Búsquedas exhaustivas de subgrupos con MapReduce en *Big Data*

Francisco Padillo, José María Luna, Sebastián Ventura

Departamento de informática y análisis numérico, Universidad de Córdoba, Campus de Rabanales, 14071 Córdoba, España.
{fpadillo,jmluna,sventura}@uco.es,

Resumen El descubrimiento de subgrupos es una tarea usada para descubrir relaciones entre diferentes atributos con respecto a una variable objetivo. Hoy en día, con el crecimiento exponencial en el almacenamiento de datos, el análisis y extracción de información puede no ser abordable en un tiempo razonable. El objetivo de este trabajo es proponer un conjunto de algoritmos de búsqueda exhaustiva para el descubrimiento de subgrupos sobre *Big Data*. Se ha usado MapReduce y en concreto Apache Spark para abordar el problema de manera eficiente. Además, se han incluido métodos para podar el espacio de búsqueda. El estudio experimental incluye 30 bases de datos y se han realizado comparaciones con otros algoritmos ampliamente conocidos. Los resultados obtenidos revelan que los algoritmos propuestos son capaces de obtener subgrupos sobre grandes cantidades de datos eficientemente.

Keywords: Descubrimiento de subgrupos, *Big Data*, MapReduce, Spark

1. Introducción

El proceso de convertir datos brutos en información interesante es conocida como análisis de datos, donde se han propuesto diferentes métodos para este fin. Aunque cada propuesta sigue su propio enfoque, todas ellas se podrían agrupar en dos categorías: aprendizaje supervisado y no supervisado. En aprendizaje supervisado el objetivo es predecir lo mejor posible una o más variables de interés, mientras que en aprendizaje no supervisado no se tiene ningún tipo de información a priori, siendo el objetivo su descripción. Sin embargo, algunas técnicas se encuentran a medio camino, agrupándose dentro de *Supervised Local Pattern mining*. Siendo el descubrimiento de subgrupos (DS) una de las más usadas [1].

El objetivo de esta técnica es identificar relaciones entre una variable dependiente (variable objetivo) y muchas variables independientes o atributos. Algunos enfoques pioneros para esta tarea estuvieron basados en adaptaciones de las técnicas de aprendizaje no supervisado. Por ejemplo, Apriori-SD [2] es una adaptación del famoso algoritmo de aprendizaje no supervisado Apriori. Aunque, otros enfoques han recibido especial interés. Los algoritmos basados en FP-Growth [3] consiguen acelerar el proceso de extraer subgrupos. Con el fin de solucionar esto, algunos autores propusieron el uso de heurísticas [4]. Aunque

estos enfoques son muy eficientes, no se tiene la garantía de que se van a obtener las mejores soluciones. En este sentido, se propusieron las estimaciones optimistas (EOs) con el objetivo de podar espacios de búsqueda garantizando que los mejores subgrupos van a ser obtenidos [5].

El objetivo de este trabajo es proponer un conjunto de algoritmos de búsqueda exhaustiva basados en técnicas tradicionales pero considerando tecnologías emergentes para *Big Data*. Se ha usado MapReduce y en concreto Apache Spark. En este trabajo introducimos dos propuestas: AprioriDS-I-EO (Apriori para el Descubrimiento de Subgrupos Iterativo considerando Estimaciones Optimistas) y DSEOFP-Tree (Descubrimiento de Subgrupos considerando Estimaciones Optimistas usando FP-Tree). En el estudio experimental, se han realizado comparaciones con otros algoritmos sobre 30 base de datos, demostrando que nuestras propuestas superan a los algoritmos tradicionales en *Big Data*.

El resto del artículo está organizado como sigue. Sección 2 presenta algunas definiciones relevantes relacionadas con esta tarea; Sección 3 describe los algoritmos propuestos; Sección 4 presenta los *datasets* usados así como los resultados obtenidos. Finalmente, las conclusiones son expuestas en la Sección 5.

2. Trabajos relacionados

En esta sección se presentan algunas definiciones relevantes, así como algunos trabajos relacionados con la tarea de descubrimiento de subgrupos.

2.1. Descubrimiento de subgrupos

El Descubrimiento de Subgrupos (DS) es una técnica de minería de datos cuyo objetivo es la extracción de relaciones interesantes de un conjunto de atributos con respecto a una variable objetivo [6]. Dado que en algunas ocasiones es difícil cuantificar los subgrupos extraídos, se han propuesto muchas métricas para tal fin [1]. DS podría definirse de manera formal, como dada una base de datos BD, una función de calidad c y un número b, determinar los b subgrupos con el máximo de calidad c.

Desafortunadamente, a diferencia de otras tareas similares como la minería de patrones, donde los algoritmos usan la propiedad de monotonicidad [7] para reducir el espacio de búsqueda, en DS esta propiedad no puede ser usada. Incluso si un subgrupo s no tiene suficiente calidad, es necesario considerar sus refinamientos [5]. Sin embargo, si los mejores b subgrupos ya han sido encontrados, y todos los refinamientos s' de un subgrupo s tienen una calidad peor o igual que los ya obtenidos, este espacio de búsqueda podría ser podado. Para tal fin, es necesario incluir las EOs [5]. Una $EO(s)$ para una métrica de calidad c es una función que permite establecer un límite superior a la calidad de los refinamientos de s. De modo formal se define como $\forall s, s'.s' \succ s \Rightarrow EO(s) \geq c(s')$. Cuando una EO es totalmente concisa, se tiene que $\forall s, s'.s' \succ s \Rightarrow EO(s) = c(s')$.

Para obtener estos mejores b subgrupos se han propuesto diferentes enfoques [2,5,8,9]. El primero de estos enfoques es AprioriSD [2], el cual está basado

en el algoritmo Apriori junto con una heurística, provocando que pueda despreciar subgrupos interesantes. Para solucionar este inconveniente otros investigadores propusieron el uso de otro enfoque basado en una estructura de datos, FP-Tree, que permite acelerar el proceso. En este sentido, SD-MAP [9] es un algoritmo para descubrir subgrupos sobre variables objetivo binarias, en este tipo de variables sólo existen dos valores (Verdadero/Positivo o False/Negativo). Una de las métricas de calidad más ampliamente usada es $c_{PS} = n(p - p_0)$, donde n es el tamaño del subgrupo, p es la frecuencia relativa de la variable objetivo y p_0 es la frecuencia relativa de la variable objetivo en todo el *dataset*. Posteriormente, *Grosskreutz et al.* [5] propusieron la estimación optimista $OE_{PS} = np(1 - p_0)$. De igual forma SD-MAP* [8] fue propuesto para variables objetivo de tipo numéricas. Para evaluar la calidad de los subgrupos obtenidos, se compara la media del *dataset* global $\mu_\emptyset$ con la media del subgrupo P, definida como $c_{imp} = n \cdot (\mu_P - \mu_\emptyset)$. Una de las estimaciones optimistas más conocidas es $OE_{c_{imp}} = \sum_{t \in P:T(t)>\mu_\emptyset} T(t) - \mu_\emptyset$. Finalmente, también se han propuesto algoritmos para extraer subgrupos sobre variables objetivas de tipo categórico, como DpSubgroup [5]. La métrica *gini* es ampliamente conocida, definida como $c_{gini}(P) = \frac{n}{N-n}\sum_i (p_i - p_{0i})^2$, y el EO [5] definida como $EO_{gini} = \sum max\{E_i^+, E_i^- E_i^g\}$, donde $E_i^- = \frac{n(1-p_i)}{N-n(1-p_i)}(p_{0i})^2$, $E_i^g = \frac{n}{N-n}(p_i - p_{0i})^2$, y $E_i^+ = \frac{np_i}{N-np_i}(1 - p_{0i})^2$. N representa el número de instancias del *dataset* ; p_i es la distribución del valor i de la variable objetivo sobre el subgrupo; p_{0i} es la distribución de la variable objetivo i en el *dataset*.

2.2. MapReduce

MapReduce es un paradigma reciente para la computación paralela. Éste permite escribir algoritmos paralelos de una forma simple, permitiendo abordar grandes cantidades de datos de una forma eficiente. Los algoritmos en MapReduce están compuestos de dos fases definidas por el programador: *map* y *reduce*; En la fase del *map*, cada *mapper* procesa un subconjunto de los datos de entrada produciendo un conjunto de $\langle k, v\rangle$. Finalmente, el *reducer* obtiene como entrada la salida del *mapper*, produciendo un nuevo conjunto de $\langle k, v\rangle$ finales. Se ha usado Apache Spark ya que es considerado como una de las implementaciones de MapReduce con más reputación para modelar algoritmos iterativos [10], permitiendo cargar los *datasets* en una estructura de datos (RDD) que permite la lectura de forma distribuida.

3. Algoritmos para el descubrimiento de Subgrupos sobre *Big Data*

En esta sección se describen nuestras propuestas para extraer subgrupos en *Big Data* usando Apache Spark. La primera propuesta (AprioriDS-I-EO) esta basada en el algoritmo Apriori para extraer subgrupos en *Big Data*. La segunda propuesta (DSEOFP-Tree) es un algoritmo para extraer subgrupos sobre grandes

cantidades de datos basado en SD-MAP, SD-MAP* y DpSubgroup. El primero de ellos es más eficiente cuando el número de subgrupos a extraer es muy bajo o muy alto, mientras que DSEOFP-Tree funciona mejor con valores medios. Esto es debido a que DSEOFP-Tree puede podar un mayor espacio de búsqueda por su estructura de *Tree*, mientras que AprioriDS-I-EO es más eficiente cuando se usa un número de subgrupos que no implica una gran poda.

3.1. AprioriDS-I-EO

AprioriDS-I-EO (Apriori para el Descubrimiento de Subgrupos Iterativo con Estimadores Optimistas) es un algoritmo iterativo para el descubrimiento de subgrupos sobre *Big Data* basado en el famoso algoritmo Apriori. Una de las mayores desventajas de Apriori es su alta complejidad computacional, ya que para k atributos donde cada uno de ellos tiene m valores, se podrían obtener $\sum_{i=1}^{k} C_i^k \times m^i$ subgrupos. Luego, para tratar con este problema una opción sería no crear todos los posibles subgrupos sino sólo aquellos de tamaño j en cada una de las iteraciones. Además, se han incluido EOs ya que existen espacios de búsqueda que podrían ser descartados porque se tiene la garantía de que no se puede obtener ningún subgrupo interesante de él. El algoritmo propuesto es capaz de obtener subgrupos sobre cualquier tipo de variable interés, cambiando la métrica de calidad en función del tipo de variable objetivo. Para variables objetivo binarias se ha usado c_{PS}; para variables numéricas y categóricas se han usado c_{imp} y c_{gini} respectivamente, así como sus EOs, EO_{PS}, EO_{imp} y EO_{gini}. Adicionalmente, se puede introducir un umbral mínimo de calidad de forma que las b mejores soluciones devueltas tendrán al menos este umbral. Para obtener estas b mejores soluciones, el algoritmo propuesto funciona como sigue.

Paso 1. El *driver* lee el *dataset* desde disco y lo almacena en un RDD, el cual lo divide en *sub-datasets*. Además, calcula el máximo tamaño de los subgrupos, es decir, el número de atributos por instancia. Iterando desde $l = 1$ hasta $maxL$.

Paso 2. Para calcular los subgrupos de tamaño l es necesario una fase de Map Reduce. Estando el *mapper* encargado de descubrir subgrupos de tamaño l para su *sub-dataset*, siendo el *reducer* el encargado de agregar los resultados de todos los *mappers* para cada subgrupo. Esto permite usar tantos *reducers* como subgrupos de tamaño l existan. Como se ha usado MapReduce, toda la comunicación se ha realizado mediante $\langle k, v \rangle$, donde k se ha usado para representar a los subgrupos de tamaño l y el valor de v depende del tipo de variable objetivo. (a) Binarias, contiene un conteo de los tp (casos en los que el subgrupo contiene la variable interés), y fp (casos en los que no se contiene la variable objetivo). (b) Numéricas, contiene la suma de la variable objetivo para el subgrupo, un contador de frecuencia, la suma de $valorVariableInteres - mediaGlobal$, donde $mediaGlobal$ es la media de la variable objetivo en todo el *dataset* , y $valorVariableInteres$ es cada valor de la variable objetivo donde se cumplía ese subgrupo, también contiene la suma de $valorVariableInteres - mediaGlobal$ donde se cumplía la condición $valorVariableInteres > mediaGlobal$. ($c$) Categóricas, contiene un contador de frecuencia para cada unos de los valores.

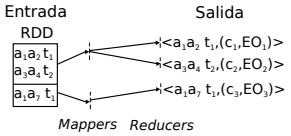


Figura 1: Ejemplo de ejecución para AprioriDS-I-EO en la iteración 2, se obtienen todos los subgrupos de tamaño 2. Donde a_i designa a valores de atributos, y t_i representa a valores de la variable objetivo.

Paso 3. Tras el *reducer*, los b mejores subgrupos son enviados al *driver* que será el encargado de unirlos con los b mejores subgrupos obtenidos hasta el momento, para seleccionar sólo los b mejores. Además, el *driver* es el encargado de detener el proceso si se tiene la garantía de que ningún subgrupo mejor va a ser obtenido. En este sentido, se compara el mejor EO de los subgrupos de tamaño l con la peor calidad de los b subgrupos obtenidos hasta el momento, si este es menor el proceso puede ser detenido ya que ningún refinamiento va a tener mejor calidad.

3.2. DSEOFP-Tree

DSEOFP-Tree (Descubrimiento de Subgrupos considerando Estimaciones Optimistas usando FP-Tree) es un algoritmo basado en la versión paralela de FP-Growth para la obtención de subgrupos. La principal ventaja de este algoritmo es que permite analizar el *dataset* con sólo dos lecturas. Esto permite hacer lecturas distribuidas y construir árboles independientes, permitiendo extraer subgrupos de forma paralela. Adicionalmente, se le han incluido EOs para reducir el espacio de búsqueda, de forma que se podría podar una gran cantidad de ramas. A continuación se describe el enfoque seguido por el algoritmo.
Paso 1. El *dataset* es dividido en partes sucesivas, y almacenado en un RDD.
Paso 2. Se calcula la frecuencia para cada uno de los valores de los atributos de forma paralela, obteniéndose la lista F-list. Como el *dataset* es leído completamente, se calculan algunas operaciones adicionales en función del tipo de variable objetivo. (a) Binarias, el valor total de TP y FP. (b) Numéricas, la media global de la variable objetivo. (c) Categóricos, se calcula la frecuencia para cada uno de los valores de las variables.
Paso 3. Los valores de F-list son ordenados por frecuencia, de forma que los más frecuentes se encontrarán al principio de la lista.
Paso 4. Este paso es primordial, ya que se crean los FP-Tree. En este paso se usa el RDD creado en el paso 1, junto con la lista F-list creada en el paso 3. Cada *mapper* produce un conjunto de $\langle k, v\rangle$, donde cada k es un *item* de F-list y v es una instancia dependiente del item [3]. Para cada *item* de F-list, si aparece en una instancia, localizar su aparición más a la derecha (L), y devuelve una $\langle k, v\rangle$ con la forma $\langle item, instancia[1]...instancia[L]\rangle$. Tras esto, cada *reducer* procesa un *item* permitiendo un mayor grado de paralelismo. Siguiendo esta filosofía,

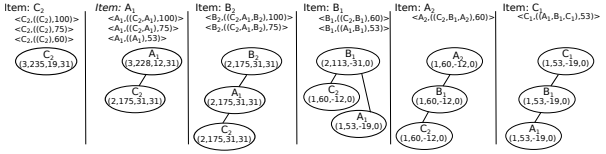


Figura 2: FP-Tree local para cada uno de los *reducers*, cada uno de ellos podría ser explotado de forma independiente con respecto al resto.

cada *reducer* crea un FP-Tree local y extrae los subgrupos representados por este árbol (Ver Figura 2). La métrica de calidad y el EO es también calculada en este paso, ya que cada nodo del árbol tiene la suficiente información para hacerlo. Cuando el árbol es explotado de forma recursiva, para cada unos de los nodos que se explora se comprueba su EO para determinar si el árbol debería de seguir siendo explotado, o los subgrupos que se van a extraer si se continúa ya no van a mejorar a los previamente obtenidos.

La información contenida en cada uno de los nodos del árbol depende del tipo de variable objetivo: (a) binarias, una tupla con los conteos de tp y fp; (b) una tupla con cuatro elementos, contador de frecuencia, suma de los valores de la variable objetivo, la suma de $valorVariableInteres - mediaGlobal$, donde $mediaGlobal$ es la media de la variable objetivo en todo el $dataset$, y $valorVariableInteres$ es cada valor de la variable objetivo donde se cumplía ese subgrupo, también contiene la suma de $valorVariableInteres - mediaGlobal$ donde se cumplía la condición $valorVariableInteres > mediaGlobal$; ($c$) Categóricas, se incluye un contador de frecuencia para cada unos de los valores.

Además del *mapper* y del *reducer*, el *driver* es el encargado de gestionar a los dos anteriores. De forma que cada *reducer* extrae los b mejores subgrupos para su FP-Tree, y los devuelve al *driver*. En este, se recibirán $b \times numeroReducers$ subgrupos, siendo la tarea de éste ordenar por calidad y seleccionar sólo los mejores b subgrupos. Aunque como resultado final sólo se devuelvan b subgrupos, el *driver* debe de trabajar con $b \times numeroReducers$ subgrupos, ya que es la única forma de devolver los b mejores subgrupos como solución final.

4. Experimentación

En esta sección, se ha estudiado el rendimiento de los algoritmos propuestos usando diferentes tamaños de datos. Es importante destacar que todos los tiempos de ejecución son la media para 10 ejecuciones diferentes, aunque todas ellas obtengan las mismas soluciones. El objetivo es suavizar las variaciones en tiempo de ejecución derivadas de la administración de recursos.

Para comprobar la escalabilidad de nuestros algoritmos se ha usado *datasets* sintéticos, de esta forma se tiene control sobre el número de instancias y

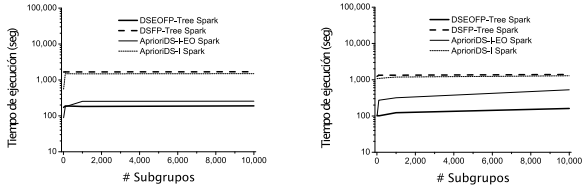
atributos. Se han usado 30 *datasets* sintéticos, donde el número de instancias variaba desde 10^5 hasta 10^7. Estos *datasets* producen un espacio de búsqueda desde $1.17648 \cdot 10^5$ hasta $1.276 \cdot 10^{15}$ subgrupos. Se han usado diferentes umbrales de calidad (α) en función del tipo de variable de objetivo ya que cada una de ellas tiene una métrica diferente y por tanto escalas diferentes. Para binarias se ha usado $\alpha_1 = 8.0 \cdot 10^{-1}$, para variables numéricas $\alpha_2 = 8.0 \cdot 10^2$ y para categóricas $\alpha_3 = 5.0 \cdot 10^{-5}$. Adicionalmente, se han usado *datasets* reales para demostrar la aplicabilidad sobre datos reales en un tiempo razonable. En este estudio experimental, nuestras propuestas se han comparados con otros algoritmos ampliamente conocidos en DS.

4.1. Estimadores optimistas: reduciendo el espacio de búsqueda

El objetivo de este estudio es demostrar que los estimadores optimistas permiten reducir el espacio de búsqueda, y por tanto mejorar el tiempo de ejecución. Además, se ha considerado interesante mostrar como la concisión de los estimadores optimistas afecta al rendimiento.

La Figura 3a muestra los resultados para variables objetivas binarias cuando se ejecutan los mismos algoritmo con y sin EO. Como se puede apreciar, AprioriDS-I-EO obtiene mejor rendimiento que AprioriDS-I, logrando un orden de magnitud de mejora en el tiempo de ejecución. De igual manera ocurre con DSEOFP-Tree con respecto a su versión sin EO. Por último, es importante destacar que DSFP-Tree obtiene peor rendimiento que AprioriDS-I, sin embargo cuando se usan EO el comportamiento cambia. Esto es debido a que DSFP-Tree necesita construir un árbol extremadamente grande, y posteriormente extraer patrones, luego podría no tener sentido realizar estos dos pasos, y simplemente extraer los subgrupos tal y como AprioriDS-I hace.

La Figura 3b muestra los resultados al usar variables objetivas numéricas. Tal y como se puede apreciar, el comportamiento es casi el mismo que el mostrado



(a) Tiempo de ejecución para los algoritmos con y sin EO. Sólo se han considerado variables objetivo de tipo binaria, y un umbral mínimo de calidad de α_1.

(b) Tiempo de ejecución para los algoritmos con y sin EO. Sólo se han considerado variables objetivo de tipo numéricas, y un umbral mínimo de calidad de α_2.

Figura 3: Estudio para comprobar el uso de estimadores optimistas.

anteriormente dado que la complejidad algorítmica es la misma, siendo la única diferencia la métrica de calidad y el EO. El mismo comportamiento se observa cuando la variable objetivo es de tipo binarío [1].

Continuando este estudio, una vez que se ha demostrado que los EOs permiten mejorar considerablemente el tiempo de búsqueda, logrando un orden de magnitud de mejora en los tiempos de ejecución [1], se va a analizar la concisión de estos. En este sentido, se va a demostrar la importancia de usar estimadores optimistas tan concisos como sea posible. Cuando la Figura 3a es comparada con la Figura 3b se puede apreciar que cuánto más conciso es el EO, mejor rendimiento obtienen todos los algoritmos, pero especialmente AprioriDS-I-EO. Como EO_{PS} es concisa AprioriDS-I-EO obtiene un rendimiento similar a DSEOFP-Tree en variables objetivo binarias, mientras que como EO_{imp} no es tan concisa, el rendimiento entre AprioriDS-I-EO y DSEOFP-Tree difiere en mayor medida con variables numéricas que con respecto al anterior caso. AprioriDS-I-EO está más afectado con la concisión de los EO ya que no es capaz de podar tanto como DSEOFP-Tree hace, mientras que este último puede descartar ramas enteras, AprioriDS-I-EO necesitaría calcular los subgrupos de un tamaño j para posteriormente realizar una poda.

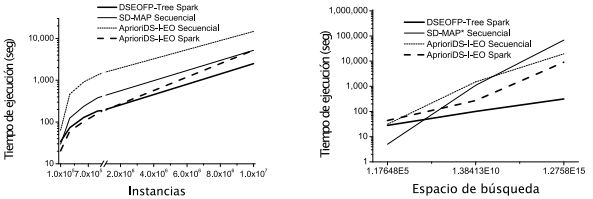
En conclusión, el uso de EO ha demostrado mejorar considerablemente el tiempo de cómputo. Permitiendo podar partes del espacio de búsqueda sin afectar a la precisión del algoritmo. Aunque AprioriDS-I-EO pudiese estar más afectado por la concisión que DSEOFP-Tree, en ambos esta justificado su uso ya que mejora el tiempo con respecto a las versiones sin EOs.

4.2. Escalabilidad de AprioriDS-I-EO y DSEOFP-Tree en *Big Data*

El objetivo de este estudio es comprobar el rendimiento de nuestros algoritmos usando *Big Data*. En este estudio sólo se han incluido algoritmos con EO, ya que han demostrado obtener los mejores resultados.

La Figura 4a muestra el resultado para variables objetivo de tipo binario. Tal y como se puede apreciar, AprioriDS-I-EO secuencial carece de sentido ya que tiene que realizar varias iteraciones, y en cada una de ellas tiene que leer el *dataset* completo. Por esta razón se propone el uso de Spark (AprioriDS-I-EO Spark), para permitir paralelizar este enfoque. El comportamiento de AprioriDS-I-EO con respecto a SD-MAP es casi el mismo cuando se usan *datasets* con $8 \cdot 10^6$ instancias. Sin embargo, aunque el comportamiento es similar, AprioriDS-I-EO Spark permite escalar horizontalmente, mientras que SD-MAP necesita de un hardware con mayor capacidad para continuar ejecutándose. DSEOFP-Tree Spark es el que mejor rendimiento obtiene, consiguiendo obtener los mismos resultados que SD-MAP en un 8.35 % del tiempo de ejecución. De igual forma, se puede apreciar que cuando se usa un número de instancias menor de 10^6, AprioriDS-I-EO obtiene mejor rendimiento que DSEOFP-Tree. Cuando se estudia el comportamiento de estos algoritmos sobre otros tipos de variables objetivo, se demuestra que es casi el mismo, ya que la complejidad computacional no varía [1].

[1] Por razones de espacio no se muestra este estudio, aunque está disponible en: http://www.uco.es/grupos/kdis/kdiswiki/DS-BigData



(a) Estudio de escalabilidad con un espacio de búsqueda de $1.38413 \cdot 10^{10}$ subgrupos y un número de instancias de 10^5 hasta 10^7

(b) Estudio de escalabilidad cuando el espacio de búsqueda varía entre $1.7648 \cdot 10^5$ y $1.2758 \cdot 10^{15}$, con 10^6 instancias.

Figura 4: Resultados de nuestra propuesta para demostrar la escalabilidad.

Continuando este estudio con otro enfoque diferente, se ha estudiado como afecta el número de atributos al rendimiento de los algoritmos. La Figura 4b muestra el resultado cuando el número de instancias es de 10^6 y el espacio de búsqueda cambia. Demostrando que DSEOFP-Tree obtiene excelentes resultados en casi todos los *datasets*, excepto cuando se usan *datasets* pequeños.

4.3. *Datasets* reales

El objetivo de esta sección es demostrar que nuestras propuestas son aplicables sobre *datasets* reales en una reducida cantidad de tiempo. La Tabla 1 muestra los resultados. Dado que se ha seguido un enfoque exhaustivo, los atributos continuos han tenido que ser discretizados previamente usando una discretización por igual amplitud. El número de *bins* usado es especificado junto al nombre del *dataset* con el sufijo $-numeroDeBins$, además de los mostrados en esta tabla se han utilizado otros *bins*. Como se puede apreciar AprioriDS-I-EO está más afectado por el número de atributos que DSEOFP-Tree, al igual que se mostró en anteriores estudios.

Dataset	Instancias	Atributos	Espacio de Búsqueda	AprioriDS-I-EO	DSEOFP-Tree
Mushroom	8,124	22	$1.6 \cdot 10^9$	$17.37 \cdot 10^3$	12.57
Nursery	12,960	8	$2.3 \cdot 10^5$	12.6	11.9
krvsk	28,056	6	$2.7 \cdot 10^4$	13.3	11.0
Adult-3	48,842	14	$3.0 \cdot 10^7$	71.02	14.61
Shuttle-3	58,000	9	$2.1 \cdot 10^4$	15.94	13.20
Fars-3	100,968	29	$1.8 \cdot 10^9$	$16.30 \cdot 10^3$	98.99
Poker	1,025,010	10	$2.0 \cdot 10^8$	72.8	46.3

Tabla 1: Tiempos de ejecución en segundos de nuestros algoritmos sobre *datasets* reales. Se ha usado un umbral de α_3, y se han extraido 100 subgrupos.

5. Conclusión

En este trabajo, se han propuesto dos algoritmos exhaustivos para la extracción de subgrupos en *Big Data*. Para abordar una enorme cantidad de datos se ha usado Apache Spark. AprioriDS-I-EO es un algoritmo basado en Apriori. Mientras que DSEOFP-Tree es un algoritmo basado en SD-MAP, SD-MAP* y DpSubgroup. Los dos algoritmos han sido probados sobre *Big Data*, considerando cualquier tipo de variables objetivo. En ambos se han usado EOs para reducir el espacio de búsqueda garantizando que ningún subgrupo interesante es perdido. Ambos se han comparado con otros algoritmos eficientes de DS, mostrando que nuestras propuestas superan a éstos sobre grandes cantidades de datos. El estudio experimental demuestra que los algoritmos se comportan bastante bien cuando tanto el espacio de búsqueda crece (Se ha usado un espacio de búsqueda de $1.276 \cdot 10^{15}$ subgrupos) como cuando se usan más de 10^7 instancias.

Agradecimientos

El presente trabajo ha sido financiado por el Ministerio de Innovación y Ciencia, y los fondos FEDER, bajo el proyecto TIN-2014-55252-P.

Referencias

1. F. Herrera, C. J. Carmona, P. González, and M. J. Jesus, "An overview on subgroup discovery: foundations and applications," *Knowledge and Information Systems*.
2. B. Kavšek, N. Lavrač, and V. Jovanoski, *5th International Symposium on Intelligent Data Analysis, IDA*, 2003, ch. APRIORI-SD: Adapting Association Rule Learning to Subgroup Discovery, pp. 230–241.
3. J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *SIGMOD Rec.*, vol. 29, no. 2, pp. 1–12, May 2000.
4. J. M. Luna, J. R. Romero, C. Romero, and S. Ventura, "On the use of genetic programming for mining comprehensible rules in subgroup discovery," *IEEE Transactions on Cybernetics*, vol. 44, no. 12, pp. 2329–2341, Dec 2014.
5. H. Grosskreutz, S. Rüping, and S. Wrobel, *ECML PKDD 2008, Proceedings*. Springer Berlin Heidelberg, 2008, ch. Tight Optimistic Estimates for Fast Subgroup Discovery.
6. W. Klösgen, "Advances in knowledge discovery and data mining." American Association for Artificial Intelligence, 1996, ch. Explora: A Multipattern and Multistrategy Discovery Assistant, pp. 249–271.
7. J. M. Luna, "Pattern mining: current status and emerging topics," *Progress in Artificial Intelligence*, pp. 1–6, 2016.
8. F. Lemmerich, M. Atzmueller, and F. Puppe, "Fast exhaustive subgroup discovery with numerical target concepts," *Data Mining and Knowledge Discovery*, 2015.
9. M. Atzmueller and F. Puppe, "Sd-map - a fast algorithm for exhaustive subgroup discovery," in *ECML/PKDD 2006*, ser. Lecture Notes on Computer Science, vol. 4213. Springer, 2006, pp. 6–17.
10. M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, Berkeley, CA, USA, 2010.