i | \alpha_j, \beta_{\phi(i)}) = \frac{1}{1 + e^{-\alpha_j \beta_{\phi(i)}}}$$

donde y_i^j es la anotación del anotador j a la instancia i e y_i es la etiqueta verdadera de la instancia. Salvo por el uso del

cluster de dificultades, la interpretación de esta expresión es similar al caso de GLAD.

III-B. Inferencia

Usamos el enfoque EM para aprender los parámetros del modelo (conocidos los *cluster* en los que se dividen las instancias). La interpretación de estos es similar al caso del algoritmo tradicional.

- **Paso E**: Sea $\mathcal{Y}_i = \{y_i^j\}$ el conjunto de anotaciones para el ejemplo i. Para obtener la probabilidad sobre la verdadera etiqueta, y_i, conocidos los valores de $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ y las anotaciones $\mathcal{Y}_i$ podemos utilizar:

$$p(y_i = k|\mathcal{Y}_i, \alpha_j, \beta_{\phi(i)}) \propto p(y_i = k) \prod_j p_{ji}^k$$

donde

$$\begin{aligned} p_{ji}^k &= p(y_i^j = k|y_i, \alpha_j, \beta_{\phi(i)}) \\ &= \begin{cases} (c_{ji})^k \cdot (1 - c_{ji})^{1-k} & \text{si } y_i = 1 \\ (c_{ji})^{1-k} \cdot (1 - c_{ji})^k & \text{si } y_i = 0 \end{cases} \end{aligned}$$

- **Paso M**: Se maximiza la siguiente función usando gradiente descendiente con respecto a los valores de $\boldsymbol{\alpha}$ y $\boldsymbol{\beta}$

$$Q(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \sum_i E[ln(p(y_i = k))] + \sum_{ij} E[ln(p_{ji}^k)].$$

A partir de esta expresión se obtienen los siguientes gradientes para cada parámetro

$$\frac{\partial Q}{\partial \alpha_j} = \sum_i (p_{ji}^1 \, y_i^j + p_{ji}^0 \, (1 - y_i^j) - \sigma(\alpha_j \beta_{\phi(i)})) \beta_{\phi(i)},$$

$$\frac{\partial Q}{\partial \beta_t} = \sum_i \delta(\phi(i), t) \sum_j (p_{ji}^1 \, y_i^j + p_{ji}^0 \, (1 - y_i^j) - \sigma(\alpha_j \beta_t)) \alpha_j,$$

donde $\delta(\phi(i), t) = 1$ si $\phi(i) = t$ y 0 en otro caso.

III-C. Inicialización

A diferencia del algoritmo GLAD, en la inicialización, aparte de obtener una estimación de las etiquetas verdaderas usando la clase de mayor frecuencia, debemos obtener los *clusters* para cada instancia $\phi(i)$. Para ello utilizamos un enfoque basado en factorización de matrices y K-Means.

- **Factorización de la matriz de anotacion**. Al igual que en problemas donde se pueden aplicar técnicas de filtrado colaborativo, podemos visualizar la matriz de anotaciones como una matriz donde las filas representan a los anotadores y las columnas a los ejemplos. Podemos obtener dos matrices que nos permitan estimar esta usando factorización de matrices. Esto nos proporciona vectores de tamaño R (tamaño que debe elegir el usuario del método) que describen tanto a los anotadores como a los ejemplos. Para la factorización en esta propuesta usamos ALS [9], obteniendo dos matrices, A y D, que representan a los anotadores y a las instancias respectivamente.
- **Clustering**. Una vez obtenidos los vectores, podemos aplicar cualquier algoritmo de *clustering* sobre estos para obtener agrupaciones sobre las instancias. Estas agrupaciones estarán basadas en las diferencias en los vectores encontrados en el apartado anterior, originados por patrones de anotación diferentes. Para este paso, en esta propuesta utilizamos K-Means, aunque se podrían utilizar otros métodos [6]. Como nuestro objetivo no es interpretar los *clusters*, si no simplemente utilizarlos para aliviar la complejidad de la inferencia, podemos utilizar un número de *clusters* K alto (su elección óptima dependerá del problema a resolver, en nuestro caso, para los experimentos, lo hemos fijado en 32).

Una vez completados los pasos anteriores, obtendríamos la función ϕ, con la que ya tendríamos todos los componentes para implementar el algoritmo.

III-D. Pseudocódigo

El pseudocódigo el algoritmo CGLAD se muestra en el Algoritmo 2

Algoritmo 2 Algoritmo CGLAD

1: **Inicialización**:
2: $(A, D) = ALS(X)$
3: $\phi(i) = KMeans(D)$
4: **Paso E inicial**: Agregación por mayoría $\rightarrow y_i$
5: **for** $i = 0$ hasta converger **do**
6: **Paso M**: Optimizar mediante gradiente descendiente:

$$\frac{\partial Q}{\partial \alpha_j} = \sum_i (p_{ji}^1 \, y_i^j + p_{ji}^0 \, (1 - y_i^j) - \sigma(\alpha_j \beta_{\phi(i)})) \beta_{\phi(i)},$$

$$\frac{\partial Q}{\partial \beta_t} = \sum_i \delta(\phi(i), t) \sum_j (p_{ji}^1 \, y_i^j + p_{ji}^0 \, (1 - y_i^j) - \sigma(\alpha_j \beta_t)) \alpha_j,$$

7: **Paso E**:

$$p(y_i = k|\mathcal{Y}_i, \alpha_j, \beta_i) \propto p(y_i = k) \prod_j p_{ji}^k)$$

8: **end for**

El algoritmo comienza aprendiendo los *clusters* dado el conjunto de anotaciones, pasando posteriormente al algoritmo EM para estimar los parámetros del modelo. Al igual que en GLAD, iteramos hasta que se produzca la convergencia o hasta realizar un máximo de iteraciones.

III-E. Escalabilidad

Aplicamos este algoritmo a los mismos conjuntos de datos que utilizamos para analizar los problemas de escalabilidad del algoritmo GLAD. También usamos la misma configuración de parámetros para el algoritmo de gradiente descendiente y el algoritmo EM. Obtenemos los resultados que mostramos en la Figura 3. Como se puede ver, obtenemos un modelo más estable con respecto al tamaño de los conjuntos de datos. A diferencia del modelo original, obtenemos resultados aceptables para todos los conjuntos de datos y comparables a los resultados que obtiene GLAD en los problemas pequeños. Asimismo, se logra un algoritmo más estable con respecto a la elección de la tasa de aprendizaje.

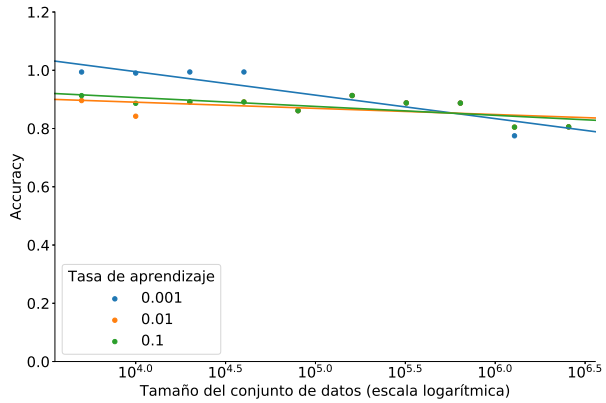


Figura 3: Escalabilidad en calidad del modelo en al propuesta CGLAD.

	Precision		F-score	
	CGLAD	GLAD	CGLAD	GLAD
Instancias				
5000	**0.9940**	**0.9940**	**0.9941**	**0.9941**
10000	0.9904	**0.9952**	0.9903	**0.9952**
20000	**0.9941**	0.9937	**0.9942**	0.9938
40000	**0.9941**	0.9937	**0.9942**	0.9938
80000	**0.8731**	0.8521	**0.8796**	0.8520
160000	**0.9132**	0.6151	**0.9191**	0.6153
320000	**0.8879**	0.7513	**0.8939**	0.7516
640000	**0.8873**	0.6613	**0.8974**	0.6566
1280000	**0.7750**	0.3077	**0.7500**	0.3086
2560000	**0.8056**	0.6423	**0.7844**	0.6418

Tabla II: Resultados GLAD y CGLAD

	CGLAD (Speedup)	GLAD
Instancias		
5000	2797.0s (1.14)	3208.0s
10000	1862.0s (2.03)	3781.0s
20000	719.0s (**5.52**)	3973.0s
40000	723.0s (**5.43**)	3925.0s

Tabla III: Resultados en tiempo GLAD y CGLAD

IV. Comparativa

Hemos realizado varias comparativas tanto entre GLAD y nuestra propuesta, CGLAD, como entre nuestra propuesta y otros algoritmos presentes en la literatura.

IV-A. CGLAD contra GLAD

En esta sección comparamos tanto GLAD y CGLAD entre sí en términos de precisión, así como en tiempo de cómputo. Para esta comparativa utilizamos los mismos conjuntos de datos que se usaron para exponer los resultados de escalabilidad en las secciones anteriores. En la Tabla II se muestran los resultados obtenidos por ambos algoritmos en estos conjuntos. Como se puede ver en la tabla, nuestra propuesta obtiene mejores resultados en la mayor parte de los conjuntos de datos, especialmente cuando los tamaños de los conjuntos de datos son mayores. Para obtener estos resultados usamos la misma configuración para ambos algoritmos, con la salvedad de que, en el caso de CGLAD, tenemos que definir el tamaño de los vectores en la factorización, 8, y el número de *clusters*, que en nuestro caso es 32. El resto de parámetros, comunes, son los siguientes:

- Maximo número de iteraciones EM: 5.
- Threshold EM: 0.1
- Máximo número de iteraciones de gradiente descendiente: 100
- Threshold gradiente descendiente: 0.1
- Tasa de aprendizaje de gradiente descendiente: 0.001

También es interesante analizar el tiempo de ejecución para los cuatro primeros casos, que presentan una precisión similar. Esto se puede ver en la Tabla III.

Como se puede ver, a pesar de que nuestra propuesta presenta una precisión similar en los casos más pequeños, esta consigue un tiempo de ejecución mucho menor. Cabe decir que usamos los primeros casos, y no los posteriores, ya que al usar CGLAD el proceso de gradiente descendiente para optimizar parámetros, para casos más grandes el algoritmo GLAD no converge, lo que lleva a que el proceso pare tras pocas iteraciones. Para estos casos, nuestra propuesta (que sí optimiza los parámetros iterando) y GLAD no se pueden comparar.

IV-B. CGLAD contra otros algoritmos del área

Existen otros algoritmos en el área del aprendizaje a partir de múltiples anotadores que podrían ser interesantes para realizar una comparativa. Sin embargo, hay que tener en cuenta que estos no presentan información de la dificultad de los ejemplos como proporcionan GLAD y CGLAD. Específicamente, vamos a comparar nuestra propuesta con MajorityVoting, el algoritmo más sencillo, que únicamente consiste en utilizar la clase más frecuente y DawidSkene [1], que utiliza como modelo una matriz de confusión para cada anotador y el algoritmo EM para realizar la estimación de esta matriz y la clase verdadera. Para estas pruebas hemos generado otros conjuntos de datos con anotadores simulados, cuyas anotaciones no solo dependen de la clase verdadera de una instancia, si no también de la dificultad de cada instancia (generada aleatoriamente). Se han generado conjuntos de los siguientes tamaños: 5000, 10000, 20000, 40000 y 80000. Para la ejecución se utilizan los siguientes parámetros:

- Maximo número de iteraciones EM: 5.
- Threshold EM: 0.1
- Máximo número de iteraciones de gradiente descendiente: 100
- Threshold gradiente descendiente: 0.1
- Tasa de aprendizaje de gradiente descendiente: 0.0003
- Tamaño de los vectores para la factorización: 8
- Tamaño de los vectores para la factorización: 32

Podemos ver los resultados en la Tabla IV. Aunque CGLAD obtiene resultados comparables e incluso mejores en varios problemas, podemos ver que no es mejor sistemáticamente que el algoritmo de DawidSkene, siendo este último mucho más rápido y sencillo. Sin embargo, CGLAD no solo nos permite modelar la precisión de los anotadores sino también la dificultad de los ejemplos, lo que puede ser interesante en

muchos problemas, incluso aunque no se obtenga una mejora en la precisión del modelo.

	Métodos		
Instancias	MajorityVoting	DawidSkene	CGLAD
5000	0.8102	0.8462	**0.8610**
10000	0.8130	0.8443	**0.8469**
20000	0.8141	**0.8478**	0.8286
40000	0.8161	**0.8494**	0.8465
80000	0.8161	**0.8494**	0.8465

Tabla IV: Resultados de CGLAD con respecto a otros algoritmos del estado del arte

V. Conclusiones y trabajo futuro

En este artículo proponemos una optimización al algoritmo GLAD para el aprendizaje a partir de múltiples anotadores donde mejoramos su escalabilidad y estabilidad a la hora de resolver problemas grandes, así como su tiempo de ejecución y, en gran parte de problemas, sus resultados. Para comprobarlo se han realizado varios experimentos que permiten comparar ambos modelos tanto en problemas con conjuntos de datos relativamente pequeños como en conjuntos de datos grandes, donde GLAD no obtiene buenos resultados. Como se ha visto en la experimentación, para este tipo de problemas nuestra propuesta funciona mejor y es más robusta ante cambios en la tasa de aprendizaje. También se ha llevado a cabo una comparativa entre nuestra propuesta y otros algoritmos del área. Se ha observado que, aunque nuestra propuesta obtiene resultados comparables a otros algoritmos del estado del arte en varios problemas, existen otros algoritmos que por su simplicidad habría que considerar a la hora de resolver un problema a partir de múltiples anotadores si el único objetivo es realizar la estimación de la verdadera etiqueta. En contrapartida, si, aparte de la precisión en la estimación de la verdadera clase, es de interés estimar la dificultad de las instancias (porque es necesaria esta información o porque en el problema que afrontamos tiene importancia la dificultad de las instancias a la hora de estimar) nuestra propuesta es una alternativa más que adecuada al algoritmo GLAD, sobre todo si queremos abordar problemas con un número de instancias relativamente elevado.

Esta mejora, así como el análisis de los problemas de este algoritmo, abren alternativas a futuras líneas de trabajo. Por un lado, en este artículo solo utilizamos algoritmos sencillos tanto para la factorización como para el *clustering*. Existen multitud de técnicas de *clustering* más potentes que podrían ser de interés para mejorar los resultados del algoritmo. Asimismo, también podría ser de interés aplicar la idea de la dificultad de los ejemplos a otros algoritmos existentes en el área pero con mejores características que GLAD en lo que respecta a la escalabilidad así como a la capacidad de abordar problemas de clase discreta o, incluso, contínua. Por último, destacar que esta propuesta solo utiliza las anotaciones para realizar el *clustering* y, posteriormente, la estimación. En problemas donde las características de cada instancia estén disponibles, podría ser interesante utilizarlas, junto con las anotaciones, para elaborar *clusters* más informados, lo que a nuestro jucio podría influir positivamente en la estimación final de la clase verdadera.

Agradecimientos

Este trabajo ha sido parcialmente financiado por la Agencia Estatal de Investigación (AEI) y el Fondo Europeo de Desarrollo Regional (FEDER, UE) mediante los proyectos TIN2016-77902-C3-1-P y TIN2016-82013-REDT. Enrique G. Rodrigo también ha sido financiado por el MECD mediante la beca FPU15/02281.

Referencias

[1] Alexander Philip Dawid and Allan M Skene. Maximum likelihood estimation of observer error-rates using the em algorithm. *Applied statistics*, 2:20–28, 1979.

[2] Jerónimo Hernández-González, Inaki Inza, and Jose A Lozano. Weak supervision and other non-standard classification problems: a taxonomy. *Pattern Recognition Letters*, 69:49–55, 2016.

[3] Qi Li, Yaliang Li, Jing Gao, Bo Zhao, Wei Fan, and Jiawei Han. Resolving conflicts in heterogeneous data by truth discovery and source reliability estimation. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1187–1198. ACM, 2014.

[4] Vikas C Raykar, Shipeng Yu, Linda H Zhao, Gerardo Hermosillo Valadez, Charles Florin, Luca Bogoni, and Linda Moy. Learning from crowds. *Journal of Machine Learning Research*, 11(Apr):1297–1322, 2010.

[5] Jacob Whitehill, Ting-fan Wu, Jacob Bergsma, Javier R Movellan, and Paul L Ruvolo. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *Advances in neural information processing systems*, pages 2035–2043, 2009.

[6] Rui Xu and Donald Wunsch. Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3):645–678, 2005.

[7] Jing Zhang, Xindong Wu, and Victor S Sheng. Learning from crowdsourced labeled data: a survey. *Artificial Intelligence Review*, 46(4):543–576, 2016.

[8] Yudian Zheng, Guoliang Li, Yuanbing Li, Caihua Shan, and Reynold Cheng. Truth inference in crowdsourcing: is the problem solved? *Proceedings of the VLDB Endowment*, 10(5):541–552, 2017.

[9] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the netflix prize. In *International Conference on Algorithmic Applications in Management*, pages 337–348. Springer, 2008.

Análisis preliminar de marcos tecnológicos en data stream

Fernando Puentes, María Dolores Pérez-Godoy, Pedro González, María José Del Jesus
Departamento de Informática
Universidad de Jaén
{fpuentes; lperez; pglez; mjjesus}@ujaen.es

***Abstract*—El análisis de datos en tiempo real está adquiriendo cada vez más importancia para ayudar en la toma de decisiones. Existen herramientas de código abierto y propietarias para el análisis de flujos continuos de datos. En este trabajo se analizan herramientas tanto para la recolección de datos de diversas fuentes como para el procesamiento de los mismos, presentando sus características más importantes con el objetivo de ayudar en la toma de decisión respecto al marco tecnológico en el desarrollo de métodos de minería de datos para *data stream*.**

Keywords*— *Streaming; Big Data; Minería de datos; Procesamiento de data stream; Recolección de datos;

I. Introducción

La minería de datos [1] se ha centrado tradicionalmente en analizar conjuntos de datos almacenados y estáticos en ambientes de procesamiento por lotes (modo *batch*). Sin embargo, cada día existen más fuentes que generan enormes cantidades de datos de forma continua, muchas veces asociados a problemas de *big data*, siendo necesario su procesamiento conforme llegan al sistema sin tener que almacenarlos (modo *streaming* o de flujo de datos).

Cada fuente va a generar un flujo de datos (*stream*) que tiene las siguientes características [2]:

- Los datos fluyen continuamente. Siempre se están generando datos, normalmente a alta velocidad.
- La distribución de los datos puede cambiar con el tiempo (cambio de concepto [3]). Debemos detectar estos cambios para poder responder a ellos lo antes posible.
- La velocidad con la que se generan los datos puede variar con el tiempo, por lo que unas veces llegarán más datos y otras menos.
- El tamaño del *stream* es teóricamente infinito, por lo que no puede ser totalmente almacenado en memoria para su procesamiento.

En los últimos tiempos, han ido apareciendo herramientas para facilitar el tratamiento de los *streams* de manera distribuida, tanto de código abierto como propietarios. La Fig. 1 muestra el esquema general de las etapas necesarias para el análisis de un *stream*. El proceso puede dividirse en 3 etapas [4]: recolección y pre-procesamiento; procesamiento; y análisis y evaluación.

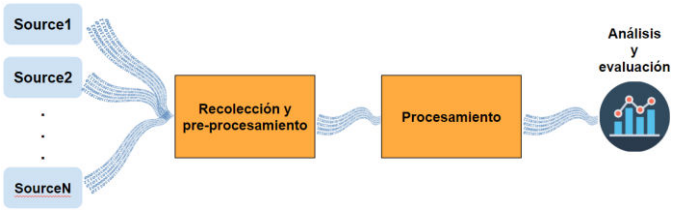


Fig. 1. Esquema general análisis en stream

En la primera etapa se utilizará una herramienta que sea capaz de realizar la recolección continua de los datos generados por una o varias fuentes. También debemos asegurarnos de que los datos estén en el formato correcto antes de pasar a la siguiente etapa, por lo que será necesario llevar a cabo un pre-procesamiento, utilizando técnicas de reducción de la dimensionalidad (reducción de características y/o de instancias) y simplificación del espacio de características [8].

Una vez que los datos han sido recolectados y pre-procesados y tienen el formato correcto, podemos pasar a la segunda etapa, el procesamiento. En esta etapa se utiliza un algoritmo de minería de datos para analizar el *stream* y llevar a cabo la extracción de conocimiento. Para mejorar el rendimiento se puede elegir una herramienta que permita realizar un procesamiento distribuido y que se adapte mejor a los requisitos y necesidades del problema a resolver.

La tercera y última etapa (cuya realización es opcional) se encarga de la visualización de los resultados obtenidos en la etapa anterior mediante alguna herramienta de visualización en tiempo real.

Aunque existen herramientas propietarias y de código abierto, en este trabajo nos centraremos fundamentalmente en las de código abierto, puesto que permiten poner el conocimiento obtenido al alcance de toda la comunidad investigadora y son cada vez más utilizadas [7]. En particular, en este trabajo se analizan herramientas que desarrollan las 2 primeras etapas, aportando información que facilite la elección de la herramienta a utilizar en función del objetivo a alcanzar.

El trabajo está organizado de la siguiente forma. En la sección 2, se detalla la etapa de recopilación y pre-procesamiento de datos y se revisan las herramientas más destacadas. En la sección 3, se introduce la etapa de procesamiento y se analizan las herramientas de esta categoría. En la sección 4, se reflejan las conclusiones obtenidas.

II. Etapa de recolección y pre-procesamiento

Antes de procesar los datos suele ser necesario recopilarlos, ya que es habitual que sean generados por diferentes fuentes simultáneas, y es necesario unificarlos para disponer de un único canal de entrada al procesamiento. Además hay que llevar a cabo un pre-procesamiento de los datos, que permita que la etapa de procesamiento reciba los datos de forma adecuada. El objetivo de esta sección es analizar y comparar diferentes herramientas para realizar estas tareas, destacando sus características principales.

A. *Apache Kafka*

Apache Kafka[1] es una plataforma de *streaming* distribuida que permite recopilar datos de diversas fuentes mediante una metodología de publicación/subscripción (el productor publica los datos y el consumidor se suscribe para recibirlos), similar a una cola de mensajes. Así, se deben definir en el clúster de Kafka un conjunto de *topics,* en los que el productor publica los mensajes, para que después un consumidor pueda recoger los datos que se van almacenando en los *topics* que se seleccionen. Estos datos se mantienen durante un tiempo (denominado periodo de retención), siendo eliminados cuando finaliza el mismo.

Los datos que se almacenan en los *topics* son pares clave-valor en formato texto, en los que la clave suele ser algún tipo de marca de tiempo (*timestamp*) y el valor el dato correspondiente. Para llevar a cabo un pre-procesamiento hay 2 opciones: hacer el pre-procesamiento en el productor o construir un consumidor-productor. La primera opción consistiría en un programa que transforme los datos antes de enviarlos y la segunda sería una aplicación que escuche un *topic*, capture todo lo que llegue a ese *topic* para transformarlo y envíe por último el resultado a otro *topic* diferente.

Las características ms importantes de Kafka se muestran en la TABLA I. Algunos puntos a destacar son:

- Tiene compatibilidad con todas las herramientas de procesamiento analizadas en este trabajo.
- Es necesario implementar un programa productor y otro consumidor, en alguno de los lenguajes soportados.
- Se dispone de una extensa documentación para desarrolladores y muchos problemas resueltos.
- No tiene una utilidad nativa para monitorizar, pero hay alguna aplicación de terceros que si lo permite, como Burrow de LinkedIn.
- Kafka es la plataforma más utilizada, ya que permite replicar y pre-procesar los datos. Además, su política de publicación/subscripción permite tener varios almacenamientos diferentes, por lo que al procesar los datos podemos elegir de qué *topic* obtenerlos, e incluso tener varias aplicaciones obteniendo datos del mismo sistema que ejecuta Kafka.

[1] https://kafka.apache.org

B. *Apache Flume*

Apache Flume[2] es un servicio distribuido que permite recolectar, agregar y mover eficientemente grandes cantidades de *logs*. Tiene una arquitectura simple y flexible basada en flujos de datos continuos. Se basa en un agente compuesto por 3 componentes: una fuente, un canal y un destino.

La fuente consume los datos de una fuente externa, como un servidor web. Esta fuente envía datos a Flume en un formato conocido por la fuente. Cuando la fuente recibe un evento, éste es almacenado en uno o más canales. Un canal es un almacén pasivo que almacena eventos hasta que son consumidos por el destino. Flume dispone de diferentes tipos de canales, dependiendo del tipo de almacenamiento que se utilice. Finalmente, el destino de los datos es el responsable de eliminar un evento del canal y ponerlo en el sistema de ficheros destino. La fuente y el destino se ejecutan asíncronamente con los eventos organizados en el canal. Hasta que un dato no se almacena en el sistema de ficheros destino, no se elimina del canal.

Las características generales más importantes de Flume se muestran en la TABLA I. Algunos puntos a destacar son:

- Se suele utilizar con Kafka (Flafka) como canal, de forma que ya no habría que implementar el productor y el receptor.
- Flume no permite replicar los datos, por lo que se suele utilizar para transportar grandes cantidades de datos de un sistema a otro.

C. *Apache Nifi*

Apache Nifi[3] es un software que permite migrar los datos de un sistema de ficheros a otro. Los datos pueden ser procesados mediante una serie de operaciones para transformarlos antes de ser enviados al destino. Cuenta con una interfaz de usuario para el navegador web, que facilita la creación de flujos y la configuración de cada una de las acciones. Nifi dispone de distintos componentes: *processor*, *reportingTask*, *controllerService*, *flowFilePrioritizer* y *authorityProvider*.

El *processor* es el componente más utilizado. Permite crear, eliminar, modificar o inspeccionar datos (denominados *flowFiles*), que incluyen un contenido y una serie de atributos que actúan como metadatos. Podemos hacer pasar los datos por diferentes *processors* para pre-procesar los datos y finalmente enviarlos al destino.

Las características ms importantes de Nifi se muestran en la TABLA I. Algunos puntos a destacar son:

- No necesita programar nada, ya que todo se configura en la interfaz.
- Permite realizar transformaciones sobre los datos, como conversión de datos/formatos, delegación de funcionalidades y operaciones de unión y difusión de los datos.

[2] https://flume.apache.org
[3] https://nifi.apache.org

- Se suele utilizar con Kafka, de forma que ya no habría que implementar el productor y el receptor.
- Permite monitorizar el estado del flujo en tiempo real, identificando posibles errores.
- Permite el envío de datos a múltiples destinos a la vez.
- Nifi no permite replicar los datos pero tiene una interfaz gráfica, por lo que se suele utilizar cuando se necesita mover datos de un sitio a otro y gestionar los *streams* de datos.

D. *Otras herramientas de recolección*

Aunque las herramientas de recolección descritas hasta ahora son de código abierto, existen otras herramientas propietarias, como Alooma o Attunity Replicate.

Alooma permite tener visibilidad y control sobre los datos, debido a su interfaz gráfica. Alooma recopila, en tiempo real, los datos de varias fuentes de datos y las unifica para aportarlas al sistema de procesamiento.

Attunity Replicate permite a las organizaciones acelerar la replicación de datos, la recolección y el *streaming* a través de diferentes bases de datos heterogéneas, almacenes de datos y plataformas de *big data*. Attunity Replicate mueve fácilmente los datos, de forma segura y eficiente.

TABLA I. Características de las herramientas de recolección

Características	**Herramienta**		
	Kafka	***Flume***	***Nifi***
Última versión	1.1.0	1.8.0	1.6.0
Fecha última modificación	28/03/2018	04/05/2018	08/04/2018
Pre-procesamiento	Si	Si	Si
Garantía de entrega	Al menos una	Al menos una	Al menos una
Tolerante a fallos	Si	Si	Si (solo con "*FileChannel*")
Lenguajes de programación	Java	Java	Java
Tiene una utilidad de monitorización	No[a]	No	Si
Plataformas	Windows Linux	Windows Linux	Windows Linux Mac OS
Documentación	Extensa	Extensa	Extensa
Replicación de datos	Si	No	No

[a] Ver información en el punto II.A

III. Etapa de Procesamiento

Una vez que se han recopilado y pre-procesado los datos, están listos para ser procesados con algún algoritmo de *streaming*. En esta sección vamos a analizar y comparar diferentes herramientas de procesamiento, destacando sus características principales. En [5] y [6] se pueden encontrar comparativas de rendimiento entre algunas de las herramientas más conocidas, como Spark, Storm y Flink.

A la hora de procesar los datos, hay que elegir de qué forma vamos a tratar los datos, utilizando alguno de los 3 tipos de garantías de procesamiento disponibles:

- Como máximo una vez: Los datos pueden ser procesados una vez o ninguna.
- Al menos una vez: Los datos pueden ser procesados 1 o más veces. Nunca tendremos datos sin procesar.
- Exactamente una vez: Los datos serán procesados una sola vez. No hay ni duplicados, ni datos no procesados.

A. *Apache Spark Streaming*

Apache Spark Streaming[4] es una extensión del núcleo (*core*) de Spark que permite procesamiento escalable, de alto rendimiento y tolerante a fallos de *streams* de datos. Los datos llegan a través de uno o varios *streams* de entrada que pueden ser procesados usando algoritmos complejos expresados con funciones a alto nivel como *map*, *reduce*, *join* y *window*. Finalmente, los datos procesados pueden ser transferidos a un sistema de ficheros o una base de datos.

Spark Streaming trabaja de la siguiente forma. Recibe un *stream* de entrada de datos y lo divide en *batches*, todos del mismo tiempo (todos los datos que lleguen en un determinado tiempo se tendrán en cuenta). Después, cada *batch* es procesado por el *Spark Engine* mediante algún algoritmo, generando a la salida un *batch* por cada *batch* de entrada.

Las principales características se muestran en la TABLA II. Algunos puntos a destacar son:

- Puede analizar los datos en modo *batch* o en modo *streaming* (una adaptación del modo *batch* en el que el *stream* se divide en *batches*).
- Permite la utilización de ventanas temporales. Éstas tienen un tamaño de ventana y un desplazamiento que debe ser múltiplo del intervalo de *batch*.
- La latencia es mayor que en otras herramientas, ya que no realiza un procesamiento continuo de los datos, sino que tiene que esperar siempre a que se cumpla el tiempo de un *batch*.
- Spark Streaming es útil en aquellos casos en los que necesitamos procesar datos por lotes en modo *batch* o en modo *streaming*. Además, la existencia de librerías de algoritmos, su extensa documentación y su activa comunidad pueden ser factores por los que elegir esta herramienta.

B. *Apache Spark Structured Streaming*

Apache Spark Structured Streaming[5] es un motor de procesamiento de *stream* escalable y tolerante a fallos construido sobre el motor de *Spark SQL*, lo que permite procesar un *stream* de manera similar a como se hace en el modo *batch* sobre datos estáticos. Structured Streaming permite hacer consultas que se resuelven incrementalmente,

[4] https://spark.apache.org
[5] https://spark.apache.org/docs/2.3.1/streaming-programming-guide.html

actualizando el resultado final, que se almacena en una tabla de salida.

En el modo *streaming*, hay una tabla ilimitada (*dataset*) a la que se van añadiendo los datos conforme llegan por el *stream*. Sobre esta tabla se realizan las consultas incrementales. Estas consultas procesan los datos en *micro-batches* de 100 milisegundos, aunque en la nueva versión 2.3 de Spark se ha introducido un modo continuo, denominado "Procesamiento continuo", que permite latencias por debajo de 1 milisegundo con garantías de procesamiento de "al menos una vez", pero es una característica experimental aún en desarrollo.

Las principales características se muestran en la TABLA II. Algunos puntos a destacar son:

- Permite procesamiento en modo *batch* y en modo *streaming*.
- En el momento de actualizar la salida hay 3 modos: *complete*, *append* y *update*. Dependiendo del tipo de consulta se podrá utilizar uno u otro.
- En otros modelos, el usuario tiene que mantener y actualizar las agregaciones ejecutadas. En este modelo, Spark es el que se encarga de mantener y actualizar las tablas con los nuevos datos.
- Permite utilizar ventanas temporales igual que Spark Streaming, con la diferencia de que aquí no hay que especificar el intervalo de *batch*.
- Junto con las ventanas temporales se puede utilizar lo que se conoce como *watermark*, que establece un límite para los datos que lleguen con retraso, basándose en un campo *timestamp*. Si un dato llega con retraso, se actualizan los valores de su instante temporal correspondiente y no en el que llega el dato.
- No dispone de ninguna librería con algoritmos de minería de datos, aunque si hay algunas implementaciones en GitHub.
- Spark Structured Streaming es útil en aquellos casos relacionados con consultas a bases de datos, gracias a sus consultas incrementales sobre los datos que llegan. Además, la utilización de *watermarks* con ventanas temporales pueden ser factores por los que elegir esta herramienta.

C. Apache Storm

Apache Storm[6] es un sistema de computación distribuida en tiempo real que procesa los datos de entrada dato a dato. Storm trabaja por topologías, de forma que el flujo va a ir pasando por una serie de nodos/operaciones hasta llegar al final de la topología. En una topología Storm existen dos tipos de nodos: Spout, que regulan la entrada de datos al sistema, y Bolt, que realizarán operaciones sobre los datos.

Storm tiene un modo de funcionamiento utilizando una abstracción superior denominada Trident, lo que permite que pueda procesar los datos por lotes (*batch*). Este modo ofrece garantías de procesamiento de "exactamente una vez", aunque aumenta la latencia al tener que esperar para recopilar un conjunto de datos.

La TABLA II muestra las principales características de Storm. Algunos puntos a destacar son:

- Solo permite procesamiento en modo *streaming*, dato a dato.
- No garantiza que los datos se procesen en orden.
- Permite la utilización de ventanas temporales y, junto a estas, se pueden utilizar *watermarks*.
- No dispone actualmente de ninguna librería con algoritmos de minería de datos, aunque hay en desarrollo una denominada SAMOA[7].
- Storm es útil en aquellos casos en los que se van a repartir las tareas por nodos según una topología y queremos procesar los datos dato a dato con garantías de procesamiento de "al menos una vez".

D. Apache Flink

Apache Flink[8] es un *framework* de procesamiento para aplicaciones de *streaming* de datos distribuidas que permite procesar los datos dato a dato o en *micro-batches*. También permite el procesamiento en modo *batch*, con un conjunto de datos estático. Funciona por topologías, al igual que Storm, en las que hay 3 tipos de nodos: operadores, fuentes y destinos. Los datos entrarán por los nodos fuente al sistema, serán procesados mediante diferentes nodos operadores y el resultado será depositado en un destino.

Flink es muy potente en cuanto al uso de ventanas, ya que permite una gran variedad. Las ventanas pueden estar basadas en número de eventos o en tiempo. Dentro de una ventana también se permite distinguir entre eventos de diferentes tipos. Hay diferentes tipos de ventana ya implementadas, pero se ofrece la posibilidad de implementar una ventana personalizada si fuese necesario.

Las características principales se muestran en la TABLA II. Algunos puntos a destacar son:

- Su mayor ventaja es la variedad de ventanas, que además permiten la utilización de *watermarks* para permitir un retardo en los datos de entrada.
- Permite procesar los datos en modo *streaming* y en modo *batch*. Además, se puede elegir entre procesar los datos de uno en uno o por lotes.
- Flink es útil en casos en los que queremos procesar los datos dato a dato o por lotes con garantías de procesamiento de "exactamente una vez", en modo *batch* o en modo *streaming*. Flink es muy similar a Storm, pero proporciona funcionalidades que en Storm habría que implementar. Además, la posibilidad de utilizar ventanas temporales junto con *watermarks* o de un tamaño concreto puede ser un factor importante por el que elegir esta herramienta.

[6] http://storm.apache.org

[7] https://samoa.incubator.apache.org

[8] https://flink.apache.org

E. Apache Samza

Apache Samza[9] es un *framework* de procesamiento distribuido de *stream* que utiliza Kafka y YARN en el proceso. Samza ofrece muchas funcionalidades similares a Storm y procesa *streams* de datos mediante tareas predefinidas, que realizan alguna operación en el *stream* de datos. Una aplicación Samza es un flujo de datos que consiste en consumidores que obtienen datos que son procesados por un grafo de trabajos, donde cada trabajo contiene una o más tareas. En Samza cada trabajo es una entidad que puede ser desplegada, iniciada o parada inmediatamente.

En Samza, cada tarea contiene un almacén clave-valor usado para almacenar el estado. Los cambios en este almacén son replicados a otras máquinas del clúster para permitir a las tareas ser recuperadas rápidamente en caso de fallo.

Las características de Samza se muestran en la TABLA II. Algunos puntos a destacar son:

- Garantiza que los mensajes son procesados en orden.
- No dispone de una librería con algoritmos de minería de datos, aunque en el futuro podrá utilizar algoritmos de SAMOA.
- Utiliza procesos *single-thread*.
- Samza es útil en casos en los que necesitamos procesar los datos dato a dato, en modo *batch* o modo *streaming*. Samza se ha realizado sobre Kafka, por lo que los datos se van a almacenar en particiones y se asegura que van a ser procesados en orden.

F. Apache Apex

Apache Apex[10] es una plataforma de procesamiento basada en YARN nativo de Hadoop. Apex proporciona un API simple, que permite a los usuarios escribir código genérico y reusable. El código se coloca como está y la plataforma automáticamente maneja las cuestiones operativas, como gestión de estados, tolerancia a fallos, escalabilidad, seguridad, métricas, etc. Esto permite que los usuarios se centren en el desarrollo.

El núcleo de la plataforma de Apex es complementado por Malhar, una librería de funciones lógicas y conectores. Proporciona acceso a diferentes sistemas de ficheros, sistemas de mensajes y bases de datos.

Las características principales se muestran en la TABLA II. Algunos puntos a destacar son:

- La topología de Apex es un grafo acíclico dirigido, cuyos nodos son Operators, y los arcos son los *streams*.
- Los datos se procesan dato a dato, aunque también soporta procesamiento por lotes de un determinado tiempo.
- No dispone de una librería con algoritmos de minería de datos, aunque en el futuro podrá utilizar algoritmos de SAMOA.
- Apex es útil en casos en los que necesitamos procesar los datos dato a dato con garantías de procesamiento de "exactamente una vez" y se quiere crear y utilizar código que sea reutilizable. Además tiene compatibilidad con la librería Malhar, que puede ser un factor importante para elegir esta herramienta.

G. Apache Beam

Apache Beam[11] es un *framework* de procesamiento de *streams* que permite definir un procesamiento independientemente del sistema de procesamiento de datos (denominado *runner*) que se utilice. Permite seguir utilizando el mismo procesamiento programado, permitiendo cambiar el *runner*. Permite utilizar los siguientes *runners*: Apache Apex, Apache Flink, Apache Spark, Apache Gearpump y Google Cloud Dataflow.

Sus características se muestran en la TABLA II. Algunos puntos a destacar son:

- La garantía de procesamiento depende del *runner* que se utilice para procesar los datos, ya que cada uno ofrece unas garantías propias de la herramienta.
- Se puede utilizar como lenguaje de programación Java y Python, aunque también hay una API de Scala en un proyecto de GitHub[12].
- La latencia varía según el *runner* que se utilice.
- Beam es útil en casos en los que se quiere probar un procesamiento con diferentes *runners* para ver cual se adapta mejor.

H. Otras herramientas de procesamiento

Aunque todas las herramientas de procesamiento analizadas aquí son de código abierto, existen otras herramientas propietarias, como Amazon Kinesis o Google Cloud Dataflow. Estas herramientas ofrecen soporte, mayor seguridad y herramientas avanzadas como entornos de desarrollo e interfaces orientadas al negocio [7].

Amazon Kinesis permite procesar y analizar datos a medida que se reciben y es capaz de ofrecer una respuesta instantánea en lugar de tener que esperar a la recopilación de datos. Una vez procesados los datos, Kinesis ofrece también herramientas para visualizar los resultados mediante gráficas o tablas.

Google Cloud Dataflow es un servicio de procesamiento de datos administrado capaz de ejecutar canalizaciones, tanto en modo *batch* como en modo *streaming*. Utiliza un modelo de programación unificado e incluye SDKs para definir flujos de procesamiento de datos.

Finalmente, hay una herramienta llamada Apache Gearpump, que aún está en desarrollo (*incubating*). Gearpump es un motor de procesamiento de *streaming* basado en eventos/mensajes que está inspirado en los avances recientes en el *framework* de Akka.

[9] http://samza.apache.org
[10] https://apex.apache.org
[11] https://beam.apache.org
[12] https://github.com/spotify/scio

IV. CONCLUSIONES

En este trabajo se han analizado diferentes herramientas tanto para recolectar datos de las fuentes que los generan como para procesarlos. Cada herramienta tiene ventajas e inconvenientes según la situación, por lo que sus características serán factores clave a la hora de seleccionar la herramienta más apropiada para nuestro objetivo.

En cuanto a recolección de datos, la herramienta más completa es apache Kafka, ya que tiene un gran rendimiento para esta tarea y permite pre-procesar los datos. También permite replicar los datos, lo que aumenta considerablemente la tolerancia a fallos. Además, Kafka puede recopilar datos para diferentes aplicaciones a la vez, ya que se pueden distinguir por un *topic* diferente. Por último, esta herramienta está siendo muy utilizada y dispone de gran cantidad de documentación y problemas resueltos para ayudar a los desarrolladores.

En cuanto a procesamiento, actualmente la mejor herramienta es Apache Spark Streaming por su gran rendimiento, a pesar de que tiene algo de latencia. Además, tiene garantías de procesamiento de "exactamente una vez" y tiene algunas librerías con algoritmos ya implementados, tanto en modo *batch* como en modo *streaming*, lo que puede ayudar a los desarrolladores a la hora de implementar nuevos algoritmos. También tiene compatibilidad con Kafka, por lo que se pueden utilizar ambos para las etapas correspondientes. Por último, esta herramienta también está siendo ampliamente utilizada y dispone de gran cantidad de documentación útil para los desarrolladores, evitando tener que invertir mucho tiempo para resolver problemas.

AGRADECIMIENTOS

Este trabajo ha sido subvencionado por el Ministerio de Economía y Competitividad bajo el proyecto TIN2015-68454-R, Fondos FEDER.

REFERENCIAS

[1] J. Han, M. Kamber y J. Pei. Data Mining: Concepts and Techniques, 3rd Edition. Morgan Kaufmann, 2011

[2] J. Gama. Knowledge Discovery from Data Streams. Chapman & Hall/CRC, 2010

[3] I. Khamassi, M. Sayed-Mouchaweh, M. Hammami, K. Ghédira. Discussion and review on evolving data streams and concept drift adapting. Evolving Systems 9:1–23, 2018

[4] Ms.D.Jayanthi, Dr.G.Sumathi. A Framework for Real-time Streaming Analytics using Machine Learning Approach. In: Proceedings of National Conference on Communication and Informatics-2016

[5] J. Samosir, M. Indrawan-Santiago, P. D. Haghighi. An evaluation of data stream processing systems for data driven applications. Procedia Computer Science 9:439-449, 2016

[6] Y. Wang. Stream Processing Systems Benchmark: StreamBench. Ph. D. dissertation. Aalto University, School of Science, 2016

[7] Gartner. Market Guide for Event Stream Processing (G00332885), 2018

[8] S. Ramírez-Gallegoa, B. Krawczyk, S. García, M. Wozniak, F. Herrera. A survey on data preprocessing for data stream mining: Current status and future directions. Neurocomputing 239:39-57, 2017

TABLA II. PRINCIPALES CARACTERÍSTICAS DE LAS HERRAMIENTAS DE PROCESAMIENTO

Características	**Herramientas**						
	Spark Streaming	***Spark Structured Streaming***	***Storm***	***Flink***	***Samza***	***Apex***	***Beam***
Última versión	2.3.1	2.3.1	1.2.2	1.5.0	0.14.1	3.7.0	2.4.0
Fecha última modificación	08/06/2018	08/06/2018	04/06/2018	25/05/2018	18/05/2018	27/04/2018	20/03/2018
Garantía de procesamiento	Exactamente una vez	Exactamente una vez	Al menos una vez / Exactamente una vez	Exactamente una vez	Al menos una vez	Exactamente una vez	Depende[a]
Tolerante a fallos	Si	Si	Si	Si	Si	Si	Si
Lenguajes de programación	Java Scala Python R	Java Scala Python R	Java Scala Python Ruby	Java Scala Python	Java	Java	Java Scala[a] Python
Utilidad de monitorización	Si	Si	Si	Si	Si	Si	Si
Plataformas	Windows Linux Mac OS	Windows Linux Mac OS	Windows Linux	Windows Linux Mac OS	Linux	Windows Linux Mac OS	Windows Linux Mac OS
Documentación	Muy extensa	Extensa	Extensa	Extensa	Poca	Poca	Extensa
Modificaciones durante la ejecución	No	No	Si	No	No	Si	No
Latencia	Segundos	Milisegundos	Milisegundos	Milisegundos	Milisegundos	Milisegundos	Depende[a]
Procesamiento modo batch	Si	Si	Si[a]	Si	Si	No	Si
Procesamiento modo streaming	Si	Si	Si	Si	Si	Si	Si
Librería Minería de Datos	MLLIB streamDM Amidst Toolbox	No[a]	SAMOA[a]	FlinkML SAMOA Amidst Toolbox	SAMOA[a]	SAMOA[a]	Si[a]

[a] Ver información en el punto correspondiente

Selección de características escalable con ReliefF mediante el uso de Hashing Sensible a la Localidad

Carlos Eiras-Franco
Grupo LIDIA. CITIC.
Universidade da Coruña
A Coruña, España
carlos.eiras.franco@udc.es

Bertha Guijarro-Berdiñas
Grupo LIDIA. CITIC.
Universidade da Coruña
A Coruña, España
cibertha@udc.es

Amparo Alonso-Betanzos
Grupo LIDIA. CITIC.
Universidade da Coruña
A Coruña, España
ciamparo@udc.es

Antonio Bahamonde
Universidad de Oviedo
Gijón, España
abahamonde@uniovi.es

***Resumen*—Los algoritmos de selección de características son de gran importancia para manejar conjuntos de datos de grandes dimensiones. Sin embargo, algoritmos efectivos y populares como ReliefF tienen una complejidad computacional que impide su uso en estos casos. En este trabajo proponemos una modificación de ReliefF que se basa en la aproximación del grafo de vecinos más cercanos usando una adaptación del algoritmo VRLSH, basado en Hashing Sensible a la Localidad. El algoritmo resultante, llamado ReliefF-LSH, es capaz de procesar conjuntos de datos masivos que están fuera del alcance del original. Detallamos experimentos que atestiguan la validez del nuevo enfoque y demuestran su buena escalabilidad.**

***Index Terms*—selección de características, escalabilidad, aprendizaje automático, Big Data**

I. Introducción

La ciencia de datos está alcanzando gran relevancia gracias, en parte, a la enorme cantidad de datos que se generan diariamente en todos los ámbitos que constituyen lo que coloquialmente se conoce como *Big Data* [1]. No obstante, este aumento en el volumen de datos constituye un reto para los científicos de datos dado que los obliga a desarrollar nuevos algoritmos y a adaptar los existentes para que sean capaces de tratar grandes cantidades de datos y obtener información relevante en un tiempo razonable.

Existen numerosas técnicas para tratar con conjuntos de datos de alta dimensionalidad, entendiendo por ello conjuntos con gran número de muestras o gran número de características para cada muestra. Cuando se dispone de muchas características para cada muestra es necesario aplicar técnicas de reducción de dimensionalidad. Entre las opciones del científico de datos están las técnicas de extracción de características, (que transforman un conjunto de características de entrada en un nuevo conjunto más pequeño en el cual cada característica es una función de varias características de entrada), o las técnicas de selección de características, que simplemente descartan las características redundantes o irrelevantes. A su vez, estas técnicas se pueden aplicar de manera explícita antes del entrenamiento como un paso de preprocesado de los datos, o se pueden realizar simultáneamente al aprendizaje.

Asimismo, para lidiar con el problema del volumen del *Big Data*, se han desarrollado plataformas de procesamiento distribuido que permiten utilizar hardware doméstico para formar clústeres de procesamiento que pueden analizar grandes cantidades de datos. La popularización de estas plataformas comenzó con el desarrollo del paradigma de programación MapReduce por Google en el año 2008 [2]. Desde entonces, han surgido varias implementaciones de código abierto que siguen ese paradigma, siendo Apache Hadoop [3] y, posteriormente, Apache Spark [4] las más populares. Su éxito dio lugar a la creación de librerías de aprendizaje automático como Mahout [5] para Hadoop y MLLib [6] para Spark y al desarrollo de versiones distribuidas de populares algoritmos, entre ellos algunos algoritmos de selección de características.

Además, en los casos en que la complejidad computacional del algoritmo no se puede reducir y el volumen de datos hace que incluso una implementación distribuida requiera un tiempo de ejecución muy alto para procesarlos, se puede recurrir a la utilización de técnicas que proporcionen aproximaciones a la solución exacta o más precisa (y también más costosa computacionalmente) pero que, dependiendo del problema, pueden obtener un rendimiento comparable.

En este trabajo hemos adaptado el popular algoritmo de selección de características ReliefF para reducir su complejidad computacional. Para ello, proponemos un cambio en el cálculo del grafo de vecinos más cercanos (construcción en la que se basa ReliefF) sustituyéndolo por un algoritmo de cálculo aproximado del grafo optimizado mediante el uso de Hashing Sensible a la Localidad. Esto, unido a la implementación del algoritmo resultante en Apache Spark que permite paralelizar gran parte de los cálculos, aumenta drásticamente su capacidad de procesar conjuntos grandes.

II. Trabajo relacionado

El algoritmo Relief es un método de ordenación (ranking) de características atendiendo a su relevancia de cara a la clasificación [7]. Es un método supervisado, dado que requiere conocer la clase de cada ejemplo, y da como resultado una ordenación por importancia de las características, que posteriormente se puede utilizar para realizar una selección de

Este trabajo ha sido parcialmente financiado por el Ministerio de Economía y Competitividad (proyectos de investigación TIN 2015-65069-C2, tanto 1-R como 2-R y Red Española de Big Data y Análisis de datos escalable, TIN2016-82013-REDT), por la Xunta de Galicia (GRC2014/035 y ED431G/01) y por Fondos de Desarrollo Regional de la Unión Europea.

las mismas estableciendo un umbral de importancia a partir del cual se desechan las características que no lo alcancen. La idea principal es asignar un peso a cada atributo de acuerdo con su capacidad a la hora de distinguir ejemplos que se encuentran muy cerca. Por ello, para cada ejemplo, Relief busca el elemento de la misma clase (llamado *hit*) más cercano y el elemento de otra clase (llamado *miss*) más cercano y actualiza el peso de cada atributo A en función de la coincidencia o no de su valor en el ejemplo con el del *hit* y el del *miss*. El peso final W de cada atributo A tiene, por tanto, una interpretación probabilística ya que es una aproximación de la siguiente diferencia de probabilidades condicionadas:

$$W[A] = P(\text{valor diferente de A}|\text{miss más cercano}) - P(\text{valor diferente de A}|\text{hit más cercano}) \quad (1)$$

Su buen funcionamiento dio lugar a extensiones capaces de lidiar con problemas multiclase y ejemplos con ruido o incompletos [8]. ReliefF es una de estas extensiones y ha terminado siendo más popular que el algoritmo original, por lo que se ha implementado en numerosas librerías y software de aprendizaje automático. Posteriormente han sido publicadas especializaciones del algoritmo [9] para adaptarlo a problemas de regresión [10], multietiqueta [11], [12] o para tener en cuenta el coste de obtener cada atributo [13]. Además también se han hecho optimizaciones para facilitar su uso con grandes conjuntos de datos, basadas en implementaciones distribuidas [14], [15], muestreo [16] y en el uso de árboles *k-d* aleatorios para aproximar el grafo de vecinos más cercanos [17], pero su aplicabilidad a conjuntos de alta dimensionalidad aún es limitada.

ReliefF, al igual que otros muchos métodos de selección de características, así como también problemas de Recuperación de Información, Minería de Datos y Aprendizaje Automático, se basa en el estudio de grafos de similitud entre elementos del conjunto de datos. Entre los grafos utilizados en aprendizaje máquina, el más popular es el grafo de los k vecinos más cercanos (*kNN* por sus siglas en inglés). Un grafo *kNN* es un grafo dirigido sobre *n* elementos, $X = \vec{x}_1, \vec{x}_2, \ldots \vec{x}_n$, en el que las aristas $(\vec{x_i}, \vec{x_j})$ indican que $\vec{x_j}$ se encuentra entre los *k* elementos más similares a $\vec{x_i}$ atendiendo a una medida de similitud $S(\vec{x_i}, \vec{x_j})$. El grafo resultante es muy versátil, pero el coste computacional de su cálculo es muy alto, ya que requiere $n(n-1)/2$ comparaciones, lo que sitúa su complejidad computacional en $\mathcal{O}(n^2)$. Se han propuesto algoritmos que calculan el grafo *kNN* exacto cuando la dimensionalidad del espacio de entrada es pequeña [18], además de algoritmos eficientes para medidas de similitud específicas [19], sin embargo, a la hora de lidiar con conjuntos de datos de alta dimensionalidad y medidas generales en tiempos manejables, solo se pueden construir soluciones aproximadas. Estas soluciones buscan replicar el grafo exacto con la mayor veracidad posible manteniendo el coste computacional bajo. Para obtener el grafo aproximado existen en la literatura algoritmos basados en diseño "divide y vencerás" [20], búsquedas locales [21] y en Hashing Sensible a la Localidad [22], [23].

En este sentido, el Hashing Sensible a la Localidad (LSH, por sus siglas en inglés) [24] es una técnica que se originó para construir estructuras de datos que permitiesen realizar consultas sobre un conjunto de datos en tiempo sublineal, aunque el método óptimo de explotar esta técnica todavía constituye un problema abierto. Básicamente, las técnicas de LSH permiten mapear datos de cualquier tamaño a un espacio, generalmente menor, utilizando una función de *hashing* que maximiza la probabilidad de que a datos similares se les asigne el mismo valor. Por este motivo, el uso de LSH para la construcción del *kNN* se ha explorado con éxito en la literatura [22], [23] con soluciones que utilizan LSH para aislar en pequeñas agrupaciones los elementos similares entre sí. A partir de estas agrupaciones se construyen subgrafos aislados que se combinan para obtener el grafo aproximado completo.

III. Algoritmo propuesto

El propósito de este trabajo es obtener un algoritmo que aproxime el resultado de ReliefF con menor esfuerzo computacional. La mayor parte de la carga de trabajo de este algoritmo se invierte en el cómputo de los *hits* y *misses* más cercanos a cada dato, por lo tanto, es esta parte del algoritmo la que se propone sustituir por el algoritmo LSH de Radio Variable (VRLSH, de sus siglas en inglés) [23]. Este algoritmo utiliza proyecciones LSH iterativas para obtener el grafo *kNN* aproximado, como se explica a continuación.

En primer lugar, se asignan una o varias claves *hash* a cada elemento $\vec{x}_i$ del conjunto de datos X usando una función LSH. Esta función está diseñada para asignar claves iguales a elementos similares de acuerdo a una medida de similitud dada y usando un nivel de resolución o *radio* de búsqueda también dado, que inicialmente será pequeño para buscar similitudes muy exactas. A continuación, se agrupan los elementos a los que se les ha asignado la misma clave. Tendremos, gracias a las propiedades de la función *hash*, pequeños grupos o "cúmulos" de elementos similares de acuerdo a la medida de similitud dada. A partir de estas agrupaciones para cada una de ellas se computa un subgrafo *kNN* exacto por el método de fuerza bruta, consistente en comparar cada elemento con todos los demás de su mismo cúmulo. Dado que un mismo elemento $\vec{x}_i$ puede pertenecer a varios cúmulos, posteriormente y si existe este solapamiento, los subgrafos de los cúmulos implicados se fusionan para incluir todos los vecinos que han sido detectados para $\vec{x}_i$. A continuación, se simplificará el conjunto de datos X eliminando aquellos elementos $\vec{x}_i$ que hayan estado involucrados en al menos un número preestablecido $C_{MAX} > k$ de comparaciones. Este proceso se repetirá, utilizando el nuevo X, para generar nuevos subgrafos que se fusionarán entre sí y con los ya existentes, hasta que el grafo fusionado contenga todos los elementos del conjunto original. En cada nueva iteración se aumentará la resolución, lo cual permitirá agrupar elementos un poco más diferentes que en la iteración anterior. Finalmente, si algún elemento queda con menos de k vecinos, se completará con los vecinos de sus vecinos o con los vecinos de elementos al azar si no tuviese ninguno. Este proceso aparece descrito en el Algoritmo 2.

Adicionalmente, para que este grafo resulte de utilidad a ReliefF en el cálculo de los pesos W de cada atributo es necesario hacer una modificación de VRLSH. El algoritmo VRLSH mantiene una única lista de vecinos para cada elemento del conjunto de datos. ReliefF, sin embargo, necesita distinguir *hits* y los *misses* más cercanos de cada clase, por lo que es necesario adaptar VRLSH para que mantenga una lista de vecinos por cada posible clase para cada elemento del conjunto de datos.

Entrada: X ←Conjunto de datos
k ← Número de vecinos a obtener
R_{INI} ← Radio de búsqueda inicial
C_{MAX} ←Máximas comparaciones por elemento
N ← Número de hashes a obtener
L ← Longitud de cada hash
Salida: G ← Grafo kNN

$G \leftarrow \emptyset$, $cúmulos \leftarrow \emptyset$, $originalX \leftarrow X$, $radio \leftarrow R_{INI}$
mientras $|cúmulos| > 1$ ***or*** $|X| > 1$ ***and*** $|X| < k$ **hacer**
  $hashElems \leftarrow LSH(X, radio, N, L)$
  $cúmulos \leftarrow hashElems.agrupaPorHash()$
  para cada c ***in*** $cúmulos$ **hacer**
    si $|c| > 1$ **entonces**
      $G \leftarrow G \cup KNNexacto(c.elems, k)$ **fin**
  fin
  $X \leftarrow X - G.nodosConAlMenosNComparaciones(C_{MAX})$
  $radio \leftarrow radio * 2$
fin
si $|X| > 1$ **entonces**
  para cada x ***in*** X **hacer**
    si $|x.vecinos| = 0$ **entonces**
      x.vecinos←alAzar(originalX,k)
    en otro caso
      $x.vecinos \leftarrow x.vecinos \cup desciendeVecinos(X, G)$
    fin
  $G \leftarrow G \cup desciendeVecinos(X, G)$
fin

Algoritmo 1: Pseudocódigo del algoritmo VRLSH

IV. EXPERIMENTACIÓN

Para comprobar la validez del método propuesto se han llevado a cabo dos baterías de experimentos. En primer lugar se ha medido el tiempo de ejecución necesario para el cómputo de los pesos de los atributos usando ReliefF en conjuntos de datos reales y se ha comparado con el tiempo que requiere ReliefF-LSH sobre los mismos conjuntos. Dado que ReliefF-LSH es un método aproximado, se ha medido además la exactitud de los resultados obtenidos comparando las ordenaciones de atributos obtenidas por los dos métodos. La segunda batería de experimentos va encaminada a comprobar la escalabilidad

Entrada: X ←Conjunto de datos
k ← Número de vecinos a utilizar
Salida: W ← Vector de pesos asignados a cada atributo

para cada A ***in*** $X.atributos$ **hacer**
  $W[A] \leftarrow 0$
fin
$G \leftarrow VRLSH(X, k, R_{INI}, C_{MAX}, N, L)$
para cada $x \in X$ ***in*** G **hacer**
  para cada A *in* $X.atributos$ **hacer**
    $c \leftarrow x.clase$
    W[A] $\leftarrow$ **W[A]** $- \sum_{j=1}^{k} \frac{diff(A,x,vecinos(c)_j)}{|X|*k} + \sum_{\gamma \neq c} \left[\frac{P(c)}{1-P(\gamma)} \sum_{j=1}^{k} \frac{diff(A,x,vecinos(\gamma)_j)}{|X|*k} \right]$
  fin
fin

Algoritmo 2: Pseudocódigo de ReliefF-LSH

del método. Para ello se compara el tiempo de ejecución de ReliefF-LSH sobre los mismos conjuntos reales utilizando un número creciente de núcleos de cómputo.

IV-A. Datos y metodología

Todos los experimentos se llevaron a cabo en máquinas con 12 núcleos de computación que forman parte de un clúster. La descripción de cada nodo de computación aparece en el Cuadro I. La versión de Spark utilizada es la 1.6.1, sobre Hadoop 2.7.1.2.4.2.0-258. El sistema operativo instalado en las máquinas es CentOS Linux release 7.4.1708.

Cuadro I
DESCRIPCIÓN DEL CLÚSTER DE COMPUTACIÓN

32 nodos con las siguientes características:	
Procesador:	2 × Intel Xeon E5-2620 v3 a 2.40Ghz
Núcleos:	6 por procesador (12 por nodo)
Threads:	2 por núcleo (24 en total por nodo)
Disco:	12 × 2TB NL SATA 6Gbps 3.5"G2HS
RAM:	64 GB
Red:	1x10Gbps + 2x1Gbps

Para realizar estos experimentos se eligieron cuatro conjuntos de datos reales de alta dimensionalidad. En primer lugar se utilizó *Higgs*, que representa propiedades cinéticas de partículas detectadas en un acelerador[1] [25]. Consta de 28 atributos numéricos y 11 millones de ejemplos. En segundo lugar se usó el conjunto artificial *Epsilon*, creado para el Pascal Large Scale Learning Challenge [26] en 2008 y que se compone de 500.000 ejemplos con 2.000 atributos cada uno. Por último, se utilizó el conjunto multiclase *Isolet* [27], que tiene 7.900 ejemplos de 617 atributos distribuidos en 27 clases. Los conjuntos utilizados y sus características aparecen reflejados en el Cuadro II.

[1]Disponible para descarga en https://archive.ics.uci.edu/ml/datasets/HIGGS

Cuadro II
Descripción de los conjuntos de datos

Conjunto	Atributos	Muestras	Clases
Higgs	28	11.000.000	2
Epsilon	2.000	500.000	2
Isolet	617	7.900	27

Cuadro III
Tiempo de ejecución de las implementaciones

	Tiempo (s)	
	ReliefF	ReliefF-LSH
# núcleos	12	12
Higgs (0.5 %)	5.550	32
Epsilon (10 %)	13.150	2.264
Isolet	320	29

El alto número de muestras de algunos conjuntos los sitúa fuera del alcance de la versión original de ReliefF, dado que su tiempo de ejecución sería de semanas, aún utilizando 12 núcleos de computación. En consecuencia, para el primer experimento consistente en comparar los tiempos de ejecución de la versión original con ReliefF-LSH se usaron versiones reducidas de los conjuntos más grandes tomando solamente los N primeros elementos del conjunto. En particular, se tomó el primer 0.5 % de los datos del conjunto Higgs (55.000 elementos) y el primer 10 % de Epsilon (50.000 elementos). No obstante, para demostrar la capacidad del método para tratar con conjuntos grandes, en el segundo experimento se utilizaron las versiones completas de los conjuntos.

IV-B. Resultados

El primer experimento consistió en comparar los resultados obtenidos con ReliefF-LSH con los obtenidos por la versión exacta. En primer lugar se compararon los tiempos de ejecución necesarios para procesar cada conjunto de datos, mostrados en el Cuadro III. Se puede apreciar que los tiempos de ejecución de ReliefF-LSH son siempre muy inferiores a los de su contrapartida exacta. Cabe destacar que, tal como se describe en [23], tanto el tiempo de ejecución de VRLSH como su precisión dependen de los hiperparámetros con que se ejecute. En este caso los hiperparámetros fueron obtenidos empíricamente para que el grafo aproximado se computase realizando en torno al 1 % de las comparaciones necesarias para calcular el grafo exacto. No obstante, existe un equilibrio entre tiempo de cómputo y exactitud del grafo obtenido que el usuario debe manejar en función de sus preferencias. Los hiperparámetros utilizados para este experimento aparecen listados en el Cuadro V.

En lo referente a la exactitud de los subconjuntos de características seleccionadas por ReliefF-LSH, estas dependen en gran medida de dos factores. En primer lugar, si ReliefF asigna a las características de un conjunto puntuaciones que difieren muy poco, es probable que el cálculo aproximado obtenga valores ligeramente distintos que las ordenen de manera distinta. Por contra, los conjuntos que muestran grandes diferencias entre sus características en términos de la puntuación otorgada por ReliefF, obtienen rankings más robustos frente a las pequeñas diferencias de puntuación derivadas del cálculo aproximado. En segundo lugar, la exactitud de las puntuaciones otorgadas por Relief-LSH dependen de la exactitud del grafo aproximado calculado con VRLSH y que, como ya se mencionó anteriormente, viene determinado por el número de comparaciones entre pares que se hayan realizado, que a su vez se determinan en función de los hiperparámetros utilizados. Para representar la exactitud de los subconjuntos de características obtenidos hemos definido el ratio de coincidencia entre el subconjunto seleccionado por el algoritmo exacto ($\mathcal{E}$) y el seleccionado por ReliefF-LSH ($\mathcal{L}$) definido como:

$$R = \frac{|\mathcal{E} \cap \mathcal{L}|}{\mathcal{E}} \tag{2}$$

En la Figura 1 hemos representado el ratio de coincidencia para distintos niveles de selección dentro del ranking devuelto por los algoritmos. La ordenación de características para el conjunto de datos *Higgs* se realizó sobre un grafo aproximado calculado con alrededor de $5 * 10^{-3}$ veces menos cálculos de los que requeriría el grafo exacto. Esto se tradujo en un tiempo de ejecución muy bajo, pero también el nivel de coincidencia es bajo cuando se seleccionan pocos atributos. Por el contrario, los niveles de coincidencia para los atributos seleccionados en el caso de los conjuntos *Epsilon* y *Isolet* son más altos, rondando el 90 % en el caso de *Epsilon* y el 80 % en *Isolet*. Es importante destacar, no obstante, que para niveles de selección muy extremos en los que nos quedamos con muy pocas variables es posible que los pequeños cambios en la ordenación de variables dejen fuera algunas que sí aparecen en la selección exacta, disminuyendo así el ratio de coincidencia.

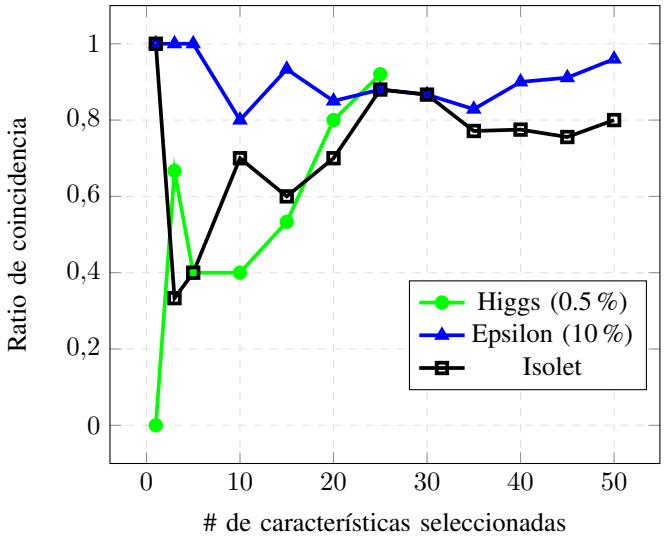


Figura 1. Ratio de coincidencia de las características seleccionadas por ReliefF vs ReliefF-LSH para los conjuntos Higgs (0.5 %), Epsilon (10 %) y Isolet.

Finalmente, realizamos un experimento con el objetivo de estudiar la escalabilidad del método. Para ello tomamos las

Cuadro IV
Tiempo de ejecución de Relief-LSH. Escalabilidad

	Tiempo (s)		
	ReliefF-LSH		
# núcleos	12	2x12	4x12
Higgs	15.726	4.206	2.580
Isolet	29	30	31

Cuadro V
Descripción de los hiperparámetros de VRLSH utilizados. *Operaciones* indica la fracción de operaciones realizadas con respecto al cálculo exacto.

Conjunto	L	N	R	Operaciones
Higgs (0.5 %)	5	20	0.5	$4{,}6 * 10^{-3}$
Higgs	8	5	0.25	$3{,}5 * 10-6$
Epsilon (10 %)	70	5	0.25	$3{,}7 * 10^{-2}$
Isolet	5	4	0.1	$2{,}8 * 10^{-3}$

versiones completas del conjunto con más muestras (*Higgs*) y con menos muestras (*Isolet*) y realizamos el cómputo del ranking de atributos repetidas veces, utilizando en cada una distinto número de nodos de computación. Los tiempos invertidos en cada caso aparecen reflejados en el Cuadro IV.

Este experimento pone de relieve la capacidad de ReliefF-LSH de procesar conjuntos que están completamente fuera del alcance de la versión exacta de ReliefF. Así, mientras que la versión exacta invirtió 5.550 segundos en procesar el 0.5 % de los ejemplos de *Higgs*, ReliefF-LSH pudo realizar el cómputo con el conjunto completo (200 veces mayor) en 15.726 segundos. Cabe recordar que, al ser la complejidad del ReliefF original de orden cuadrático en el número de muestras, cabría esperar un tiempo de ejecución del orden de 10^7 segundos, lo cual es inabarcable en la práctica. Además, se puede apreciar en los resultados que, gracias a que gran parte de los cálculos se pueden realizar independientemente de manera paralela, los nodos de cómputo que se añaden son aprovechados cuando el conjunto es grande, lo que se traduce en una relación inversamente proporcional de pendiente cercana a -1 entre el número de nodos de cómputo y el tiempo de ejecución, como es deseable. Esto no es así en conjuntos pequeños como *Isolet*, dado que el número de operaciones requeridas es bajo y la aceleración derivada de la paralelización se ve compensada por el sobreesfuerzo de comunicación que implica la distribución del trabajo a los nodos de cómputo.

Por otra parte, hay que recordar que ReliefF-LSH utiliza proyecciones aleatorias a la hora de computar el grafo de vecinos más cercanos, lo cual introduce un factor de azar que hace que cada ejecución sea distinta incluso cuando utilice los mismos hiperparámetros. Esto provoca que los tiempos de cómputo puedan variar entre ejecuciones. Finalmente, también en este apartado entra en juego el balance entre exactitud y rapidez mencionado anteriormente. Los valores de los hiperparámetros utilizados aparecen reflejados en el Cuadro V. Se puede comprobar que en el caso del conjunto de datos *Higgs* se realizaron $3{,}5 * 10^{-6}$ veces menos operaciones de las que se necesitarían para calcular el grafo exacto. Es posible que esto diese como resultado una ordenación de las características con un bajo ratio de coincidencia con el eventual resultado exacto - que resulta impracticable calcular. Dependiendo del nivel de selección deseado y del rendimiento de la selección realizada en el problema posterior para el que se precise, el usuario podría decidir obtener una ordenación más coincidente con la eventual solución exacta. Por suerte, gracias a la escalabilidad del método, el cálculo de un grafo más exacto alterando los hiperparámetros se puede realizar en un tiempo similar al registrado con tan solo añadir más nodos al cómputo.

V. Conclusiones

En este trabajo se ha propuesto una adaptación del popular algoritmo de selección de características ReliefF para posibilitar el tratamiento de grandes volúmenes de datos. Se ha hecho uso del algoritmo VRLSH para aproximar el cálculo de vecinos más cercanos, que constituye la parte más costosa computacionalmente de ReliefF, y se obtiene así una ordenación aproximada de las características del conjunto de datos. El algoritmo resultante, llamado ReliefF-LSH, se ha implementado en la plataforma Apache Spark y se han realizado experimentos para verificar la validez del método y su escalabilidad. Los experimentos muestran que ReliefF-LSH es un algoritmo que puede tratar conjuntos que están muy fuera del alcance de ReliefF. Asimismo, el algoritmo propuesto ofrece al usuario la posibilidad de balancear la rapidez de ejecución con la exactitud de la solución, lo que lo convierte en una versátil herramienta para problemas de distintas complejidades.

Actualmente estamos trabajando en simplificar el manejo de los hiperparámetros de VRLSH por parte del usuario, para facilitar el mencionado balanceo de rapidez y precisión. Igualmente, como trabajo futuro queremos añadir estructuras de datos que eviten cómputos duplicados, acelerando la ejecución.

Agradecimientos

Los autores desean agradecer a la Fundación Pública Galega Centro Tecnolóxico de Supercomputación de Galicia (CESGA) el uso de sus recursos de computación.

Referencias

[1] Saint John Walker. Big data: A revolution that will transform how we live, work, and think, 2014.
[2] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
[3] Apache Hadoop Project. http://hadoop.apache. org/. Accessed: 2016-04-19.
[4] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.
[5] Apache Mahout Project. http://mahout.apache. org/. Accessed: 2016-04-19.
[6] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, 2016.
[7] Kenji Kira and Larry A Rendell. A practical approach to feature selection. In *Machine Learning Proceedings 1992*, pages 249–256. Elsevier, 1992.

[8] Igor Kononenko. Estimating attributes: analysis and extensions of relief. In *European conference on machine learning*, pages 171–182. Springer, 1994.

[9] Ryan J Urbanowicz, Melissa Meeker, William LaCava, Randal S Olson, and Jason H Moore. Relief-based feature selection: introduction and review. *arXiv preprint arXiv:1711.08421*, 2017.

[10] Marko Robnik-Šikonja and Igor Kononenko. An adaptation of relief for attribute estimation in regression. In *Machine Learning: Proceedings of the Fourteenth International Conference (ICML'97)*, pages 296–304, 1997.

[11] Newton Spolaôr, Everton Alvares Cherman, Maria Carolina Monard, and Huei Diana Lee. Relieff for multi-label feature selection. In *Intelligent Systems (BRACIS), 2013 Brazilian Conference on*, pages 6–11. IEEE, 2013.

[12] Ivica Slavkov, Jana Karcheska, Dragi Kocev, Slobodan Kalajdziski, and Sašo Džeroski. Relieff for hierarchical multi-label classification. In *International Workshop on New Frontiers in Mining Complex Patterns*, pages 148–161. Springer, 2013.

[13] Verónica Bolón-Canedo, Beatriz Remeseiro, Noelia Sánchez-Maroño, and Amparo Alonso-Betanzos. mc-relieff-an extension of relieff for cost-based feature selection. In *ICAART (1)*, pages 42–51, 2014.

[14] Carlos Eiras-Franco, Verónica Bolón-Canedo, Sabela Ramos, Jorge González-Domínguez, Amparo Alonso-Betanzos, and Juan Touriño. Multithreaded and spark parallelization of feature selection filters. *Journal of Computational Science*, 17:609–619, 2016.

[15] Raul-Jose Palma-Mendoza, Daniel Rodriguez, and Luis de Marcos. Distributed relieff-based feature selection in spark. *Knowledge and Information Systems*, pages 1–20, 2018.

[16] Margaret J Eppstein and Paul Haake. Very large scale relieff for genome-wide association analysis. In *Computational Intelligence in Bioinformatics and Computational Biology, 2008. CIBCB'08. IEEE Symposium on*, pages 112–119. IEEE, 2008.

[17] Sitong Xu, Xiang Li, and Wen Feng Lu. Randomized kd tree relieff algorithm for feature selection in handling high dimensional process parameter data. In *Emerging Technologies and Factory Automation (ETFA), 2016 IEEE 21st International Conference on*, pages 1–8. IEEE, 2016.

[18] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[19] David C Anastasiu and George Karypis. L2knng: Fast exact k-nearest neighbor graph construction with l2-norm pruning. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 791–800. ACM, 2015.

[20] Jie Chen, Haw-ren Fang, and Yousef Saad. Fast approximate knn graph construction for high dimensional data via recursive lanczos bisection. *Journal of Machine Learning Research*, 10(Sep):1989–2012, 2009.

[21] Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*, pages 577–586. ACM, 2011.

[22] Yan-Ming Zhang, Kaizhu Huang, Guanggang Geng, and Cheng-Lin Liu. Fast knn graph construction with locality sensitive hashing. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 660–674. Springer, 2013.

[23] Carlos Eiras-Franco, Leslie Kanthan, Amparo Alonso-Betanzos, and David Martınez-Rego. Scalable approximate k-nn graph construction based on locality sensitive hashing.

[24] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 459–468. IEEE, 2006.

[25] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5:4308, 2014.

[26] Soeren Sonnenburg, Vojtech Franc, Elad Yom-Tov, and Michele Sebag. Pascal large scale learning challenge. In *25th International Conference on Machine Learning (ICML2008) Workshop. http://largescale. first. fraunhofer. de. J. Mach. Learn. Res*, volume 10, pages 1937–1953, 2008.

[27] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017.

II Workshop en Big Data y Análisis de Datos Escalable (II BigDADE)

SESIÓN 2

Smart Data: Filtrado de Ruido para Big Data

Diego García-Gil*, Julián Luengo*, Salvador García* y Francisco Herrera*
*Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad de Granada, Granada, España, 18071
Email: {djgarcia,julianlm,salvagl,herrera}@decsai.ugr.es

***Resumen*—En cualquier proceso de minería de datos, el valor del conocimiento extraído está directamente relacionado con la calidad de los datos utilizados. Los problemas Big Data, generados por el crecimiento masivo de los datos en los últimos años, siguen el mismo dictado. Un problema común que afecta a la calidad de los datos es la presencia de ruido, especialmente en los problemas de clasificación, en los que el ruido de clase se refiere al etiquetado incorrecto de las instancias de entrenamiento. Sin embargo, en la era del Big Data, las grandes cantidades de datos plantean un reto a las propuestas tradicionales creadas para hacer frente al ruido, ya que tienen dificultades para procesar tal cantidad de datos. Es necesario proponer nuevos algoritmos para el tratamiento de ruido en Big Data, obteniendo datos limpios y de calidad, también conocidos como Smart Data. En este trabajo se proponen dos enfoques de preprocesamiento de datos en Big Data para eliminar instancias ruidosas: un ensemble homogéneo y un ensemble heterogéneo, con especial énfasis en su escalabilidad y rendimiento. Los resultados obtenidos muestran que estas propuestas permiten obtener eficientemente un Smart Dataset a partir de cualquier problema de clasificación Big Data.**

***Index Terms*—Big Data, Smart Data, Clasificación, Preprocesamiento, Ruido de clase**

I. Preliminares

Hoy en día estamos rodeados por enormes cantidades de datos. Está previsto que para 2020 el universo digital sea 10 veces más grande que en 2013, totalizando unos asombrosos 44 zettabytes [1]. El volumen actual de datos ha superado las capacidades de procesamiento de los sistemas clásicos de minería de datos y ha creado la necesidad de nuevos frameworks para almacenarlos y procesarlos. Es un hecho ampliamente aceptado que hemos entrado en la era del Big Data. El Big Data es el conjunto de tecnologías que hacen posible el procesamiento de grandes cantidades de datos.

En Big Data, las técnicas preprocesamiento clásicas para mejorar la calidad de los datos consumen aún más tiempo y exigen más recursos, volviéndose inviables. La falta de técnicas de preprocesamiento eficientes y asequibles implica que las alteraciones en los datos afectarán a los modelos extraídos de ellos. Entre todos los problemas que pueden aparecer en los datos, la presencia de *ruido* es uno de los más frecuentes. El ruido puede definirse como la alteración parcial o total de la información recogida en una instancia causada por un factor externo. El ruido conduce a modelos excesivamente complejos con un rendimiento degradado.

Recientemente, se ha introducido el concepto Smart Data (centrado en la veracidad y el valor de los datos), con el objetivo de filtrar el ruido y resaltar los datos valiosos. Hay tres atributos clave para que los datos sean *inteligentes*, deben ser:

- Precisos: los datos deben ser lo que dicen que son con suficiente precisión para generar valor. La calidad de los datos es importante.
- Procesables: los datos deben ser escalables inmediatamente para maximizar objetivos empresariales.
- Rápidos: los datos deben estar disponibles en tiempo real y listos para adaptarse a las cambiantes exigencias empresariales.

El modelado y análisis avanzado en Big Data es indispensable para descubrir la estructura subyacente en los datos con el fin de obtener Smart Data. En este trabajo aportamos varias técnicas de preprocesamiento para Big Data, transformando conjuntos de datos sin procesar en Smart Data. Nos hemos centrado en la tarea de clasificación, donde se distinguen dos tipos de ruido: *ruido de clase*, cuando afecta a la etiqueta de una instancia, y *ruido de atributo*, cuando afecta a los atributos. El primero es conocido por ser el más perjudicial [1]. Como consecuencia, muchos trabajos, incluido este, han sido dedicados a resolver este problema, o al menos a minimizar sus efectos (en [2] encontramos una completa revisión).

A pesar de que se han propuesto algunos diseños para tratar el ruido en Big Data [3], no se ha propuesto ningún algoritmo para eliminar el ruido ni se ha llevado a cabo ninguna comparación en Big Data.

Por esto proponemos un framework para eliminar instancias ruidosas en Big Data sobre Apache Spark, compuesto de dos algoritmos basados en ensembles de clasificadores. El primero es un Ensemble Homogéneo (EHO), el cual usa un clasificador base (Random Forest [4]) sobre un dataset particionado. El segundo es un Ensemble Heterogéneo (EHT), el cual usa tres clasificadores para identificar instancias ruidosas: Random Forest, Regresión Logística y k-Nearest Neighbors (kNN). Para una comparativa más completa, hemos implementado un filtro basado en similaridad de instancias, llamado Edited Nearest Neighbor (ENN). ENN examina los vecinos más cercanos de cada ejemplo de entrenamiento y elimina aquellos cuyo vecindario pertenece a una clase diferente. Todas estas técnicas han sido implementadas en Apache Spark [5], y pueden ser descargadas de su repositorio comunitario [2].

Para mostrar el rendimiento de los métodos propuestos, hemos llevado a cabo experimentos con cuatro grandes datasets: *SUSY*, *HIGGS*, *Epsilon* y *ECBDL14*. Hemos creado diferentes niveles de ruido de clase para evaluar los efectos de aplicar los

[1] IDC: The Digital Universe of Opportunities. 2018 [Online] http://www.emc.com/infographics/digital-universe-2014.htm

[2] https://spark-packages.org/package/djgarcia/NoiseFramework

métodos, así como la mejora obtenida en términos de precisión para dos clasificadores: un árbol de decisión y kNN. También mostramos que, en los problemas estudiados, los clasificadores se benefician del filtrado de ruido incluso cuando no hay ruido añadido, ya que los problemas Big Data contienen ruido propio debido a la automatización en la recogida de los datos. Los resultados indican que la propuesta es capaz de eliminar exitosamente el ruido. En concreto, EHO es la primera técnica válida para tratar el ruido en problemas Big Data, mejorando la precisión con un bajo coste computacional.

El resto del trabajo está organizado como sigue: la Sección II explica el framework propuesto. La Sección III describe los experimentos llevados a cabo para probar el rendimiento del framework. Finalmente, la Sección IV concluye el trabajo.

II. Filtrado de Ruido en Big Data

En esta sección, presentamos el primer framework para Big Data sobre Apache Spark para eliminar instancias ruidosas basado en el modelo MapReduce. Se trata de un diseño MapReduce donde todos los procesos de filtrado de ruido se realizan de forma distribuida. En la Sección II-A describimos el modelo MapReduce. En la Sección II-B listamos las primitivas de Spark usadas en la implementación del framework. Hemos diseñado dos algoritmos basados en ensembles. Ambos realizan un k-fold a los datos de entrada, aprenden un modelo en cada partición de entrenamiento y limpian las instancias ruidosas de la partición de test. El primero es un ensemble homogéneo que utiliza Random Forest como clasificador, llamado EHO (Sección II-C). El segundo, llamado EHT (Sección II-D), es un ensemble heterogéneo basado en tres clasificadores: Random Forest, Regresión Logística y kNN.

II-A. Modelo MapReduce

MapReduce es un framework diseñado por Google en 2003 [6], [7]. Este modelo se compone de una función Map, que realiza una transformación, y un método Reduce, que realiza una agregación. El flujo de trabajo de una tarea MapReduce es el siguiente: el nodo máster divide el conjunto de datos y lo distribuye por el clúster. Después cada nodo aplica la función Map a sus datos. A continuación, los datos se redistribuyen por el clúster en función de los pares clave-valor generados en la fase Map. Una vez que todos los pares de una misma clave están en el mismo nodo, se procesan en paralelo.

Apache Hadoop[3] es el framework open-source más conocido para el almacenamiento y procesamiento de grandes cantidades de datos, basado en el modelo MapReduce. El framework está diseñado para tratar los errores de hardware automáticamente. A pesar de su popularidad y rendimiento, Hadoop tiene algunas limitaciones [8]: bajo rendimiento para tareas iterativas e imposibilidad de trabajar en memoria.

Apache Spark[4] es un framework open-source centrado en la velocidad, facilidad de uso y procesamiento en memoria [9]. Spark está construido sobre una estructura de datos distribuida e inmutable llamada Resilient Distributed Datasets (RDDs) [10]. Los RDDs se pueden describir como un conjunto de particiones de datos distribuidas por el clúster. Los RDDS soportan dos tipos de operaciones: acciones, que evalúan y devuelven un valor. Y transformaciones, que no son evaluadas cuando se definen, se aplican sobre un RDD y producen un nuevo RDD. Cuando se llama a una acción sobre un RDD, todas las transformaciones previas se aplican en paralelo a cada partición del RDD.

[3]Apache Hadoop Project 2018 [Online] https://hadoop.apache.org/

[4]Apache Spark Project 2018 [Online] https://spark.apache.org/

II-B. Primitivas Spark

Para la implementación del framework, hemos utilizado algunas primitivas de la API de Spark. Estas funciones ofrecen operaciones más complejas al extender el modelo MapReduce. A continuación, describimos las empleadas en los algoritmos:

- map: Aplica una transformación a cada elemento de un RDD.
- $zipWithIndex$: Para cada elemento de un RDD crea una tupla con el elemento y su índice, empezando en 0.
- $join$: Devuelve un RDD que contiene todos los pares de elementos cuyas claves coinciden entre dos RDDs.
- $filter$: Devuelve un nuevo RDD que contiene solo los elementos que satisfacen una condición.
- $union$: Devuelve un RDD resultante de la unión de dos RDDs.
- $kFold$: Devuelve una lista de k pares de RDDs, siendo el primer elemento los datos de $train$, y el segundo elemento los datos de $test$. Donde k es el número de particiones.
- $randomForest$: Método para aprender un modelo de Random Forest.
- $predict$: Devuelve un RDD que contiene las instancias y clases predichas para un dataset usando un modelo.
- $learnClassifiers$: A pesar de no ser una primitiva de Spark, la usamos para simplificar la descripción de los algoritmos. Esta primitiva aprende un modelo de Random Forest, Regresión Logística y 1NN a partir de los datos de entrada.

Estas primitivas de Spark se usan en las siguiente secciones donde describimos EHO y EHT.

II-C. Ensemble Homogéneo: EHO

El ensemble homogéneo está inspirado en Cross-Validated Committees Filter (CVCF) [11]. Este filtro elimina instancias ruidosas realizando un k-fold a los datos. Después entrena un árbol de decisión k veces, dejando fuera uno de los subconjuntos cada vez. Esto da como resultado k clasificadores que se utilizan para predecir los datos de entrenamiento k veces. Luego, utilizando una estrategia de voto, se eliminan las instancias mal clasificadas.

EHO también está basado en un esquema de particionamiento de los datos. Hay una diferencia importante con respecto a CVCF: el uso de Random Forest en lugar de un árbol de decisión como clasificador. CVCF crea un ensemble particionando los datos de entrenamiento. EHO también divide

Figura 1. Algoritmo EHO

```
1: Entrada: data RDD de tuplas (clase, atributos)
2: Entrada: P número de particiones
3: Entrada: nTrees número de árboles para Random Forest
4: Salida: RDD filtrado sin ruido
5: partitions ← kFold(data, P)
6: filteredData ← ∅
7: for all train, test in partitions do
8:     rfModel ← randomForest(train, nTrees)
9:     rfPred ← predict(rfModel, test)
10:    joinedData ← join(zipWithIndex(test), zipWithIndex(rfPred))
11:    markedData ←
12:    map original, prediction ∈ joinedData
13:        if label(original) = label(prediction) then
14:            original
15:        else
16:            (label = ∅, features(original))
17:        end if
18:    end map
19:    filteredData ← union(filteredData, markedData)
20: end for
21: return (filter(filteredData, label ≠ ∅))
```

los datos de entrenamiento, pero el uso de Random Forest nos permite mejorar el paso de la votación:

- CVCF predice el dataset entero k veces. EHO solo predice las instancias de la partición $test$ del k-fold. Este paso se repite k veces. Con este cambio no solo mejoramos el rendimiento, sino también el tiempo de cómputo, ya que solo tiene que predecir una pequeña parte de los datos de entrenamiento en cada iteración.
- No necesitamos implementar una estrategia de votación, la decisión de si una instancia es ruidosa está asociada con la predicción del Random Forest.

La Figura 1 describe el proceso de filtrado de ruido en EHO. Los parámetros de entrada son los siguientes: el dataset ($data$), el número de particiones (P) y el número de árboles para el Random Forest ($nTrees$):

- El algoritmo realiza un $kFold$ en los datos de entrada. Como se definió antes, la función $kFold$ de Spark devuelve una lista de $(train, test)$ para un P dado.
- Iteramos sobre cada partición, aprendiendo un Random Forest usando $train$ como datos de entrada, y prediciendo $test$ usando el modelo aprendido.
- Para unir los datos de $test$ y los datos predichos para comparar las clases, usamos la operación $zipWithIndex$ en ambos RDDs. Con esta operación, añadimos un índice a cada elemento de ambos RDDs. Este índice se utiliza como clave para la operación $join$.
- El siguiente paso es aplicar una función map al RDD anterior y comprobar, para cada instancia, la clase original y la predicha. Si la clase predicha y el original son diferentes, la instancia se marca como ruido y se devuelven sus atributos y la clase (línea 16).
- El resultado de la función map anterior es un RDD donde las instancias ruidosas están marcadas. Estas instancias son finalmente eliminadas usando una función $filter$ y el conjunto de datos resultante es devuelto.

Figura 2. Algoritmo EHT

```
1: Entrada: data RDD de tuplas (clase, atributos)
2: Entrada: P número de particiones
3: Entrada: nTrees número de árboles para Random Forest
4: Entrada: vote estrategia de voto (mayoría o consenso)
5: Salida: RDD filtrado sin ruido
6: partitions ← kFold(data, P)
7: filteredData ← ∅
8: for all train, test in partitions do
9:     classifiersModel ← learnClassifiers(train, nTrees)
10:    predictions ← predict(classifiersModel, test)
11:    joinedData ← join(zipWithIndex(predictions), zipWithIndex(test))
12:    markedData ←
13:    map rf, lr, knn, orig ∈ joinedData
14:        count ← 0
15:        if rf ≠ label(orig) then count ← count + 1
16:        if lr ≠ label(orig) then count ← count + 1
17:        if knn ≠ label(orig) then count ← count + 1
18:        if vote = majority then
19:            if count ≥ 2 then (label = ∅, features(orig))
20:            if count < 2 then orig
21:        else
22:            if count = 3 then (label = ∅, features(orig))
23:            if count ≠ 3 then orig
24:        end if
25:    end map
26:    filteredData ← union(filteredData, markedData)
27: end for
28: return (filter(filteredData, label ≠ ∅))
```

II-D. Ensemble Heterogéneo: EHT

El ensemble heterogéneo está inspirado en Ensemble Filter (EF) [12]. Este filtro de ruido usa tres algoritmos de aprendizaje para identificar instancias ruidosas: un árbol de decisión, kNN y un clasificador lineal. Realiza un k-fold sobre los datos y para cada una de las k particiones, aprende tres modelos en las otras $k-1$ particiones. Cada uno de los clasificadores da una etiqueta a los ejemplos de $test$. Finalmente, usando una votación, se eliminan las instancias ruidosas.

EHT sigue el mismo esquema que EF. La principal diferencia es la elección de los tres algoritmos de aprendizaje: en lugar de un árbol de decisión, usamos Random Forest. Como clasificador lineal se ha usado la Regresión Logística. Finalmente, kNN se ha mantenido como clasificador.

En la Figura 2 se describe el filtrado de ruido en EHT. Los parámetros de entrada son los siguientes: el dataset ($data$), el número de particiones (P), el número de árboles para Random Forest ($nTrees$) y la estrategia de voto ($vote$):

- Para cada partición de $train$ y $test$ del k-fold, se aprenden tres modelos: Random Forest, Regresión Logística y 1NN usando $train$ como datos de entrada.
- Después se predicen los datos de $test$ usando los tres modelos aprendidos. Esto crea un RDD de tripletas (rf, lr, knn) con la predicción de cada algoritmo para cada instancia.
- Las predicción y los datos de $test$ se unen por índice para poder comparar las predicciones con la etiqueta original.
- Se comparan las tres predicciones de cada instancia con la etiqueta original usando una función map y, dependiendo de la estrategia de voto, las instancias se marcan como ruidosas o limpias.
- Las instancias marcadas como ruidosas son eliminadas usando la función $filter$ y el dataset es devuelto.

III. Resultados Experimentales

Esta sección describe los experimentos llevados a cabo sobre cuatro problemas Big Data. En la Sección III-A mostramos los detalles de los datasets y parámetros usados por los filtros de ruido. Los estudios de precisión e instancias eliminadas se encuentran en la Sección III-B. Finalmente, la Sección III-C analiza los tiempos de cómputo de las propuestas.

III-A. Framework Experimental

En nuestros experimentos hemos utilizado cuatro datasets de clasificación:

- SUSY, compuesto por 5,000,000 de instancias y 18 atributos. El objetivo es distinguir entre una señal que produce partículas supersimétricas (SUSY) y una que no.
- HIGGS, formado por 11,000,000 de instancias y 28 atributos. Este dataset es un problema de clasificación en el que se distingue una señal que produce bosones de Higgs y otra que no.
- Epsilon, con 500,000 instancias y 2,000 atributos. Este dataset fue creado artificialmente para la Pascal Large Scale Learning Challenge en 2008.
- ECBDL14, con 32 millones de instancias y 631 atributos. Este dataset fue usado como referencia en la competición de machine learning del Evolutionary Computation for Big Data and Big Learning llevado a cabo el 14 de Julio, 2014, bajo la conferencia internacional GECCO-2014. Es un problema de clasificación binaria donde la distribución de clases está altamente desbalanceada: 98 % de instancias negativas. Para este problema, usamos una versión reducida con 1,000,000 de instancias y 30 % de instancias positivas.

Hemos llevado a cabo experimentos en cinco niveles de ruido: en cada uno, un porcentaje de las instancias de entrenamiento han sido alteradas, sustituyendo su clase por otra de las disponibles. Los niveles de ruido elegidos son 0 %, 5 %, 10 %, 15 % y 20 %. Un nivel de 0 % de ruido indica que el dataset no ha sido alterado. Dadas las limitaciones del algoritmo kNN, hemos realizado una validación hold-out.

Hemos elegido 4 particiones para EHO y EHT. Para el filtro heterogéneo, hemos usado dos estrategias de voto: *mayoría* (mismo resultado para al menos la mitad de los clasificadores) y *consenso* (mismo resultado para todos los clasificadores). Para ENN, con *k = 5* tenemos una carga de red mayor, y dada la alta redundancia de datos en Big Data, no hemos apreciado diferencia con respecto a *k = 1*. Por esto hemos elegido la opción más eficiente.

Tabla I
Precisión en test de kNN. Los valores más altos de cada dataset y nivel ruido se muestran en negrita

Dataset	Ruido	Original	EHO	EHT		ENN
Voto				Mayor.	Consen.	
SUSY	0 %	71,79	**78,73**	77,86	74,64	72,02
	5 %	69,62	**78,68**	77,68	73,38	69,84
	10 %	67,44	**78,63**	77,44	72,01	67,66
	15 %	65,27	**78,62**	77,19	70,52	65,28
	20 %	63,10	**78,56**	76,93	69,10	63,25
HIGGS	0 %	61,21	**64,26**	63,94	62,30	60,65
	5 %	60,10	**64,06**	63,63	61,45	59,60
	10 %	58,97	**63,83**	63,29	60,65	58,56
	15 %	57,84	**63,65**	62,86	59,81	57,52
	20 %	56,69	**63,53**	62,55	58,89	56,45
Epsilon	0 %	56,55	**58,11**	57,43	55,19	56,21
	5 %	55,71	**58,64**	57,47	55,47	55,43
	10 %	55,20	**58,51**	57,26	55,25	54,79
	15 %	54,54	**58,39**	57,00	55,00	54,30
	20 %	54,05	**58,02**	56,75	54,72	53,68
ECBDL14	0 %	74,83	**76,06**	75,12	73,54	73,94
	5 %	72,36	**75,60**	74,59	72,89	72,77
	10 %	69,86	**75,31**	74,19	72,50	71,40
	15 %	67,39	**75,11**	73,99	72,11	69,68
	20 %	64,90	**74,82**	73,70	71,89	67,64

Utilizamos dos clasificadores para probar la efectividad de los filtros de ruido, un árbol de decisión y kNN. El árbol de decisión puede adaptar su profundidad para detectar mejor las instancias ruidosas, mientras que kNN es sensible al ruido cuando el número de vecinos el bajo. Para evaluar el rendimiento de los modelos usamos la precisión (número de instancias correctamente clasificadas divido por el total de instancias). Se ha aumentado la profundidad del árbol de decisión ($depth = 20$) para mejorar la detección de ruido. Para kNN usamos *k = 1* ya que es más sensible al ruido.

Los experimentos se han realizado en un clúster compuesto por 20 nodos de cómputo con la siguiente configuración: 2 x Intel Xeon E5-2620, 6 núcleos, 2.00 GHz, 2 TB HDD, 64 GB RAM. Hemos utilizado la siguiente configuración software: Hadoop 2.6.0-cdh5.4.3 de la distribución open source de Apache Hadoop de Cloudera, Apache Spark y MLlib 2.2.0, 460 núcleos (23 núcleos/nodo), 960 RAM GB (48 GB/nodo).

III-B. Análisis de Precisión e Instancias Eliminadas

En esta sección se presenta el análisis de los resultados obtenidos por los clasificadores después de aplicar el framework propuesto. Denotamos con *Original* el uso del clasificador sin utilizar ninguna técnica de tratamiento de ruido.

La Tabla I muestra la precisión en test para los cuatro datasets y los cinco niveles de ruido usando kNN como clasificador. En vista de estos resultados podemos señalar que:

Tabla II
Precisión en test del árbol de decisión. Los valores más altos de cada dataset y nivel ruido se muestran en negrita

Dataset	Ruido	Original	EHO	EHT		ENN
Voto				Mayor.	Consen.	
SUSY	0 %	80,24	79,78	79,69	**80,27**	78,56
	5 %	79,94	79,99	80,07	**80,36**	77,49
	10 %	79,15	79,85	79,81	**80,04**	77,00
	15 %	78,21	**79,81**	79,32	79,47	75,81
	20 %	77,09	**79,71**	79,35	78,95	74,21
HIGGS	0 %	70,17	**71,16**	69,61	70,41	68,85
	5 %	69,61	**71,14**	69,34	69,98	68,29
	10 %	69,22	**71,06**	68,95	69,56	67,52
	15 %	68,65	**71,03**	68,52	69,04	66,93
	20 %	67,82	**71,05**	68,18	68,38	66,05
Epsilon	0 %	62,39	**66,86**	65,13	66,07	61,54
	5 %	61,10	**66,64**	65,32	66,09	60,41
	10 %	60,09	**66,87**	65,46	66,11	59,20
	15 %	59,02	**66,62**	65,33	65,99	58,09
	20 %	57,73	**66,46**	65,08	65,69	56,71
ECBDL14	0 %	73,98	**74,59**	74,21	74,51	73,66
	5 %	72,87	**74,64**	74,16	74,54	73,48
	10 %	71,67	**74,59**	73,84	74,51	72,75
	15 %	70,28	**74,61**	73,82	73,91	71,68
	20 %	68,66	**74,83**	73,78	73,82	70,16

Tabla III
Tasa de Reducción (%) para EHO, EHT y ENN

Dataset	Ruido	EHO	EHT		ENN
Voto			Mayoría	Consenso	
SUSY	0 %	20,62	21,04	8,74	49,51
	5 %	23,57	25,09	10,33	49,57
	10 %	26,50	27,94	11,68	49,66
	15 %	29,44	30,85	13,04	49,74
	20 %	32,35	33,71	14,22	49,82
HIGGS	0 %	29,08	35,13	8,20	49,71
	5 %	31,15	36,65	8,83	49,75
	10 %	33,23	38,11	9,59	49,81
	15 %	35,38	39,55	10,35	49,92
	20 %	37,34	41,05	11,11	49,88
Epsilon	0 %	34,31	22,30	2,90	49,97
	5 %	25,32	25,24	4,32	50,01
	10 %	27,80	27,88	5,83	49,97
	15 %	30,79	30,71	7,22	50,01
	20 %	33,52	33,44	8,74	50,17
ECBDL14	0 %	22,44	21,35	5,85	26,58
	5 %	25,80	24,51	8,25	31,06
	10 %	28,49	27,68	10,31	35,07
	15 %	31,13	30,71	12,07	38,63
	20 %	33,86	33,69	13,91	41,60

- El uso de cualquier técnica de filtrado de ruido siempre mejora la precisión *Original* para un mismo nivel de ruido. Hay que tener en cuenta que el uso de los filtros de ruido permiten a kNN obtener mejor rendimiento para cualquier nivel de ruido que *Original* a un nivel del 0 % para cualquier conjunto de datos.
- Si nos fijamos en la mejor estrategia de filtrado de ruido para kNN, el filtro homogéneo EHO, permite a kNN obtener los mejores valores de precisión.
- EHT obtiene un rendimiento inferior a EHO. Por otro lado, la estrategia de voto es crucial para EHT, ya que el voto por consenso alcanza una precisión menor, cercana al 2 % con respecto al voto por mayoría.
- Mientras que la precisión para *Original* cae en torno a un 2 % cada 5 % de incremento de ruido, llegando a un total de un 10 % menos precisión al 20 % de ruido, EHO decrece menos de un 1 % en total.
- El método base, ENN, es la peor opción para kNN, ya que obtiene la precisión más baja de los tres filtros de ruido. Para ENN, la precisión cae alrededor de un 2 % cada 5 % de incremento de ruido. Sin embargo, ENN es preferible a no tratar el ruido.

La Tabla II recoge los resultados de precisión en test para los tres métodos de ruido usando un árbol de decisión profundo como clasificador. De estos resultados podemos señalar:

- De nuevo, evitar tratar el ruido no es la mejor opción, ya que usar el filtro adecuado supone una mejora de precisión. Sin embargo, como el árbol de decisión es más robusto al ruido que kNN, no todos los filtros son mejores que si no tratamos el ruido (*Original*). Cuando los filtros eliminan demasiadas instancias, limpias y ruidosas, el árbol de decisión se ve más afectado ya que es capaz de tolerar pequeñas cantidades de ruido mientras explora las instancias limpias. kNN se ve más afectado que el árbol de decisión por las instancias ruidosas mantenidas. Por esto, una mala elección del filtro de ruido penalizará el rendimiento del árbol de decisión.
- Para niveles bajos de ruido, EHT puede rendir ligeramente mejor que EHO para algún dataset. Sin embargo, del 10 % en adelante, EHO supera a EHT, haciéndolo el mejor filtro para lidiar con ruido en árboles de decisión.
- Como observamos anteriormente, la precisión *Original* cae con cada incremento de ruido. En este caso, EHO rinde incluso mejor que con kNN, ya que la precisión se mantiene de 0 % al 20 % de ruido.
- En cuanto a la estrategia de voto de EHT, el voto por consenso obtiene mejores resultados que el voto por mayoría. Hay que resaltar que en kNN se ha observado lo contrario: dado que kNN es mucho más sensible y necesita fronteras más limpias (logradas con el voto por mayoría), el árbol de decisión se beneficia de la eliminación de ruido más precisa del voto por consenso.
- ENN, alcanza en torno a un 1 % menos de precisión que el resto para niveles de ruido bajos, incrementándose esta diferencia hasta un 5 % para niveles altos.

Los resultados mostrados han demostrado la importancia de aplicar un filtrado de ruido, independientemente de la cantidad de ruido presente en el dataset. Para explicar mejor por qué EHO es la mejor estrategia de filtrado, debemos estudiar la cantidad de instancias eliminadas.

En la Tabla III mostramos la tasa de reducción de instancias después de aplicar los tres filtros de ruido sobre los cuatro datasets. Cuanto mayor sea el porcentaje de ruido, mayor será la cantidad de instancias eliminadas. Sin embargo, observamos

Tabla IV
Tiempo Medio de Cómputo para EHO, EHT y ENN en segundos

Dataset	EHO	EHT		ENN
Voto		Mayoría	Consenso	
SUSY	513,46	5.511,15	5.855,66	8.956,71
HIGGS	587,72	15.300,62	15.232,99	25,441,09
Epsilon	1.868,75	4.120,79	7.201,05	2.718,97
ECBDL14	1.228,24	9.710,70	11.217,02	14.080,03

diferentes patrones dependiendo de la técnica utilizada:

- De media, EHO elimina en torno a un 25 % de instancias al 0 % de ruido. Cada nivel de ruido, se eliminan un 3 % más de instancias.
- En EHT la estrategia de voto tiene un gran impacto en el número de instancias eliminadas. Mientras que el voto por mayoría elimina casi tantas instancias como EHO, el voto por consenso es mucho más conservativo.
- ENN es el filtro que más instancias elimina. De media elimina la mitad de las instancias del dataset al 0 % de ruido, incrementándose un 1 % cada nivel. Este comportamiento tan agresivo lacra el rendimiento de los clasificadores que toleran algo de ruido, como el árbol de decisión.
- EHO es el filtro más equilibrado en términos de instancias eliminadas. A pesar de que la cantidad de instancias eliminadas por EHT con el voto por mayoría es muy similar a EHO, las instancias seleccionadas son diferentes, afectando al clasificador usado posteriormente.

III-C. Tiempos de Cómputo

Para constituir una propuesta válida en Big Data, este framework también tiene que ser escalable. Esta sección mostramos los tiempos de cómputo de los filtros EHO, EHT y ENN.

En la Tabla IV podemos ver los tiempos medios de ejecución para los tres filtros en segundos. Como el nivel de ruido no es un factor que afecte a estos tiempos, mostramos la media para las cinco ejecuciones realizadas para cada dataset.

Los tiempos medidos muestran que EHO es también el método más rápido. EHO es unas diez veces más rápido que EHT y ENN. Esto se debe al uso del clasificador kNN en EHT y ENN, el cual es un algoritmo muy pesado computacionalmente. Como resultado, EHO es la opción más recomendada para tratar con el ruido en problemas Big Data.

En vista de estos resultados concluimos que:

- El uso de cualquier técnica de filtrado de ruido siempre mejora la precisión *Original* al mismo nivel de ruido.
- EHO ha sido el método que mejor rendimiento ha obtenido para ambos clasificadores. También es el más eficiente términos de tiempo.
- La estrategia de voto tiene un gran impacto en el número de instancias eliminadas en EHT.
- kNN es un método muy pesado computacionalmente, afectando a los tiempos de cómputo de EHT y ENN.

IV. Conclusiones

Este trabajo presenta el primer filtro de ruido para Big Data, donde la alta redundancia de las instancias y los problemas de alta dimensionalidad plantean nuevos retos a los algoritmos clásicos de preprocesamiento de ruido. Hemos propuesto dos algoritmos de filtrado de ruido, implementados en un framework Big Data: Apache Spark. Diferentes estrategias de particionamiento y ensembles de clasificadores han llevado a tres enfoques diferentes: un ensemble homogéneo, un ensemble heterogéneo y un enfoque de filtrado simple basado en similitud entre instancias.

La idoneidad de estas técnicas propuestas ha sido analizada utilizando varios conjuntos de datos. El ensemble homogéneo ha demostrado ser el enfoque más adecuado en la mayoría de los casos, tanto en la mejora de la precisión como en la mejora de los tiempos de ejecución. También muestra el mejor equilibrio entre instancias eliminadas y mantenidas.

El problema de ruido en Big Data es un paso crucial en la transformación de estos datos brutos en Smart Data. Con esta propuesta hemos permitido obtener Smart Data a partir de datos imperfectos. Nuestra propuesta puede resolver problemas con millones de instancias y miles de características en poco tiempo, obteniendo datasets limpios de ruido.

Financiación

Este trabajo está financiado por el Proyecto Nacional TIN2017-89517-P, y el Proyecto BigDaP-TOOLS - Ayudas Fundación BBVA a Equipos de Investigación Científica 2016.

Referencias

[1] X. Zhu and X. Wu, "Class Noise vs. Attribute Noise: A Quantitative Study," *Artificial Intelligence Review*, vol. 22, pp. 177–210, 2004.

[2] B. Frénay and M. Verleysen, "Classification in the presence of label noise: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 5, pp. 845–869, 2014.

[3] B. Zerhari, "Class noise elimination approach for large datasets based on a combination of classifiers," in *Cloud Computing Technologies and Applications (CloudTech), 2016 2nd International Conference on*. IEEE, 2016, pp. 125–130.

[4] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct 2001.

[5] M. Hamstra, H. Karau, M. Zaharia, A. Konwinski, and P. Wendell, *Learning Spark: Lightning-Fast Big Data Analytics*. O'Reilly Media, 2015.

[6] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[7] S. Ramírez-Gallego, A. Fernández, S. García, M. Chen, and F. Herrera, "Big data: Tutorial and guidelines on information and process fusion for analytics algorithms with mapreduce," *Information Fusion*, vol. 42, pp. 51 – 61, 2018.

[8] J. Lin, "Mapreduce is good enough? if all you have is a hammer, throw away everything that's not a nail!" *Big Data*, vol. 1, no. 1, pp. 28–37, 2013.

[9] D. García-Gil, S. Ramírez-Gallego, S. García, and F. Herrera, "Principal Components Analysis Random Discretization Ensemble for Big Data," *Knowledge-Based Systems*, vol. 150, pp. 166 – 174, 2018.

[10] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedins of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. San Jose, CA: USENIX, 2012, pp. 15–28.

[11] S. Verbaeten and A. Assche, "Ensemble methods for noise elimination in classification problems," in *4th International Workshop on Multiple Classifier Systems*, ser. Lecture Notes on Computer Science, vol. 2709. Springer, 2003, pp. 317–325.

[12] C. E. Brodley and M. A. Friedl, "Identifying Mislabeled Training Data," *Journal of Artificial Intelligence Research*, vol. 11, pp. 131–167, 1999.

Un análisis crítico del clasificador AkDE como *ensemble* y sus implicaciones para tratar con grandes volúmenes de datos

Jacinto Arias
Dpto. de Sistemas Informáticos
Universidad de Castilla-La Mancha
Albacete, 02071, España
jacinto.arias@uclm.es

José A. Gámez
Dpto. de Sistemas Informáticos
Universidad de Castilla-La Mancha
Albacete, 02071, España
jose.gamez@uclm.es

José M. Puerta
Dpto. de Sistemas Informáticos
Universidad de Castilla-La Mancha
Albacete, 02071, España
jose.puerta@uclm.es

***Abstract*—En clasificación supervisada el tamaño muestral condiciona enormemente el modelo a utilizar, especialmente para volúmenes de datos masivos donde la eficiencia del modelo y su potencia predictiva constituyen un equilibrio entre rendimiento y complejidad computacional. Los clasificadores basados en Redes Bayesianas permiten ajustarlo parametrizando su aprendizaje para estimar distribuciones de probabilidad cada vez más complejas. El rendimiento puede descomponerse en sesgo, que se reduce al aumentar la complejidad, y varianza, que aumenta de manera inversamente proporcional. El clasificador AkDE es uno de los ejemplos más estudiados ya que puede aprenderse en una única pasada sobre los datos y al tratarse de un modelo de tipo ensemble reduce la varianza agregando las predicciones de clasificadores individuales. En la práctica es necesario reducir su complejidad espacial y es utilizado junto a técnicas de selección de modelos basadas en la Teoría de la Información lo que implica pasadas adicionales sobre los datos. Este trabajo estudia el rendimiento de este clasificador comparándolo a otras técnicas de ensemble populares y cuestiona el impacto real de la agregación en la reducción del sesgo y la varianza. Comprobaremos empíricamente como en problemas con muestras grandes los resultados no se ajustan al modelo teórico y cómo la selección de modelos difiere del comportamiento básico de un ensemble. Los resultados obtenidos se utilizarán para proponer un modelo alternativo que sí capture las propiedades deseadas para un ensemble.**

***Index Terms*—Clasificación supervisada; Clasificadores basados en redes Bayesianas; Clasificadores ensemble.**

I. Introducción

La actual abundancia de datos y potencia computacional motiva a los investigadores en aprendizaje automático a desarrollar nuevos métodos buscando el balance entre escalabilidad y precisión. Sin embargo, no deberíamos caer en el error de conseguir este balance a cambio de producir algoritmos complejos de usar y que generen modelos difíciles de interpretar, o de otra manera, no tendrán aceptación por parte de la industria.

De hecho, gran parte de los algoritmos incluidos en los paquetes software que son referencia en la actualidad (e.g. [12]), fueron propuestos hace más de una década, pero todavía son competitivos. Parte de su éxito, sin duda se debe a sus fundamentos teóricos, pero también a su funcionamiento intuitivo y a la facilidad de interpretar y fijar los hiperparámetros necesarios para su uso. Un ejemplo claro son los clasificadores tipo *ensembles* [9] y, en particular, Random Forest (RF) [5], bien considerados tanto por investigadores como por usuarios. Desde el punto de vista de su usabilidad, el éxito recae en que un único parámetro, el número de modelos en el ensemble, sirve para controlar tanto la complejidad del modelo resultante como su nivel de precisión. P.e. en RF está demostrado que incrementar el número de árboles reduce la varianza en la clasificación manteniendo un sesgo estable. Esto garantiza que el rendimiento del clasificador mejorará o se estabilizará con un mayor número de árboles, lo que permite al usuario fijarlo basándose únicamente en la complejidad del problema abordado y/o en los recursos disponibles (tiempo y espacio).

En este trabajo nos centramos en los clasificadores basados en redes Bayesianas (BNCs), los cuáles al poderse aprender *out of core* son, en principio, excelentes candidatos para abordar grandes volúmenes de datos [1]. Además, en algunos de estos algoritmos, un único parámetro, k, controla la complejidad de los modelos aprendidos. Por ejemplo en el algoritmo kDB [14], incrementar el valor de k permite aumentar el número de dependencias permitidas y, por tanto, la complejidad de la red resultante. De forma similar, en el clasificador tipo ensemble AkDE [18], aumentar el valor de k implica también aumentar el número de dependencias, pero también el número de modelos, por lo que el orden de incremento en la complejidad es incluso superior al caso de kDB. La implicación, sin embargo, en términos del análisis en términos de sesgo y varianza es diferente: en kDB aumentar k permite reducir el sesgo a cambio de incrementar la varianza, mientras que en AkDB un aumento en k reduce la varianza pero aumenta mucho la complejidad del modelo resultante. Además, en AkDE la influencia de incrementar k se traslada en un aumento de las necesidades computacionales, sobre todo espaciales, que hacen obligatorio realizar una selección de modelos [6], [7], [11] para permitir su uso con bases de datos

Este artículo ha sido parcialmente financiado por fondos FEDER y la Agencia Estatal de Investigación (AEI/MINECO) mediante los proyectos TIN2016-77902-C3-1-P y TIN2016-82013-REDT. Jacinto Arias también está financiado por el MECD mediante la beca FPU13/00202.

de tamaño medio-grande. En este trabajo también estudiamos de forma crítica algunos de los principios en los que se basan estos procesos de selección de modelos.

Aunque el análisis basado en sesgo y varianza es habitual en otro tipo de ensembles, no es común en los ensembles de BNCs, por ello, la novedad de este trabajo reside en que proponemos evaluar los ensembles (y sus modelos constituyentes) de BNCs usando el enfoque de sesgo y varianza, estudiando su comportamiento en dominios de diferente tamaño. Los resultados que obtenemos indican que el algoritmo AkDE exhibe un comportamiento completamente diferente en muestras grandes que al considerar los benchmark habituales en aprendizaje automático, tradicionalmente formados por conjuntos de pequeño tamaño. Por otra parte, el estudio de los modelos constituyentes del ensemble, nos muestra discrepancias importantes con respecto al funcionamiento interno (y agregado) de otros ensembles típicos como RF. Estas observaciones podrían hacer a los usuarios replantearse la forma que se entiende el clasificador AkDE, especialmente en el caso de grandes conjuntos de datos. A partir del estudio, proponemos un nuevo ensemble de BNCs que si posee las propiedades esperadas en clasificadores clásicos tipo ensemble como bagging y RF.

El resto del trabajo contiene una breve descripción de los BNCs usados en el estudio, seguido de la descripción del análisis de sesgo y varianza seguido en el artículo y su aplicación al estudio del algoritmo AkDE desde la perspectiva *ensemble*. A continuación realizamos una experimentación usando tanto datos masivos como benchmarks clásicos en aprendizaje automático para evaluar los clasificadores objeto del estudio. Por último, a la luz de los resultados, proponemos un ensemble de BNCs cuyo comportamiento se asemeja más a los ensembles clásicos (bagging y RF).

II. Clasificadores basados en Redes Bayesianas

El problema de clasificación supervisada consiste en predecir la *etiqueta* $y \in \Omega_Y = \{y_1, \ldots, y_c\}$ para la variable de respuesta (o clase) Y, para un ejemplo $\mathbf{x} = (\mathbf{x_1}, \ldots, \mathbf{x_d})$ definido sobre d atributos $\{X_1, \ldots, X_d\}$. Para resolverlo, el objetivo es inducir un modelo (o clasificador) a partir de un conjunto de datos formado por m ejemplos previamente *etiquetados* $D = \{(x^{(1)}, y^1), \ldots, (x^{(m)}, y^{(m)})\}$.

El formalismo de las Redes Bayesianas (RBs) [13] usa un grafo dirigido acíclico (DAG) para representar de forma eficiente la distribución de probabilidad conjunta (DPC) definida sobre el conjunto de variables. En particular una RB factoriza la DPC mediante el producto $p(y, \mathbf{x}) = \prod_{\mathbf{i=1}}^{\mathbf{d}} \mathbf{p}(\mathbf{x_i} \mid \pi_{\mathbf{x_i}})$, donde π_{x_i} denota el conjunto de padres del atributo X_i en el DAG. Desde un punto de vista probabilístico, la etiqueta a asignar a un ejemplo $\mathbf{x}$ es la que maximiza la probabilidad a posteriori, es decir, $\arg\max_{y \in \Omega_Y} p(y \mid \mathbf{x})$.

En la práctica, los algoritmos que han demostrado un mejor rendimiento en esta tarea son los denominados BNCs [3], redes Bayesianas cuya estructura otorga un papel distinguido a la variable clase y que limitan de distinta forma el número de dependencias permitido. De ellos, el más popular por su simplicidad es Naive Bayes (NB), que asume que todas las variables predictoras son independientes dada la clase. Sin embargo, la suposición realizada resta capacidad discriminativa a NB lo que lo convierte en un clasificador con mucho sesgo. Otros modelos de BNCs se basan en imponer la estructura base de NB pero permitiendo algunas dependencias adicionales entre las variables predictoras.

A. k-dependence BN Classifier (kDB)

El algoritmo kDB [14] se basa en el uso de estimadores k-dependientes, es decir, cada variable puede depender de otros k atributos y de la clase. Requiere aprendizaje estructural y el modelo producido es una única RB.

El algoritmo propuesto en [14] usa los conceptos de información mutua (MI) e información mutua condicional (CMI) para guiar el proceso de aprendizaje estructural. Primero se ordenan las variables en función de $I(X_i; C)$. Después, a la variable i-ésima de dicho orden se le ponen como padres en el grafo la variable clase C y las k variables X_j con mayor $I(X_j, X_i|C)$, t.q. sólo se consideran las que preceden a X_i en el orden. La posterior estimación de parámetros requiere una pasada adicional por los datos cuando $k \geq 2$.

Fijando adecuadamente el valor de k se mejora el ajuste del modelo a los datos, sin embargo, aumentar k implica también obtener modelos más complejos, habitualmente con menor sesgo pero mayor varianza debido al sobreajuste. En [11] se estudia que en bases de datos grandes el riesgo de sobreajuste (e incremento de la varianza) disminuye considerablemente. El algoritmo kDB *selectivo* (SkDB) combate la complejidad del modelo eligiendo el k adecuado para cada variable [11], lo que requiere una pasada adicional por los datos.

B. Averaged k Dependence Estimators

El clasificador AkDE puede modelarse como un ensemble formado por un conjunto de BNCs $H = \{h_i(\pi_i), i = 1, \ldots, K\}$, donde el número de modelos K se fija restringiendo el espacio de los posibles modelos a una familia de clasificadores k-dependientes. En particular, cada modelo es un NB aumentado en el que además de depender de la clase, todos los atributos dependen de un conjunto fijo (el mismo para todos) de k padres, denotado por π_i. El ensemble consiste entonces en la familia *completa* de este tipo de clasificadores, por lo que $K = \binom{d}{k}$. La etapa de inferencia consiste en promediar la distribución de probabilidad de la clase a posteriori en todos los clasificadores del ensemble. Como caso particular, el clasificador A1DE (AODE) asume que todos los atributos dependen de la clase y otro atributo, conocido como *super padre* (SP). En AODE los modelos constituyentes se conocen como SPODE y en general como SPkDE. Al fijarse la estructura, no se requiere aprendizaje estructural, siendo un clasificador *out of core* real.

Al igual que en kDB, en AkDE podemos ajustar el balance sesgo-varianza variando el parámetro k, lo que permite representar desde clasificadores con mucho sesgo y poca varianza como NB (A0DE), hasta clasificadores con poco sesgo pero alta varianza, obtenidos al incrementar k y poder usar distribuciones de probabilidad de alta dimensionalidad

[18]. Sin embargo, en el caso de AkDE la influencia de incrementar k en la complejidad del modelo resultante se traslada al incremento del tamaño de las tablas de probabilidad y, sobre todo, al incremento en el número de modelos del ensemble. Por ejemplo, con 100 variables predictoras A1DE contendrá 100 modelos individuales pero A2DE contendrá 5000. Indudablemente esto representa un problema muy importante de cara a la escalabilidad del algoritmo en problemas de media-alta dimensionalidad (número de variables) por lo que se han propuesto diferentes soluciones basadas en realizar selección de modelos [6], [7], [11]. Es importante remarcar que si en kDB la selección de modelos tenía como objetivo un mejor ajuste de los datos, en AkDE la selección de modelos es obligatoria para convertirlo en un modelo usable en la práctica.

Estas técnicas de selección de modelos se basan en una aproximación híbrida entre el enfoque filter y wrapper, guiados por el uso de conceptos de Teoría de la Información, mayormente Información Mutua (MI). En particular, los algoritmos *Sample Attribute Selective AkDE (SASAkDE)* [6] y **Selective AkDE (SAkDE)** [7] han mostrado que es posible reducir la complejidad espacial, pero también (sorprendentemente) obtener mejor predicción (accuracy) que el ensemble formado por la familia completa de SPkDEs. Sin embargo, también hay algunos puntos débiles que deben ser estudiados: (1) este proceso añade pasadas adicionales por los datos, lo que supone un inconveniente en el caso de grandes muestras (e.g. big data); (2) la idea subyacente es asumir que la información mutua condicional del conjunto de super-padres dada la clase es un buen indicador del rendimiento en términos de clasificación del sub-modelo resultante. En este trabajo analizamos empíricamente esta hipótesis; y (3) volviendo a nuestra discusión inicial y las ventajas de tener hiperparámetros interpretables, indicar que este proceso de selección de modelos también añade cierta confusión al proceso, puesto que a igual k la complejidad del modelo resultante puede diferir mucho de un problema a otro.

III. Sesgo y Varianza en clasificación

Distintos estudios [2], [4] han analizado la capacidad predictiva de los clasificadores ensemble mediante la descomposición del error en términos de *sesgo* y *varianza*. Intuitivamente, un algoritmo de aprendizaje *sesgado* muestra un error persistente (similar) al entrenar con distintos conjuntos de datos, mientras que un algoritmo con alta varianza muestra un error que fluctúa entre los distintos conjuntos de datos.

Breiman [4] presenta una taxonomía de clasificadores estables (poca varianza a riesgo de tener mucho sesgo) e inestables (poco sesgo y alta varianza). Los clasificadores inestables, *en media* tienen un buen rendimiento, lo que les convierte en el marco ideal para los ensembles basados en voto por la mayoría, puesto que su uso consigue reducir la varianza [4]. Ejemplos son RF o bagging con árboles de decisión, que son algoritmos estado-del-arte en clasificación supervisada. Por otra parte, los modelos inestables son buenos candidatos para su aplicación en problemas grandes [11], ya que el impacto de la varianza se suaviza al disponer de más datos.

Sin embargo, la descomposición del error en sesgo y varianza proviene de la regresión numérica, y aunque su interpretación es también intuitiva en el caso de la clasificación, su aplicación no lo es. De hecho, mientras que en regresión promediar funciones independientes reduce la varianza sin modificar el sesgo, en clasificación promediar los modelos podría incrementar el error de clasificación [15].

Existen distintas formulaciones de la descomposición en sesgo/varianza [15], [16]. En este trabajo hemos optado por la implementación realizada en [16][1] de la descrita en [2], [4]. Las métricas utilizadas se obtienen a partir de la estabilidad de un clasificador $\mathcal{L}$ cuando se entrena y testea repetidamente sobre un número de conjuntos de datos $\mathcal{T}$. Definimos la tendencia central $C^{\circ}_{\mathcal{LT}}(\mathbf{x})$ como la clase con máxima probabilidad de ser seleccionada para un ejemplo determinado $\mathbf{x}$ por parte de todos los clasificadores aprendidos a partir de $\mathcal{T}$: $C^{\circ}_{\mathcal{LT}}(\mathbf{x}) = \arg\max_y P_{\mathcal{T}}(\mathcal{L}(\mathbf{x}) = y)$

El sesgo puede entonces medirse como el error introducido por la tendencia central del algoritmo, es decir, el error de la clasificación más frecuente, mientras que la varianza es el error introducido por la desviación de dicha tendencia central. Habitualmente se habla de estos valores en términos de *contribución* del sesgo y la varianza al error de clasificación, ya que éste puede expresarse como la suma de ambos. Para calcularlos, primero se obtiene una estimación de la tendencia central a partir de nuestra muestra $\mathcal{D}$, para lo que se ejecuta una 10x3 validación cruzada, induciendo así 30 modelos $\mathcal{L}(T^i_k)$ y el correspondiente conjunto de test f^i_k para cada una de las $i \in \{1, \ldots, 10\}$ repeticiones y $k \in \{1, 2, 3\}$ conjuntos (folds) de entrenamiento. Por tanto, se obtienen 10 predicciones independientes para cada instancia $\mathbf{x}$ y fijamos la tendencia central $C^{\circ}_{\mathcal{LT}}(\mathbf{x})$ para dicho ejemplo como la media. Formalmente, la tendencia central se obtiene como:

$$C^{\circ}_{\mathcal{LT}}(\mathbf{x}) = \arg\max_y P\left(\sum_{i=1}^{10}\sum_{k=1}^{3} 1\left[\mathbf{x} \in f^i_k \wedge \mathcal{L}(T^i_k)(\mathbf{x}) = y\right]\right)$$

La contribución del sesgo y la varianza al error se calculan para cada instancia y se agregan para todo el conjunto de datos:

$$\begin{aligned} sesgo &= P_{(\mathbf{x},y),\mathcal{T}}(\mathcal{L}(\mathcal{T})(\mathbf{x}) \neq y \wedge \mathcal{L}(\mathcal{T})(\mathbf{x}) = C^{\circ}_{\mathcal{LT}}(\mathbf{x}) \\ varianza &= P_{(\mathbf{x},y),\mathcal{T}}(\mathcal{L}(\mathcal{T})(\mathbf{x}) \neq y \wedge \mathcal{L}(\mathcal{T})(\mathbf{x}) \neq C^{\circ}_{\mathcal{LT}}(\mathbf{x}) \end{aligned}$$

IV. AkDE Bajo la Perspectiva de un Ensemble

Entre los modelos ensemble más populares encontramos el clasificador RF [5] basado en agregar el voto de árboles de decisión aplicando Bagging: Aprender cada modelo a partir de una muestra obtenida realizando bootstraping, y Random Subspaces: Seleccionar un conjunto de nodos subóptimo a la hora de estimar la partición de cada nodo del árbol. Esto incrementa la diversidad de los modelos de un modo predecible, especialmente en árboles completamente desarrollados. El estudio original demuestra que, si se incluyen modelos

[1]Esta metodología se ha usado p.e. para estudiar la propuesta inicial del clasificador AkDE [17], [18]

suficientes el sesgo del ensemble converge asintóticamente a niveles relativos a un árbol de decisión individual.

En el caso de AkDE se aprende un número finito de modelos en función de k, imponiendo una cota para la reducción de varianza y siendo imposible estabilizar el sesgo asintóticamente. Un número fijo de modelos reducirá al mismo tiempo la diversidad, deteriorando los resultados de la agregación. Además, los modelos del ensemble no se aprenden mediante una estrategia aleatoria subóptima sino que su estructura se fija heurísticamente conforme a cada super-padre. En general un clasificador k-dependiente H reduce el sesgo ajustando su estructura de la manera más fiel posible a un modelo óptimo sin restringir H', mientras que en el caso de AkDE no hay garantía de que las distribuciones de probabilidad modeladas aproximen a dicho clasificador, por lo que podemos asumir que muchos parámetros impactarán negativamente en el sesgo.

La literatura muestra que AkDE obtiene baja varianza en la práctica, sin embargo, no existe evidencia de que esto sea un resultado directo de la agregación o es inherente a los modelos individuales. Para profundizar realizaremos un experimento, para nuestro conocimiento inédito, evaluando la contribución particular al error de sesgo y varianza para una colección de ensembles y los modelos que los componen.

A. Sesgo y la Varianza en Problemas de Gran Tamaño

Un BNC con bajo sesgo es idóneo para grandes volúmenes de datos ya que obtendrá distribuciones de probabilidad mejor calibradas. Para determinar el sesgo de un modelo podemos medir su estabilidad para muestras de tamaño incremental. Para ello utilizaremos la base de datos sintética *pokerhand* [8] como benchmark, evaluando la descomposición en sesgo y varianza del error sobre 20 muestreos desde 50k instancias hasta 1M [2].

En la Figura 1 podemos ver que RF obtiene menor varianza que los modelos individuales, mientras que es máxima en el caso de árbol totalmente desarrollado. Al contrario, un árbol sin aleatorizar, es un modelo menos sesgado que el ensemble por estar compuesto por modelos que no se ajustan perfectamente a los datos. No obstante, la suma confirma que el ensemble es el de menor error y mayor estabilidad, especialmente para dominios pequeños donde la varianza es más dificil de reducir. En los modelos BNC confirmamos que los modelos con mayor sesgo como naive Bayes o kDB con $k = 1$ no mejoran conforme el tamaño de la muestra aumenta, mientras que al aumentar k la mejora es clara y constante, en presencia de suficientes datos que permitan calibrar los parámetros correctamente. Respecto a la varianza, confirmamos que los modelos sencillos son más estables, mientras que los clasificadores complejos requieren de más datos para estabilizarse.

Aunque los resultados para A1DE deberían ser superiores a los de kDB con $k = 1$ o $k = 2$, vemos que empeoran al aumentar la muestra. Si utilizamos el voto por la mayoría en lugar de la agregación numérica de probabilidades, Figura 1 bajo el nombre *a1de-majority*, vemos que la agregación de probabilidades suaviza el error cometido por los modelos de peor calidad, mientras que el voto por la mayoría los imita, implicando que si hay más modelos sesgados que acertados el ensemble lo estará también. Un tamaño de muestra mayor calibra las probabilidades de dichos modelos hacia valores extremos, convirtiendo al ensemble en un modelo equivalente al voto por la mayoría para muestras grandes.

B. Diseño Alternativo de Ensembles Basados en BNCs

Recientemente se han propuesto ensembles de BNCs alternativos, dado que en la práctica AkDE requiere aplicar selección de modelos y con ello una fase de aprendizaje adicional, podemos utilizar otros modelos que ya la realizan como kDB. Cabe destacar el clasificador kDF (k-dependent forest) [10] basado en construir un ensemble compuesto por n modelos kDB, uno para cada atributo X donde se aplica un proceso de ordenación de los atributos más sofisticado que introduce diversidad. Aunque esta propuesta supera en rendimiento a A1DE sufre de los mismos problemas descritos anteriormente ya que solo considera un número finito de modelos obtenidos de un modo no aleatorio.

Un modelo ensemble basado en agregación o voto por mayoría debería capturar ambas propiedades, para lo que proponemos una alternativa básica, el clasificador k-dependiente aleatorio (RkDB). Este ensemble tendá $h \in [1, \inf)$ modelos independientes aprendidos por una versión alterada de kDB, considerando solo un subconjunto en una proporción $\alpha \in [0, 1]$ de los padres disponibles para cada nodo en cada modelo, tomando como inspiración el particionado subóptimo que realiza RF. Así, añadimos diversidad mediante aleatoriedad controlada, preservando el sesgo del clasificador original.

C. Sesgo y Varianza en los Modelos Individuales

En nuestro segundo experimento evaluaremos el comportamiento de la descomposición del error para un benchmark clásico (véase la Tabla I) obtenido a partir del repositorio UCI [8]. Este conjunto de problemas es el utilizado en la mayoría de propuestas basadas en el clasificador AkDE. Estos modelos han obtenido siempre un buen rendimiento en este contexto, como hemos podido reproducir según muestran los resultados de la Tabla II. Comprobamos que no existe diferencia significativa entre el rendimiento de A1DE y RF y la nueva propuesta RkDB cuando k=1, no obstante para sucesivos valores de k los resultados empeoran, lo que achacamos al reducido tamaño de muestra de los problemas que conforman el benchmark. Si se observa la Figura 1 podemos comprobar que ocurre lo mismo en muestras de poco tamaño.

A la vista de que los ensembles mejoran a los modelos individuales, estudiaremos el efecto de la agregación en la descomposición en sesgo y varianza. La Figura 2 muestra la distancia entre el ensemble y los modelos individuales, podemos ver como RF reduce la varianza como esperabamos seguido de RkDB, mientras que en A1DE esta reducción es

[2]Experimentos realizados en un cluster Apache Spark de 7 nodos con procesadores hexacore Intel Xeon E5-2609v3 1.90GHz y 64GB de RAM. El software está basado en [1] y el código está disponible en http://github.com/jacintoArias/pgm2018.

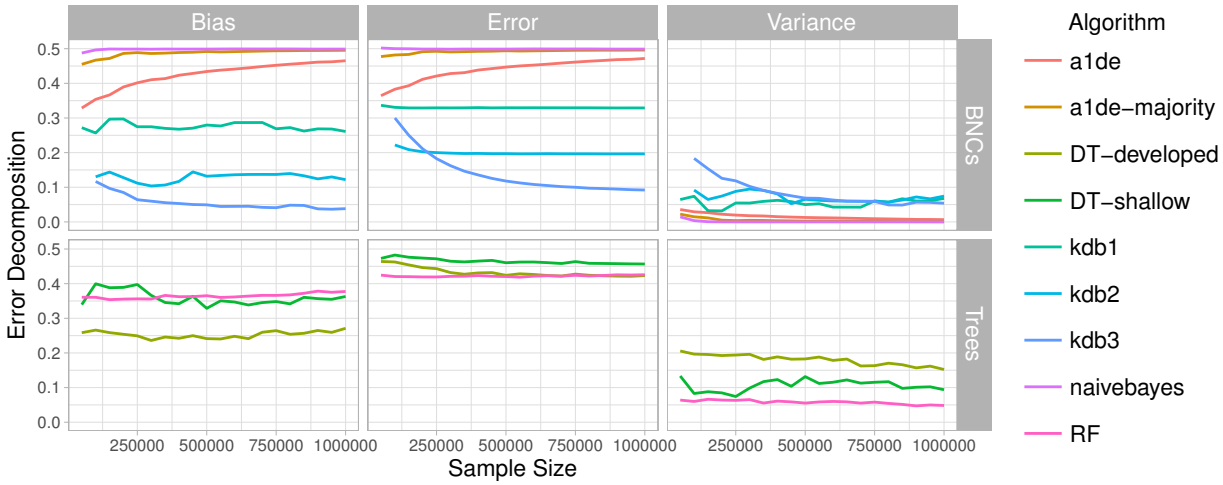


Figura 1. Evolución de sesgo y varianza para experimentos repetidos sobre muestras de mayor tamaño incrementalmente. La gráfica muestra en el eje y los valores para varianza, sesgo y error de izquierda a derecha, para la familia de modelos BNC en la parte superior y los basados en árboles en la inferior.

Problema	Casos	Atr.	Etiq.
Pokerhand	1000000	10	8
Adult	48842	15	2
Chess	28056	6	18
Letter	20000	17	26
Nursery	12960	9	5
PenDigits	10992	17	10
CensusIncome	10419	14	2
Mushrooms	8124	23	2
Musk	6598	168	2
OpticalDigits	5620	49	10
PageBlocks	5473	11	5
Spambase	4601	58	2
Hypothyroid	3772	30	4
Kr.vs.kp	3196	37	2
Splice	3190	62	3
Segment	2310	20	7
Mfeat	2000	6	2

Problema	Casos	Atr.	Etiq.
Car	1728	8	4
Contraceptive-mc	1473	10	3
German	1000	21	2
Vowel	990	14	11
Tic-Tac-Toe	958	10	2
Anneal	898	39	6
Vehicle	846	19	4
PimaIndiansDiabetes	768	9	2
BreastCancer-w	699	10	2
BalanceScale	625	5	3
CreditApproval	690	15	2
Cylinder-bands	512	39	2
Haberman	306	3	2
HouseVotes84	435	17	2
HorseColic	368	22	2
Ionosphere	351	35	2
PrimaryTumor	339	18	22

Problema	Casos	Atr.	Etiq.
Soybean	307	35	15
Haberman	306	3	2
HeartDisease-c	303	14	2
Audiology	226	70	24
New-Thyroid	215	6	3
Glass-id	214	10	3
Sonar	208	61	2
Autos	205	26	7
Wine	178	14	3
Hepatitis	155	20	2
TeachingAssistant	151	6	3
Iris	150	5	3
Promoters	106	58	2
Zoo	101	17	7
Post-operative	90	9	3
LaborNegotiations	57	17	2
LungCancer	32	57	3

Tabla I
Propiedades de los problemas utilizados en los experimentos. Los atributos continuos han sido discretizados en cuatro intervalos por igual frecuencia para los modelos BNC.

casi imperceptible, incluso la mejora parece venir de una reducción en sesgo. Observando una muestra de estas diferencias a nivel individual, Figura 3, podemos ver dos distribuciones muy diferentes, muy consistente en el caso de RF y RkDB pero casi aleatoria en A1DE. Este experimento saca a luz que A1DE al no seguir los mismos principios de diseño que otros ensembles tampoco se comporta de la manera esperada, lo que explica porque la selección de subconjuntos puede mejorar el rendimiento respecto al conjunto entero, una propiedad nada deseable y difícil de controlar en un ensemble [15].

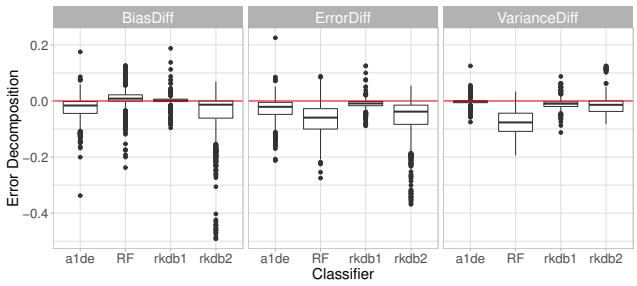


Figura 2. Distribución de las diferencias en la descomposición del error entre el modelo agregado y los diferentes modelos individuales. La línea roja en y=0 ayuda a distinguir entre diferencias positivas y negativas.

D. Sobre la Efectividad de la Selección de Modelos

Hemos observado baja consistencia en los modelos que componen un ensemble AkDE, donde unos modelos muestran propiedades deseables como bajo sesgo mientras que otros parecen ser propensos a error y tienen poco margen de varianza que reducir. Los algoritmos de selección de modelos introducidos aprovechan esta circunstancia para reducir el espacio de modelos y al mismo tiempo mejorar su rendimiento utilizando heurísticas basadas en la hipótesis de que la Información Mutua (MI) entre los atributos predictores que guían los modelos y la clase es un buen indicador de su rendimiento.

Cabe realizar entonces la pregunta de si estos métodos seleccionan entonces aquellos clasificadores propensos a mejorar en la agregación, poco sesgo y mayor varianza, o solo aquellos que minimizan el error independientemente. Para responder empíricamente hemos evaluado diversos criterios de selección de modelos de manera incremental, es decir, añadiendo cada vez un modelo adicional. Se han evaluado tres criterios wrapper: mínimo error, bias y varianza; un criterio filter basado en MI y un criterio aleatorio para establecer una base de comparación. La Tabla III muestra la suma del

Figura 3. Distribución bidimensional de sesgo (x) y varianza (y) para los modelos individuales (cruces azules) y el ensemble (punto rojo).

error sobre todos los incrementos para cada criterio y base de datos del benchmark anterior. Los resultados muestran que los mejores criterios son aquellos que minimizan el error o su componente del sesgo, mientras que la varianza, la IM y el resultado aleatorio son equivalentes. Esta evidencia reafirma nuestra hipótesis y revela que la los criterios filter basados en Información Mutua pueden no ser los más acertados.

V. Conclusiones y Trabajo Futuro

Hemos visto que los buenos resultados de AkDE en la práctica son inconsistente con la definición de un ensemble, y que la mejora en rendimiento al seleccionar modelos solo es relevante en la aplicación de técnicas wrapper que son muy costosas de computar en un escenario real. En su lugar hemos propuesto y comparado favorablemente un clasificador sencillo de tipo ensemble basado en BNCs. Los resultados muestran que es comparable a AkDE y que proporciona un recorrido de mejora que exploraremos en trabajos posteriores, especialmente en problemas de gran tamaño, donde hemos visto una clara superioridad del clasificador básico kDB en calibración de probabilidades y reducción del sesgo.

Algorithm	rank	pvalue	win	tie	loss
A1DE	2.48	-	-	-	-
RF	2.49	**9.7929e-01**	26	0	27
RkDB1	2.76	**8.7224e-01**	30	2	21
kDB1	3.58	7.1644e-03	37	0	16
RkDB2	4.75	1.8647e-09	46	0	7
kDB2	4.93	7.4299e-11	46	1	6

Tabla II

Las columnas gana, empata y pierde comparan el error obtenido por el mejor modelo (A1DE) contra los demás (p.e, A1DE gana 24 veces contra RF y pierde 17). El rango y el p-valor corresponden al test de Friedman y post-hoc (Holm) con $\alpha = 0.05$. En negrita las hipótesis rechazadas.

Criterio	Rango	p-valor	Gana	Empata	Pierde
error	1.52	-	-	-	-
bias	1.83	**3.6273e-01**	24	3	17
mi	3.75	7.8353e-11	43	0	1
variance	3.81	3.7139e-11	41	1	2
random	4.09	1.0269e-13	43	0	1

Tabla III

Comparativa de la suma del error para distintos criterios de selección en A1DE. Métricas descritas en la Tabla II.

References

[1] Jacinto Arias, Jose A. Gamez, and Jose M. Puerta. Learning distributed discrete Bayesian Network Classifiers under MapReduce with Apache Spark. *Knowledge-Based Systems*, 117:16 – 26, 2017.

[2] Eric Bauer, Ron Kohavi, Philip Chan, Salvatore Stolfo, and David Wolpert. An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants. *Machine Learning*, 36:105–139, 1999.

[3] Concha Bielza and Pedro Larrañaga. Discrete Bayesian Network Classifiers: A Survey. *ACM Comput. Surv.*, 47(1):5:1–5:43, jul 2014.

[4] Leo Breiman. Arcing classifiers. *Annals of Statistics*, 26(3):801–849, 1998.

[5] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[6] Shenglei Chen, Ana M. Martinez, Geoffrey I. Webb, and Limin Wang. Sample-Based Attribute Selective AnDE for Large Data. *IEEE Transactions on Knowledge and Data Engineering*, 29(1):172–185, 2017.

[7] Shenglei Chen, Ana M. Martínez, Geoffrey I. Webb, and Limin Wang. Selective AnDE for large data learning: a low-bias memory constrained approach. *Knowledge and Information Systems*, 50(2):475–503, 2017.

[8] Dua Dheeru and Efi Karra. UCI Machine Learning Repository, 2017.

[9] Thomas G. Dietterich. An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees. *Machine Learning*, 40:139–157, 2000.

[10] Zhiyi Duan and Limin Wang. K-Dependence Bayesian Classifier Ensemble. *Entropy*, 19(12), 2017.

[11] A M Martínez, G I Webb, S Chen, and N A Zaidi. Scalable learning of Bayesian network classifiers. *Journal of Machine Learning Research*, 17:1–30, 2016.

[12] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. MLlib: Machine Learning in Apache Spark. *Journal of Machine Learning Research*, 17(34):1–7, 2016.

[13] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. 2014.

[14] Mehran Sahami. Learning Limited Dependence Bayesian Classifiers. In *Proc. of the 2nd Int. Conf. on Knowledge Discovery and Data Mining*, KDD'96, pages 335–338. AAAI Press, 1996.

[15] Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26(5):1651–1686, 1998.

[16] Geoffrey I. Webb. MultiBoosting: a technique for combining boosting and wagging. *Machine Learning*, 40(2):159–196, 2000.

[17] Geoffrey I. Webb, Janice R. Boughton, and Zhihai Wang. Not so naive Bayes: Aggregating one-dependence estimators. *Machine Learning*, 58(1):5–24, 2005.

[18] Geoffrey I. Webb, Janice R. Boughton, Fei Zheng, Kai Ming Ting, and Houssam Salem. Learning by extrapolation from marginal to full-multivariate probability distributions: Decreasingly naive Bayesian classification. *Machine Learning*, 86(2):233–272, oct 2012.

Un enfoque aproximado para acelerar el algoritmo de clasificación Fuzzy kNN para Big Data

Jesus Maillo, Julián Luengo, Salvador García, Francisco Herrera
Dept. de Ciencias de la Computación e Inteligencía Artificial
Universidad de Granada. 18071, Granada, España
Email: {jesusmh, julianlm, salvagl, herrera}@decsai.ugr.es

Isaac Triguero
School of Computer Science
University of Nottingham, Jubilee Campus
Nottingham NG8 1BB, United Kingdom
Email: Isaac.Triguero@nottingham.ac.uk

***Resumen*—El clasificador difuso de los k vecinos más cercanos (Fuzzy k Nearest Neighbor - Fuzzy kNN) es reconocido por su eficacia en problemas de aprendizaje supervisado. El algoritmo kNN clasifica comparando una nueva instancia con los datos etiquetados mediante una función de similitud. Su versión difusa se compone de dos etapas. La primera etapa calcula el grado de pertenencia a cada clase para cada instancia del problema, con el objetivo de detectar con mayor precisión las fronteras entre clases. La segunda etapa clasifica siguiendo el mismo esquema que el algoritmo kNN, aunque haciendo uso del grado de pertenencia a clase previamente calculado. La propuesta existente de Fuzzy kNN para afrontar conjuntos de datos muy grandes no es completamente escalable, debido a que replica el comportamiento de Fuzzy kNN original. Con el objetivo de agilizar este algoritmo, en esta contribución se propone un enfoque aproximado que acelera los tiempos de ejecución sin perder calidad en la clasificación, mediante el uso de árboles e implementado en Spark. En la experimentación realizada se comparan varios enfoques de Fuzzy kNN para big data, con conjuntos de datos de hasta 11 millones de instancias. Los resultados muestran una mejora en tiempo de ejecución y precisión con respecto al modelo de la literatura.**

***Palabras Clave*—Conjuntos difusos, K-vecinos más cercanos, Clasificación, MapReduce, Apache Spark, Big Data**

I. Introducción

El algoritmo de los k vecinos más cercanos (*k Nearest Neighbor* - kNN) [1] es catalogado como clasificador basado en instancias. Para clasificar, compara las instancias no vistas con aquellas etiquetadas del conjunto de entrenamiento utilizando una función de similitud. Generalmente la similitud es medida mediante una función de distancia (Euclídea o Manhattan). A pesar de su simplicidad, el clasificador kNN es uno de los diez algoritmos de clasificación más relevantes [2].

Sin embargo, kNN suponiendo todos los vecinos igual de importantes en la clasificación, considerando que las fronteras entre clases están perfectamente definidas. Existen diferentes propuestas basadas en la teoría de los conjuntos difusos que abordan este problema. En [3], el algoritmo clásico Fuzzy kNN [4] destaca como uno de los enfoques más eficaces. Fuzzy kNN se compone de dos etapas: cálculo de grado de pertenencia y clasificación. La primera etapa, cambia la etiqueta de la clase por un grado de pertenencia a cada clase, de acuerdo a las instancias más cercanas. La segunda etapa, calcula el kNN con la información del grado de pertenencia. Así, se consigue detectar con mayor precisión las fronteras, viéndose menos afectado por el ruido y mejorando al kNN en la mayoría de los problemas de clasificación.

En el ámbito del big data, tanto el algoritmo kNN como Fuzzy kNN encuentran problemas para manejar grandes conjuntos de datos con respecto al tiempo de ejecución y al consumo de memoria. Existe una propuesta exacta del algoritmo kNN para abordar problemas big data y se denomina *k Nearest Neighbor - Iterative Spark* (kNN-IS) [5]. Además de esta versión exacta, también existen versiones aproximadas que reducen los tiempos de ejecución: Metric-Tree [6] y Spill-Tree. En [7], los autores estudian estos modelos y proponen el modelo Hybrid Spill-Tree (HS) como hibridación de ambos con el objetivo de mejorar el tiempo de ejecución sin afectar a los resultados debido a la redundancia de datos implícita en big data. Por otro lado, existe una solución exacta para aplicar Fuzzy kNN en big data, *Global Exact Fuzzy k Nearest Neighbors* (GE-FkNN) [8]. Aunque es capaz de escalar hasta conjuntos de datos muy grandes, los tiempos de ejecución de la primera etapa son sustancialmente altos, provocando un cuello de botella. Sin embargo, no se han aplicado enfoques aproximados para el algoritmos Fuzzy kNN en big data.

En este trabajo se propone una variación aproximada del algoritmo Fuzzy kNN desarrollado en Spark. Para aliviar el cuello de botella de la primera etapa del algoritmo Fuzzy kNN, se estudiarán dos enfoques: local y global. El enfoque local consiste en dividir el conjunto de datos en diferentes partes y calcular el grado de pertenencia a clase internamente en cada partición (operación map), sin considerar otras particiones. El enfoque global está basado en el modelo HS. Éste genera un árbol con las instancias del conjunto de entrenamiento y lo distribuye entre todos los nodos de cómputo, considerando todas las instancias para el cálculo del grado de pertenencia. La segunda etapa es responsable de la clasificación y común para las dos propuestas. En ésta, el esquema de trabajo es igual a la versión global de cálculo de pertenencia, pero considera la pertenencia previamente calculada clase para predecir. Para comprobar el rendimiento de este modelo, los experimentos realizados se han llevado a cabo en 4 conjuntos de datos con hasta 11 millones de instancias. El estudio experimental analiza la precisión y tiempo de ejecución realizando una comparativa con otros algoritmos de la literatura.

El documento está estructurado de la siguiente manera. La

sección II introduce el estado del arte en los algoritmos Fuzzy kNN y Hybrid Spill-Tree. La sección III detalla la propuesta aproximada del algoritmo Fuzzy kNN. El estudio experimental se describe en la Sección IV. La sección V concluye el documento y esboza el trabajo futuro.

II. Preliminares

Esta sección proporciona el conocimiento básico del algoritmo Fuzzy kNN (Sección II-A), el Hybrid Spill-Tree (Sección II-B) y las tecnologías big data utilizadas (Sección II-C).

II-A. k vecinos más cercanos difuso y su complejidad

El algoritmo Fuzzy kNN [4] se propone como una mejora del algoritmo kNN, llegando a mejorarlo en términos de precisión para la mayoría de los problemas de clasificación. Necesita una etapa de pre-cómputo en el conjunto de entrenamiento, que calcula el grado de pertenencia a la clase. Posteriormente, calcula kNN para cada instancia no vista y decide la clase predicha con el grado de pertenencia. Una notación formal para el algoritmo Fuzzy kNN es la siguiente:

Sea CE un Conjunto de Entrenamiento y CP un Conjunto de Prueba, compuestos de **n** y **t** instancias respectivamente. Cada instancia $\mathbf{x}_i$ es un vector $(\mathbf{x}_{i1}, \mathbf{x}_{i2}, \mathbf{x}_{i3}, \ldots, \mathbf{x}_{ij})$, donde $\mathbf{x}_{ij}$ es el valor de la i-ésima instancia y j-ésima característica. Para cada instancia de CE se conoce su clase ω. Sin embargo, para las instancias de CP la clase es desconocida.

Fuzzy kNN posee dos etapas: cálculo de pertenencia y clasificación. La primera etapa calcula los k vecinos más cercanos para cada instancia del CE, manteniendo un esquema *leave-one-out* seleccionando las k instancias con una distancia menor. Finalmente, calcula el grado de pertenencia de acuerdo a la Ecuación 1. El resultado de la primera etapa será CE modificando la etiqueta de clase ω, por un vector de pertenencia a cada clase $(\omega_1, \omega_2, \ldots, \omega_l)$ donde l es el número de clases. Este nuevo conjunto se llamará Conjunto de Entrenamiento Difuso, CED.

$$u_j(x) = \begin{cases} 0{,}51 + (n_j/k_{pert}) \cdot 0{,}49 & if \quad j = i \\ (n_j/k_{pert}) \cdot 0{,}49 & if \quad j \neq i \end{cases} \tag{1}$$

La etapa de clasificación calcula para cada instancia de CP sus kNN como se describe en la primera etapa, para cada instancias de CP en el CED y se obtiene el grado de pertenencia a clase. Posteriormente, obtiene la clase resultante de acuerdo a la Ecuación 2.

$$u_i(x) = \frac{\sum_{j=1}^{K} u_{ij}(1/|x - x_j|^{2/(m-1)})}{\sum_{j=1}^{K}(1/|x - x_j|^{2/(m-1)})} \tag{2}$$

La etapa extra respecto a kNN y el aumento de la complejidad computacional generan dos problemas big data:

- Tiempo de ejecución: La complejidad de calcular kNN para una instancia es $mathcalO(n \cdot c)$, donde n es el número de instancias de CE y c el número de características. Para más de un vecino, aumenta a $\mathcal{O}(n \cdot log(N))$. Además, Fuzzy kNN tiene una etapa extra de cómputo para el cálculo de grado de pertenencia.
- Consumo de memoria: Para acelerar el cálculo, se requiere disponer de los conjuntos CE y CP almacenados en memoria principal. Cuando ambos conjuntos son grandes, es fácil exceder la memoria principal disponible.

Para aliviar estas dificultades, trabajamos en el diseño de un modelo aproximado basado en Hybrid Spill-Tree desarrollado bajo las tecnologías big data de MapReduce y Spark.

II-B. Hybrid Spill-Tree

El algoritmo Hybrid Spill-Tree (HS) [9] es una propuesta aproximada para calcular el algoritmo kNN de forma distribuida. HS se compone de dos estructuras: Metric-Tree y Spill-Tree. La estructura de datos Metric-Tree (MT) organiza el conjunto de datos en una jerarquía espacial. Es un árbol binario cuya raíz contiene todos los elementos, y donde cada hijo representa un subconjunto de elementos. La Figura 1 ilustra como dividir los elementos entre los dos hijos, seleccionando cada hijo como las instancias más lejanas posibles (representados por ◯). Los nodos hijos de un MT no contienen instancias repetidas (representadas por $+$). El árbol tendrá una profundidad de $\mathcal{O}log(N))$. Para realizar la búsqueda de la instancia más cercana, se conserva al candidato con menor distancia C y su distancia d. Si la distancia a una rama es superior a d, la poda y continúa la búsqueda. Una vez no existe una rama del árbol con distancia menor que d, se finaliza la búsqueda y se devuelve C y d. Sin embargo, invierte tiempo y cómputo en asegurar que C es el más cercano, volviendo atrás en el árbol si fuese necesario. Con el objetivo de reducir el tiempo de búsqueda emerge Spill-Tree.

La estructura de datos Spill-Tree (SP) es una variación de MT. La diferencia principal consiste en permitir instancias compartidas entre los nodos hijos. La Figura 1 presenta cómo se dividen los datos con el mismo procedimiento que MT, permitiendo un conjunto de instancias duplicadas en los nodos hijos. El área de solapamiento es dependiente del parámetro τ. Cuando τ es 0, no se comparten instancias, sería un MT. Si τ es demasiado alto, el solapamiento es alto y la profundidad del árbol tiende a infinito. SP reduce los tiempos de búsqueda respecto a MT sacrificando la *vuelta atrás* en el árbol para comprobar que C es el más cercano. Debido al solapamiento entre hijos, la instancia que encuentra, aunque aproximada, es muy representativa del problema.

HS aparece con el objetivo de obtener un balance entre precisión y tiempo de ejecución. Para ello, fusiona los modelos MT y SP. Para construir la estructura hibrida de HS, se construye un SP, y si el número de instancias en el área de solapamiento es menor que el Umbral de Equilibrio (UE), se mantiene como SP. Si las instancias repetidas superan el UE, se reconstruye como MT. Para calcular kNN sobre esta estructura, se realizará *vuelta atrás* para asegurar el más cercano si la rama es MT, mientras que los nodos SP no harán esta comprobación. Destacar la implementación

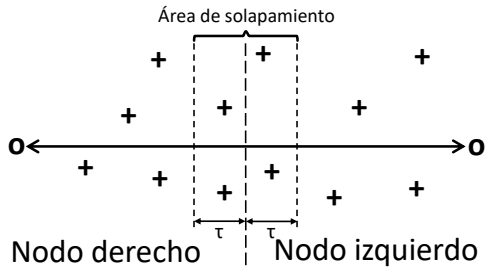


Figura 1: Construcción de Metric-Tree y Spill-Tree

disponible en el repositorio de software libre spark-package[1], que es el punto de partida en la etapa de desarrollo del presente trabajo.

II-C. Paradigma de programación MapReduce: Spark

Para el desarrollo del algoritmo propuesto en este trabajo se utilizará el paradigma de programación MapReduce [10]. MapReduce tiene como objetivo procesar grandes conjuntos de datos a través de la distribución del almacenamiento de datos y ejecución a través de un conjunto de máquinas.

La implementación de MapReduce seleccionada es Apache Spark [11]. Spark paraleliza el cálculo de forma transparente a través de una estructura de datos distribuidos llamada *Resilient Distributed Datasets* (RDD). Los RDDs permiten persistir y reutilizar estructuras de datos almacenadas en memoria principal. Adicionalmente, Spark fue desarrollado para cooperar con el sistema de archivos distribuidos de Hadoop[2] (Hadoop Distributed File System - HDFS). Con esta configuración, se puede aprovechar la división de instancias, la tolerancia a fallos y la comunicación de trabajos que proporciona Spark.

Spark incluye una librería de aprendizaje automático llamada MLlib[3]. Dispone de multitud de algoritmos de aprendizaje automático y técnicas estadísticas de diferentes áreas como clasificación, regresión o preprocesamiento de datos.

III. Fuzzy kNN acelerado para Big data

En esta sección se presenta una propuesta aproximada y distribuida del algoritmo Fuzzy kNN basado en el método HS para tratar problemas big data implementado en Spark. La presente propuesta tiene las mismas dos etapas que el modelo Fuzzy kNN: grado de pertenencia a clase (Sección III-A) y clasificación (Sección III-B).

III-A. Etapa de cálculo de grado de pertenencia a clase

Esta sección explica el flujo de trabajo de dos enfoques aproximados para calcular el grado de pertenencia a clase: Local basado en kNN (Sección III-A1) y global basado en HS (Sección III-A2) . El resultado de ambos enfoques es modificar la etiqueta de clase del Conjunto de Entrenamiento (CE) por un vector de pertenencia a clase, formando el Conjunto de Entrenamiento Difuso (CED)

[1]Hybrid Spill-Tree. https://spark-packages.org/package/saurfang/spark-knn
[2]Apache Hadoop. Web: http://hadoop.apache.org/
[3]Machine Learning Library for Spark. Web: http://spark.apache.org/mllib/

Algorithm 1 Etapa de cálculo de grado de pertenencia - Local

Require: CE, k, $\#Maps$
1: $CEP \leftarrow$ repartition(CE, #Maps)
2: $CEPD \leftarrow$ mapPartition(calcularPertenencia(CEP, k))
3: $CED \leftarrow$ unir($CEPD$)
4: **return** CED
5: **COMIENZA calcularPertenencia**
6: **for** y: CEP_i **do**
7: $Vecinos_y \leftarrow$ calcularkNNLocal ($modelo$, k, y)
8: $pertenencia_y \leftarrow$ calcularPertenencia ($Vecinos_y$)
9: $CEPD \leftarrow$ unir(y, $pertenencia_y$)
10: **end for**
11: **return** $CEPD$
12: **FIN calcularPertenencia**

Algorithm 2 Etapa de cálculo de grado de pertenencia - Global

Require: CE, k
1: $muestreas \leftarrow$ muestrear(CE,0, 2 %)
2: $arbolSuperior \leftarrow$ construirMT($muestreas$)
3: $\tau \leftarrow$ estimarTau($TopTree$)
4: $arbol \leftarrow$ repartir(CE, $arbolSuperior$, τ, $UE = 70\,\%$)
5: $modelo \leftarrow$ (difundir($arbolSuperior$),arbol)
6: **for** y: CE **do**
7: $Vecinos_y \leftarrow$ calcularkNN ($modelo$, k, y)
8: $pertenencia_y \leftarrow$ calcularPertenencia ($Vecinos_y$)
9: $resultado_y \leftarrow$ unir(y, $pertenencia_y$)
10: **end for**
11: **return** $resultado$

III-A1. Enfoque local: El enfoque local conjuntamente con la etapa de clasificación recibe el nombre *Local Hybrid Spill-Tree Fuzzy kNN* (L-HSFkNN). El Algoritmo 1 muestra los pasos y operaciones en Spark para el cálculo del grado de pertenencia. Comienza leyendo el CE de HDFS y es dividido en $\#Maps$ partes. Posteriormente, utiliza una operación mapPartition de Spark para calcular de forma distribuida el grado de pertenencia a clase para cada partición CEP_i. Para cada instancia y de cada partición CEP_i, calcula kNN y finalmente obtiene el grado de pertenencia aplicando la Ecuación 1. Una vez obtenida la pertenencia para cada partición, se agrupan los resultados y forman el CED, que será la entrada de la etapa de clasificación.

III-A2. Enfoque global basado en HS: El enfoque global sumado a la etapa de clasificación recibe el nombre *Global Approximate Hybrid Spill-Tree Fuzzy kNN* (GA-HSFkNN). El Algoritmo 1 muestra los pasos y operaciones en Spark para el cálculo del grado de pertenencia.

El algoritmo 2 muestra los pasos de la etapa de cálculo de pertenencia. Las líneas 1-5 corresponden a la etapa de ajuste del modelo, y las restantes al cálculo del grado de pertenencia.

La etapa de ajuste del modelo comienza leyendo el CE de HDFS. En primer lugar, toma un submuestreo aleatorio para construir un MT como se describe en la Sección II-B (Los autores recomiendan un 0,2 %). Este MT recibe el nombre de árbol superior (AS) y es usado para estimar el valor del parámetro τ y particionar todo el CE. La estimación de τ es la distancia media entre las instancias. Para acelerar este cálculo, se realiza con las instancias del AS.

El siguiente paso es repartir el CE. Para ello, a partir

Algorithm 3 Calcular kNN

Require: $modelo$, k, x
1: $Indices \leftarrow$ x.flatMap (buscaIndices($modelo.arbol$))
2: $Vecinos \leftarrow$ consulta($modelo.arbol$, $Indices$, k)
3: **return** $Vecinos$
4:
5: **COMIENZA buscaIndices**
6: $distIzq \leftarrow$ nodoIzq.dist(x)
7: $distDch \leftarrow$ nodoDch.dist(x)
8: **if** $nodo! = HOJA$ **then**
9: **if** $distIzq < distDch$ **then**
10: buscaIndices($nodoIzq$, ID)
11: **else**
12: buscaIndices($nodoDch$, $ID + hijoIzq$)
13: **end if**
14: **else**
15: **return** $Indices$
16: **end if**
17: **FIN buscaIndices**

Algorithm 4 Etapa de clasificación

Require: CED, CP, k
1: $muestreas \leftarrow$ muestrear(CED,0, 2 %)
2: $arbolSuperior \leftarrow$ construirMT($muestreas$)
3: $\tau \leftarrow$ estimarTau($TopTree$)
4: $arbol \leftarrow$ repartir(CED, $arbolSuperior$, τ, $UE = 70\,\%$)
5: $modelo \leftarrow$ (difundir($arbolSuperior$),arbol)
6: **for** y: CE **do**
7: $VecinosP_y \leftarrow$ calcularFuzzykNN ($modelo$, k, y)
8: $prediccion_y \leftarrow$ calcularPredicción ($Vecinos_y$)
9: **end for**
10: **return** $prediccion_y$

del AS se distribuyen las instancias en el espacio. El valor de τ define el área de solapamiento. Comienza construyendo un SP, y comprueba si el número de instancias en el área de solapamiento es menor que el 70 %. En otro caso, se reconstruye un MT. Al realizar la búsqueda, aquellas ramas SP realizan una búsqueda más rápida al no tener que volver atrás en el árbol. Sin embargo, las construidas como MT realizan *vuelta atrás* para asegurar el más cercano. La etapa de construcción del modelo termina distribuyendo el AS y el árbol asociado al CE.

La etapa de cálculo de grado de pertenencia se muestra en las líneas 6-10. Para cada instancia de CE, se calcula kNN siguiendo el modelo generado. El Algoritmo 3 describe como se realiza la búsqueda con operaciones nativas de Spark. Mediante una operación flatMap, se computan y obtienen los índices de las instancias más cercanas del CE. Para ello, se calcula la distancia hasta el nodo derecho e izquierdo, y continúa por el nodo con una distancia menor. Cuando alcanza un nodo hoja, devuelve el índice de la instancia seleccionada.

Dados los vecinos, se calcula el grado de pertenencia siguiendo la Ecuación 1. (Línea 8). El resultado de esta fase es el CED, pasando a ser la entrada de la etapa de clasificación.

III-B. Clasificación con Hybrid Spill-Tree

Esta sección describe la etapa de clasificación, la cual recibe como entrada el CED calculado en la etapa previa. La etapa de clasificación está basada en HS y sigue la misma estructura que la aproximación global de la primera etapa (Sección III-A2). Tiene dos fases diferenciadas: creación del modelo y clasificación. La primera construye el árbol y divide las instancias entre los nodos de cómputo. La segunda busca las k instancias más cercanas de CED calculado en la primera etapa, y devuelve como salida la clase predicha de acuerdo al vector de grado de pertenencia.

El Algoritmo 4 muestra los pasos de la etapa de clasificación. Se especificarán las diferencias con respecto a la sección III-A2, ya que el esquema principal es el mismo y sólo se ven afectados pequeños detalles de la estructura de datos.

La primera diferencia se encuentra en los conjuntos de datos de entrada. En este caso, se utilizarán CED y CP. La fase de ajuste del modelo no se ve afectada pues no se modifican las características, manteniendo las distancias de las instancias. Así, se construye el modelo con la misma metodología.

El cálculo de Fuzzy kNN se hace de la misma manera, con la diferencia de que en lugar de devolver la etiqueta de clase para cada vecino, se devuelve el vector de grado de pertenencia a la clase (Línea 7). La línea 8 calcula la clase predicha aplicando la Ecuación 2. El resultado final es la clase prevista para cada instancia de CP.

IV. Estudio experimental

En esta sección se presentan las cuestiones planteadas en el estudio experimental. La Sección IV-A presenta los algoritmos de comparación utilizados para la experimentación. La Sección IV-B determina el marco de los experimentos. Finalmente, la Sección IV-C expone y analiza los resultados obtenidos.

IV-A. Algoritmos de comparación

Esta sección expone los algoritmos de comparación utilizados en los experimentos y sus acrónimos. Se compara con otras propuestas de Fuzzy kNN y sus análogos no *Fuzzy* siguiendo dos enfoques: local y global. El enfoque local divide las instancias y los distribuye entre los nodos de cómputo, ejecutando de forma independiente en cada partición, obteniendo resultados aproximados. El enfoque global si considera todas las instancias, aunque realice el cómputo de forma distribuida y a su vez presenta dos posibilidades: exacto y aproximado. El exacto invierte cálculo en asegurar que el resultado es igual que la variante secuencial. El aproximado, no asegura el mismo resultado acelerando el tiempo de ejecución.

Algoritmos utilizados en la experimentación para realizar la comparativa:

- *Global Exact Fuzzy kNN (GE-FkNN)* [8]: modelo exacto del algoritmo Fuzzy kNN para abordar problemas big data, obteniendo los mismos resultados que el Fuzzy kNN original. Sus dos etapas son globales y exactas.
- *Local Fuzzy kNN (L-FkNN)*: propuesta desarrollada del algoritmo Fuzzy kNN. La primera etapa es la descrita en la Sección III-A1 y su segunda etapa sería global y exacta, idéntica a la de GE-FkNN.

- *k Nearest Neighbor - Iterative Spark (kNN-IS)* [5]: propuesta exacta de kNN para abordar problemas big data, obteniendo los mismos resultados que el kNN original.
- *Hybrid Spill-Tree kNN (HS-kNN)* [9]: propuesta aproximada de kNN para big data. Aunque aproximada, considera todas las instancias en la búsqueda.

IV-B. Marco experimental

Para el estudio experimental, se han seleccionado cuatro conjuntos de datos de un elevado número de instancias. El conjunto de datos ECBDL14 se extrae de la competición [12]. A pesar de que tiene una relación de desbalanceo superior a 45, hemos seleccionado este conjunto de datos por su número relativamente alto de características. Sin embargo, en este documento no abordamos el problema de la clasificación desequilibrada, por lo que se ha submuestreado dejándolo en una relación de desbalanceo de dos. Los otros tres conjuntos han sido extraídos del repositorio UCI [13]. La Tabla I presenta el número de instancias, características y clases ($\#\omega$). Se seguirá el esquema de validación cruzada en 5 particiones.

Tabla I: Descripción de los conjuntos de datos

Conjunto de datos	#Instancias	# Características	$\#\omega$
Poker	1.025.010	10	10
ECBDL14	2.063.187	631	2
Susy	5.000.000	18	2
Higgs	11.000.000	28	2

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

- *Precisión:* Contabiliza el número de clasificaciones correctas en relación con el número total de instancias.
- *Tiempo de ejecución:* Tiempo consumido considerando lecturas y comunicaciones por red de Spark.

El parámetro más conocido para el algoritmo original Fuzzy kNN es el número de vecinos (k). k podría ser diferente en cada etapa, pero mantendremos ambos iguales por simplicidad. En nuestros experimentos, el parámetro k está ajustado a 3, 5 y 7. Se necesita un parámetro adicional debido a su comportamiento distribuido. Este es el número de particiones del conjunto de entrenamiento. Para exprimir la componente distribuida, se utilizará el máximo posible para la configuración del cluster, 256 operaciones maps/particiones.

Los algoritmos basados en HS tienen dos parámetros adicionales: el número de instancias para construir el árbol principal AP y el umbral de equilibrio UE. Los valores recomendados por los autores son AP igual al 0,2 % y UE igual al 70 %.

Todos los experimentos se han realizado en un cluster compuesto por 14 nodos de cálculo gestionados por un nodo maestro. Todos los nodos tienen la misma configuración. 2 procesadores Intel Xeon CPU E5-2620, 6 núcleos (12 hilos) por procesador, 2 GHz y 64 GB de RAM. Red Infiniband 40Gb/s. Con la configuración actual, se disponen de 256 tareas map como máximo. La versión de Spark es 2.2.1.

IV-C. Resultados obtenidos y análisis

Esta sección presenta una comparativa de los modelos propuestos y los detallados en la Sección IV-A, analizando el tiempo de ejecución y la precisión de cada uno de ellos. Para realizar un análisis detallado de los algoritmos, se utilizarán los 4 conjuntos de datos expuestos en la Tabla I. La Sección IV-C1 presenta el tiempo de ejecución y precisión. La Sección IV-C2 compara los algoritmos propuestos con otros algoritmos de la literatura.

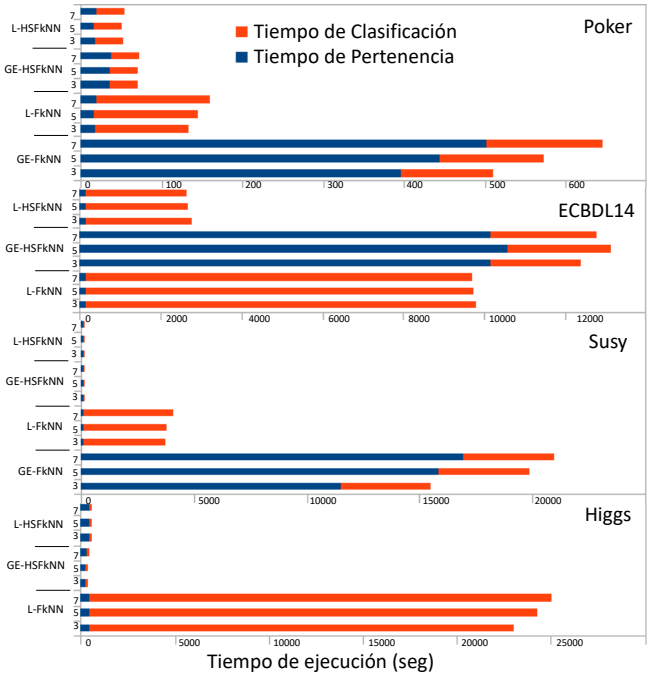


Figura 2: Tiempo de ejecución para cada conjunto y algoritmo

IV-C1. Estudio de tiempo y precisión: Esta sección presenta y estudia el tiempo de ejecución y precisión obtenida para los 4 conjuntos de datos y los diferentes algoritmos.

La Tabla II muestra una comparación de precisión entre los algoritmos en relación al número de vecinos (k) y para todos los conjuntos de datos.

Tabla II: Comparativa entre algoritmos - Precisión

Algoritmo	k	Poker	ECBDL14	Susy	Higgs
GE-FkNN	3	0,5257	-	0,7301	-
	5	0,5316	-	0,7306	-
	7	0,5338	-	0,7268	-
L-FkNN	3	0,5218	0,7636	0,7178	0,5936
	5	0,5243	0,7492	0,7190	0,5953
	7	0,5243	0,7423	0,7179	0,5959
GA-HSFkNN	3	0,5233	**0,8051**	0,7302	0,5968
	5	0,5371	0,8050	0,7457	0,6081
	7	0,5459	0,8014	**0,7510**	0,6162
L-HSFkNN	3	0,5324	0,8035	0,7320	0,6038
	5	0,5432	0,8027	0,7438	0,6139
	3	**0,5487**	0,7969	0,7471	**0,6198**

La Figura 2 muestra el tiempo de la etapa de cálculo de pertenencia y el tiempo de la etapa de clasificación en segundos, para cada conjunto de datos, algoritmo y los valores de k igual a 3, 5 y 7.

De acuerdo con la tabla y figura presentadas, se observa:

Tabla III: Comparativa contra propuestas kNN

Conjunto de datos	Algoritmo	k	Tiempo Total	Precisión
Poker	kNN-IS	3	113,2370	0,4758
		5	123,8311	0,4952
		7	136,5830	0,4937
	HS-kNN	3	32,9661	0,5201
		5	33,4674	0,5305
		7	35,5575	0,5369
	GA-HSFkNN	7	72,1978	0,5459
	L-HSFkNN	7	54,3435	**0,5487**
ECBDL14	kNN-IS	3	27121,0583	0,7833
		5	28673,7015	0,7797
		7	28918,2992	0,7683
	HS-kNN	3	3151,6242	0,8020
		5	2608,1722	0,8017
		7	2691,5595	0,7986
	GA-HSFkNN	3	12413,8265	**0,8051**
	L-HSFkNN	3	2787,1230	0,8035
Susy	kNN-IS	3	2615,0150	0,6675
		5	2273,6377	0,6784
		7	2372,4100	0,6861
	HS-kNN	3	128,1860	0,7223
		5	131,0210	0,7360
		7	133,2428	0,7431
	GA-HSFkNN	7	133,2568	**0,7510**
	L-HSFkNN	7	146,2076	0,7471
Higgs	kNN-IS	3	10262,9178	0,5454
		5	13446,0519	0,5458
		7	14285,6442	0,5559
	HS-kNN	3	641,4434	0,5885
		5	651,1573	0,5936
		7	664,9204	0,5981
	GA-HSFkNN	7	429,3507	0,6162
	L-HSFkNN	7	608,2571	**0,6198**

- El valor de k no afecta significativamente a los tiempos de ejecución en ninguna de las etapas.
- El algoritmo GE-FkNN encuentra su límite de escalabilidad en su primera etapa, no llegando a ejecutar para los datasets ECBDL14 y Higgs.
- Centrándonos en los tiempos de ejecución, el modelo L-FkNN escala mejor en la primera etapa y se ve menos afectado por el número de características. El algoritmo GA-HSFkNN consigue mejores tiempos en la etapa de clasificación. Así, los mejores tiempos de ejecución son obtenidos por el algoritmo L-HSFkNN, que toma la primera etapa del algoritmo L-FkNN y la etapa de clasificación de GA-HSFkNN.
- En precisión los ganadores son GA-HSFkNN y L-HSFkNN con diferencias poco significativas. Teniendo en cuenta que los tiempos de ejecución totales son menores para L-HSFkNN, es nuestra propuesta más escalable y de mayor calidad.

IV-C2. Comparativa con versiones kNN big data: En esta sección se comparan los modelos basados en conjuntos difuso con las versiones clásicas de kNN para big data. La Tabla III muestra una comparativa entre el mejor resultado obtenido por los dos algoritmos propuestos y las dos alternativas no *Fuzzy*, explorando los mismos valores de k. Se muestra para cada conjunto de datos y algoritmo, el tiempo total de ejecución y el valor de k utilizado asociado a la precisión obtenida.

De acuerdo con la tabla presentada, se puede observar que los mejores resultados los obtienen las versiones *Fuzzy*. Además, kNN-IS obtiene resultados significativamente peores que los demás algoritmos. Aunque HS-kNN mejora respecto al algoritmo kNN-IS, siempre queda por debajo de los modelos propuestos, sin verse incrementado en exceso el tiempo de ejecución debido a la optimización realizada en la etapa de clasificación del algoritmo GA-HSFkNN.

V. Conclusiones y trabajo futuro

En esta contribución se ha propuesto un enfoque MapReduce para acelerar el algoritmo Fuzzy kNN en problemas Big data. Gracias al diseño realizado y al uso de tecnologías big data, se consigue ejecutar con conjuntos de datos muy grandes, que de otra forma sería inviable. Los experimentos realizados comparan la propuesta con otros enfoques de comportamiento exacto y aproximado para conseguir un equilibrio entre eficiencia y precisión. Asimismo, se ha comparado con las versiones no *Fuzzy* para estudiar la mejora del modelo propuesto. El algoritmo L-HSFkNN demuestra una escalabilidad muy elevada, así como resultados en precisión de alta calidad. Como trabajo futuro, se plantea adaptar el modelo desarrollado a otros problemas de minería de datos como regresión, o aprendizaje semi-supervisado.

Agradecimientos

Este trabajo se ha sustentado por el proyecto de investigación TIN2017-89517-P. J. Maillo disfruta de una beca FPU del Ministerio de Educación de España.

Referencias

[1] T. M. Cover, P. E. Hart, Nearest neighbor pattern classification, IEEE Transactions on Information Theory 13 (1) (1967) 21–27.
[2] X. Wu, V. Kumar (Eds.), The Top Ten Algorithms in Data Mining, Chapman & Hall/CRC Data Mining and Knowledge Discovery, 2009.
[3] J. Derrac, S. García, F. Herrera, Fuzzy nearest neighbor algorithms: Taxonomy, experimental analysis and prospects, Information Sciences 260 (2014) 98 – 119.
[4] J. M. Keller, M. R. Gray, J. A. Givens, A fuzzy k-nearest neighbor algorithm, IEEE Transactions on Systems, Man, and Cybernetics SMC-15 (4) (1985) 580–585.
[5] J. Maillo, S. Ramírez, I. Triguero, F. Herrera, kNN-IS: an iterative spark-based design of the k-Nearest Neighbors classifier for big data, Knowledge-Based Systems 117 (Supplement C) (2017) 3 – 15, volume, Variety and Velocity in Data Science.
[6] J. K. Uhlmann, Satisfying general proximity / similarity queries with metric trees, Information Processing Letters 40 (4) (1991) 175 – 179.
[7] T. Liu, A. W. Moore, K. Yang, A. G. Gray, An investigation of practical approximate nearest neighbor algorithms, in: Advances in neural information processing systems, 2005, pp. 825–832.
[8] J. Maillo, J. Luengo, S. García, F. Herrera, I. Triguero, Exact fuzzy k-nearest neighbor classification for big datasets, in: 2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2017, pp. 1–6.
[9] T. Liu, C. J. Rosenberg, H. A. Rowley, Performing a parallel nearest-neighbor matching operation using a parallel hybrid spill tree, uS Patent 7,475,071 (Jan. 6 2009).
[10] J. Dean, S. Ghemawat, Map reduce: A flexible data processing tool, Communications of the ACM 53 (1) (2010) 72–77.
[11] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, in: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, 2012, pp. 1–14.
[12] ECBDL14 dataset: Protein structure prediction and contact map for the ECBDL2014 big data competition (2014).
URL http://cruncher.ncl.ac.uk/bdcomp/
[13] M. Lichman, UCI machine learning repository (2013).
URL http://archive.ics.uci.edu/ml

Aplicando la transformada integral de la probabilidad para reducir la complejidad de los árboles de decisión difusos multi-vía en problemas de clasificación Big Data

Mikel Elkano*†‡, Mikel Uriz*†, Humberto Bustince*†‡, Mikel Galar*†‡
*Departamento de Estadística, Informática y Matemáticas, Universidad Pública de Navarra, 31006 Pamplona, España
†GIARA, Navarrabiomed, Complejo Hospitalario de Navarra (CHN), Universidad Pública de Navarra (UPNA), IdiSNA Irunlarrea 3, 31008 Pamplona, España
‡Institute of Smart Cities, Universidad Pública de Navarra, 31006 Pamplona, España
Emails: {mikel.elkano, mikelxabier.uriz, bustince, mikel.galar}@unavarra.es

***Resumen*—Presentamos un nuevo método de particionamiento difuso distribuido para reducir la complejidad de los árboles de decisión difusos multi-vía en problemas de clasificación Big Data. El algoritmo propuesto construye un número fijo de conjuntos difusos para todas las variables y ajusta su forma y posición a la distribución real de los datos de entrenamiento. Para ello se aplica un proceso compuesto por dos fases: 1) transformación de la distribución original en una distribución uniforme estándar mediante la transformada integral de la probabilidad. Dado que generalmente la distribución original se desconoce, se calcula una aproximación de la función de distribución acumulada extrayendo los q-cuantiles del conjunto de entrenamiento; 2) construcción de una partición difusa de Ruspini en el espacio de características transformado empleando un número fijo de funciones de pertenencia triangulares uniformemente distribuidas. A pesar de realizar una transformación sobre los datos originales, la definición de los conjuntos difusos en el espacio original se puede recuperar aplicando la función cuantil. Los resultados experimentales revelan que el método de particionamiento propuesto permite mantener la precisión del árbol de decisión difuso multi-vía del estado del arte FMDT empleando hasta 6 millones menos de hojas.**

***Index Terms*—Árboles de Decisión Difusos; Transformada Integral de la Probabilidad; Función Cuantil; MapReduce; Apache Spark; Big Data**

I. Introducción

Los árboles de decisión (ADs) [1] son uno de los algoritmos de aprendizaje automático más populares, habiendo sido aplicados en una gran variedad de problemas como las finanzas [2], la clasificación de imágenes [3], la bioinformática [4], o la medicina [5]. La principal característica de los ADs es la habilidad para explicar el razonamiento de sus predicciones mediante grafos en forma de árboles. Cada nodo es una pregunta o test en un atributo determinado (por ejemplo, ¿$x > 0{,}5$?), cada rama representa la respuesta o el resultado del test, y los nodos terminales (hojas) contienen la decisión final. Generalmente los árboles se construyen aplicando una estrategia *top-down* llamada *particionamiento recursivo* [1], en donde los datos de entrada se dividen recursivamente en dos o más sub-espacios que aumentan la homogeneidad de las distribución de las clases.

La lógica difusa [6] ha mostrado ser una forma efectiva para mejorar el rendimiento de clasificación de los algoritmos de aprendizaje automático cuando se trabaja con datos imprecisos que generan incertidumbre [7], [8], incluyendo los árboles de decisión difusos (ADDs). Sin embargo, en entornos Big Data los excesivos requerimientos de tiempo y espacio de los ADDs afectan seriamente la escalabilidad de estos algoritmos. Segatori et al. propusieron una solución MapReduce que consiste en un nuevo método de particionamiento difuso y en un algoritmo de aprendizaje de ADD distribuido [9]. El discretizador genera una partición difusa fuerte y triangular para cada atributo continuo basándose en la entropía difusa, las cuales son empleadas posteriormente para construir el árbol. Los autores propusieron dos versiones de ADD que difieren en la estrategia de particionamiento recursivo: el ADD binario (o bi-vía) (FBDT) y el ADD multi-vía (FMDT). El ADD binario divide el espacio del atributo en dos sub-espacios (nodos hijo), mientras que el ADD multi-vía puede generar más de dos sub-espacios. A pesar de que la solución propuesta por Segatori et al. consigue buenos resultados en términos de precisión, normalmente los árboles construidos por este algoritmo están compuestos por miles e incluso millones de hojas, por lo que la complejidad del modelo es generalmente elevada.

En este trabajo presentamos un nuevo método de particionamiento difuso que reduce la complejidad de los árboles construidos por FMDT en términos de número de conjuntos difusos empleado por variable y número de hojas. El algoritmo propuesto aplica la *transformada integral de la probabilidad* [10] para ajustar un número fijo de conjuntos difusos a la distribución real del conjunto de entrenamiento. Esta transformación permite convertir las variables del conjunto de entrenamiento en variables aleatorias uniformes independientemente de su distribución original. Posteriormente, se construyen la particiones difusas de Ruspini [11] en el nuevo espacio transformado empleando funciones de pertenencia

triangulares uniformemente distribuidas. Los conjuntos difusos resultantes se utilizan posteriormente por el FMDT original para construir el árbol.

Para evaluar los beneficios de nuestra propuesta, hemos llevado a cabo un estudio empírico que consiste en 4 problemas de clasificación Big Data disponibles en los repositorios de UCI [12] y OpenML[1]. En este estudio comparamos la precisión y la complejidad del modelo de FMDT cuando se considera el método de particionamiento difuso original y el propuesto. Los resultados experimentales muestran una reducción significativa en la complejidad del modelo cuando se aplica nuestro método.

La estructura de este trabajo es la siguiente. La Sección II repasa los conceptos básicos de los ADDs y describe brevemente la solución distribuida de Segatori et al. para construir ADDs para Big Data. En la Sección III introducimos el método de particionamiento difuso propuesto. El marco y el estudio experimental se muestran en las Secciones IV y V. Finalmente, la Sección VI presenta las conclusiones de este trabajo.

II. Preliminares: Árboles de decisión difusos para Big Data

Los árboles de decisión (ADs) [1] son algoritmos de aprendizaje automático supervisado no-paramétricos empleados para tareas de clasificación y regresión. En este trabajo nos centramos en las tareas de clasificación, las cuales consisten en construir un modelo predictivo llamado *clasificador* que es capaz de clasificar ejemplos no etiquetados (desconocidos) en base a un conjunto de entrenamiento formado por ejemplos previamente etiquetados. Cada ejemplo $x = (x_1, \ldots, x_F)$ contenido en el conjunto de entrenamiento TR pertenece a una clase $y \in \mathbb{C} = \{C_1, ..., C_M\}$ (siendo M el número de clases del problema) y se caracteriza por un conjunto de F variables (también llamadas atributos o características), donde cada variable x_f puede tomar cualquier valor contenido en el conjunto $\mathcal{F}_f$. Por tanto, la construcción de un clasificador consiste en encontrar una función de decisión $h : \mathcal{F}_1 \times \ldots \times \mathcal{F}_F \rightarrow \mathbb{C}$ que maximice la precisión en la clasificación.

Un AD es un grafo dirigido acíclico donde cada nodo interno es un test sobre un atributo, cada rama representa el resultado del test, y cada nodo terminal (hoja) contiene la decisión final (etiqueta de la clase). Los ADs se construyen aplicando un *particionamiento recursivo* [1] del espacio de atributos. La selección del atributo considerado en el nodo de decisión está basada en métricas que miden la diferencia entre el nivel de homogeneidad de las clases contenidas en el nodo padre y en los nodos hijos. Para los atributos continuos se pueden aplicar soluciones basadas tanto en fuerza bruta como en estrategias de discretización. Las primeras comprueban todos los posibles puntos de corte en el conjunto de entrenamiento, mientras que las últimas dividen el dominio del atributo en un conjunto discreto de intervalos (también llamados *bins*). Dado que las soluciones por fuerza bruta pueden ser computacionalmente costosas, los ADs diseñados para Big Data suelen aplicar estrategias de discretización para acelerar la ejecución del algoritmo y reducir la complejidad del modelo.

Los árboles de decisión difusos (ADDs) [7] hacen uso de la lógica difusa [6] para manejar mejor la incertidumbre y crear fronteras de decisión suaves que mejoran la precisión de la clasificación. En este caso los atributos continuos están caracterizados por particiones difusas en lugar de un conjunto discreto de intervalos. Por consiguiente, un valor de entrada determinado puede pertenecer a uno o más conjuntos difusos con un grado de pertenencia determinado y activar múltiples ramas al mismo tiempo. Las particiones difusas permiten manejar transiciones progresivas entre los intervalos adyacentes y mejorar así la precisión de las predicciones cuando se manejan datos numéricos. A la hora de clasificar un nuevo ejemplo x, se calcula el grado de activación de cada hoja. Para ello se debe calcular el grado de activación de todos los nodos internos que pertenecen a la ruta de la hoja correspondiente de la siguiente forma. Dado el nodo actual CN que considera x_f como el atributo a dividir, se calcula el grado de activación $md^{CN}(x)$ de CN para x:

$$md^{CN}(x) = T\left(\mu^{CN}(x_f), md^{PN}(x)\right), \tag{1}$$

donde T es una T-norma, $\mu^{CN}(x_f)$ es el grado de pertenencia de x_f al conjunto difuso asociado con el nodo CN, y $md^{PN}(x)$ es el grado de activación del nodo padre PN para x. Posteriormente, se obtiene el grado de asociación $AD_m^{LN}(x)$ de la clase C_m en la hoja LN para x:

$$AD_m^{LN}(x) = md^{LN}(x) \cdot w_m^{LN}, \tag{2}$$

donde $md^{LN}(x)$ es el grado de activación de la hoja LN para x y w_m^{LN} es el peso de la clase C_m en LN. En la literatura se han propuesto diferentes definiciones para w_m^{LN} [13]. En este trabajo consideramos

$$w_m^{LN} = \frac{\sum_{x \in TR_{C_m}} md^{LN}(x)}{\sum_{x \in TR} md^{LN}(x)}, \tag{3}$$

donde TR_{C_m} es el conjunto de todos los ejemplos de entrenamiento que pertenecen a la clase C_m. Finalmente, se predice la clase de x siguiendo un determinado criterio, siendo los siguientes los más comunes:

- *Máxima activación*: se predice la clase que corresponde al máximo grado de asociación.
- *Voto ponderado*: para cada clase se calcula la suma de todos los grados de asociación correspondientes a la clase y se predice aquella que obtiene la mayor suma.

Los requerimientos de tiempo y memoria de los ADDs pueden causar serios problemas de escalabilidad cuando se tratan grandes conjuntos de datos. En este trabajo consideramos la solución distribuida propuesta por Segatori et al. en [9] para construir ADDs en Big Data, la cual consta de dos fases:

[1] https://www.openml.org/search?type=data

1. *Particionamiento difuso*. Se genera una partición difusa triangular fuerte para cada atributo continuo basándose en la entropía difusa. Para ello, el algoritmo selecciona la partición difusa candidata que minimiza la entropía difusa y divide el dominio del atributo en dos subconjuntos de forma recursiva hasta que se cumple una cierta condición de parada. A pesar de que las particiones difusas construidas por este método suelen ser precisas, el alto número de conjuntos difusos que habitualmente contienen aumenta la complejidad del modelo.
2. *Aprendizaje del ADD*. Se construye un ADD aplicando una de las dos estrategias de división consideradas por los autores: binaria (FBDT), que genera dos nodos hijos, o multi-vía (FMDT), que puede generar más de dos hijos. Ambos métodos emplean la ganancia de información difusa [14] para la selección del atributo. En este trabajo nos centramos en los árboles FMDT.

III. APLICANDO LA TRANSFORMADA INTEGRAL DE LA PROBABILIDAD PARA REDUCIR LA COMPLEJIDAD DE LOS ÁRBOLES DE DECISIÓN DIFUSOS MULTI-VÍA

En este trabajo proponemos un nuevo método de particionamiento difuso distribuido que reduce la complejidad de los ADDs generados por el algoritmo FMDT [9]. La solución propuesta reemplaza el método de particionamiento difuso original empleado por FMDT sin alterar el algoritmo de aprendizaje del ADD. Los objetivos de nuestra propuesta son los siguientes:

- Construir un número reducido de conjuntos difusos por atributo. El método original añade conjuntos difusos a la partición hasta que la ganancia de información difusa es menor que un cierto umbral, aumentando la complejidad del modelo. Nuestro método emplea un número fijo de conjuntos difusos para todos los atributos.
- Ajustar los conjuntos difusos a la distribución real de los atributos. La solución propuesta modifica tanto la forma como la posición de los conjuntos difusos para mejorar la capacidad de discriminación del modelo

Para cumplir estos objetivos proponemos un algoritmo que consiste en una fase de pre-procesamiento que resulta en un particionamiento difuso auto-adaptativo.

- Pre-procesamiento: las variables del conjunto de entrenamiento son transformadas en variables aleatorias uniformes aplicando la *transformada integral de la probabilidad* [10] descrita en el Teorema 1. Este teorema afirma que cualquier variable aleatoria continua puede ser convertida en una variable aleatoria uniforme estándar.

  Teorema 1. *Si X es una variable aleatoria continua con una función de distribución acumulada (FDA) $F_X(x)$ y si $Y = F_X(X)$, entonces Y es una variable aleatoria uniforme en el intervalo [0,1].*

  Demostración. Supongamos que $Y = g(X)$ es una función de X donde g es derivable y estrictamente creciente. Por tanto, su inversa g^{-1} existe únicamente. La FDA de Y se puede derivar usando

$$F_Y(y) = Prob\,(Y \leq y) = Prob\left(X \leq g^{-1}(y)\right) = F_X\left(g^{-1}(y)\right)$$

  y su densidad viene dada por

$$f_Y(y) = \frac{d}{dy}F_Y(y) = \frac{d}{dy}F_X(g^{-1}(y)) = f_X(g^{-1}(y)) \cdot \frac{d}{dy}g^{-1}(y).$$

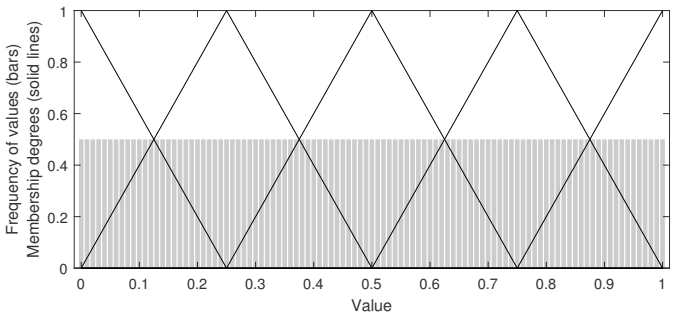
  Este procedimiento se llama la técnica de la FDA y permite que la distribución de Y se derive de la siguiente forma:

$$F_Y(y) = Prob\,(Y \leq y) = Prob\left(X \leq F_X^{-1}(y)\right) = F_X\left(F_X^{-1}(y)\right) = y$$

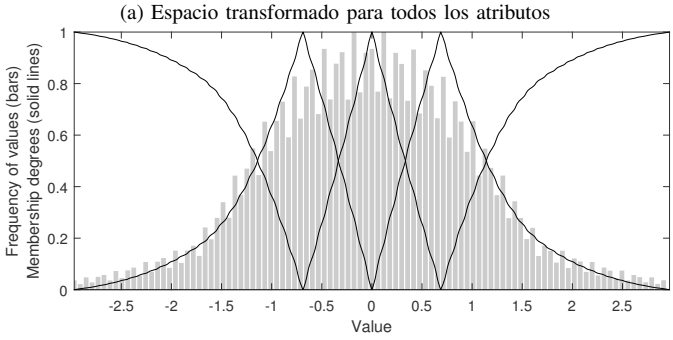
  ■

  Sin embargo, dado que la distribución original del conjunto de entrenamiento es desconocida, la FDA no puede ser calculada exactamente. Como solución proponemos calcular los q-cuantiles del conjunto de entrenamiento para obtener una FDA aproximada. Para ello, para cada variable, se ordenan todos los valores y se extrae cada cuantil. Si q es menor que el número de ejemplos en el conjunto de entrenamiento, el valor de la FDA para un valor dado se interpola linealmente en el intervalo [Q_{i-1}, Q_i], siendo Q_i el primer cuantil mayor que el valor. Si el valor es menor que el primer cuantil (Q_1) o mayor que el último cuantil (Q_{q-1}), el valor de la FDA será 0 o 1, respectivamente. De esta forma, las variables del nuevo conjunto de datos transformado seguirán una distribución aproximadamente uniforme independientemente de su distribución original. Por supuesto, la transformación sobre el conjunto de test se realiza interpolando la FDA con los cuantiles extraídos del conjunto de entrenamiento.
- Particionamiento: se construye una partición difusa fuerte de Ruspini [11] distribuyendo uniformemente un número fijo de funciones de pertenencia triangulares a lo largo del intervalo [0,1]. Cabe destacar que la definición de cada uno de los conjuntos difusos en el espacio original puede recuperarse aplicando la función cuantil [15]. En este caso, para cada punto que define la función de pertenencia triangular, el valor correspondiente se interpolaría calculando la inversa de la función lineal empleada para calcular la FDA entre los dos cuantiles más próximos. La Figura 1 muestra un ejemplo ilustrativo de cómo se distribuyen los conjuntos difusos en el espacio original y en el transformado del atributo *jet_1_eta* del conjunto de datos HIGGS. Las líneas sólidas y las barras representan las funciones de pertenencia y la distribución original de las variables, respectivamente.

Nótese que ambas fases (pre-procesamiento y particionamiento) están estrechamente relacionadas. Dado que, bajo nuestro punto de vista, una partición difusa de Ruspini con



(a) Espacio transformado para todos los atributos



(b) Espacio original del atributo *jet_1_eta*

Figura 1: Conjuntos difusos construidos para *jet_1_eta* en HIGGS.

funciones de pertenencia uniformemente distribuidas es una buena forma de modelar una distribución uniforme, nuestra hipótesis es que si es posible transformar la distribución de cualquier atributo en una distribución uniforme y al mismo tiempo poder realizar el proceso inverso, se obtiene una partición auto-adaptada para la distribución original del atributo. El hecho más interesante es que este resultado se obtiene sin necesidad de diseñar un nuevo método de particionamiento específico. El código fuente de nuestra propuesta está escrito en Scala 2.11[2] para Apache Spark 2.0.2 y está disponible en GitHub[3] bajo la licencia GPL.

IV. Marco experimental

En esta sección primero describimos los conjuntos de datos y las medidas de rendimiento empleadas para evaluar los métodos considerados en los experimentos (Sección IV-A). Posteriormente detallamos los parámetros y la configuración del entorno utilizados para la ejecución de los algoritmos (Sección IV-B).

IV-A. Conjuntos de datos y medidas de rendimiento

Para llevar a cabo el estudio experimental hemos considerado 4 problemas de clasificación Big Data disponibles en los repositorios de UCI [12] y OpenML[4]. La Tabla I muestra la descripción de los conjuntos de datos indicando el número de instancias (#Instancias), atributos (#Atributos) reales (R)/enteros(E)/categóricos(C)/totales(T), y clases (#Clases). Los nombres de BNG Australian (BNG) y HEPMASS (HEPM) han sido abreviados. En todos los experimentos se ha empleado un esquema de validación cruzada estratificada de 5 particiones.

Tabla I: Descripción de los conjuntos de datos.

Dataset	#Instancias	#Atributos				#Clases
		R	E	C	T	
BNG	1,000,000	8	6	0	14	2
HEPM	10,500,000	28	0	0	28	2
HIGGS	11,000,000	28	0	0	28	2
SUSY	5,000,000	18	0	0	18	2

El rendimiento en clasificación se ha medido empleando la denominada matriz de confusión (Tabla II), la cual almacena el número de ejemplos clasificados correcta e incorrectamente para cada clase. De esta matriz podemos obtener las siguientes

Tabla II: Matriz de confusión para un problema binario.

	Predicción positiva	Predicción negativa
Clase positiva	Verdadero positivo (TP)	Falso Negativo (FN)
Clase negativa	Falso Positivo (FP)	Verdadero Negativo (TN)

cuatro medidas:

- Ratio de verdaderos positivos: porcentaje de ejemplos positivos clasificados correctamente.
  $TP_{rate} = TP \ / \ (TP + FN)$.
- Ratio de verdaderos negativos: porcentaje de ejemplos negativos clasificados correctamente.
  $TN_{rate} = TN \ / \ (TN + FP)$.
- Ratio de falsos positivos: porcentaje de ejemplos negativos clasificados incorrectamente.
  $FP_{rate} = FP \ / \ (FP + TN)$.
- Ratio de falsos negativos: porcentaje de ejemplos positivos clasificados incorrectamente.
  $FN_{rate} = FN \ / \ (FN + TP)$.

Basados en estas métricas, el rendimiento de clasificación de cada método se ha medido empleando el porcentaje de precisión y el Área Bajo la Curva ROC (AUC) definidas como:

$$\%Precision = (TP+TN) \ / \ (TP+FN+FP+TN) \quad (4)$$

$$AUC = (1 + TP_{rate} + FP_{rate}) \ / \ 2 \quad (5)$$

IV-B. Parámetros y configuración del entorno

En cuanto a los parámetros utilizados para FMDT, hemos fijado los valores sugeridos por los autores en el artículo original [9]:

- Medida para calcular la impureza de los nodos: entropía difusa
- T-norma: producto
- Máximo número de bins para los atributos numéricos: 32

[2]http://www.scala-lang.org/
[3]https://github.com/melkano/uniform-fuzzy-partitioning
[4]https://www.openml.org/search?type=data

- Máxima profundidad del árbol: 5
- $\gamma = 0.1\,\%$; $\phi = 0.02 \cdot N$; $\lambda = 10^{-4} \cdot N$

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, Apache Hadoop 2.6.0, y Apache Spark 2.0.2.

De acuerdo con los autores de FMDT, emplear más de 24 núcleos de CPU tiene un impacto negativo en el tiempo de ejecución de este método. Por consiguiente, hemos empleado 6 *executors* con 4 núcleos cada uno para ejecutar FMDT.

V. Estudio experimental

Para evaluar el rendimiento de nuestra propuesta hemos llevado a cabo un estudio empírico que cubre tres aspectos: rendimiento en la clasificación (Tabla III), complejidad del modelo (Tabla IV), y tiempo de ejecución (Tabla V). En todos los casos hemos considerado cuatro métodos: el FMDT original [9] y tres configuraciones diferentes del método propuesto que difieren en el número de conjuntos difusos (X) utilizados para los atributos numéricos (denotado como FMDT$_X$). Cabe señalar que el método FMDT original se quedó sin memoria mientras procesaba HEPMASS debido al gran número de hojas que se generaron a lo largo del entrenamiento, y por tanto no se muestran resultados para este método en HEPMASS.

Las Tablas III y IV revelan que el método de particionamiento difuso propuesto (FMDT$_X$) es capaz de mantener el rendimiento de clasificación de FMDT con modelos notablemente más simples. Las diferentes configuraciones de nuestra aproximación rinden de forma similar en términos de precisión y AUC (excepto en HIGGS), a pesar de que hay una tendencia positiva a favor del uso de más conjuntos difusos. Sin embargo, emplear más conjuntos difusos suele causar que el algoritmo de aprendizaje genere más hojas, aumentando la complejidad del modelo. A continuación analizamos los resultados obtenidos en cada uno de los conjuntos de datos:

- BNG: nuestro método mejora la precisión y el AUC de FMDT un 6 % y un 8 %, respectivamente. A pesar de que los árboles construidos por FMDT$_X$ son más profundos, éstos tienen entre 8 mil y 80 mil veces menos de hojas que FMDT.
- HEPM: el FMDT original construye demasiadas hojas como para poder afrontar este problema con el cluster descrito en la Sección IV-B, quedándose sin memoria durante la ejecución. Este hecho sugiere que nuestra aproximación es una solución candidata para evitar la explosión del número de hojas durante la inducción de los ADDs.
- HIGGS: el rendimiento de clasificación de FMDT$_5$ en este problema cae casi un 1 % con respecto al resto de métodos, revelando que 5 conjuntos difusos no son suficientes para capturar la complejidad de este problema. Sin embargo, el resto de configuraciones (FMDT$_7$ y FMDT$_9$) son capaces de mantener el rendimiento de clasificación de FMDT con 15 mil y 50 mil hojas, respectivamente, mientras que FMDT genera 6 millones de hojas. Además, el método de particionamiento difuso original genera casi el doble de conjuntos difusos que FMDT$_7$.
- SUSY: todas las configuraciones rinden de forma similar a FMDT en términos de capacidad de discriminación. Sin embargo, nuestro método construye árboles de 3 mil, 15 mil, y 50 mil hojas, mientras que FMDT genera 5 millones de hojas. En este caso, la diferencia en el número de conjuntos difusos empleados por cada método es aún mayor, ya que FMDT utiliza casi 23 conjuntos de datos por atributo.

Tabla III: Rendimiento de clasificación de cada método.

		%Precision		
Dataset	FMDT	FMDT$_5$	FMDT$_7$	FMDT$_9$
BNG	$80.23^{\pm 0,05}$	$86.79^{\pm 0,06}$	$86.93^{\pm 0,07}$	$86.97^{\pm 0,06}$
HEPM	-	$91.13^{\pm 0,02}$	$91.25^{\pm 0,02}$	$91.33^{\pm 0,02}$
HIGGS	$71.54^{\pm 0,02}$	$70.61^{\pm 0,02}$	$71.32^{\pm 0,03}$	$71.69^{\pm 0,03}$
SUSY	$79.29^{\pm 0,05}$	$79.15^{\pm 0,04}$	$79.49^{\pm 0,04}$	$79.66^{\pm 0,04}$
		AUC		
Dataset	FMDT	FMDT$_5$	FMDT$_7$	FMDT$_9$
BNG	$.7896^{\pm ,0004}$	$.8649^{\pm ,0006}$	$.8658^{\pm ,0007}$	$.8662^{\pm ,0007}$
HEPM	-	$.9113^{\pm ,0002}$	$.9125^{\pm ,0002}$	$.9133^{\pm ,0002}$
HIGGS	$.7143^{\pm ,0001}$	$.7033^{\pm ,0002}$	$.7114^{\pm ,0003}$	$.7155^{\pm ,0003}$
SUSY	$.7859^{\pm ,0004}$	$.7847^{\pm ,0004}$	$.7880^{\pm ,0004}$	$.7898^{\pm ,0004}$

La Tabla V muestra el tiempo requerido por cada método para las tres fases: particionamiento difuso, la inducción del ADD, y el tiempo total. En general, no hay diferencias significativas entre los diferentes métodos cuando se trata de la fase de particionamiento, a pesar de que el algoritmo propuesto es un 30 % más rápido que el método original en SUSY. Sin embargo, cuando se considera la inducción del ADD, la reducción en la complejidad del modelo cuando se emplea nuestro método resulta en tiempos de ejecución mucho más rápidos.

VI. Conclusiones

En este trabajo hemos presentado un nuevo método de particionamiento difuso distribuido que reduce la complejidad del modelo de los árboles de decisión difusos (ADDs) multivía en problemas de clasificación Big Data. El algoritmo

Tabla IV: Complejidad de cada modelo.

Dataset	FMDT	$FMDT_5$	$FMDT_7$	$FMDT_9$
Número de hojas				
BNG	83,044	1,211	4,807	9,492
HEPM	-	2,854	13,472	43,339
HIGGS	6,414,575	3,005	15,876	53,489
SUSY	5,225,134	2,977	14,989	49,038
Profundidad media				
BNG	3.02	4.67	5.00	4.35
HEPM	-	4.52	4.03	3.93
HIGGS	3.25	5.00	5.00	4.89
SUSY	3.68	5.00	5.00	4.76
Promedio de número de conjuntos difusos				
BNG	6.04	5.00	7.00	9.00
HEPM	-	5.00	7.00	9.00
HIGGS	13.01	5.00	7.00	9.00
SUSY	22.60	5.00	7.00	9.00

Tabla V: Tiempos de ejecución(s) de cada método.

Dataset	FMDT	$FMDT_5$	$FMDT_7$	$FMDT_9$
Particionamiento				
BNG	58	41	40	40
HEPM	-	295	292	294
HIGGS	252	273	274	276
SUSY	110	77	72	77
Aprendizaje				
BNG	25	23	22	24
HEPM	-	149	158	153
HIGGS	4,984	176	167	158
SUSY	1,282	76	75	77
Tiempo total				
BNG	84	65	63	65
HEPM	-	445	450	448
HIGGS	5,238	450	441	435
SUSY	1,392	154	148	155

propuesto consiste en transformar el conjunto de entrenamiento original de tal forma que todas las variables numéricas sigan una distribución uniforme estándar. Para ello se aplica la transformada integral de la probabilidad, la cual afirma que cualquier variable aleatoria continua puede ser convertida en una variable aleatoria uniforme en base a la función de distribución acumulada (FDA) original. Dado que la FDA es generalmente desconocida, proponemos aproximar esta función calculando los q-cuantiles del conjunto de entrenamiento e interpolando linealmente entre dichos cuantiles. Después de esta transformación, se crean las particiones difusas de Ruspini distribuyendo un número fijo de funciones de pertenencia triangulares de forma uniforme a lo largo del intervalo [0,1]. Para recuperar los puntos que definen los conjuntos difusos en el espacio original se aplica la función cuantil. El método de particionamiento propuesto es capaz de ajustar tanto la posición como la forma de los conjuntos difusos a la distribución real del conjunto de entrenamiento.

Para evaluar el rendimiento de nuestra propuesta, hemos realizado un estudio empírico que se centra en el algoritmo de inducción de ADDs multi-vía propuesto por Segatori et al. para Big Data (FMDT). Para ello, hemos reemplazado el método de particionamiento difuso original por el método propuesto, sin modificar la fase de inducción del ADD. Los resultados experimentales revelan que nuestro algoritmo genera árboles significativamente más simples que son capaces de mantener el rendimiento de clasificación del método original.

Agradecimientos

Este trabajo ha sido financiado parcialmente por el Ministerio de Ciencia, Innovación y Universidades de España con el proyecto TIN2016-77356-P.

Referencias

[1] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[2] M.-Y. Chen, "Predicting corporate financial distress based on integration of decision tree classification and logistic regression," *Expert Systems with Applications*, vol. 38, no. 9, pp. 11 261–11 272, 2011.

[3] C.-C. Yang, S. Prasher, P. Enright, C. Madramootoo, M. Burgess, P. Goel, and I. Callum, "Application of decision tree technology for image classification using remote sensing data," *Agricultural Systems*, vol. 76, no. 3, pp. 1101–1117, 2003.

[4] D. Che, Q. Liu, K. Rasheed, and X. Tao, "Decision tree and ensemble learning algorithms with their applications in bioinformatics," *Advances in Experimental Medicine and Biology*, vol. 696, pp. 191–199, 2011.

[5] J. Sanz, D. Paternain, M. Galar, J. Fernandez, D. Reyero, and T. Belzunegui, "A New Survival Status Prediction System for Severe Trauma Patients Based on a Multiple Classifier System," *Computer Methods and Programs in Biomedicine*, vol. 142, no. C, pp. 1–8, 2017.

[6] L. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, no. 3, pp. 338 – 353, 1965.

[7] C. Janikow, "Fuzzy decision trees: Issues and methods," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 28, no. 1, pp. 1–14, 1998.

[8] J. Sanz, H. Bustince, A. Fernández, and F. Herrera, "IIVFDT: Ignorance functions based interval-valued fuzzy decision tree with genetic tuning," *International Journal of Uncertainty, Fuzziness and Knowlege-Based Systems*, vol. 20, no. SUPPL. 2, pp. 1–30, 2012.

[9] A. Segatori, F. Marcelloni, and W. Pedrycz, "On Distributed Fuzzy Decision Trees for Big Data," *IEEE Transactions on Fuzzy Systems*, vol. 26, no. 1, pp. 174–192, 2018.

[10] J. E. Angus, "The Probability Integral Transform and Related Results," *SIAM Review*, vol. 36, no. 4, pp. 652–654, 1994.

[11] E. H. Ruspini, "A new approach to clustering," *Information and Control*, vol. 15, no. 1, pp. 22–32, 1969.

[12] D. Dheeru and E. Karra Taniskidou, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[13] H. Ishibuchi, T. Nakashima, and M. Nii, *Classification and Modeling with Linguistic Information Granules: Advanced Approaches to Linguistic Data Mining (Advanced Information Processing)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2004.

[14] M. Zeinalkhani and M. Eftekhari, "Fuzzy partitioning of continuous attributes through discretization methods to construct fuzzy decision tree classifiers," *Information Sciences*, vol. 278, pp. 715–735, 2014.

[15] N. U. Nair, P. G. Sankaran, and N. Balakrishnan, *Quantile-Based Reliability Analysis*. New York, NY: Springer New York, 2013, ch. Quantile Functions, pp. 1–28.

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.

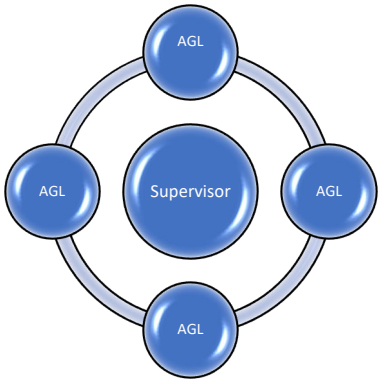


Fig. 1. Modelo distribuido de EDGAR: poblaciones de algoritmos geneticos y datos con particiones en nodos y nodo central con conjunto completo

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.

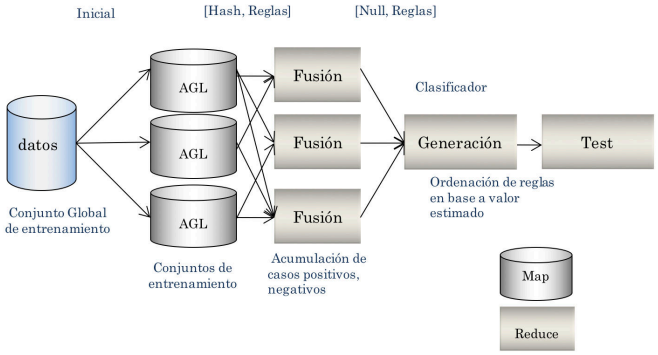


Fig. 3. Esquema *MapReduce* de las fases de aprendizaje, agregación, generacion de clasificador y *test*

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*.

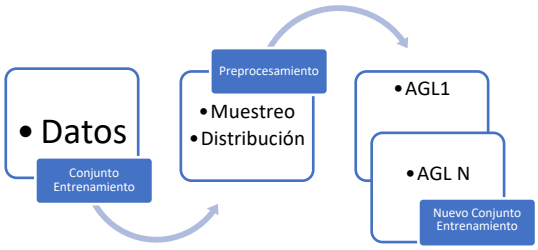


Fig. 4. Esquema de rebalanceo inicial de clases en fase de carga inicial

```
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).

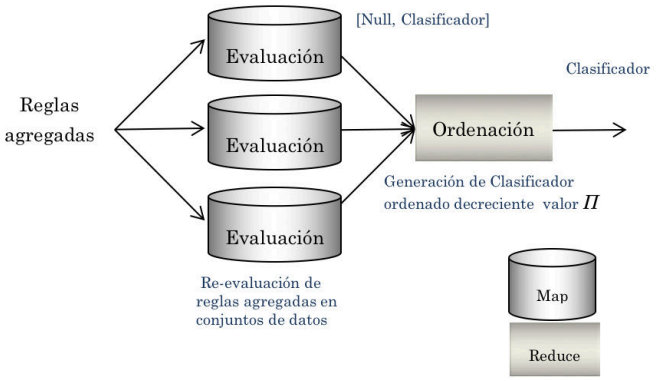


Fig. 5. Esquema *MapReduce* de la re-evaluación de reglas en variante de evaluación global en la fase de generación del clasificador

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.

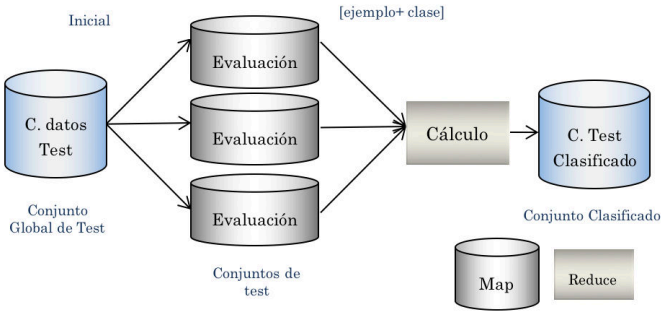


Fig. 6. Cálculo de la precisión en fase *test*

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.