Reglas de Asociación

José Tomás Palma Méndez

Dept. de Ingeniería de la Información y las Comunicaciones. Universidad de Murcia
Contacting author: `jtpalma@um.es`

1. Introducción

En esta práctica vamos a analizar las librerías `arules` y `arulesViz` centradas en el descubrimiento de reglas de asociación y en su visualización.

2. Manipulación de conjuntos de datos

El paquete `arules` nos ofrece diferentes funciones que permiten leer y escribir bases de datos de transacciones: `read.transactions()` y `write()`. La función `read.transactions()` nos permite leer ficheros de transacciones en los dos formatos que comúnmente se suelen utilizar: una transacción por linea (cesta de la compra) o una linea por artículo o item. Esta función tiene los siguientes parámetros:

- **file**: nombre del fichero a leer.
- **format**: string indicando el formato de la base de datos de transacciones, "`basket`" o "`single`".
- **sep**: para indicar qué carácter hace de separador de los campos (por defecto un espacio).
- **cols**: para el formato "`basket`" es un entero indicando la columna que contiene los identificadores de las transacciones. Para el formato "`single`" pueden sor los nombres de las dos columnas, o dos números indicando cuál es la columna con las transacciones y cuál con los items, respectivamente.
- **rm.duplicates**: valor lógico indicando si se eliminan los items duplicados en una misma transacción.
- **quote**: caracter que se utiliza como comillas.
- **skip**: número de líneas que hay que saltar desde el comienzo del fichero.

Por ejemplo, si queremos leer un fichero con el formato cesta de la compra, bastaría con la siguiente instrucción:

```
> transacciones <- read.transactions("1000i.csv", format = "basket" )
```

Donde el objeto `transacciones` es de tipo `transactions`.

> Ejercicio 1. Examina el contenido de los ficheros `titanic1` y `titanic2`:
>
> 1.a) ¿En qué formato están almacenados los datos en cada fichero?

1.b) Crea dos objetos llamados `titanic.basket` y `titanic.single` que contengan los datos de los ficheros anteriormente mencionados.

Además de poder leer bases de datos de transacciones a través de la función `read.transactions()`, el paquete `arules` nos permite transformar un objeto de la clase `data.frame` en un objeto de la clase `transactions`. Por ejemplo, si queremos convertir el `data frame tabla` en un objeto de la clase `transactions` podemos utilizar la siguiente instrucción:

```
> transacciones <- as(tabla, "transactions" )
```

Recordad que los objetos de la clase `transactions` sólo trabajan con información booleana, es decir, con la presencia o no de cada uno de los items en la transacción. Esto nos obliga a qué, en principio, el **data.frame** sólo contenga atributos booleanos. Por lo tanto, si tenemos datos numéricos estos deben ser discretizados. Una vez discretizados, estos atributos y los categóricos son transformados automáticamente. Por ejemplo, si tenemos un atributo `fiebre` que puede tomar los valores `alta`, `normal` y `baja`, se crearan los siguientes atributos binarios: `fiebre=alta`, `fiebre=normal` y `fiebre=baja`.

Para poder discretizar un atributo numérico, el paquete `arules` nos ofrece la función `discretize()`, que tiene los siguientes parámetros:

- El nombre del fichero como una string.
- **method** para indicar el tipo de discretización: "`interval`" para intervalos de la misma anchura, "`frecuency`" para intervalos con la misma frecuencia, "`cluster`" para discretizar mediante el k-means y "`fixed`" para discretizar definiendo los puntos de corte de cada intervalo.
- **categories**: un entero para indicar el número de categorías o un vector de enteros indicando los límites de cada intervalo (para `method = fixed`).
- **labels:** vector de caracteres para definir las etiquetas para cada una de las categorías.
- **ordered**: TRUE si queremos que indicar que las categorías definidas tiene un orden.
- **onlycuts**: TRUE si queremos que la función devuelva sólo los puntos de corte.

Por ejemplo, si quiere discretizar el atributo fiebre en tres categorías, utilizando el algoritmo k-means podemos utilizar la siguiente instrucción:

```
> fiebre_dis <- discretize(tabla$fiebre,
                           method = "cluster",
                           categories = 3,
                           ordered = TRUE)
```

Ejercicio 2. Lee el fichero `titanic.csv` y examina los atributos que tiene y el tipo de cada uno de ellos.

2.a) Elimina la columna `tarifa`, esa información la tenemos discretizada en el atributo `clase`.
2.b) Discretiza el atributo `edad` en tres categorías: `Child` (menor que 18), `Adult` (entre 18 y 65) y `Old` (mayor que 65).
2.c) Recodifica el atributo `superviviente` a `Si` para el valor 1 y `No` para el valor 0.
2.d) Crea un objeto de la clase `transactions`, llamado `titanic.trans`, a partir del `data frame titanic`

Para hacer una primera inspección visual sobre los objetos de la clase `transactions` tenemos las funciones `itemFrequency()` y `itemFrequencyPlot()` que muestran la frecuencia de cada uno de los items en una base de transacciones.

Por último, el paquete `arules` nos proporciona la función `write()`, para generar ficheros que contengan la información sobre las transacciones y los items. La función `write()` tiene los siguientes parámetros:

- El objeto de tipo `transactions` que se va a exportar.
- **file**: nombre del fichero que se va a generar.
- **format**: string indicando el formato de exportación de la base de datos de transacciones, "`basket`" o "`single`".
- **sep**: para indicar qué caracter hace de separador de los campos (por defecto un espacio).
- **quote**: TRUE si queremos que los campos aparezcan entre comillas.

Por ejemplo, con `quote=TRUE` y `sep=","`, estaríamos generando un fichero en con el formato `csv`.

Ejercicio 3. Utilizando la función `write()` genera dos ficheros en formato `csv` denominados `titanic.basket` y `titanic.single`, cada uno conteniendo las transacciones en los formatos indicados en su nombre.

3. Descubrimiento de reglas de asociación

El paquete `arules` nos ofrece dos funciones para descubrir itemsets frecuentes y reglas de asociación: `apriori()`[Agrawal *et al.*, 1993] y `eclat()`[Zaki *et al.*, 1997]. Ambas funciones se basan en las implementaciones desarrolladas por Christian Borgelt [Borgelt, 2003]. La función `eclat()` sólo nos devuelve los itemsets frecuentes, con lo que es necesario utilizar la función `ruleInduction()` para generar las reglas de asociación a partir del conjunto de itemsets frecuentes. La función `apriori()` nos permite obtener tanto reglas de asociación como itemsets frecuentes.

La función `apriori()` tiene los siguientes parámetros:

- Un objeto del tipo `transactions` con la base de datos de transacciones.
- **parameter**: lista en la que se indican los distintos parámetros del algoritmo.

- **appearance**: lista que nos permite definir patrones de reglas para restringir el espacio de búsqueda de reglas.
- **control**: lista que nos permite modificar la forma en la que se ejecuta el algoritmo.

A través del parámetro `parameter` podemos indicar el soporte, número máximo/mínimos de items en cada itemsets, el objeto del algoritmo (itemsets, reglas, ...), generar medidas de calidad adicionales, etc. El parámetro `appearance` nos permite definir qué items pueden aparecer (no aparecer) en los itemsets o reglas. El parámetro `control` non permite definir aspectos internos de algoritmo, como la ordenación de los itemsets, si se construye un árbol con las transacciones, aspectos relacionados con el uso de memoria, etc.

Por ejemplo, para generar el conjunto de itemsets frecuentes, podemos utilizar el siguiente código:

```
> titanic.itemsets <- apriori(titanic.trans,
                              parameter = list(target = "frequent itemset"))

Apriori

Parameter specification:
 confidence minval smax arem  aval
         NA    0.1    1 none FALSE
 originalSupport support minlen maxlen
            TRUE     0.1      1     10
            target   ext
 frequent itemsets FALSE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 104

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[13 item(s), 1043 transaction(s)] done [0.00s].
sorting and recoding items ... [11 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 5 done [0.00s].
writing ... [87 set(s)] done [0.00s].
creating S4 object  ... done [0.00s].
```

Si lo que queremos es generar reglas de asociaciones con un soporte superior a 0.2, hubiera sido suficiente con:

```
> titanic.rules <- apriori(titanic.trans,
                           parameter = list(support = 0.2))

Apriori
```

```
Parameter specification:
 confidence minval smax arem  aval
       0.8    0.1    1 none FALSE
 originalSupport support minlen maxlen target
           TRUE     0.2      1     10  rules
   ext
 FALSE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 208

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[13 item(s), 1043 transaction(s)] done [0.00s].
sorting and recoding items ... [10 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 5 done [0.00s].
writing ... [43 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
```

Supongamos qué solo nos interesan generar reglas que solo contengan en el consecuente la información relativa a la supervivencia:

```
> reglas.sup <- apriori(titanic.trans,
                        appearance = list(rhs=c("superviviente=Si",
                                                "superviviente=No"),
                                          default="lhs"))

Apriori

Parameter specification:
 confidence minval smax arem  aval
       0.8    0.1    1 none FALSE
 originalSupport support minlen maxlen target
           TRUE     0.1      1     10  rules
   ext
 FALSE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 104

set item appearances ...[2 item(s)] done [0.00s].
set transactions ...[13 item(s), 1043 transaction(s)] done [0.00s].
sorting and recoding items ... [11 item(s)] done [0.00s].
```

```
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 5 done [0.00s].
writing ... [13 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
```

Con `rhs=c("superviviente=Si", "superviviente=No")` estamos diciendo que sólo queremos las reglas que contengan los items `superviviente=Si` y `superviviente=No` en el consecuente (`rhs`, right-hand side). Con `default="lhs"` estamos indicando que el resto de items que no se han incluido de forma explícita sólo pueden aparecer en el antecedente (`lhs`, left-hand side). Otro términos que se pueden utilizar son `none`, `both` y `items`.

Ejercicio 4. Partiendo de la base de transacciones `titanic.trans`:

- 4.a) Aplica las funciones `apriori()` y `eclat()` y genera los objetos `titanic.rules` y `titanic.eclat` respectivamente.
- 4.b) Indica cuáles son los valores de los parámetros por defecto.
- 4.c) Genera una tabla en R en la que en cada fila indique un valor real entre 0 y 1 (empezando por 0.1 y con incrementos de 0.1), el número de reglas generadas para un soporte igual a dicho número en la segunda columna y lo mismo pero para la confianza en la tercera columna. Para poder realizar esta cuestión hay que utilizar la función `length()` sobre el conjunto de reglas.
- 4.d) Representa gráficamente la tabla anterior.
- 4.e) Genera un conjunto de reglas de asociación para determinar qué items están relacionados con el sexo.

4. Análisis de los resultados

.

Existen muchas formas de explotar los resultados devueltos por las funciones `apriori()` y `eclat()` tenemos las funciones `inspect()` y `subset()`. La función `inspect()`, simplemente nos muestra por pantalla el conjunto de reglas/itemsets generados. Mediante los parámetros que nos ofrece podemos modificar la apariencia de las reglas.

```
> inspect(titanic.rules[1:5])

  lhs                 rhs                       support confidence      lift
1 {}               => {edad=Adult}           0.8398849  0.8398849 1.0000000
2 {clase=segunda}  => {embarque=Southampton} 0.2224353  0.8888889 1.1870821
3 {clase=segunda}  => {edad=Adult}           0.2166826  0.8659004 1.0309750
4 {clase=primera}  => {edad=Adult}           0.2483221  0.9184397 1.0935304
5 {sexo=mujer}     => {edad=Adult}           0.3000959  0.8108808 0.9654666
```

Utilizando la función `subset()` o accediendo directamente al objeto que contiene las reglas podemos seleccionar un subconjunto de las reglas de acuerdo con algún criterio. La función `subset()` requiere como primer parámetro el conjunto de reglas y el parámetro `subset` que consiste en una expresión lógica indicando la condición que tienen que cumplir las reglas/itemsets seleccionados. En la Tabla 1 se pueden apreciar los operadores permitios.

Operador	**Significado**
`&`	AND
`||`	OR
`%in%`	contiene cualquier de los siguientes elementos?
`%ain%`	contiene todos de los siguientes elementos?
`%pin%`	contiene parcialmente los siguientes elementos?

Tabla 1. Operadores lógicos permitidos en la función `subset()`

Por ejemplo, para encontrar el conjunto de reglas que nos permitan determinar cuál fue el destino de las personas qué embarcaron en Southampton, podemos utilizar el siguiente código

```
> reglas.sub = subset(titanic.rules,
                      subset = lhs %pin% "Southampton" &
                               rhs %pin% "superviviente")
> inspect(reglas.sub)

  lhs                        rhs                   support confidence     lift
1 {embarque=Southampton,
   sexo=hombre}           => {superviviente=No} 0.4074784  0.8220503 1.387376
2 {clase=tercera,
   embarque=Southampton,
   sexo=hombre}           => {superviviente=No} 0.2329818  0.8408304 1.419071
3 {edad=Adult,
   embarque=Southampton,
   sexo=hombre}           => {superviviente=No} 0.3614573  0.8490991 1.433026
```

Si quisiera saber cuál es fue el destino de los hombres adultos con un lift superior a 1.4, tendríamos que utilizar el siguiente código.

```
> reglas.sub2 = subset(titanic.rules,
                      subset = lhs %ain% c("sexo=hombre","edad=Adult") &
                               rhs %pin% "superviviente" &
                               lift > 1.4)
> inspect(reglas.sub2)

  lhs                        rhs                   support confidence     lift
1 {clase=tercera,
```

```
   edad=Adult,
   sexo=hombre}          => {superviviente=No} 0.2301055  0.8421053 1.421223
2 {edad=Adult,
   embarque=Southampton,
   sexo=hombre}          => {superviviente=No} 0.3614573  0.8490991 1.433026
```

Algunas operaciones de selección también se pueden hacer también operando directamente sobre el conjunto de reglas. Por ejemplo, las siguientes instrucciones son equivalentes:

```
> reglas.sop1 <- subset(titanic.rules, subset = support > 0.5)
> reglas.sop2 <- titanic.rules[quality(titanic.rules)$support > 0.5]
> match(reglas.sop1,reglas.sop2)

[1] 1 2 3 4 5

> setequal(reglas.sop1,reglas.sop2)

[1] TRUE
```

La función `match()` nos indica qué reglas/itemsets son idénticas en los dos conjuntos. Funciones parecidas a esta que nos permiten manipular conjuntos de reglas/itemsets son: `union()`, `intersect()` y `setequal()`. Otra función interesante es la función `sort()` que nos permite ordenar las reglas por las diferentes medidas de calidad. Por ejemplo, para ordenar en orden decreciente las reglas según su soporte:

```
> reglas.ord <- sort(titanic.rules, by = "support")
> inspect(head(reglas.ord))

  lhs                       rhs                       support confidence     lift
1 {}                     => {edad=Adult}           0.8398849  0.8398849 1.000000
2 {embarque=Southampton} => {edad=Adult}           0.6318313  0.8437900 1.004650
3 {sexo=hombre}          => {edad=Adult}           0.5397891  0.8569254 1.020289
4 {superviviente=No}     => {edad=Adult}           0.5119847  0.8640777 1.028805
5 {superviviente=No}     => {sexo=hombre}          0.5004794  0.8446602 1.340914
6 {superviviente=No}     => {embarque=Southampton} 0.4803452  0.8106796 1.082636
```

Para ordenar en orden decreciente sólo habría que incluir el parámetro `decreasing=FALSE`.

Como se puede observar, por defecto sólo se trabajo con tres medidas de calidad: soporte, confianza y lift. Sin embargo, gracias a la función `interestMeasures()` podemos obtener muchas más medidas. Por ejemplo, si queremos obtener las medidas para los índices de jacaard, coseno, kappa y el índice de correlación hay que ejecutar la siguiente instrucción:

```
> head(interestMeasure(titanic.rules,
                 c("jaccard", "cosine", "kappa", "phi"),
                 titanic.trans))

    jaccard    cosine     kappa         phi
1 0.8398849 0.9164524 -913668.0          NA
2 0.2864198 0.5138569 -408724.9  0.18660529
3 0.2480790 0.4726462 -359230.0  0.04098472
4 0.2880979 0.5211025 -374119.0  0.13040036
5 0.3298209 0.5382681 -447855.0 -0.06062401
6 0.3566215 0.5605051 -475506.0 -0.07955336
```

Si queremos añadir dichas medidas al conjunto de medidas ya registradas en el conjunto de reglas, bastaría con la siguiente instrucción:

```
> quality(titanic.rules) <- cbind(quality(titanic.rules),
                                  coseno = interestMeasure(titanic.rules,
                                                           c("cosine"),
                                                           titanic.trans))
> inspect(head(sort(titanic.rules, by ="coseno")))

  lhs                       rhs                    support confidence     lift    coseno
1 {}                     => {edad=Adult}       0.8398849  0.8398849 1.000000 0.9164524
2 {superviviente=No}     => {sexo=hombre}      0.5004794  0.8446602 1.340914 0.8192069
3 {embarque=Southampton} => {edad=Adult}       0.6318313  0.8437900 1.004650 0.7967239
4 {edad=Adult,
   sexo=hombre}          => {superviviente=No} 0.4410355  0.8170515 1.378940 0.7798470
5 {edad=Adult,
   superviviente=No}     => {sexo=hombre}      0.4410355  0.8614232 1.367526 0.7766128
6 {embarque=Southampton,
   sexo=hombre}          => {superviviente=No} 0.4074784  0.8220503 1.387376 0.7518815
```

Otro conjunto importante de funciones son aquellas que nos permiten determinar el tipo de las reglas/itemsets. Para ello tenemos las siguientes funciones booleanas: `is.subset()`, `is.superset()`, `is.maximal()`, `is.redundant()` e `is.close()`.

Ejercicio 5. Partiendo del del conjunto de reglas generado por el algoritmo `apriori`:

- 5.a) Selecciona el conjunto de reglas que permitan determinar el destino a partir de la ciudad de embarque ¿Cuántas reglas se han seleccionado?.
- 5.b) Del conjunto anterior, selecciona las reglas con una confianza superior al 0.83.
- 5.c) Calcula el índice gini, hyperlift e hyperConfidence. Agrégalos al conjunto de reglas `titanic.rules` y muestra por pantalla, para cada índice, las 5 reglas que lo tengan más alto.

5.d) Genera diferentes conjuntos de reglas para las reglas no redundantes y maximales.
5.e) Genera los conjuntos de los itemsets frecuentes maximales y cerrados.

5. Visualización

El paquete `aruleViz` nos ofrece un gran abanico de posiblidades de analizar visualmente un conjunto de reglas de asociación. Los distintos gráficos que podemos generar se realizan a través de la función `plot()`:

```
> plot(x ,method = ... , measure = "suport", shading = "lift",
     interactive = FALSE, data = ..., control = ....)
```

donde el `x` es el conjunto de reglas a visualizar, `method` es la técnica de visualización que vamos a utilizar, `interactive` indica si queremos realizar una exploración interactiva de las reglas o simplemente las queremos mostrar, `data` es la base de datos de transacciones (sólo necesaria para algunas técnicas) y `control` agrupa a diferentes parámetros para personalizar el gráfico.

5.1. Gráficos de dispersión

Es la forma más directa de visualizar un conjunto de reglas. Se representan las reglas en un gráfico bidimensional con una medida de calidad en cada eje. También se puede incorporar una tercera medida, que se representa mediante un código de colores (Figura 2).

```
> plot(titanic.rules)
```

Cualquier medida almacenada junto a las reglas (lo podemos saber con la instrucción `head(quality(titanic.rules))`) puede ser utiliza en el gráfico.

```
> plot(titanic.rules, measure = c("support", "lift"), shading = "coseno")
```

Un tipo interesante de gráfico de dispersión es el gráfico de dos claves (*two-key plot*), en el que el código de color se utiliza para indicar el número de items de la regla. Para obtener este gráfico basta con indicar `shading = "order"`. Los gráficos de dispersión también permiten la inspección interactiva.

Ejercicio 6. Partiendo del del conjunto de reglas generado por el algoritmo `apriori`:

6.a) Prueba la versión interactiva de este gráfico.

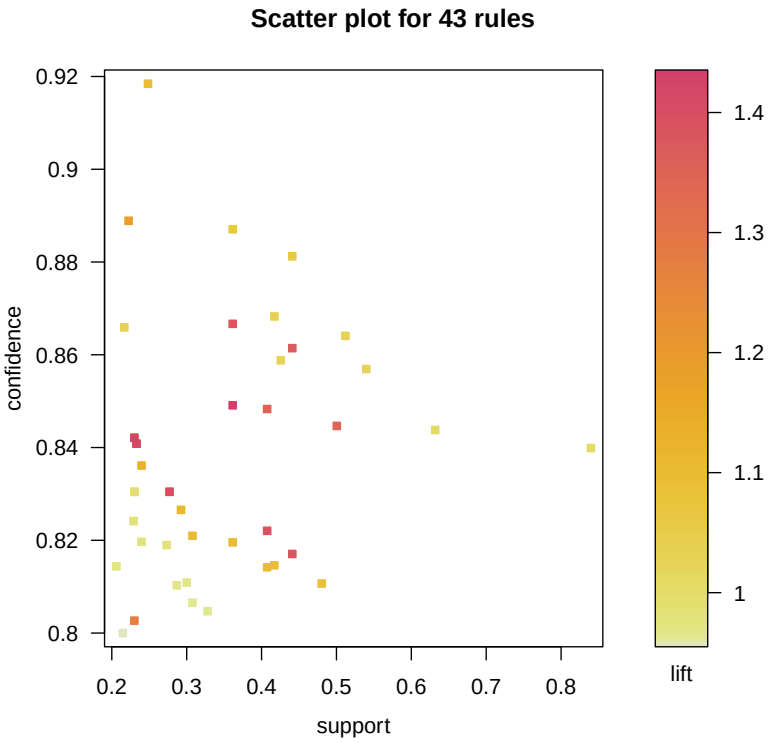

Figura 1. Gráfico de dispersión

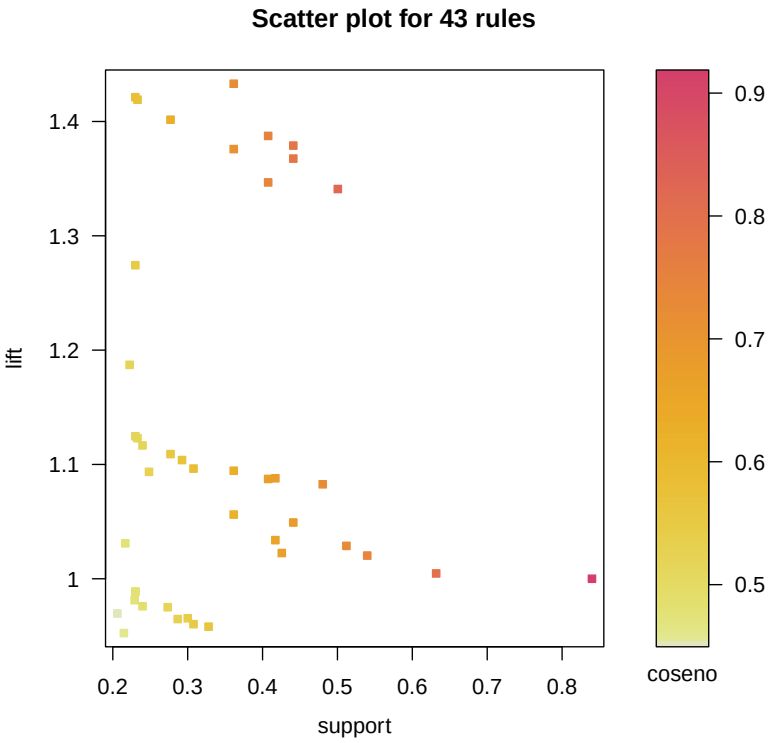

Figura 2. Gráfico de dispersión

6.b) Genera un gráfico de dos claves.

5.2. Gráficos basados en matrices

En este tipo de gráficos, los itemsets antecedentes y consecuentes se colocan en los ejes `x` e `y` respectivamente. Si dos itemsets forman parte de una regla, esto se marca en el gráfico con una indicación de la medida de calidad utilizada. Este gráfico permite utilizar dos medidas calidad diferentes.

En la Figura 3 podemos ver como incorporamos las medidas de calidad soporte y lift en el gráfico. Por cuestiones de espacio se ha suprimido la salida textual que genera esta gráfica en el que se indica qué itemsets se corresponden con cada índice de la figura.

```
> plot(titanic.rules, method = "matrix",measure = c("support", "lift"))
```

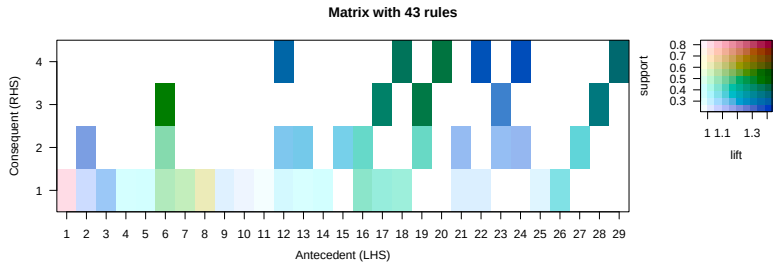

Figura 3. Gráfico matricial teniendo en cuenta el soporte y el lift.

Este tipo de representación también se puede hacer en 3 dimensiones con la opción `method = "matrix3D"` (en este caso sólo se puede utilizar un única medida de calidad). Para evitar la fragmentación en el gráfico se puede utilizar la opción `control = list(reorder=TRUE)`. Sólo la versión en dos dimensiones admite la inspección interactiva.

Ejercicio 7. Partiendo del del conjunto de reglas generado por el algoritmo `apriori`:

7.a) Representa dichas reglas en una matriz de 3 dimensiones para el soporte.
7.b) Prueba la versión interactiva.

5.3. Gráficos basados en matrices con datos agrupados

Los gráficos basados en matrices no son eficientes para visualizar grandes conjuntos de reglas, sobre todo teniendo en cuenta que muchas reglas tendrán solamente un

único antecedente o consecuente. Una buena forma de evitar este problema consiste en agrupar en la matriz aquellas reglas que sean similares. Para crear este gráfico debemos utilizar la opción `method = "grouped"` (ver Figura 4):

```
> plot(titanic.rules, method = "grouped")
```

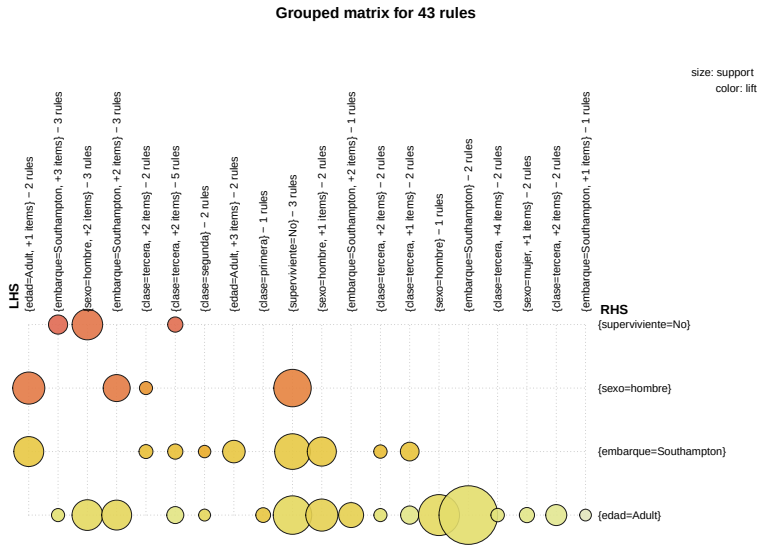

Figura 4. Gráfico matricial con agrupamiento de reglas.

Se puede modificar el número de grupos a través de la opción `k` en el parámetro `control`, por ejemplo `control=list(k=30)`. Este tipo de gráfico también permite la visualización interactiva y hacer zoom sobre cada grupo.

Ejercicio 8. Partiendo del conjunto de reglas generado por el algoritmo `apriori`:

8.a) Genera gráficas para distintos número de grupos.
8.b) Prueba la versión interactiva.

5.4. Gráficos basados en grafos

En las gráficas basadas en grafos, los itemsets se representan mediante vértices y los arcos indican los itemsets que están incluidos en cada regla. En este caso, las medidas

de calidad se representan como vértices que unen el antecedente y el consecuente. Otros herramientas representan las medidas de calidad mediante etiquetas, colores o anchura de los arcos. Sin embargo, sólo son viables para un conjunto reducido de reglas. Para crear este gráfico debemos utilizar la opción `method = "graph"` (ver Figura 4):

```
> titanic.rules2 <- subset(titanic.rules, subset = support > 0.4)
> plot(titanic.rules2, method = "graph")
```

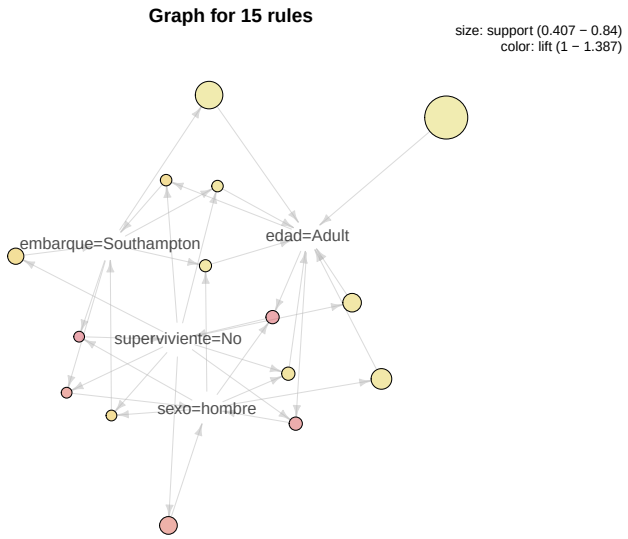

Figura 5. Grafo con las 15 reglas con el soporte más alto.

Para obtener un grafo en el qué los itemsets estén representados mediante vértices y las reglas mediante arcos que unen los itemsets implicados, bastaría con la opción `type="itemsets"` en la lista del parámetro `control`. Este tipo de gráficos también permite la visualización interactiva. Además, se permite la exportación del grafo en el formato `GraphML` para poder utilizar herramientas específicas para la visualización de redes y grafos. Para ello, se podría utilizar la función `saveAsGraph()`:

```
> saveAsGraph(titanic.rules, file="reglas.graphml")
```

Ejercicio 9. Para las 10 reglas con la confianza más alta:

9.a) Genera un grafo que las represente.
9.b) Genera un grafo utilizando la opción `type="itemsets"`.
9.c) Prueba la versión interactiva.

5.5. Gráficos de coordenadas paralelas

Este tipo de gráficos permite la visualización en dos dimensiones de datos multi-dimensionales, representando cada dimensión en el eje X y compartiendo todas ellas el mismo eje Y. Este tipo de gráficas ya las analizamos cuando vimos técnicas de visualización de agrupamientos. En este caso, en el eje Y se representa cada uno de los items y en el eje X su posición dentro de la regla. De esta forma, cada regla queda representada mediante una recta que va de izquierda a derecha, quedando representada la confianza de la regla por el color. Para crear este gráfico debemos utilizar la opción `method = "graph"` (ver Figura 6).

```
> titanic.rules3 <- head(sort(titanic.rules, by = "confidence",), n=10)
> plot(titanic.rules3, method = "paracoord")
```

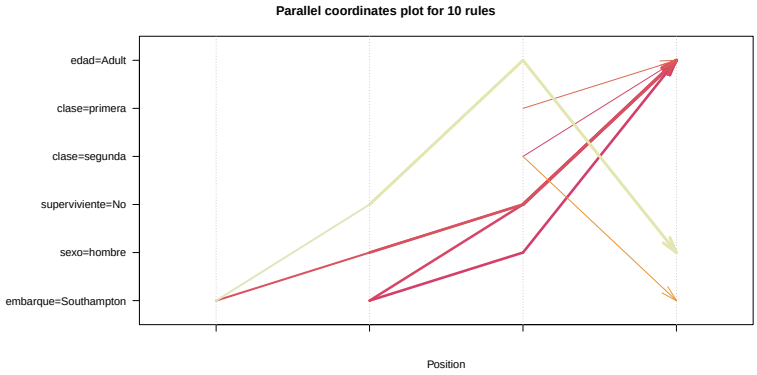

Figura 6. Gráfico de coordenadas paralelas para las 10 reglas con la confianza más alta.

Con la opción de `control=list(reorder=TRUE)` podemos reducir el número de cruces. Este tipo de gráficos no admite la versión interactiva.

Ejercicio 10. Para las 10 reglas con el lift más alto:

10.a) Genera un gráfico de coordenadas paralelas.
10.b) Utilizar la opción `control=list(reorder=TRUE)` y compara los resultados.

5.6. Gráficos de mosaicos

Los gráficos de mosaicos (*double-decker plots*) permiten representar una matriz de contingencia, usando rectángulos dentro del mosaico cuya área es proporcional al valor correspondiente en la tabla de contingencia. En este caso, nos permite visualizar una sola regla y representar la frecuencia de cada subconjunto de items en el antecedente y en el consecuente en el conjunto de transacciones. Los items en el antecedentes se utilizan para realizar las particiones verticales y el item del consecuente para el horizontal. Para crear este gráfico debemos utilizar la opción `method = "doubledecker"` y indicar cuál es el conjunto de transacciones sobre el que calcular las frecuencias (ver Figura 7). En este gráfico, el área de cada rectángulo es proporcional al soporte y la altura del bloque marcado como `"yes"` es proporcional a la confianza de la regla que contiene dicho item.

```
> regla <- sample(titanic.rules, 1)
> plot(regla, method = "doubledecker", data = titanic.trans)
```

Este gráfico es importante para detectar aquellos items que presentan grandes saltos en la confianza de una regla cuando dejan de estar presentes en la misma.

Ejercicio 11. Representa, mediante un gráfico de mosaicos, la regla con el lift más alta que tenga al menos 3 items en el antecedentes.

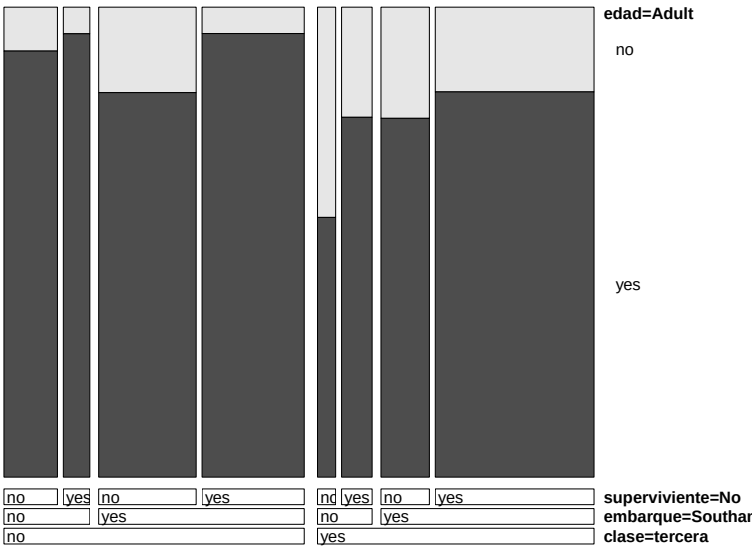

Figura 7. Gráfico de mosaico.

Referencias

Agrawal *et al.*, 1993. Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993.*, pages 207–216. ACM Press, 1993.

Borgelt, 2003. Christian Borgelt. Efficient implementations of apriori and eclat. In *FIMI'03: Proceedings of the IEEE ICDM workshop on frequent itemset mining implementations*, 2003.

Hahsler and Chelluboina, 2015. Michael Hahsler and Sudheer Chelluboina. *arulesViz: Visualizing Association Rules and Frequent Itemsets*, 2015. R package version 1.1-0.

Zaki *et al.*, 1997. Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. New algorithms for fast discovery of association rules. In David Heckerman, Heikki Mannila, and Daryl Pregibon, editors, *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97), Newport Beach, California, USA, August 14-17, 1997*, pages 283–286. AAAI Press, 1997.