Reglas de asociación a priori

Cedric Prieels

10/11/2016

Reglas de asociación a priori

Abrimos primero todas las reglas de asociación del conjunto de datos Groceries.

```
rm(list=ls())
require("arulesViz")
```

```
## Loading required package: arulesViz

## Loading required package: arules

## Loading required package: Matrix

##
## Attaching package: 'arules'

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

## Loading required package: grid

## Warning: failed to assign NativeSymbolInfo for lhs since lhs is already
## defined in the 'lazyeval' namespace

## Warning: failed to assign NativeSymbolInfo for rhs since rhs is already
## defined in the 'lazyeval' namespace
```

```
data(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 6 ...
##   ..@ itemsetInfo:'data.frame':  0 obs. of  0 variables
```

Vemos que en este caso tenemos 169 objetos considerados, y vamos a considerar algunos compras de una tienda (que están de momento en formato de matriz dispersa, que tiene muchos zeros). Estudiamos primero el soporte, la proporción de compras que cumplen una regla que consideramos. Aquí enseñamos por ejemplo las 10 reglas que tienen un soporte más grande (y que tienen por lo tanto la estadística más grande).

```
rules <- apriori(Groceries, parameter=list(support = 0.001, confidence = 0.5))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.5    0.1    1 none FALSE            TRUE       5   0.001      1
##  maxlen target   ext
##      10  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 9
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## 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 ... [5668 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
inspect(head(sort(rules, by="support"), 10))
```

```
##      lhs                     rhs                   support confidence     lift
## [1]  {other vegetables,
##       yogurt}             => {whole milk}       0.02226741  0.5128806 2.007235
## [2]  {tropical fruit,
##       yogurt}             => {whole milk}       0.01514997  0.5173611 2.024770
## [3]  {other vegetables,
##       whipped/sour cream} => {whole milk}       0.01464159  0.5070423 1.984385
## [4]  {root vegetables,
##       yogurt}             => {whole milk}       0.01453991  0.5629921 2.203354
## [5]  {pip fruit,
##       other vegetables}   => {whole milk}       0.01352313  0.5175097 2.025351
## [6]  {root vegetables,
##       yogurt}             => {other vegetables} 0.01291307  0.5000000 2.584078
## [7]  {root vegetables,
##       rolls/buns}         => {whole milk}       0.01270971  0.5230126 2.046888
## [8]  {other vegetables,
##       domestic eggs}      => {whole milk}       0.01230300  0.5525114 2.162336
## [9]  {tropical fruit,
##       root vegetables}    => {other vegetables} 0.01230300  0.5845411 3.020999
## [10] {root vegetables,
##       rolls/buns}         => {other vegetables} 0.01220132  0.5020921 2.594890
```

En este caso, vemos por ejemplo que la regla "other vegetables, yogurt" => "whole milk" es la regla que tiene el soporte más grande (del orden de 0.22), y aparece entonces en muchas de las compras hechas.

Para intentar sacar algo de información sobre lo que tenemos, tenemos que elegir unos valores de soporte y de confianza (que da información sobre la cantidad de personas que compran Y, una vez comprado X de la regla X=>Y) que tengan sentido. Lo primero que hicimos es enseñar todas las reglas que tienen un soporte mayor que 0,001 y una confianza mayor que 0,5. Enseñamos aquí también las 10 reglas que cumplen estas condiciones y que tienen el soporte más grande (en este caso, son exactamente las mismas reglas que en el primer ejemplo porque las 10 reglas que enseñamos ya complían estas reglas).

```
rules <- apriori(Groceries, parameter=list(support = 0.001, confidence = 0.5))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.5    0.1    1 none FALSE            TRUE       5   0.001      1
##  maxlen target   ext
##      10  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 9
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## 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 ... [5668 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
inspect(head(sort(rules, by="support"), 10))
```

```
##      lhs                     rhs                   support confidence     lift
## [1]  {other vegetables,
##       yogurt}             => {whole milk}       0.02226741  0.5128806 2.007235
## [2]  {tropical fruit,
##       yogurt}             => {whole milk}       0.01514997  0.5173611 2.024770
## [3]  {other vegetables,
##       whipped/sour cream} => {whole milk}       0.01464159  0.5070423 1.984385
## [4]  {root vegetables,
##       yogurt}             => {whole milk}       0.01453991  0.5629921 2.203354
## [5]  {pip fruit,
##       other vegetables}   => {whole milk}       0.01352313  0.5175097 2.025351
## [6]  {root vegetables,
##       yogurt}             => {other vegetables} 0.01291307  0.5000000 2.584078
## [7]  {root vegetables,
##       rolls/buns}         => {whole milk}       0.01270971  0.5230126 2.046888
## [8]  {other vegetables,
##       domestic eggs}      => {whole milk}       0.01230300  0.5525114 2.162336
```

```
## [9]  {tropical fruit,
##       root vegetables}    => {other vegetables} 0.01230300  0.5845411 3.020999
## [10] {root vegetables,
##       rolls/buns}         => {other vegetables} 0.01220132  0.5020921 2.594890
```

Lo que pasa es que el soporte elegido es un poco pequeño en este caso (buscamos algo que tenga algo de estadística, y que tenga entonces un soporte mayor que 0,001). Del otro lado, la confianza no tiene porque tener un valor tan alto (por ejemplo, si el 30% de la gente hace algo, ya es algo que nos puede interesar, dependiendo del problema). Volvemos a mirar las reglas que cumplen estos dos nuevos requisitos :

```
rules <- apriori(Groceries, parameter=list(support = 0.01, confidence = 0.3))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.3    0.1    1 none FALSE            TRUE       5    0.01      1
##  maxlen target   ext
##      10  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 98
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [125 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
inspect(head(sort(rules, by="support"), 10))
```

```
##      lhs                   rhs                support    confidence
## [1]  {other vegetables} => {whole milk}       0.07483477 0.3867578
## [2]  {rolls/buns}       => {whole milk}       0.05663447 0.3079049
## [3]  {yogurt}           => {whole milk}       0.05602440 0.4016035
## [4]  {root vegetables}  => {whole milk}       0.04890696 0.4486940
## [5]  {root vegetables}  => {other vegetables} 0.04738180 0.4347015
## [6]  {yogurt}           => {other vegetables} 0.04341637 0.3112245
## [7]  {tropical fruit}   => {whole milk}       0.04229792 0.4031008
## [8]  {tropical fruit}   => {other vegetables} 0.03589222 0.3420543
## [9]  {bottled water}    => {whole milk}       0.03436706 0.3109476
## [10] {pastry}           => {whole milk}       0.03324860 0.3737143
##      lift
## [1]  1.513634
## [2]  1.205032
## [3]  1.571735
## [4]  1.756031
```

```
## [5]  2.246605
## [6]  1.608457
## [7]  1.577595
## [8]  1.767790
## [9]  1.216940
## [10] 1.462587
```

Podemos ahora empezar a mirar el lift. El lift es una variable que combina el soporte y la confianza juntos, y que nos indica la potencia de la regla. El lift más pequeño posible vale 1. Podemos buscar reglas raras que tengan por lo tanto un lift alto.

```
rules <- apriori(Groceries, parameter=list(support = 0.001, confidence = 0.3))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.3    0.1    1 none FALSE            TRUE       5   0.001      1
##  maxlen target   ext
##      10  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 9
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## 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 ... [13770 rule(s)] done [0.00s].
## creating S4 object  ... done [0.01s].
```

```
inspect(head(sort(rules, by="lift", decreasing = T), 3))
```

```
##     lhs                              rhs                support
## [1] {bottled beer,red/blush wine} => {liquor}           0.001931876
## [2] {ham,white bread}             => {processed cheese} 0.001931876
## [3] {bottled beer,liquor}         => {red/blush wine}   0.001931876
##     confidence lift
## [1] 0.3958333  35.71579
## [2] 0.3800000  22.92822
## [3] 0.4130435  21.49356
```

Por fin, lo que podemos hacer es pintar un histograma de las tres variables (soporte en x, confianza en y y el lift en z, representado por un gradiente de color en este plot) que corresponden a las reglas que tenemos.

```
plot(rules)
```


```
plot(rules, method = NULL, measure = "support", shading = "lift",
     interactive = F, data = NULL, control = NULL)
```

Con este último plot, podemos ver directamente algunos reglas interesantes, depediendo del caso. Por ejemplo, parece que haya dos puntos que tienen una confianza 1. Vemos también que algunos puntos tienen un lift mucho mayor que los otros (hasta llegar a más o menos 35). Por fin, un solo punto tiene un soporte muy grande (alrededor de 0,25) y corresponde a una regla que aparece muchas veces en las compras, y que tiene una confianza del orden del 50%.

Respuestas a las preguntas

¿Cuantas reglas de asociación se han generado?

El número total de reglas generadas sin tener ningún requisito de soporte o de confianza se puede calcular fácilmente.

```
rules <- apriori(Groceries, parameter=list(support = 0.01, confidence = 0.3))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.3    0.1    1 none FALSE            TRUE       5    0.01      1
##  maxlen target   ext
##      10  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 98
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [125 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

Con este linea de código, vemos que en este caso tenemos 169 objetos considerados, 9835 compras registradas, y 125 reglas de asociación creadas. Este número de reglas cambia por supuesto en función de los valores dados a los parámetros de soporte y de confianza.

Teniendo en cuenta el conjunto de datos de partida, ¿resulta útil el conjunto de reglas generado?

Podemos comparar el número de reglas obtenido antes con el número de reglas que obtenemos poniendo un valor de corte muy pequeño a los parámetros de soporte y de confianza.

```
#rules <- apriori(Groceries, parameter=list(support = 0.000001, confidence = 0.000001))
```

La linea de código está comentada porque su ejecución dura mucho tiempo pero nos devuelvo un número de reglas igual a 189393896. Por lo tanto, vemos claramente que el conjunto de reglas generado ayuda mucho al analisis de estos datos, para simplificarlos y encontrar rápidamente lo que nos interesa.

¿Cual es la regla con mayor confianza? ¿Y con menor?

```
rules <- apriori(Groceries, parameter=list(support = 0.001, confidence = 0))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##           0    0.1    1 none FALSE            TRUE       5   0.001      1
##  maxlen target   ext
##      10  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 9
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## 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 ... [41100 rule(s)] done [0.00s].
## creating S4 object  ... done [0.01s].
```

```
inspect(head(sort(rules, by="confidence"), 1))
```

```
##     lhs             rhs          support     confidence lift
## [1] {rice,sugar} => {whole milk} 0.001220132 1          3.913649
```

```
inspect(head(sort(rules, by="confidence", decreasing = F), 1))
```

```
##     lhs    rhs               support     confidence  lift
## [1] {}  => {rubbing alcohol} 0.001016777 0.001016777 1
```

Vemos directamente con estas lineas la regla que tiene la mayor confianza (comprar arroz y azúcar implica comprar leche, con una confianza exactamente igual a 1) y la regla que tiene la menor confianza (comprar alcohol, con una confianza de 0.001).

En el objeto aparece una medida llamada lift, ¿qué mide?

El lift es una manera de quantificar la potencia de la regla. El lift más pequeño posible vale 1, y cuanto más grande es, más potente es la regla. Se obtiene dividiendo el soporte por la probabilidad del antecedente por la probabilidad del consecuente de la regla considerada.

Escribe el podium asociado a cada una de las tres medidas: support, confidence y lift.

Primero, se puede empezar por escribir el podium asociado al soporte.

```
rules <- apriori(Groceries, parameter=list(support = 0.01, confidence = 0.3))
```

```
## Apriori
##
## Parameter specification:
```

```
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.3    0.1    1 none FALSE            TRUE       5    0.01      1
##  maxlen target   ext
##      10  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 98
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [125 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
inspect(head(sort(rules, by="support"), 3))
```

```
##     lhs                   rhs          support    confidence lift
## [1] {other vegetables} => {whole milk} 0.07483477 0.3867578  1.513634
## [2] {rolls/buns}       => {whole milk} 0.05663447 0.3079049  1.205032
## [3] {yogurt}           => {whole milk} 0.05602440 0.4016035  1.571735
```

Después, pintamos por pantalla el podium asociado a la confianza.

```
rules <- apriori(Groceries, parameter=list(support = 0.01, confidence = 0.3))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.3    0.1    1 none FALSE            TRUE       5    0.01      1
##  maxlen target   ext
##      10  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 98
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [125 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```r
inspect(head(sort(rules, by="confidence"), 3))
```

```
##     lhs                   rhs                    support confidence     lift
## [1] {citrus fruit,
##      root vegetables} => {other vegetables} 0.01037112  0.5862069 3.029608
## [2] {tropical fruit,
##      root vegetables} => {other vegetables} 0.01230300  0.5845411 3.020999
## [3] {curd,
##      yogurt}          => {whole milk}       0.01006609  0.5823529 2.279125
```

Y por fin, se estudia el podium asociado al lift.

```r
rules <- apriori(Groceries, parameter=list(support = 0.01, confidence = 0.3))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.3    0.1    1 none FALSE            TRUE       5    0.01      1
##  maxlen target   ext
##      10  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 98
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [125 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

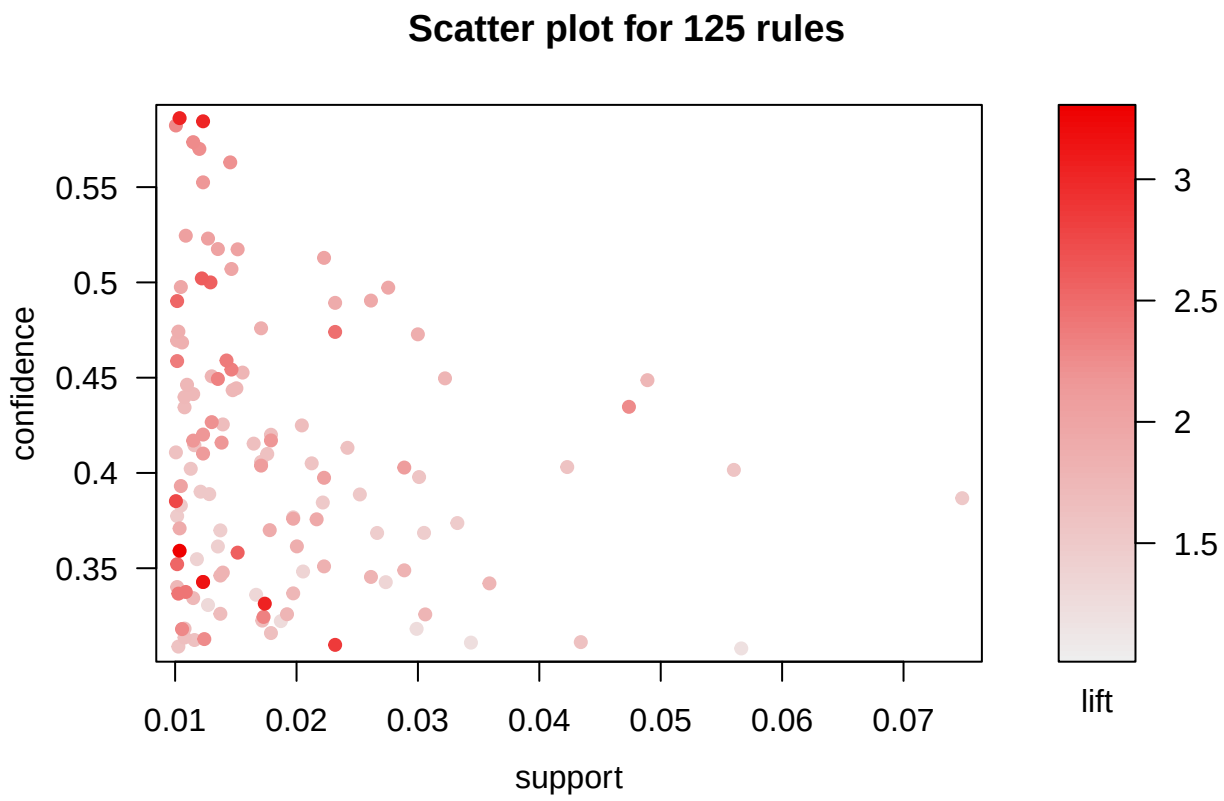
```r
inspect(head(sort(rules, by="lift"), 3))
```

```
##     lhs                    rhs                    support confidence     lift
## [1] {citrus fruit,
##      other vegetables} => {root vegetables} 0.01037112  0.3591549 3.295045
## [2] {tropical fruit,
##      other vegetables} => {root vegetables} 0.01230300  0.3427762 3.144780
## [3] {beef}             => {root vegetables} 0.01738688  0.3313953 3.040367
```

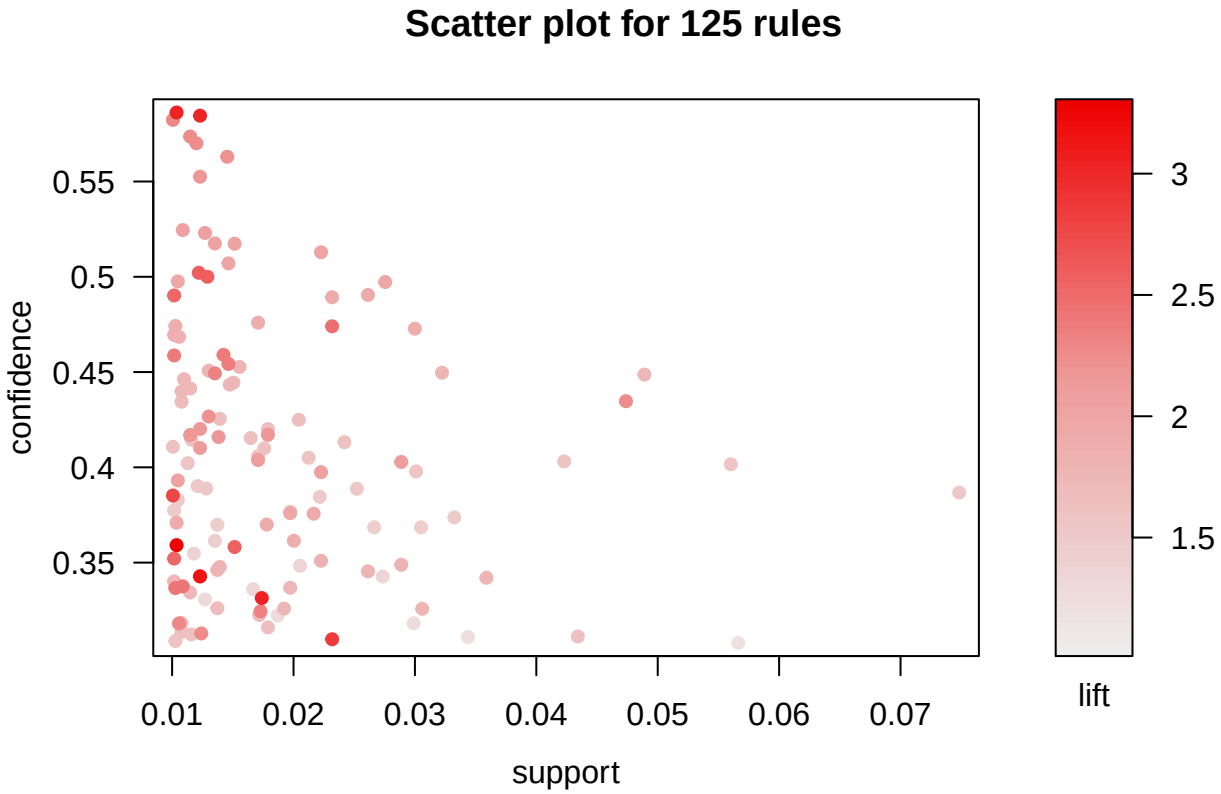
Explorar los ejemplos de la función plot.

Como ya lo hicimos antes, podemos primero pintar un plot sencillo con el soporte en x, la confianza en y y el lift con un código de color.

```
plot(rules)
```

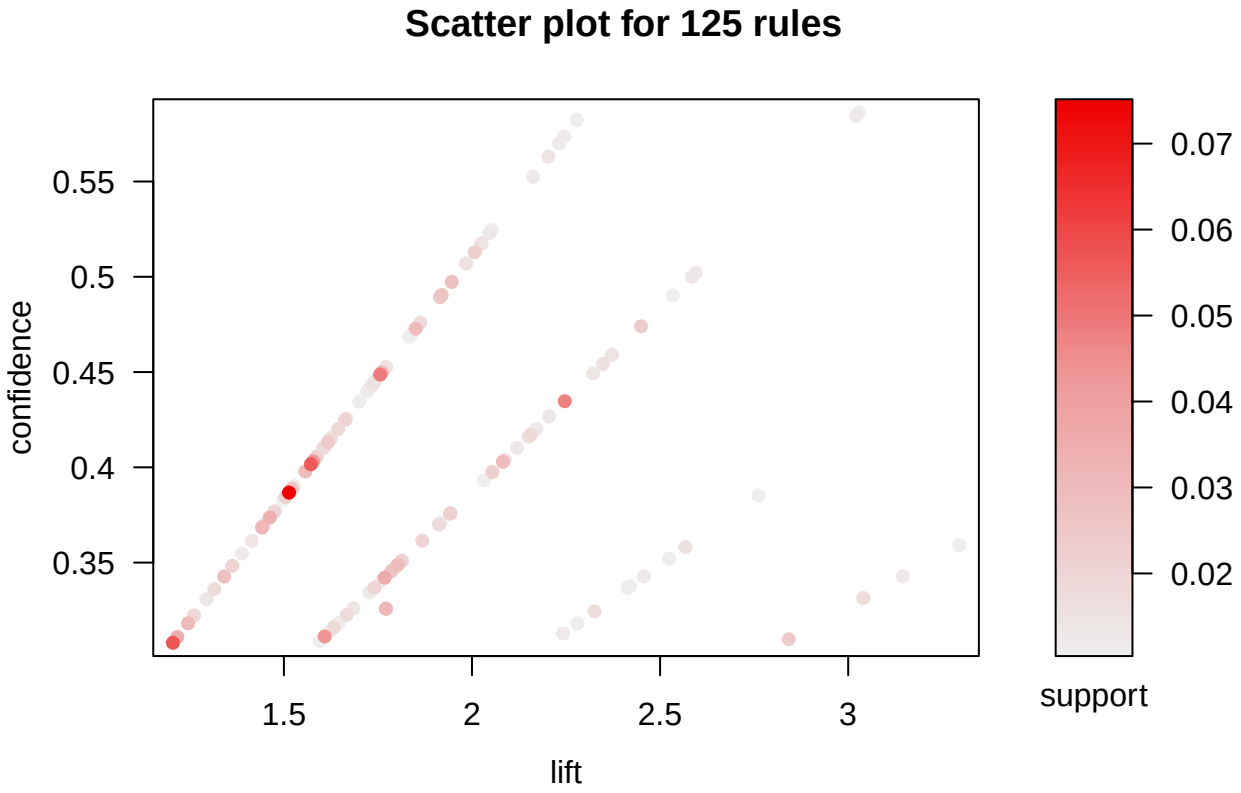



```
plot(rules, method = NULL, measure = "support", shading = "lift",
     interactive = F, data = NULL, control = NULL)
```



Podemos poner el argumento *interactive* a true para jugar con los datos directamente en el plot (seleccionar unos datos que nos interessan, o hacer un zoom por ejemplo). También podemos por supuesto cambiar las variables de eje (puede que nos interese pintar el soporte con el código de color, por ejemplo).

```
plot(rules, method = NULL, measure = "lift", shading = "support",
     interactive = F, data = NULL, control = NULL)
```



También se puede cambiar el método usado para pintar los datos, pero en este caso no he encontrado ningún método que nos permite representar la información que tenemos de mejor manera que poniendo el parámetro "method" a NULL.