

Ejercicios de Visualización de Association Rules

Contents

Instalación y configuración de arulesViz	1
Transactions: Elementos de la clase y ejemplo con data set Groceries	2
myGroceries: Muestra de las transacciones de Groceries	4
Visualización básica de transacciones	5
Algoritmo apriori: Minería de reglas de asociación	6
Estructura de la clase rules	8
Técnicas y ejemplos de visualización con arulesViz	9

Instalación y configuración de arulesViz

EJERCICIO-ARV01

Instala los paquetes requeridos para este notebook.

En caso que los no tengas será necesario que retires los comentarios y ejecutes los comandos de la siguiente celda.

Tip: Coloca nuevamente en comentario las líneas de abajo en cuanto hayas instalado los paquetes.

```
#TODO: Retira los comentarios la primera vez para instalar los paquetes.
#install.packages("arulesViz")
#install.packages("devtools")
#devtools::install_github("hadley/emo")
```

Carga la librería arulesViz (la cual carga automáticamente arules).

```
library("arulesViz")

## Warning: package 'arulesViz' was built under R version 3.6.2

## Loading required package: arules

## Warning: package 'arules' was built under R version 3.6.2

## Loading required package: Matrix

##
## Attaching package: 'arules'
```

```

## The following objects are masked from 'package:base':
##
##     abbreviate, write

```

Carga la librería emo (opcional, para desplegar emoticones).

```
library("emo")
```

Transactions: Elementos de la clase y ejemplo con data set Groceries

Groceries contiene información de ventas de una tienda de abarrotes con 9835 transacciones y 169 items (grupos de productos). Carga Groceries y visualiza su información general.

```

data("Groceries")
summary(Groceries)

```

```

## transactions as itemMatrix in sparse format with
## 9835 rows (elements/itemsets/transactions) and
## 169 columns (items) and a density of 0.02609146
##
## most frequent items:
##      whole milk other vegetables      rolls/buns      soda
##      2513           1903            1809          1715
##      yogurt        (Other)
##      1372           34055
##
## element (itemset/transaction) length distribution:
## sizes
##   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16
## 2159 1643 1299 1005 855 645 545 438 350 246 182 117 78 77 55 46
##   17  18  19  20  21  22  23  24  26  27  28  29  32
##   29  14  14    9   11    4    6    1    1    1    1    3    1
##
##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
##      1.000  2.000  3.000  4.409  6.000 32.000
##
## includes extended item information - examples:
##      labels level2      level1
## 1 frankfurter sausage meat and sausage
## 2 sausage sausage meat and sausage
## 3 liver loaf sausage meat and sausage

```

Examina la estructura de Groceries

```
str(Groceries)
```

```

## Formal class 'transactions' [package "arules"] with 3 slots
##   ..@ data      :Formal class 'ngCMatrix' [package "Matrix"] with 5 slots
##   ... . . . @ i      : int [1:43367] 13 60 69 78 14 29 98 24 15 29 ...
##   ... . . . @ p      : int [1:9836] 0 4 7 8 12 16 21 22 27 28 ...
##   ... . . . @ Dim     : int [1:2] 169 9835

```

```

## ... .@ Dimnames:List of 2
## ... .@ ... .$ : NULL
## ... .@ ... .$ : NULL
## ... .@ factors : list()
## ..@ itemInfo : 'data.frame': 169 obs. of 3 variables:
## ... $ labels: chr [1:169] "frankfurter" "sausage" "liver loaf" "ham" ...
## ... $ level2: Factor w/ 55 levels "baby food","bags",...: 44 44 44 44 44 44 44 42 42 41 ...
## ... $ level1: Factor w/ 10 levels "canned food",...: 6 6 6 6 6 6 6 6 6 ...
## ..@ itemsetInfo:'data.frame': 0 obs. of 0 variables

```

Observa que la clase de Groceries es de tipo `transactions` y contiene tres slots identificados como `@data`, `@itemInfo` e `@itemsetInfo`.

`data` es un objeto de la clase `ngCMatrix` que almacena una matriz de incidencia binaria.

`itemInfo` es un `data.frame` que almacena las etiquetas de los ítems.

El método `inspect()` permite desplegar información de transacciones al igual que asociaciones.

```
inspect(Groceries[1:10])
```

```

##      items
## [1] {citrus fruit,
##       semi-finished bread,
##       margarine,
##       ready soups}
## [2] {tropical fruit,
##       yogurt,
##       coffee}
## [3] {whole milk}
## [4] {pip fruit,
##       yogurt,
##       cream cheese ,
##       meat spreads}
## [5] {other vegetables,
##       whole milk,
##       condensed milk,
##       long life bakery product}
## [6] {whole milk,
##       butter,
##       yogurt,
##       rice,
##       abrasive cleaner}
## [7] {rolls/buns}
## [8] {other vegetables,
##       UHT-milk,
##       rolls/buns,
##       bottled beer,
##       liquor (appetizer)}
## [9] {pot plants}
## [10] {whole milk,
##        cereals}

```

myGroceries: Muestra de las transacciones de Groceries

EJERCICIO-ARV02

Trabajaremos con una muestra del 80% de las transacciones de Groceries. Para lograr resultados reproducibles asigna el valor de los dígitos de tu matrícula en el siguiente bloque de código (p.e. 430000).

```
#TODO: Asigna a STUDENT_ID los dígitos de tu matrícula (p.e. 430000).
STUDENT_ID = 01375441
```

Ejecuta el siguiente bloque de código para obtener una muestra del 80% de las transacciones de Groceries, las cuales estarán referenciadas en la variable myGroceries.

```
numTransactions <- round(length(Groceries) * 0.8)
set.seed(STUDENT_ID)
myGroceries <- sample(Groceries, numTransactions)
summary(myGroceries)

## transactions as itemMatrix in sparse format with
## 7868 rows (elements/itemsets/transactions) and
## 169 columns (items) and a density of 0.02612259
##
## most frequent items:
##      whole milk other vegetables          rolls/buns        soda
##           2021          1521            1459         1383
##      yogurt          (Other)
##           1110          27241
##
## element (itemset/transaction) length distribution:
## sizes
##    1    2    3    4    5    6    7    8    9    10   11   12   13   14   15   16
## 1727 1305 1037  809  683  514  436  353  285  202  141  101   61   60   45   35
##    17    18    19    20    21    22    23    27    28    29    32
##    25    10    10     6     8     4     5     1     1     3     1
##
##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
##      1.000  2.000  3.000  4.415  6.000 32.000
##
## includes extended item information - examples:
##      labels level2          level1
## 1 frankfurter sausage meat and sausage
## 2 sausage sausage meat and sausage
## 3 liver loaf sausage meat and sausage
```

DOCUMENTO ARV-01

Elabora una tabla comparando los valores de las siguientes métricas de acuerdo a la información reportada por `summary(Groceries)` y `summary(myGroceries)`:

- número de reglones.
- número de columnas.
- densidad.

- item más frecuente (nombre y frecuencia).
- Media aritmética (Mean).

Responde a las siguientes preguntas:

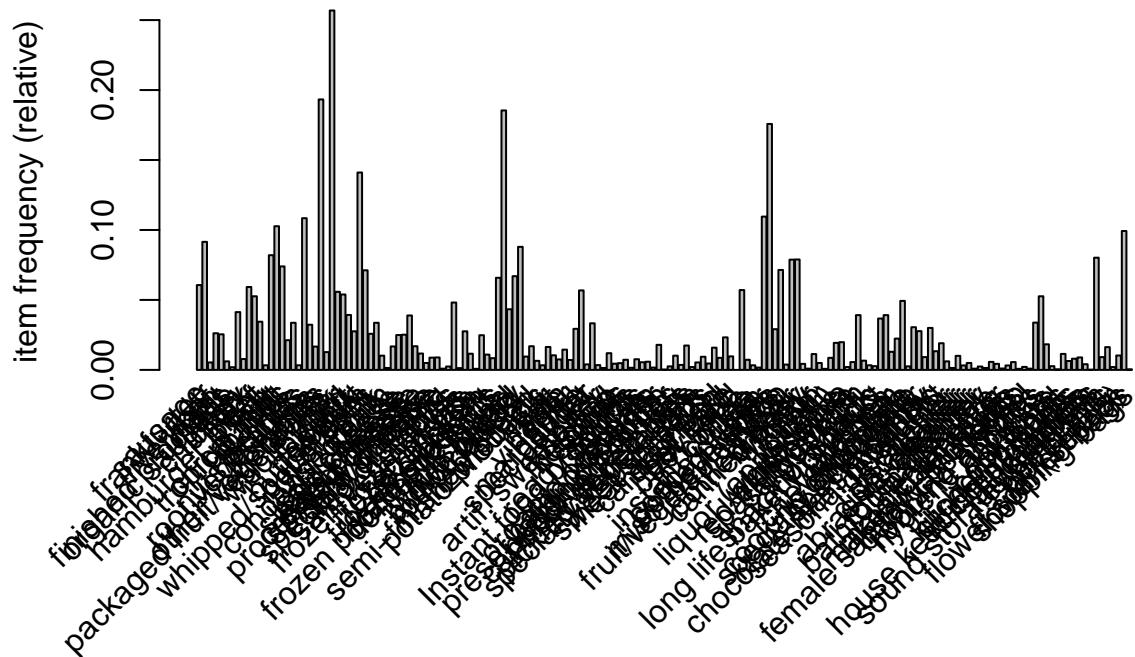
- De acuerdo al valor de density, ¿consideras que la matriz de transacciones de myGroceries es densa o dispersa?
- ¿Cuántos items hay en la transacción promedio?
- ¿Cuál es el producto de mayor frecuencia?

Visualización básica de transacciones

El método `itemFrequencyPlot` crea una gráfica de barras que permite inspeccionar la frecuencia de items para objetos basados en `itemMatrix` como `transactions`, `itemsets` y `rules`.

Visualizaremos la gráfica de frecuencia de items en myGroceries con los parámetros por omisión.

```
itemFrequencyPlot(myGroceries)
```



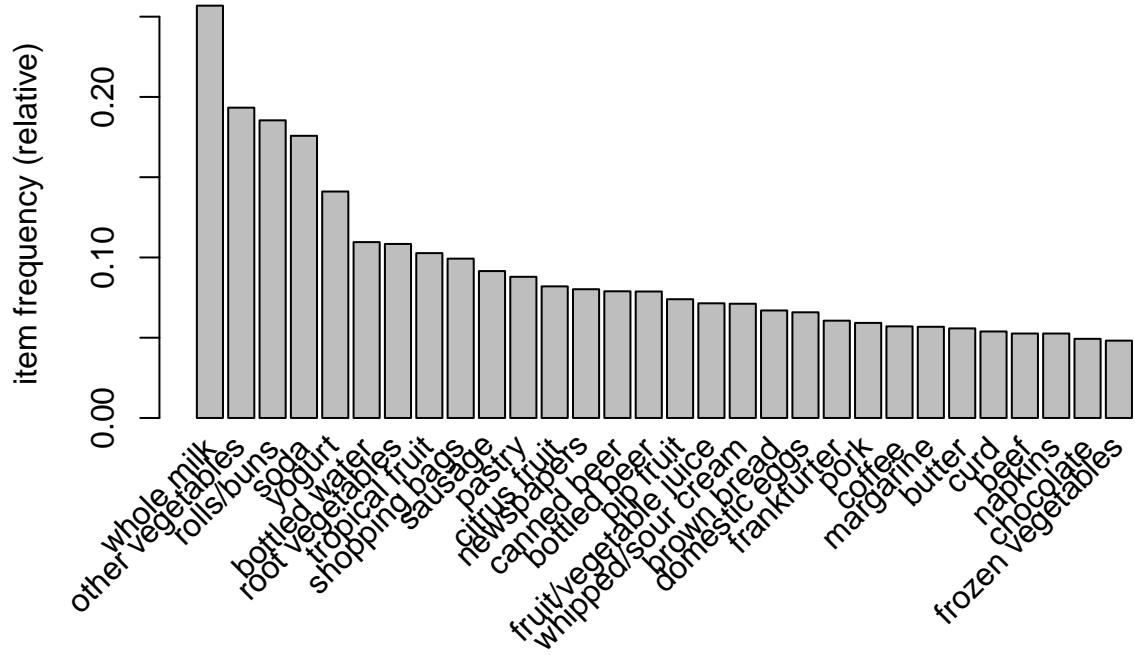
Para este caso, la gráfica tiene demasiada información por lo que su interpretación es difícil.

`itemFrequencyPlot` cuenta con parámetros adicionales para filtrar items.

EJERCICIO-ARV03

Visualiza la frecuencia de items en myGroceries filtrando por los primeros 30 items.

```
#TODO: Modifica el valor de topN a 30
itemFrequencyPlot(myGroceries, topN = 30)
```



Algoritmo apriori: Minería de reglas de asociación

Apriori es un algoritmo que cuenta transacciones por medio de una estrategia orientada a nivel y por amplitud primero (en contraste con profundidad). Fue desarrollado por Agrawal y Srikant (1994). Se utiliza para generar reglas de asociación, identificar itemsets, itemsets de frecuencia máxima e itemsets de frecuencia cerrada.

Efectuaremos minería de reglas de asociación por medio del algoritmo Apriori implementado en arules.

Visualiza la salida de la consola de R para examinar la salida del algoritmo.

```
rules <- apriori(myGroceries, parameter = list(support = 0.001, confidence = 0.05))
```

```
## Apriori
##
## Parameter specification:
##   confidence minval smax arem  aval originalSupport maxtime support minlen
##             0.05    0.1     1 none FALSE              TRUE      5  0.001     1
##   maxlen target ext
##         10  rules TRUE
##
## Algorithmic control:
```

```

## filter tree heap memopt load sort verbose
##      0.1 TRUE TRUE FALSE TRUE    2     TRUE
##
## Absolute minimum support count: 7
##
## set item appearances ... [0 item(s)] done [0.00s].
## set transactions ... [168 item(s), 7868 transaction(s)] done [0.01s].
## sorting and recoding items ... [157 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 done [0.01s].
## writing ... [40386 rule(s)] done [0.01s].
## creating S4 object ... done [0.01s].
```

summary(rules)

```

## set of 40386 rules
##
## rule length distribution (lhs + rhs):sizes
##      1     2     3     4     5     6
##      28   3911  20271  13685  2425    66
##
##      Min. 1st Qu. Median     Mean 3rd Qu.     Max.
##      1.000  3.000  3.000   3.366  4.000   6.000
##
## summary of quality measures:
##      support      confidence      coverage      lift
##      Min. :0.001017  Min. :0.0500  Min. :0.001017  Min. : 0.4027
##      1st Qu.:0.001144 1st Qu.:0.1397  1st Qu.:0.003559 1st Qu.: 1.9068
##      Median :0.001398 Median :0.2353  Median :0.006482 Median : 2.5806
##      Mean   :0.002082 Mean   :0.2913  Mean   :0.012365 Mean   : 2.9373
##      3rd Qu.:0.002034 3rd Qu.:0.4000  3rd Qu.:0.012328 3rd Qu.: 3.5125
##      Max.  :0.256863 Max.  :1.0000  Max.  :1.000000 Max.  :50.5168
##
##      count
##      Min.   : 8.00
##      1st Qu.: 9.00
##      Median :11.00
##      Mean   :16.38
##      3rd Qu.:16.00
##      Max.  :2021.00
##
## mining info:
##      data ntransactions support confidence
## myGroceries          7868     0.001       0.05
```

DOCUMENTO ARV-02

Documenta los siguientes datos de la salida de arules:

- Número de reglas producidas.
- Distribución del tamaño de las reglas (lhs + rhs).
- Media aritmética de la distribución de reglas.

Estructura de la clase rules

Examinaremos la estructura del objeto rules

```
str(rules)
```

```
## Formal class 'rules' [package "arules"] with 4 slots
## ..@ lhs      :Formal class 'itemMatrix' [package "arules"] with 3 slots
## ... .@ data     :Formal class 'ngCMatrix' [package "Matrix"] with 5 slots
## ... . . . . @ i       : int [1:95538] 73 149 122 122 45 45 12 141 18 18 ...
## ... . . . . @ p       : int [1:40387] 0 0 0 0 0 0 0 0 0 0 ...
## ... . . . . @ Dim     : int [1:2] 169 40386
## ... . . . . @ Dimnames:List of 2
## ... . . . . @ factors : list()
## ... . . . . @ itemInfo  :'data.frame':   169 obs. of  3 variables:
## ... . . . . $ labels: chr [1:169] "frankfurter" "sausage" "liver loaf" "ham" ...
## ... . . . . $ level2: Factor w/ 55 levels "baby food","bags",...: 44 44 44 44 44 44 44 44 42 42 41 ...
## ... . . . . $ level1: Factor w/ 10 levels "canned food",...: 6 6 6 6 6 6 6 6 6 6 ...
## ... . . . . @ itemsetInfo:'data.frame':   0 obs. of  0 variables
## ..@ rhs      :Formal class 'itemMatrix' [package "arules"] with 3 slots
## ... .@ data     :Formal class 'ngCMatrix' [package "Matrix"] with 5 slots
## ... . . . . @ i       : int [1:40386] 108 98 10 152 107 26 9 69 0 57 ...
## ... . . . . @ p       : int [1:40387] 0 1 2 3 4 5 6 7 8 9 ...
## ... . . . . @ Dim     : int [1:2] 169 40386
## ... . . . . @ Dimnames:List of 2
## ... . . . . @ factors : list()
## ... . . . . @ itemInfo  :'data.frame':   169 obs. of  3 variables:
## ... . . . . $ labels: chr [1:169] "frankfurter" "sausage" "liver loaf" "ham" ...
## ... . . . . $ level2: Factor w/ 55 levels "baby food","bags",...: 44 44 44 44 44 44 44 44 42 42 41 ...
## ... . . . . $ level1: Factor w/ 10 levels "canned food",...: 6 6 6 6 6 6 6 6 6 6 ...
## ... . . . . @ itemsetInfo:'data.frame':   0 obs. of  0 variables
## ..@ quality:'data.frame': 40386 obs. of  5 variables:
## ... . $ support   : num [1:40386] 0.0789 0.0571 0.0526 0.0526 0.0788 ...
## ... . $ confidence: num [1:40386] 0.0789 0.0571 0.0526 0.0526 0.0788 ...
## ... . $ coverage   : num [1:40386] 1 1 1 1 1 1 1 1 1 ...
## ... . $ lift       : num [1:40386] 1 1 1 1 1 1 1 1 1 ...
## ... . $ count      : int [1:40386] 621 449 414 414 620 424 466 447 477 527 ...
## ..@ info      :List of 4
## ... . $ data       : symbol myGroceries
## ... . $ ntransactions: int 7868
## ... . $ support     : num 0.001
## ... . $ confidence   : num 0.05
```

La clase **rules** representa a un conjunto de reglas las cuales generalmente se crean por medio de la invocación de un algoritmo de minería de reglas de asociación como **apriori**. Las reglas almacenan el LHS y el RHS de forma separada como objetos de la clase **itemMatrix**.

Las reglas minadas típicamente contienen diversas mediciones de interés las cuales pueden accederse por medio del método **quality**. Es posible calcular mediciones adicionales por medio de **interestMeasure**

Obtenemos las primeras tres reglas respecto a la métrica lift, que es una métrica popular de la fortaleza de la regla.

```
inspect(head(rules, n = 3, by = "lift"))

##      lhs                      rhs          support
## [1] {soda,bottled beer,red/brush wine} => {liquor} 0.001016777
## [2] {bottled beer,red/brush wine}      => {liquor} 0.002160651
## [3] {soda,bottled beer,liquor}        => {red/brush wine} 0.001016777
##      confidence coverage    lift   count
## [1] 0.5714286 0.001779359 50.51685 8
## [2] 0.4146341 0.005210981 36.65552 17
## [3] 0.6666667 0.001525165 33.62393 8
```

Para conocer las medidas disponibles en el conjunto de reglas empleamos el método `quality`.

```
head(quality(rules))

##      support confidence coverage lift   count
## 1 0.07892730 0.07892730     1     1   621
## 2 0.05706660 0.05706660     1     1   449
## 3 0.05261820 0.05261820     1     1   414
## 4 0.05261820 0.05261820     1     1   414
## 5 0.07880020 0.07880020     1     1   620
## 6 0.05388917 0.05388917     1     1   424
```

Estas son las medidas generadas por omisión con el algoritmo Apriori. Para añadir otras mediciones es posible emplear la función `interestMeasures()` de la librería arules.

Resulta evidente que explorar las decenas de miles de transacciones de forma manual no es una opción viable. Emplearemos diversas técnicas de visualización del paquete arulesViz las cuales comparten la siguiente interfaz:

```
args(getS3method("plot", "rules"))

## function (x, method = NULL, measure = "support", shading = "lift",
##           interactive = NULL, engine = "default", data = NULL, control = NULL,
##           ...)
## NULL
```

`x` representa el conjunto de reglas a ser vizualizadas, `method` es el método de visualización y `measure` y `shading` contienen las mediciones de interés empleadas por la gráfica. Por medio de `engine` es posible usar diversos motores de graficación para visualizar las reglas. El motor predeterminado típicamente usa `grid`, muchas gráficas pueden ser visualizadas con el motor “`htmlwidget`” en el cual resulta en un widget HTML.

Técnicas y ejemplos de visualización con arulesViz

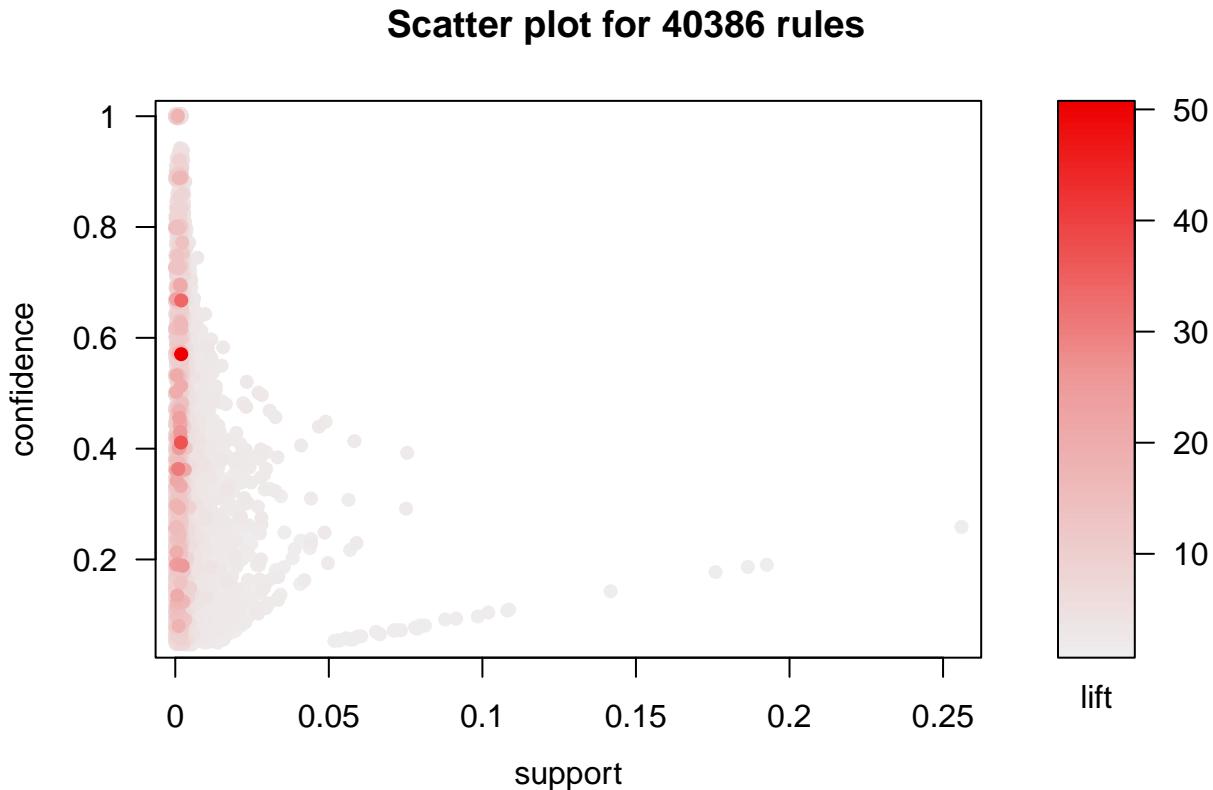
Scatter plot

Es posible lograr una visualización simple de reglas de asociación por medio de la gráfica scatter plot con dos medidas de interés en los ejes. El método por omisión de reglas de asociación en arulesViz es el scatter

plot empleando support y confidence en los ejes. También se usa una tercera medición (lift, por omisión) como el color (o escala de grises de los puntos). Se incluye una escala del rango de valores que corresponde a los colores.

```
plot(rules)

## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.
```



Es posible emplear en los ejes del scatter plot cualquier medición de calidad almacenada en el slot de calidad del conjunto de reglas (vector de longitud 2 para el parámetro measure) o para el gradiente de color (shading).

DOCUMENTO ARV-03

Analiza la gráfica, identifica las reglas que tengan un valores altos en la medición lift e indica para ellas si el valor de support es alto o bajo.

EJERCICIO ARV-04

Modifica el código de abajo para crear un scatter plot con las siguientes propiedades:

- support en el eje horizontal (h-axis)
- lift en el eje vertical (v-xaix)
- confidence (measure) como gradiente de color (shading)

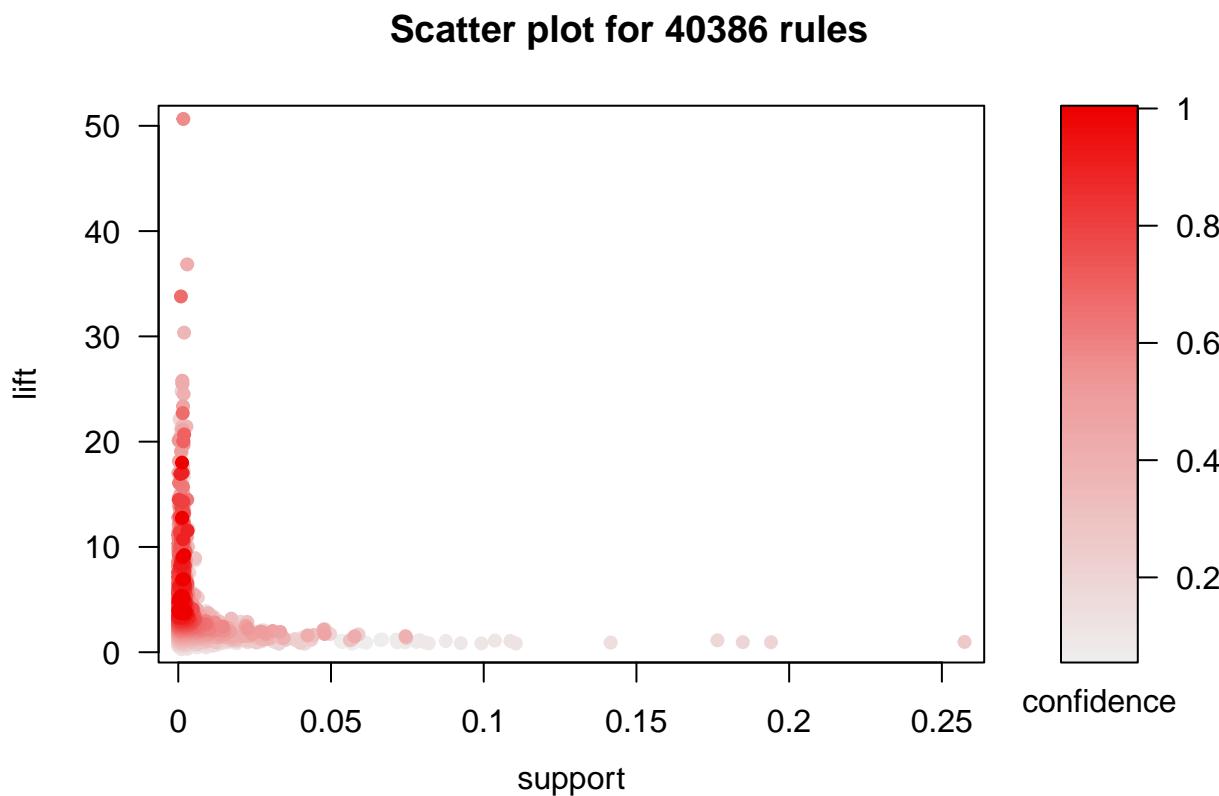
```

#TODO: Asigna los parámetros a plot para:
# support en el eje horizontal (h-axis)
# lift en el eje vertical (v-xaix)
# confidence (measure) como gradiente de color (shading)

#plot(rules, measure = c("h-axis", "v-axis"), shading = "measure")
plot(rules, measure = c("support", "lift"), shading = "confidence")

```

To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.



EJERCICIO ARV-05

Modifica el código de abajo para crear una gráfica scatter plot con support y confidence en los ejes horizontal y vertical, empleando coverage como medición asociada a la escala de colores.

```

#TODO: Scatter plot con support y confidence en los ejes horizontal y vertical, empleando coverage como medida

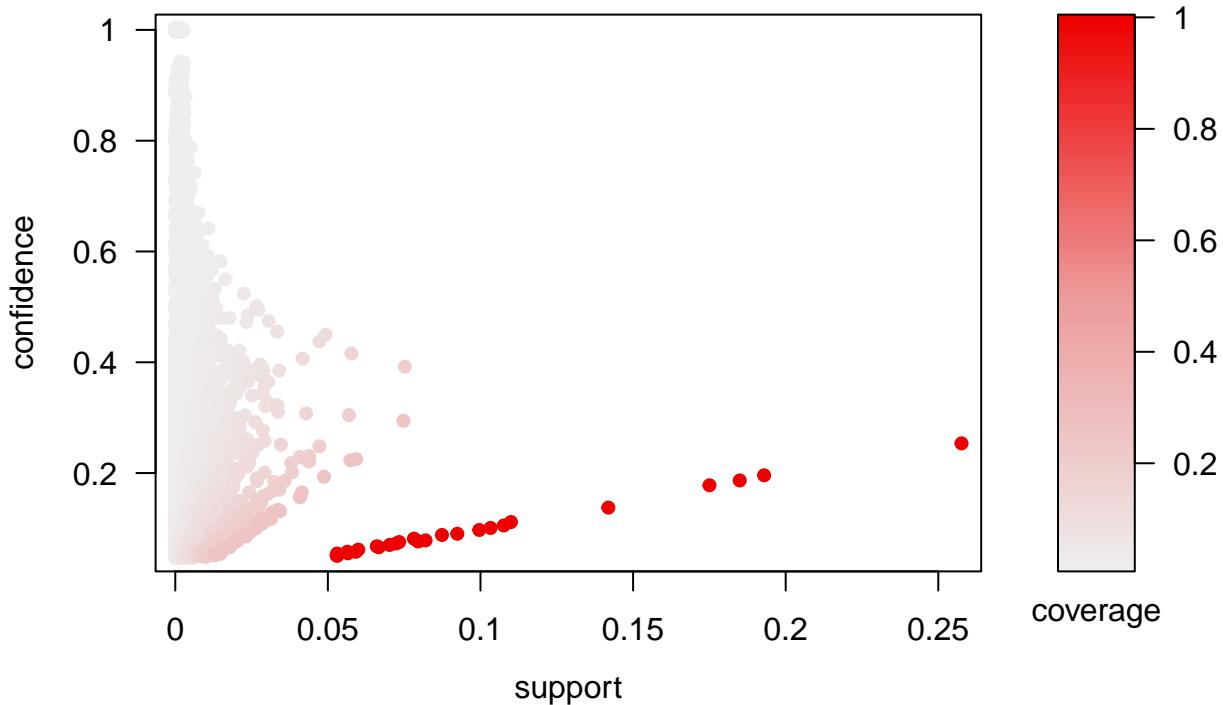
# Por omisión la gráficas scatter plot en arulesViz presentan support en el eje horizontal y confidence en el eje vertical

#plot(rules, shading="measure")
plot(rules, shading = "coverage")

## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.

```

Scatter plot for 40386 rules



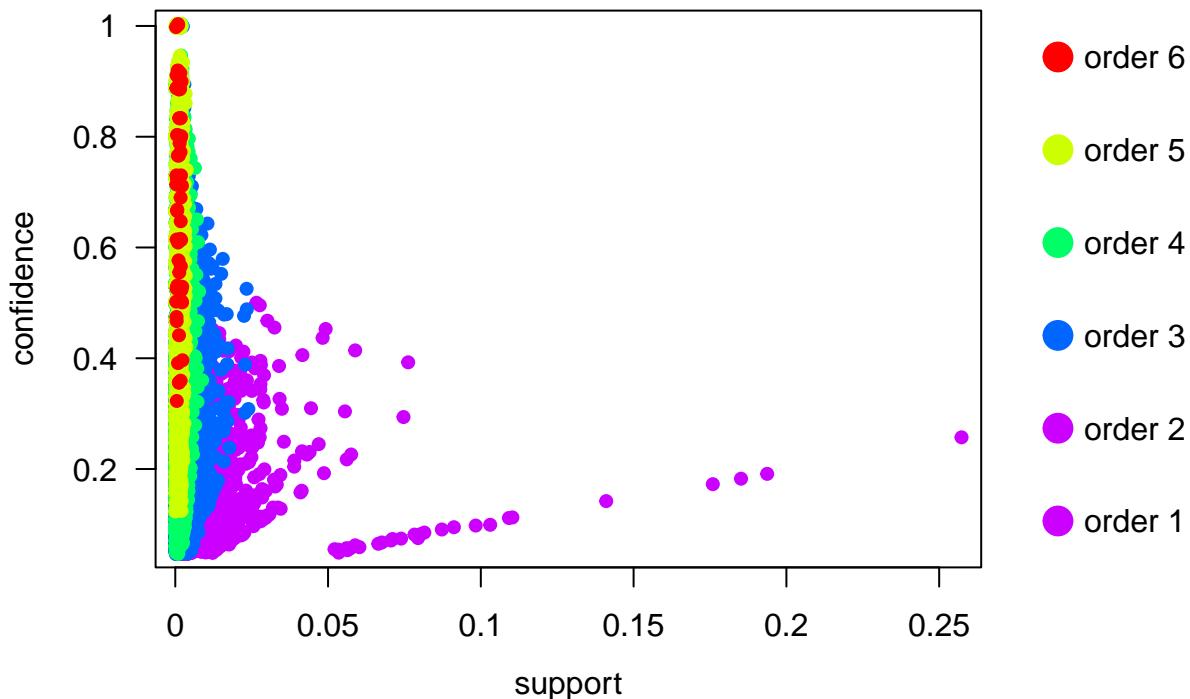
Two-key plot

Introducida por Unwin, Hoffmann y Bernt (2001), el Two-key plplot es una versión especial de un scatter plot en la que support y confidence están asociados al eje horizontal y vertical y el color de los puntos indica el “orden” (número de elementos contenidos en la regla). Este tipo de gráficas pueden ser creadas por medio de la interfaz unificada.

```
plot(rules, method= "two-key plot")
```

```
## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.
```

Scatter plot for 40386 rules



DOCUMENTO ARV-04

Responde: ¿Qué tipo de relación muestran order y support en esta visualización?

Scatter plots interactivos en arulesViz

El método plot ofrece características interactivas para seleccionar reglas y mostrar mayor o menor detalle en la gráfica (zoom in y zoom out) las cuales son útiles tareas de exploración. La interacción se activa por medio del parámetro engine = "interactive".

```
#sel <- plot(rules, measure = c("support", "lift"), shading = "confidence", interactive = TRUE)
```

Las funciones interactivas incluyen:

- Inspeccionar reglas individuales al seleccionarlas y dar click al botón inspect.
- Inspeccionar conjuntos de reglas al seleccionar una región rectangular de la gráfica y dar click en el botón inspect.
- Mostrar mayor o menor detalle en una región seleccionada (botones zoom in / zoom out).
- Regresar a la última selección para mayor análisis, (botón end).