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.

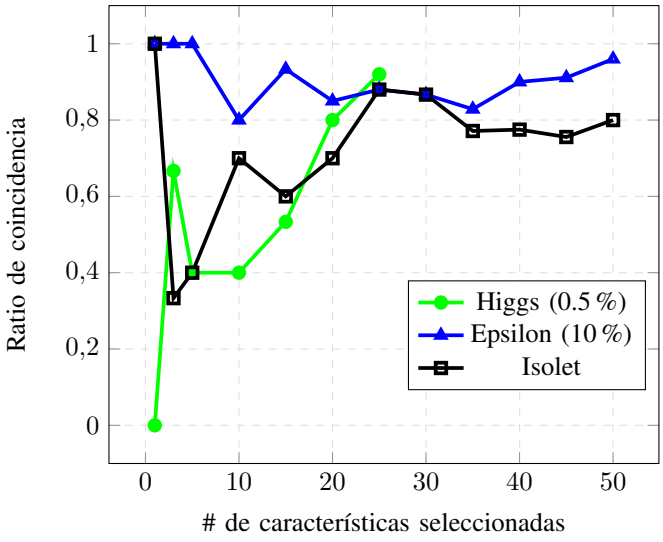

Figura 1. Ratio de coincidencia de las características seleccionadas por ReliefF vs ReliefF-LSH para los conjuntos Higgs (0.5 %), Epsilon (10 %) y Isolet.

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.