Análisis eficiente de Transacciones para la mejora de Recomendaciones Web

Enrique Lazcorreta
Instituto Universitario Centro de Investigación Operativa (CIO)
Universidad Miguel Hernández
enrique@umh.es

Federico Botella
Instituto Universitario Centro de Investigación Operativa (CIO)
Universidad Miguel Hernández
federico@umh.es

Antonio Fernández-Caballero
Instituto de Investigación en Informática de Albacete (I3A)
Universidad de Castilla-La Mancha
caballer@dsi.uclm.es

RESUMEN

Si trabajamos con grandes colecciones de datos y queremos obtener en poco tiempo la información más relevante que contienen podemos recurrir a la extracción de patrones mediante Minería de Datos. Entre los patrones más utilizados están las Reglas de Asociación, que miden la co-ocurrencia de ítems en grandes colecciones de transacciones. Tras muchos ensayos hemos encontrado un tipo de transacciones que pueden ser tratadas de un modo más eficiente que el que se han venido utilizando hasta hoy día. En este trabajo hemos aplicado un nuevo método a este tipo de transacciones, lo que nos ha permitido obtener en nuestros primeros ensayos tiempos de ejecución mucho más rápidos y que ofrecen más información que la obtenida con los algoritmos clásicos de Minería de Reglas de Asociación. Esto nos permitirá mejorar los tiempos de respuesta de un sistema de recomendación web para ofrecer respuestas a nuestros usuarios en tiempo real.

.

Categories and Subject Descriptors

H4.0 **[Information Systems Applications]**: Miscellaneous; H3.3 **[Information Storage and Retrieval]**: General; F.2.2 **[Analysis of algorithms and problem complexity]** Nonnumerical Algorithms and Problems - *Pattern matching*.

General Terms

Algorithms, Measurement, Experimentation.

Keywords

Association Rules Mining (ARM), Sistemas de Recomendación (RS), Modelado de transacciones

1. INTRODUCCIÓN

Desde sus inicios la Minería de Reglas de Asociación (ARM) [1] se ha protegido del *dilema del ítem raro* mediante la definición de un soporte mínimo que impide estudiar los ítems de menor soporte (ítems raros) para evitar problemas de desbordamiento de memoria y poder ofrecer resultados en poco tiempo. En la evolución del ARM hay múltiples aportaciones que alivian este problema, pero siempre a costa de renunciar al estudio de algunas relaciones existentes entre los datos en estudio.

En los primeros trabajos sobre ARM se trataba de evitar la búsqueda de relaciones entre los ítems raros, [1,2,3]. En [4] proponen el uso de múltiples soportes mínimos para que un itemset sea considerado frecuente sólo si su soporte es grande en relación al soporte de los ítems que lo forman. De este modo se ahorra espacio en memoria que puede ser utilizado para analizar ítems con menor soporte. El uso de un soporte relativo para cada itemset basado en la confianza de las reglas que generan sus ítems, lo que permite guardar información de los itemsets con menor soporte que el soporte mínimo que superen su soporte relativo [5], garantizando que las reglas que generan sean de calidad. En [6] se propone una mejora del algoritmo propuesto en [4] para dotarlo de escalabilidad. En [7] trabajan con las mismas ideas de soporte múltiple pero reducen el número de itemsets a guardar en función de su *lift*, que mide la mejora que produce la presencia de un ítem en el soporte del resto de ítems que contiene. En [8] utilizan múltiples soportes basados en la distribución conjunta de los ítems y no sólo en su distribución individual como proponían en [4]. Todas estas aportaciones reducen el número de itemsets almacenados en memoria, con lo que pueden almacenarse otros itemsets menos frecuentes pero con mejor información sobre la población en estudio.

Cuando trabajamos con un número grande de ítems diferentes y una gran cantidad de transacciones el dilema del ítem raro sólo es resoluble utilizando mucho tiempo y/o potentes computadores. Lo que sorprende es que este dilema aparezca en colecciones de datos con dimensiones reducidas, como ocurre con los repositorios utilizados en los trabajos analizados sobre esta temática destacando `chess` y `mushroom`. [9,10,11].

`chess` recoge información sobre la posición final de una serie de piezas en el juego del ajedrez. Contiene tan solo 75 ítems distintos en un total de 3196 transacciones. `mushroom` recoge el valor de una serie de atributos medidos en ciertas setas, utilizando 119 ítems distintos en 8124 transacciones.

Todos los investigadores que han trabajado con estas colecciones han tratado sus transacciones del mismo modo que la clásica transacción formada por la "cesta de la compra". En nuestra investigación comprobamos que se puede dar un tratamiento diferente sin perder las ventajas del ARM ni aplicar nuevos algoritmos.

En la sección 2 daremos una definición formal para este tipo de transacciones. En la sección 3 mostraremos un método que resuelve el dilema del ítem raro en este tipo de transacciones utilizando los algoritmos clásicos de ARM. En la sección 4 exponemos los resultados experimentales obtenidos con `chess` y `mushroom` y en la sección 5 planteamos las conclusiones de este artículo.

2. TIPOS DE TRANSACCIONES

Una transacción es una serie de elementos o características (que llamaremos ítems) que han sido observados conjuntamente bajo ciertas circunstancias. Esta simple definición permite modelar muchas colecciones de datos mediante transacciones, desde la clásica ”cesta de la compra" hasta el estudio de una sesión de navegación web, como si se tratara de un simple conjunto de páginas web visitadas conjuntamente, pasando por la observación de las características de individuos que tienen enfermedades similares o la detección de fraude en el uso de tarjetas de crédito.

Si suponemos que no es casual el hecho de formar una transacción agrupando unos ítems y no otros, y observamos un conjunto grande de transacciones, no es difícil pensar que encontraremos en ellas conjuntos de ítems que se repiten, conjuntos que llamaremos itemsets frecuentes o patrones. Por ejemplo, en la cesta de la compra estos patrones nos pueden sugerir que “el cliente que compra el producto *A* suele comprar conjuntamente el producto *B*". En el análisis de navegación en un portal web podemos encontrar la regla que sugiere que “el usuario que visita la página *A* suele acudir en la misma visita a la página *B*" o incluso que “la página *A* debe tratar sobre el mismo tema que la página *B”*, pues observamos que los usuarios las visitan conjuntamente con mucha frecuencia.

Es importante destacar que ARM suele trabajar sin conocimiento previo de la población en estudio, dejando que sean los datos quienes muestren esa información. Todos los algoritmos de ARM buscan extraer el máximo conocimiento de una gran cantidad de transacciones en el menor tiempo posible. Al aplicarlos se pueden hacer ajustes para cada estudio concreto, generalmente en base a estudios previos hechos sobre la misma población. Pero al plantearlos se debe procurar que sean aplicables a todas las poblaciones en estudio.

A lo largo de su evolución, ARM se ha ido desarrollando en base a conceptos teóricos y a su aplicación en ciertos repositorios. Muchos desarrollos han resultado poco competitivos al ser aplicados en repositorios concretos, lo que genera una constante duda sobre si existe un algoritmo mejor que otro o si todo depende de los datos sobre los que son aplicados. Es el caso de `chess` y `mushroom`: pese a tener dimensiones reducidas no se suelen utilizar con soporte mínimo bajo, pues su análisis mediante ARM produce una explosión de reglas que o bien no pueden ser guardadas en memoria o bien tardan tanto tiempo en obtenerse que se pierde su utilidad en servicios que deben ser ágiles, como los Sistemas de Recomendación.

En ninguno de esos trabajos se ha tenido en cuenta que realmente estamos analizando dos tipos bien diferenciados de transacciones:

- la clásica “cesta de la compra” de la que surge el ARM
- y las transacciones que llamaremos en este artículo “de tipo II”.

Para entenderlo mejor nos referiremos a la colección de datos `mushroom`. En su diseño se consideraron 22 atributos diferentes (color, textura…) de un total de 8124 setas diferentes, clasificadas cada una de ellas como “venenosa” o “comestible”. Para dar a estos datos forma de transacción y poder estudiarlos mediante ARM se asigna un código único a cada uno de los pares atributo-valor. Así, los códigos 1 y 2 se corresponden a la clasificación realizada. Los códigos 3, 4, 5, 6, 7 y 8 indican el color. Cada seta es observada y se anota mediante una transacción su clasificación, su color, etc. de modo que todas las transacciones están compuestas por 23 ítems (su clase más el valor de los 22 atributos).

Una característica de este tipo de transacciones que la diferencia de la cesta de la compra es la dependencia estructural entre los ítems de la población en estudio. Si una transacción tradicional contiene un ítem determinado no podemos asegurar que no contenga cualquier otro ítem de la población. Sin embargo si una transacción formada por códigos atributo-valor contiene el código de un atributo-valor concreto sabemos con seguridad que no contendrá otro código perteneciente al mismo atributo. En el caso de `mushroom`, si una transacción contiene el código 1 (*venenosa*) no puede contener por definición el código 2 (*comestible*).

Otra característica de este tipo de transacciones es la gran densidad de su matriz de correlación. En la práctica nos encontramos con que la dificultad de trabajar con transacciones de este tipo se debe a la gran cantidad de relaciones existentes en muchas colecciones de datos utilizadas. Cada atributo-valor está relacionado con casi todos los valores del resto de atributos, hecho que no ocurre en la clásica cesta de la compra. Es esto lo que provoca la falta de recursos y el exceso de tiempo empleado cuando intentamos analizar a fondo estas colecciones de datos con los algoritmos clásicos de ARM, si una transacción de este tipo no tuviera una matriz de correlación densa y tuviera un número reducido de ítems su análisis no presentaría ningún problema.

En [12] convierten las sesiones de navegación de los usuarios de un portal de comercio electrónico en transacciones del tipo II y utilizan las reglas de asociación obtenidas para sugerir mejoras en el diseño del portal. Los atributos estudiados en este análisis recogen información generada por los usuarios del portal: navegador, tipo de visitante, palabras clave usadas, fuente, etc. Todos los atributos se han dividido en un pequeño conjunto de valores representativos y para formar las transacciones han anotado el valor concreto de cada atributo en una visita al portal.

Otros portales de comercio electrónico pueden modelar sus datos mediante transacciones del tipo II, de modo que se beneficien de las ventajas propuestas en este trabajo. Por ejemplo en un portal de venta de muebles se podría utilizar en el diseño de una habitación. Si el usuario selecciona un modelo determinado de cama se podría analizar qué otros muebles combinan el resto de usuarios con ese modelo de cama para mejorar las recomendaciones del portal al usuario. Si el análisis se hace usando algoritmos suficientemente rápidos se podrían hacer las recomendaciones en tiempo real, conforme el usuario va haciendo sus selecciones.

3. ANALISIS EFICIENTE DE TRANSACCIONES DEL TIPO II

Las transacciones de tipo II pueden presentar problemas en su análisis cuando existe relación entre la mayoría de sus ítems. Sin embargo, antes de proceder a su análisis podemos asegurar que los distintos ítems que forman un atributo son, por definición, excluyentes. Esta característica no ha sido utilizada en ninguno de los estudios previos sobre ARM que hemos revisado, por lo que proponemos su uso para que no se pierda información contenida en los datos y además nos permitirá abordar problemas que con las otras propuestas no se pueden resolver por falta de recursos de memoria o bien por tardar mucho más tiempo del aceptado por un usuario medio ante un Sistema de Recomendación.

La relación existente entre los códigos asignados a los distintos valores de un atributo es una relación establecida antes de la toma de los datos, por lo que no depende de los datos obtenidos. Todas

las transacciones de tipo II tienen la misma longitud, el número de atributos utilizado (y de clases en su caso). Y en todas ellas se da la característica de que si contienen un valor determinado de un atributo no pueden contener ninguno del resto de valores asignados a ese atributo en el experimento.

Para tratar convenientemente estas transacciones sólo hay que prescindir de uno de los ítems utilizados para modelar los distintos valores de cada atributo. Si sumamos el soporte de cada uno de los valores de un atributo siempre obtendremos el mismo número, el número de transacciones en estudio (*N*). Si un atributo sólo tuviera dos valores y el primero se presentara en *x* transacciones el segundo estará en las *N*-*x* restantes.

Una vez eliminados estos valores del repositorio en estudio podemos aplicar cualquiera de los algoritmos planteados para ARM, tanto los algoritmos tipo Apriori con soporte mínimo unitario como los que usan múltiples soportes mínimos descritos en la primera sección de este trabajo.

Al eliminar todos los ítems propuestos del repositorio de transacciones de tipo II no estamos eliminando ninguna información que no podamos reconstruir una vez analizado el análisis, simplemente reducimos sus dimensiones provocando un análisis más rápido y en muchas ocasiones más profundo.

Supongamos que tenemos una población con tres atributos en estudio cuyos valores codificamos con {1,2} para el primer atributo, {3,4,5} para el segundo y {6,7} para el tercero. Suponiendo que tenemos un repositorio de transacciones (*D*) presentando máxima variabilidad, en el algoritmo Apriori clásico [2] se generará el árbol *L* mostrado en la figura 1 para recoger todas las co-ocurrencias encontradas en el repositorio:

1 (n_1)	-	2 (n_{12})		
	-	3 (n_{13})	-	6 (n_{136})
			-	7 (n_{137})
	-	4 (n_{14})	-	6 (n_{146})
			-	7 (n_{147})
	-	5 (n_{15})	-	6 (n_{156})
			-	7 (n_{157})
	-	6 (n_{16})		
	-	7 (n_{17})		
2 (n_2)	-	3 (n_{23})	-	6 (n_{236})
			-	7 (n_{237})
	-	4 (n_{24})	-	6 (n_{246})
			-	7 (n_{247})
	-	5 (n_{25})	-	6 (n_{256})
			-	7 (n_{257})
	-	6 (n_{26})		
	-	7 (n_{27})		
3 (n_3)	-	4 (n_{34})		
	-	5 (n_{35})		
	-	6 (n_{36})		
	-	7 (n_{37})		
4 (n_4)	-	5 (n_{45})		
	-	6 (n_{46})		
	-	7 (n_{47})		
5 (n_5)	-	6 (n_{56})		
	-	7 (n_{57})		
6 (n_6)	-	7 (n_{67})		
7 (n_7)				

Fig. 1. Árbol *L*

Aunque trabajemos con millones de transacciones en *D* el árbol *L* de la figura 1 guarda toda la información necesaria para obtener todas las reglas de asociación de *D*.

Si tenemos en cuenta lo planteado en nuestra propuesta y eliminamos del repositorio los ítems 1, 3 y 6, el árbol *L* se reducirá al mostrado en la figura 2.

2 (n_2)	-	4 (n_{24})	-	7 (n_{247})
	-	5 (n_{25})	-	7 (n_{257})
	-	7 (n_{27})		
4 (n_4)	-	7 (n_{47})		
5 (n_5)	-	7 (n_{57})		
7 (n_7)				

Fig. 2. Árbol *L* reducido.

Pasamos así de tener 40 nodos a sólo 11 lo que permite obtener, en mucho menos tiempo y utilizando mucho menos espacio de memoria, todas las relaciones existentes entre cualquier par de ítems del repositorio reducido.

Para obtener información sobre los ítems que no están presentes en el árbol *L* obtenido basta con recurrir al diseño utilizado en la codificación de los valores de cada atributo en estudio:

- El soporte del ítem 1 se obtiene restando a *N* el soporte del ítem 2. $n_1 = N - n_2$
- $n_3 = N - n_4 - n_5$
- $n_{23} = n_2 - n_{24} - n_{25}$
- $n_{13} = n_3 - n_{23}$
- …

El soporte de cualquier nodo del árbol *L* completo se obtiene mediante un par de búsquedas dirigidas en el árbol reducido y una diferencia, por lo que apenas consume recursos de tiempo o memoria.

3.1 Refinamiento

Si queremos reducir aún más el tiempo empleado en el análisis podemos forzar a que la reducción de ítems en *D* sea máxima sin que cambien los resultados obtenidos. En el ejemplo anterior hemos eliminado al azar los ítems 1, 3 y 6. Si aprovechamos la primera lectura de *D* para comprobar qué ítem es más frecuente en cada atributo podemos seleccionar ese ítem para eliminarlo del estudio, con lo que conseguiremos reducir más el tamaño de *D* sin perder ninguna información útil.

Esta reducción de *D* implicará que en cada transacción habrá menos ítems que estudiar y, por tanto, muchos menos itemsets que recorrer para contar su soporte y menos búsquedas en el árbol *L* para actualizarlo. Estos son los procesos que más tiempo consumen en los algoritmos de ARM por lo que, sin perder información, ganaremos en tiempo de respuesta al realizar el análisis.

Esta reducción puede también provocar un tamaño menor del árbol *L*, ya que los ítems eliminados son los más frecuentes y, por tanto, los que más relaciones suelen presentar con otros ítems del estudio llevado a cabo.

3.2 Ítems no relacionados

A menudo, en el análisis clásico de obtención de reglas de asociación nos vernos forzados a utilizar un soporte mínimo superior al 0% para poder completarlo. Esto tiene como consecuencia que si no encontramos ninguna relación entre dos ítems concretos puede deberse a dos explicaciones distintas:

- el número de veces que aparecen relacionados en *D* es menor al soporte mínimo fijado
- realmente no aparecen relacionados en ninguna transacción de *D*.

El método que proponemos permitirá reducir el soporte mínimo del estudio hasta el 0% en muchos casos, lo que producirá un nuevo conocimiento que no podíamos adquirir antes: si no encontramos ninguna relación entre dos ítems es porque realmente no existe dicha relación en el repositorio. En algunos estudios esta relación es importante.

4. EXPERIMENTACIÓN

Hemos utilizado en experimentos anteriores transacciones de este tipo, concretamente `chess` y `mushroom`, disponibles en el UCI Machine Learnning Repository FIMI[1], pero siempre hemos tenido que renunciar al análisis completo de los datos debido a la alta co-ocurrencia de sus ítems y la gran cantidad de itemsets largos que producen. En otros trabajos consultados que utilizan estos datos también han renunciado al análisis completo del repositorio utilizando soporte mínimo superior al 0%.

En la tabla 1 pueden observarse las características de nuestros experimentos. Al ejecutar Apriori clásico [2] partimos de soporte mínimo 0. Es valor se fue incrementando, hasta conseguir que se ejecutara el algoritmo por completo, sin que se desbordara la memoria. A continuación utilizamos nuestra propuesta eliminando en el caso (1) un valor para cada atributo seleccionado aleatoriamente, y en el caso (2) eliminando el valor más presente en *D* para cada atributo. En ambos casos pudimos trabajar con un soporte mínimo del 0% con lo que se obtuvo información sobre el 100% de los ítems en estudio.

	Apriori clásico			Tipo II (1)	Tipo II (2)
	Soporte mínimo	Ítems raros	Tiempo (sg)		
`chess`	30%	33.3%	7660	192	117
`mushroom`	1%	23%	2130	183	140

Tabla 1. Comparativa entre el algoritmo clásico y nuestra propuesta

Lo importante en ambos casos es que hemos podido realizar un análisis completo sin renunciar a la información que poseen los datos sobre todos los ítems en estudio y en tiempos razonables.

Además de encontrar todas las relaciones en que intervienen ítems poco frecuentes podemos detectar qué ítems no están relacionados, algo que con los algoritmos clásicos no era posible debido al uso de soporte mínimo no nulo.

5. CONCLUSIÓN Y TRABAJO FUTURO

Los experimentos realizados con las transacciones que hemos definido tipo II nos muestran que podemos obtener un gran aumento de eficiencia de los algoritmos de ARM cuando analizamos este tipo de transacciones. El método propuesto implica una reducción de una gran cantidad de los datos que queremos procesar sin ninguna pérdida de la información que contienen, permitiendo aplicar a los nuevos datos cualquiera de los algoritmos de ARM existentes. Incluso en muchos experimentos podemos descubrir lo que algunos autores llaman reglas negativas de asociación, es decir, la ausencia de relación entre ciertos ítems del repositorio en estudio, sin necesidad de modificar los criterios de búsqueda de los algoritmos.

Como trabajo futuro vamos a aplicar esta nueva metodología a portales web pequeños para encontrar las reglas de asociación que verifican sus usuarios en tiempos muy pequeños, al objeto de intentar generar en tiempo casi-real información relevante que nos permitirá afinar un Sistema de Recomendaciones Web..

6. AGRADECIMIENTOS

Este trabajo ha sido financiado en parte por el Ministerio de Economía y Competitividad / FEDER bajo el proyecto TIN2010-20845-C03-01.

7. BIBLIOGRAFÍA

[1] Agrawal, R., Imielinsky, T., and Swami, A. Mining Association Rules between sets of items in large databases. *ACM SIGMOD Int. Conf. on Management of data.* (1993), 207-216.

[2] Agrawal, R., Srikant, R. *Fast algorithms for mining association rules*. Proc. of the 20th Very Large Data Bases Conference. VLDB, Morgan Kaufmann Publishers Inc., (1994) 487-499.

[3] Park, J., Chen, M. and Yu, P. *Using a Hash-Based Method with Transaction Trimming for Mining Association Rules*. IEEE Transactions on Knowledge and Data Engineering, 9-5 (sept 1997). 813-825.

[4] Liu, B., Hsu, W., Ma, Y. *Mining association rules with multiple minimum supports*. In: KDD '99: Proceedings of the 5th ACM SIGKDD Int. Conf. on Knowledge discovery and data mining. (1999). 337-341.

[5] Yun, H., Ha, D., Hwang, B., Ryu, K. H. *Mining association rules on significant rare data using relative support*. J. Syst. Softw. 67-3 (2003), 181-191.

[6] Hu, Y.-H., Chen, Y.-L. Mining association rules with multiple minimum supports: a new mining algorithm and a support tuning mechanism. Decision Support Systems 42-1 (2006), 1-24.

[7] Tseng, M.-C., Lin, W.-Y. Efficient mining of generalized association rules with non-uniform minimum support. Data & Knowledge Engineering 62-1 (2007), 41-64.

[8] Kiran, R. U., Reddy, P. K. *An improved multiple minimum support based approach to mine rare association rules*. Tech. rep., IEEE Symposium on Computational Intelligence and Data Mining (jan. 2009).

[9] Dong, J. and Han, M. *BitTableFI: An efficient mining frequent itemsets algorithm*. Knowledge-Based Systems 20-4 (may 2007). 329-335.

[10] Luo, K. and Zhang, XM. *An Efficient Frequent Itemset Mining Algorithm*. Int. Conf. on Machine Learning and Cybernetics vol. 2 (aug. 2007). 756-761.

[11] Borgelt, C. *Efficient Implementations of Apriori and Eclat.* Workshop of Frequent Item Set Mining Implementations (2003)

[12] Carmona, C.J., Ramírez-Gallego, S., Torres, F., Bernal, E., del Jesús, M.J. y García, S. *Web usage mining to improve the design of an e-commerce website: OrOliveSur.com*. Expert Systems with Applications (april 2012).

[1] http://fimi.cs.helsinki.fi/