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:

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

- 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.

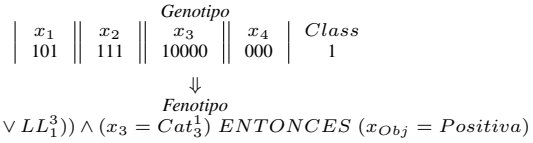

Figura 1. Representación de una regla DNF en EvAEP.

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.

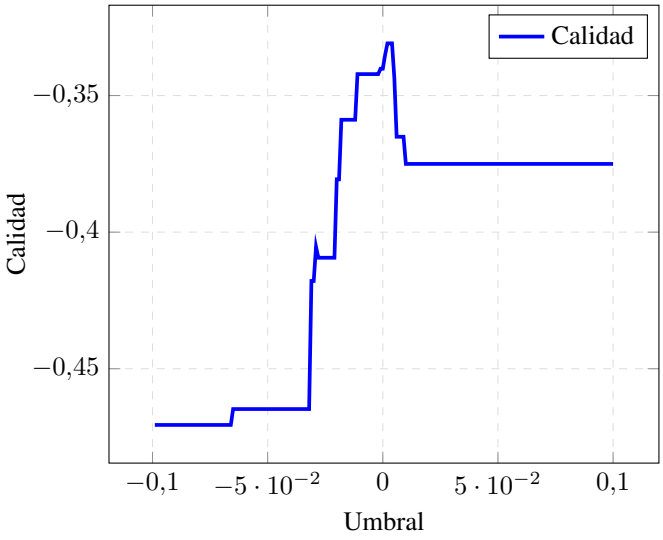

Figura 1. Calidad del agrupamiento vs umbral de bifurcación para el conjunto DIA1 con $\lambda = 0{,}001$.

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

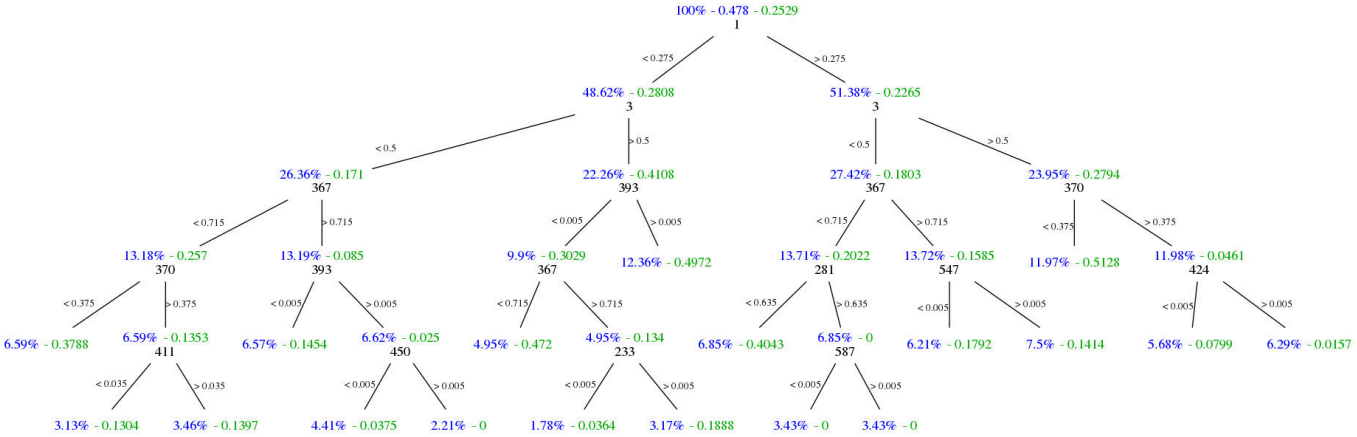

Figura 2. Árbol de decisión que determina el agrupamiento de los usuarios del conjunto de datos DIA1. En cada nodo se listan, respectivamente, la proporción de elementos que representa respecto al total (indicada en color azul), la entropía ponderada de ese nodo (verde) y la variable utilizada para la 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

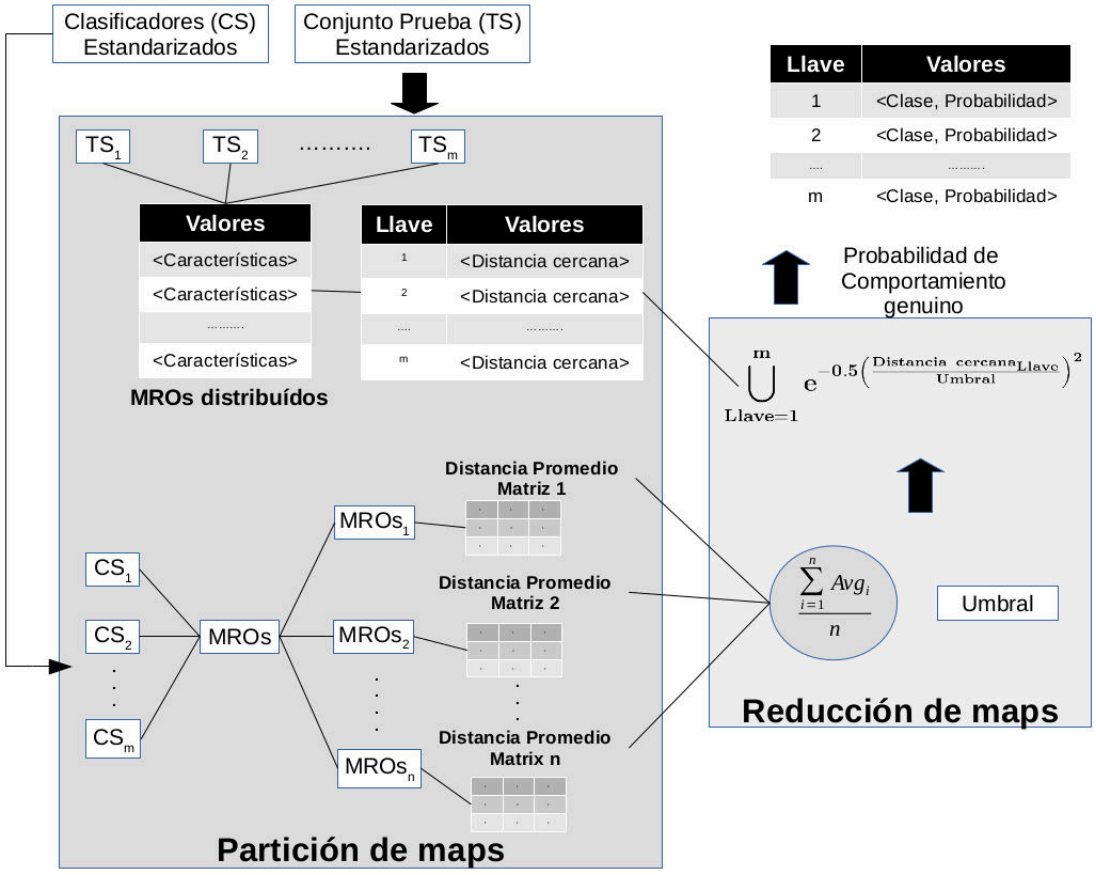

Figura 1. Diagrama de aprendizaje y predicción de Bagging-RandomMiner en MapReduce.

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.

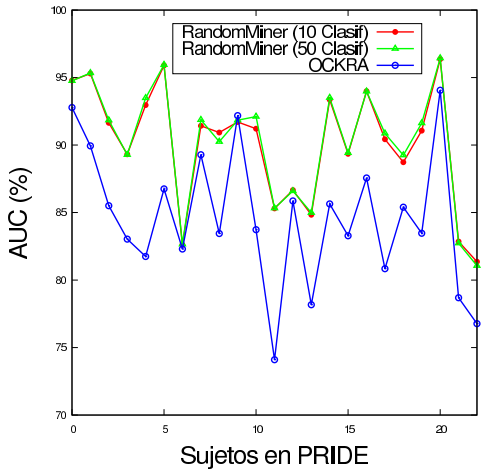

Figura 2. Porcentaje de Área bajo la curva (AUC) para RandomMiner (2 configuraciones) y OCKRA para cada uno de los 23 sujetos en PRIDE

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.

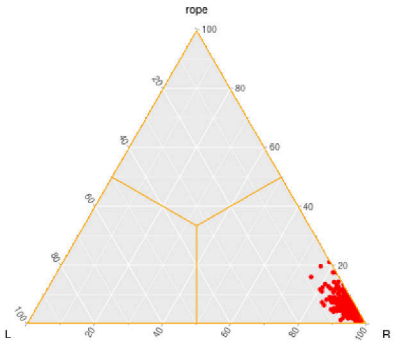

Figura 3. Resultados del Bayesian Signed Rank test sobre los resultados de OCKRA y Bagging-RandomMiner. La aglutinación de datos en la esquina inferior derecha (asociada a RandomMiner) sugiere fuertemente que éste es mejor algoritmo que OCKRA.

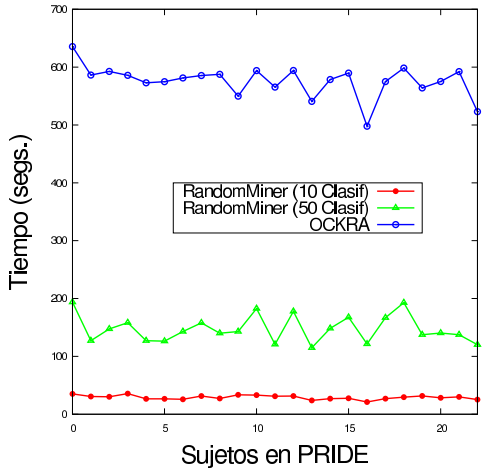

Figura 4. Tiempo de ejecución de los algoritmos RandomMiner (2 configuraciones) y OCKRA para cada uno de los 23 sujetos en PRIDE

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

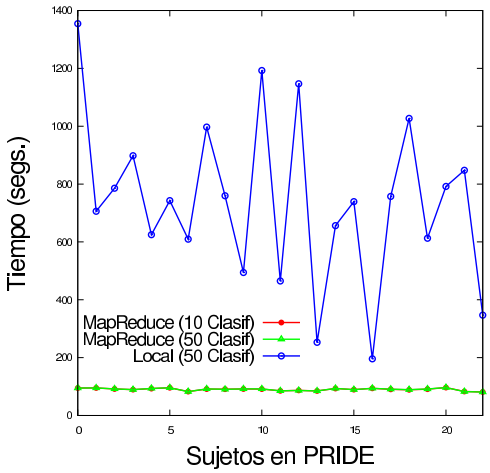

Figura 5. Tiempo de ejecución de los algoritmos RandomMiner en versión MapReduce y local.

[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

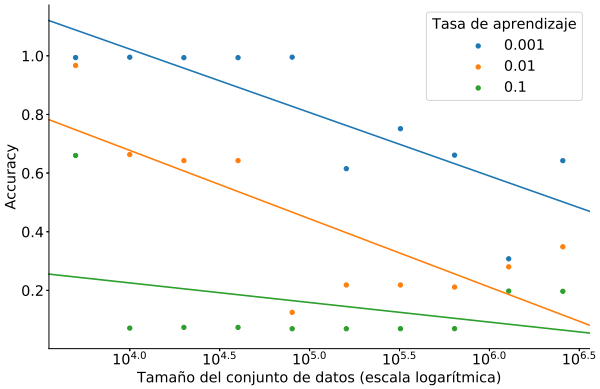

Figura 2: Problemas de escalabilidad del algoritmo GLAD.

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.

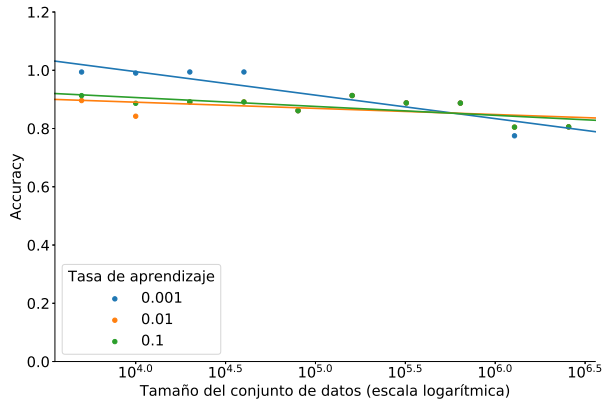

Figura 3: Escalabilidad en calidad del modelo en al propuesta CGLAD.

	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.

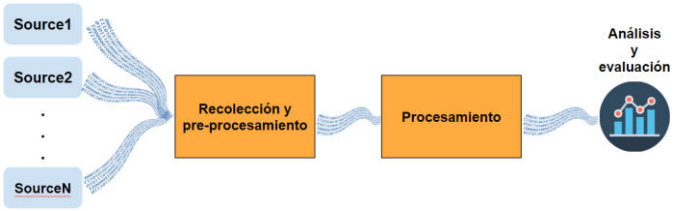

Fig. 1. Esquema general análisis en stream

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 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.

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

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.

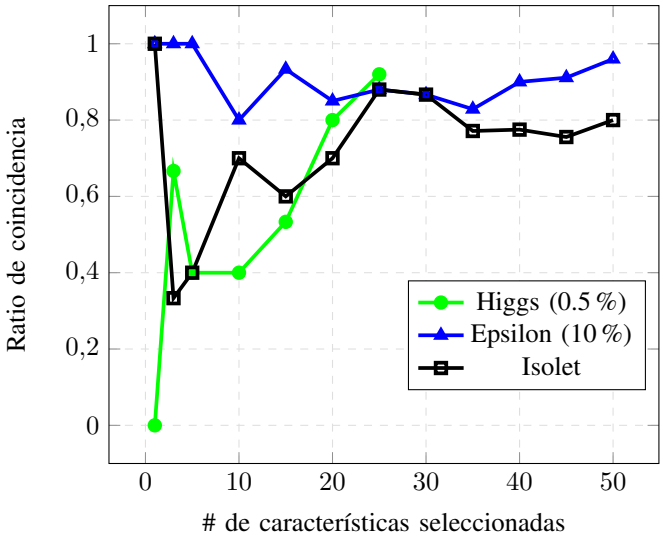

Figura 1. Ratio de coincidencia de las características seleccionadas por ReliefF vs ReliefF-LSH para los conjuntos Higgs (0.5 %), Epsilon (10 %) y Isolet.

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.