Tutorial sobre extracción de reglas de asociación

Seminario Permanente de Formación en Inteligencia Artificial Aplicada a la Defensa

Pedro González y Cristóbal J. Carmona pglez@ujaen.es, ccarmona@ujaen.es
DaSCI, Universidad de Jaén

Junio de 2020

Contents

Este Cuaderno de R desarrolla un tutorial práctico sobre *Reglas de asociación. Aplicación al Análisis de la Cesta de la Compra* dentro del *Seminario Permanente de Formación en Inteligencia Artificial Aplicada a la Defensa* conjunto entre el Mando de Adiestramiento y Doctrina y el Instituto Andaluz Interuniversitario DaSCI (Universidad de Granada y Universidad de Jaén).

Introducción

El objetivo de este tutorial es desarrollar paso a paso un ejemplo de utilización de algortmos de extracción de reglas de asociación aplicado al problema del análisis de la cesta de la compra.

Para ello, se seguirán los siguientes pasos:

1. Introducir el concepto de reglas de asociacion.
2. Definir el problema del análisis de la cesta de la compra.
3. Instalar y cargar los paquetes de R necesarios.
4. Cargar un conjunto de datos de transacciones y entender su estructura mediante resúmenes estadísticos y visualización de datos.
5. Convertir los datos al formato necesario para la aplicación de algoritmos de reglas de asociación.
6. Aplicar algoritmos de reglas de asociación y ver los resultados.
7. Ajustar los parámetros para obtener mejores resultados.

Para facilitar el uso de este tutorial, los fragmentos de código fuente de R pueden ser ejecutados directamente. Para ello, basta con hacer clic en el botón *Run* dentro del trozo de código que se quiere ejecutar, o colocar el cursor dentro de él y pulsar *Cmd+Shift+Enter* (*Ctrl+Shift+Enter* en Windows).

Puede guardar el cuaderno con sus modificaciones (tanto texto como código R). Cuando guarde el cuaderno, se guardará un archivo HTML con el código y la salida junto a él (haga clic en el botón *Previsualizar* (*preview*) o pulsando *Cmd+Mayús+K* para obtener una vista previa del archivo HTML). Esta es una buena manera de compilar todas las tareas desarrolladas durante el tutorial.

Reglas de asociación

Los algoritmos de reglas de asociación tienen como objetivo encontrar relaciones dentro un conjunto de transacciones, en concreto, *items* o atributos que tienden a ocurrir de forma conjunta. En este contexto, el término transacción hace referencia a cada grupo de eventos que están asociados de alguna forma, por ejemplo:

- La cesta de la compra en un supermercado.
- Los libros que compra un cliente en una librería.
- Las páginas web visitadas por un usuario.
- Las características que aparecen de forma conjunta.

A cada uno de los eventos o elementos que forman parte de una transacción se le conoce como *item* y a un conjunto de ellos itemset. Una transacción puede estar formada por uno o varios *items*, en el caso de ser varios, cada posible subconjunto de ellos es un *itemset* distinto. Por ejemplo, la transacción T = *{A,B,C}* está formada por 3 *items (A, B y C)* y sus posibles *itemsets* son: *{A,B,C}*, *{A,B}*, *{B,C}*, *{A,C}*, *{A}*, *{B}* y *{C}*.

Una regla de asociación se define como una implicación del tipo "si X entonces Y" ($X => Y$), donde X e Y son *itemsets* o *items* individuales. El lado izquierdo de la regla recibe el nombre de antecedente o *lenft-hand-side (LHS)* y el lado derecho el nombre de consecuente o *right-hand- side (RHS)*. Por ejemplo, la regla *{A,B}* => *{C}* significa que, cuando ocurren A y B, también ocurre C.

Existen distintos algoritmos diseñados para identificar itemsets frecuentes y reglas de asociación. A continuación, se describen dos de los algoritmos de reglas de asociación más utilizados, que disponen de implementación en R.

Apriori

Apriori fue uno de los primeros algoritmos desarrollados para la búsqueda de reglas de asociación y sigue siendo uno de los más empleados, tiene dos etapas:

- Identificar todos los *itemsets* que ocurren con una frecuencia por encima de un determinado límite (*itemsets* frecuentes).
- Convertir esos *itemsets* frecuentes en reglas de asociación.

Encontrar *itemsets* frecuentes (*itemsets* con una frecuencia mayor o igual a un determinado soporte mínimo) no es un proceso trivial debido la explosión combinatoria de posibilidades, sin embargo, una vez identificados, es relativamente directo generar reglas de asociación que presenten una confianza mínima. El algoritmo *Apriori* hace una búsqueda exhaustiva por niveles de complejidad (de menor a mayor tamaño de *itemsets*). Para reducir el espacio de búsqueda aplica la norma de "si un *itemset* no es frecuente, ninguno de sus superconjuntos (*itemsets* de mayor tamaño que contengan al primero) puede ser frecuente". Visto de otra forma, si un conjunto es infrecuente, entonces, todos los conjuntos donde este último se encuentre, también son infrecuentes. Por ejemplo, si el *itemset {A,B}* es infrecuente, entonces, *{A,B,C}* y *{A,B,E}* también son infrecuentes ya que todos ellos contienen *{A,B}*.

El algorimto comienza identificando los *items* individuales que aparecen en el total de transacciones con una frecuencia por encima de un mínimo establecido por el usuario. A continuación, se sigue una estrategia bottom-up en la que se extienden los candidatos añadiendo un nuevo *item* y se eliminan aquellos que contienen un subconjunto infrecuente o que no alcanzan el soporte mínimo. Este proceso se repite hasta que el algoritmo no encuentra más ampliaciones exitosas de los *itemsets* previos o cuando se alcanza un tamaño máximo. Por último, se identican los *itemsets* frecuentes y se crean las reglas de asociación a partir de ellos.

FP-Growth

El algorimto *FP-Growth* fue propuesto por Han et al. (2000). Permite extraer reglas de asociación a partir de *itemsets* frecuentes pero, a diferencia del algoritmo *Apriori*, estos se identifican sin necesidad de generar candidatos para cada tamaño.

El algoritmo emplea una estructura de árbol (*Frequent Pattern Tree*) donde almacena toda la información de las transacciones. Esta estructura permite comprimir la información de una base de datos de transacciones hasta 200 veces, haciendo posible que pueda ser cargada en memoria RAM. Una vez que la base de datos ha sido comprimida en una estructura *FP-Tree*, se divide en varias bases de datos condicionales, cada una asociada con un patrón frecuente. Finalmente, cada partición se analiza de forma separada y se concatenan los resultados obtenidos. En la mayoría de casos, *FP-Growth* es más rápido que *Apriori*.

Instalar y cargar los paquetes R necesarios

A continación se describen los paquetes que contienen algoritmos de reglas de asociacion, y después se describe el proceso de instalacion y carga de los paquetes necesarios.

Paquete arules

El paquete `arules` implementa el algoritmo *Apriori* para la identificación de *itemsets* frecuentes y la creación de reglas de asociación a través de la función `apriori()`. `apriori()` recibe como argumento un objeto `transaction` con los datos de las transacciones, un argumento `parameter` que determina las características de los *itemsets* o reglas generadas (por ejemplo, el soporte mínimo) y un argumento `control` que determina el comportamiento del algoritmo (por ejemplo, ordenación de los resultados).

La función `apriori()` también incluye el argumento `aparence` que impone restricciones sobre las reglas generadas (como crear solo reglas que contengan un determinado *item*).

El resultado de ambas funciones es un objeto de tipo `association` que puede ser manipulado por las funciones que ofrece `arules`, entre las que destacan:

- `summary()`: muestra un resumen de los resultados.
- `inspect()`: muestra los resultados.
- `lenght()`: número de elementos (reglas o *itemsets*) almacenados.

- `items()`: extrae los *items* que forman un *itemset* o una regla. En el caso de reglas, combina los *items* de antecedente (*LHS*) y del consecuente (*RHS*).
- `sort()`: ordena los resultados.
- `subset`: filtra los resultados.

Paquete rCBA

El paquete `rCBA` proporciona implementaciones de un clasificador basado en la *"Clasificación Basada en Asociaciones" (CBA)*. Una de las funciones que ofrece, `fpgrowth`, implementa el algoritmo *FP-Growth*, propuesto por J. Han, J. Pei e Y. Yin. Para utilizar este algormito debemos utilizar la siguiente sintaxis:

```
fpgrowth(train, support = 0.01, confidence = 1, maxLength = 5,
         consequent = NULL, verbose = TRUE, parallel = TRUE)
```

Estos paámetros significan lo siguiente:

- `train`: es un `data.frame` o `transactions` de `arules` con datos de entrada.
- `support`: soporte mínimo.
- `confidence`: confianza mínima.
- `maxLength`+ : longitud máxima de las reglas
- `consequent`: para filtrar el consecuente, contiene el nombre de la columna que formará el consecuente.
- `verbose`: indica si se mostrará información adicional.
- `parallel`: indicador de ejecución paralela.

Intrucciones para la instalación y carga de los paquetes

Para la instalación de los paquetes, se debe ejecutar el siguiente código R (es normal que la instalación de un paquete requiera paquetes adicionales, pero la instalación procederá a instalar los paquetes adicionales necesarios):

```
if(! "arules" %in% installed.packages())
  install.packages("arules", depend = TRUE, repos = "http://cran.r-project.org")
if(! "rCBA" %in% installed.packages())
  install.packages("rCBA", depend = TRUE, repos = "http://cran.r-project.org")
if(! "tidyverse" %in% installed.packages())
  install.packages("tidyverse", depend = TRUE, repos = "http://cran.r-project.org")
```

Una vez instalados los paquetes, es necesario cargarlos:

```
library(arules)
```

```
## Loading required package: Matrix
```

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

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

```
library(rCBA)
```

```
## Loading required package: rJava
```

```
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.1     v dplyr   1.0.0
```

```
## v tidyr   1.1.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------------------ tidyverse_conflicts() --
## x tidyr::expand() masks Matrix::expand()
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x tidyr::pack()   masks Matrix::pack()
## x dplyr::recode() masks arules::recode()
## x tidyr::unpack() masks Matrix::unpack()
```

Descripción del problema de la cesta de la compra

Supongamos que tenemos el siguiente problema:

Se dispone del registro de todas las compras que se han realizado en un supermercado. El objetivo del análisis es identificar productos que tiendan a comprarse de forma conjunta para así poder situarlos en posiciones cercanas dentro de la tienda y maximizar la probabilidad de que los clientes compren.

Para este ejemplo se emplea el conjunto de datos `groceries` del paquete `arules`, que contiene un registro de todas las ventas realizadas por un supermercado durante 30 días. En total se dispone de 9835 transacciones formadas por combinaciones de 169 productos. El objeto `Groceries` almacena la información en un formato propio de este paquete (descrito más adelante).

Para representar mejor lo que suele ocurrir en la realidad, se ha reestructurado la información en formato de tabla. Los datos están disponibles junto con este fichero.

```
datos <- read_csv(file = "./datos_groceries.csv", col_names = TRUE)
```

```
## Parsed with column specification:
## cols(
##   id_compra = col_double(),
##   item = col_character()
## )
```

```
head(datos)
```

```
## # A tibble: 6 x 2
##   id_compra item
##       <dbl> <chr>
## 1         1 citrus fruit
## 2         1 semi-finished bread
## 3         1 margarine
## 4         1 ready soups
## 5         2 tropical fruit
## 6         2 yogurt
```

Cada línea del archivo contiene la información de un *item* y el identificador de la transacción (compra) a la que pertenece. Esta es la estructura en la que comúnmente se almacenan los datos dentro de una base de datos y que, en el ámbito de las transacciones, recibe el nombre de tabla larga o *single*. Otra forma en la que se pueden encontrar los datos de transacciones es en formato matriz o tabla ancha, en el que cada fila contiene todos los items que forman parte de una misma transacción, este formato recibe el nombre de cesta o *basket*.

Tal y como se ha definido previamente, el concepto de transacción hace referencia al conjunto de *items* o eventos que ocurren de forma conjunta. Para este caso de estudio, compras de supermercado, cada transacción está formada por todos los productos que se compran a la vez, es decir, el vínculo de unión no es el cliente sino cada una de las "cestas de la compra". Por ejemplo, la transacción con `id_compra == 14` está formada por 3 *items*.

```r
datos %>% filter(id_compra == 14) %>% pull(item)
```

```
## [1] "frankfurter" "rolls/buns"  "soda"
```

Lectura de datos

Con la función `read.transactions()` se pueden leer directamente los datos de archivos tipo texto y almacenarlos en un objeto de tipo `transactions`, que es la estructura de almacenamiento que emplea `arules`. Esta función tiene los siguientes argumentos:

- `file`: nombre del archivo que se quiere leer.
- `format`: estructura en la que se encuentran almacenados los datos, *"basket"* si cada línea del archivo es una transacción completa, o *"single"* si cada línea representa un *item*.
- `sep`: tipo de separación de los campos.
- `cols`: si el formato es de tipo *"basket"*, un entero que indica la columna que contiene el identificador de las transacciones. Para el formato *"single"*, es un vector con los nombres (o posiciones) de las dos columnas que identifican las transacciones y los *items*, respectivamente.
- `rm.duplicates`: valor lógico indicando si se eliminan los *items* duplicados en una misma transacción. Por lo general, es conveniente eliminarlos, el interés suele estar en qué *items* ocurren de forma conjunta, no en qué cantidad.
- `quote`: carácter que se utiliza como comillas.
- `skip`: número de líneas que hay que saltar desde el comienzo del fichero.

Es importante recordar que los objetos de tipo `transactions`solo trabajan con información booleana, es decir, con la presencia o no de cada uno de los *items* en la transacción. Por lo tanto, si el set de datos contiene variables numéricas, estas deben de ser discretizadas en intervalos o niveles. Para discretizar una variable numérica, se puede emplear la función `discretize()` o bien otras alternativas como la función `case_when()` del paquete `dplyr`.

Los objetos `transactions`, se almacenan internamente como un tipo de matriz binaria. Se trata de una matriz de valores 0/1, con una fila por cada transacción, en este caso cada compra, y una columna por cada posible *item*, en este caso productos. La posición de la matriz *(i,j)* tiene el valor 1 si la transacción *i* contiene el *item j*.

```r
# IMPORTACIÓN DIRECTA DE LOS DATOS A UN OBJETO TIPO TRANSACTION
# ==============================================================================
library(arules)
transacciones <- read.transactions(file = "./datos_groceries.csv",
                                   format = "single",
                                   sep = ",",
                                   header = TRUE,
                                   cols = c(1,2),
                                   rm.duplicates = TRUE)
transacciones
```

```
## transactions in sparse format with
##  9835 transactions (rows) and
##  169 items (columns)
```

```r
colnames(transacciones)[1:5]
```

```
## [1] "abrasive cleaner" "artif. sweetener" "baby cosmetics"   "baby food"
## [5] "bags"
```

```r
rownames(transacciones)[1:5]
```

```
## [1] "1"    "10"   "100"  "1000" "1001"
```

También es posible convertir un objeto `dataframe` en uno de tipo `transactions` con la función `as(dataframe, "transactions")`. Para lograrlo, primero hay que convertir el `dataframe` en una lista en la que cada elemento contiene los *items* de una transacción. Este proceso puede ser muy lento si el `dataframe` tiene muchos registros, por lo que suele ser mejor crear un archivo de texto con los datos e importarlo mediante `read.transactions()`.

```
# CONVERSIÓN DE UN DATAFRAME A UN OBJETO TIPO TRANSACTION
# ==============================================================================
# Se convierte el dataframe a una lista en la que cada elemento contiene los
# items de una transacción
datos_split <- split(x = datos$item, f = datos$id_compra)
transacciones <- as(datos_split, Class = "transactions")
transacciones
```

```
## transactions in sparse format with
##  9835 transactions (rows) and
##  169 items (columns)
```

Otra alternativa es convertir el `dataframe` en una matriz 0/1 en la que cada fila es una transacción y cada columna uno de los posibles *items*.

```
# CONVERSIÓN DE UNA MATRIZ A UN OBJETO TIPO TRANSACTION
# ==============================================================================
datos_matriz <- datos %>%
                as.data.frame() %>%
                mutate(valor = 1) %>%
                spread(key = item, value = valor, fill = 0) %>%
                column_to_rownames(var = "id_compra") %>%
                as.matrix()
transacciones <- as(datos_matriz, Class = "transactions")
transacciones
```

```
## transactions in sparse format with
##  9835 transactions (rows) and
##  169 items (columns)
```

Exploración de items

Uno de los primeros aspectos a analizar cuando se trabaja con transacciones es explorar su contenido y tamaño; es decir, el número de *items* que las forman y cuáles son. La función `inspect()` muestra los *items* que forman cada transacción.

```
inspect(transacciones[1:5])
```

```
##     items                      transactionID
## [1] {citrus fruit,
##      margarine,
##      ready soups,
##      semi-finished bread}                1
## [2] {coffee,
##      tropical fruit,
##      yogurt}                             2
## [3] {whole milk}                         3
## [4] {cream cheese,
##      meat spreads,
##      pip fruit,
##      yogurt}                             4
```

```
## [5] {condensed milk,
##      long life bakery product,
##      other vegetables,
##      whole milk}                              5
```

También es posible mostrar los resultados en formato de *dataframe* con la función `DATAFRAME()` o con `as(transacciones, "dataframe")`.

```
df_transacciones <- as(transacciones, Class = "data.frame")
# Para que el tamaño de la tabla se ajuste mejor
as.tibble(df_transacciones) %>% head()
```

```
## Warning: `as.tibble()` is deprecated as of tibble 2.0.0.
## Please use `as_tibble()` instead.
## The signature and semantics have changed, see `?as_tibble`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
## # A tibble: 6 x 2
##   items                                                          transactionID
##   <chr>                                                          <chr>
## 1 {citrus fruit,margarine,ready soups,semi-finished bread}       1
## 2 {coffee,tropical fruit,yogurt}                                 2
## 3 {whole milk}                                                   3
## 4 {cream cheese,meat spreads,pip fruit,yogurt}                   4
## 5 {condensed milk,long life bakery product,other vegetables,whole~ 5
## 6 {abrasive cleaner,butter,rice,whole milk,yogurt}               6
```

Para extraer el tamaño de cada transacción se emplea la función `size()`.

```
tamanos <- size(transacciones)
summary(tamanos)
```

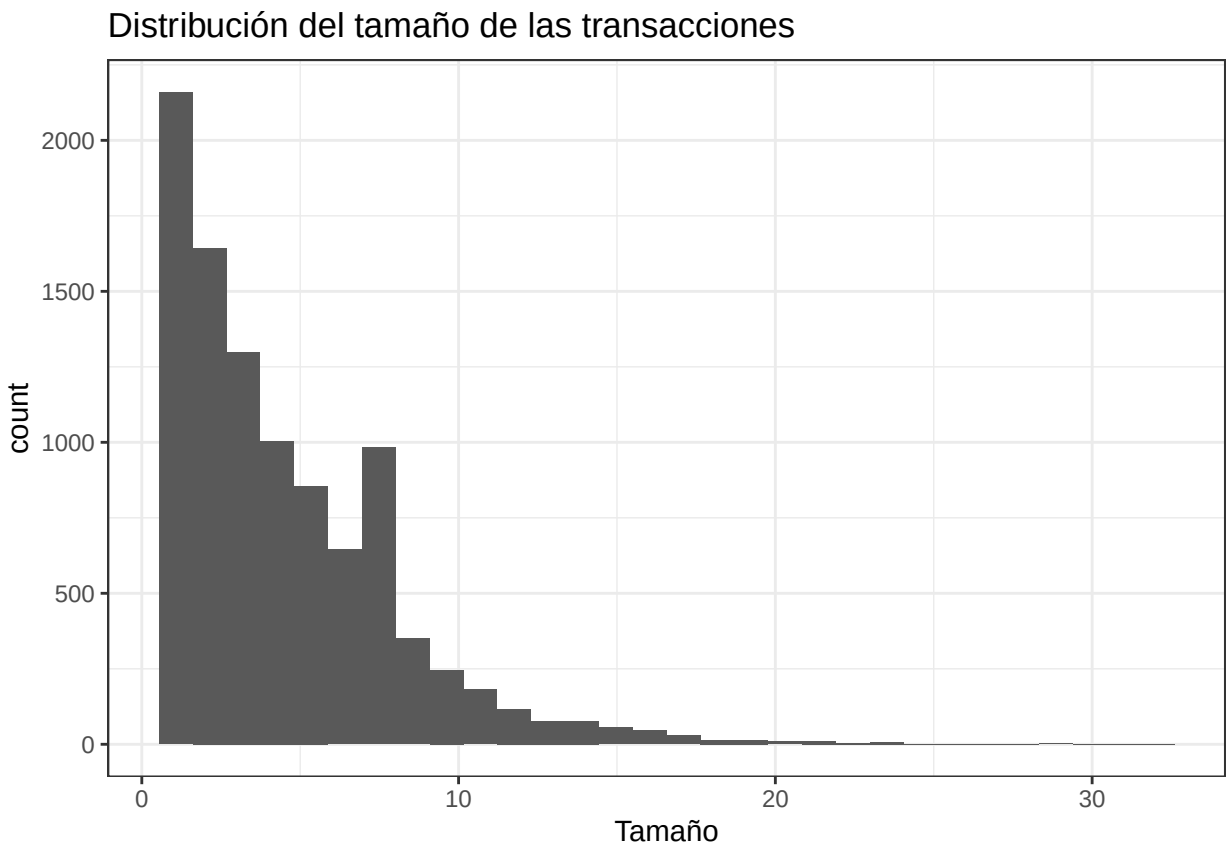
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   3.000   4.409   6.000  32.000
```

```
quantile(tamanos, probs = seq(0,1,0.1))
```

```
##   0%  10%  20%  30%  40%  50%  60%  70%  80%  90% 100%
##    1    1    1    2    3    3    4    5    7    9   32
```

```
data.frame(tamanos) %>%
  ggplot(aes(x = tamanos)) +
  geom_histogram() +
  labs(title = "Distribución del tamaño de las transacciones", x = "Tamaño") +
  theme_bw()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



La gran mayoría de clientes compra entre 3 y 4 productos y el 90% de ellos compra como máximo 9.

El siguiente análisis básico consiste en identificar cuáles son los items más frecuentes (los que tienen mayor soporte) dentro del conjunto de todas las transacciones. Con la función `itemFrequency()`se puede extraer esta información de un objeto tipo `transactions`. El nombre de esta función puede causar confusión. Por "frecuencia" se hace referencia al soporte de cada *item*, que es la fracción de transacciones que contienen dicho *item* respecto al total de todas las transacciones. Esto es distinto a la frecuencia de un *item* respecto al total de *items*, de ahí que la suma de todos los soportes no sea 1.

```
frecuencia_items <- itemFrequency(x = transacciones, type = "relative")
frecuencia_items %>% sort(decreasing = TRUE) %>% head(5)
```

```
##       whole milk other vegetables       rolls/buns             soda
##        0.2555160        0.1934926        0.1839349        0.1743772
##           yogurt
##        0.1395018
```

Si se indica el argumento `type = "absolute"`, la función `itemFrequency()`devuelve el número de transacciones en las que aparece cada *item*.

```
frecuencia_items <- itemFrequency(x = transacciones, type = "absolute")
frecuencia_items %>% sort(decreasing = TRUE) %>% head(5)
```

```
##       whole milk other vegetables       rolls/buns             soda
##             2513             1903             1809             1715
##           yogurt
##             1372
```

El listado anterior muestra que los 5 productos que más se venden son: *whole milk*, *other vegetables*, *rolls/buns*

y *soda*.

Es muy importante estudiar cómo se distribuye el soporte de los *items* individuales en un conjunto de transacciones antes identificar *itemsets* frecuentes o crear reglas de asociación, ya que, dependiendo del caso, tendrá sentido emplear un límite de soporte u otro. Por lo general, cuando el número de posibles *items* es muy grande (varios miles) prácticamente todos los artículos son raros, por lo que los soportes son muy bajos.

Itemsets

Con la función `apriori()` se puede aplicar el algoritmo *Apriori* a un objeto de tipo `transactions`y extraer tanto *itemsets* frecuentes como reglas de asociación que superen un determinado soporte y confianza. Los argumentos de esta función son: * `data`: un objeto del tipo transactions o un objeto que pueda ser convertido a tipo `transactions`, por ejemplo un `dataframe` o una matriz binaria. * `parameter`: lista en la que se indican los parámetros del algoritmo. * `support`: soporte mínimo que debe tener un *itemset* para ser considerado frecuente. Por defecto es 0.1. * `minlen`: número mínimo de *items* que debe tener un *itemset* para ser incluido en los resultados. Por defecto 1. * `maxlen`: número máximo de *items* que puede tener un *itemset* para ser incluido en los resultados. Por defecto 10. * `target`: tipo de resultados que debe de generar el algoritmo, pueden ser "frequent itemsets", "maximally frequent itemsets", "closed frequent itemsets", "rules" o "hyperedgesets". * `confidence`: confianza mínima que debe de tener una regla para ser incluida en los resultados. Por defecto 0.8. * `maxtime`: tiempo máximo que puede estar el algoritmo buscando subsets. Por defecto 5 segundos. * `appearance`: lista que permite definir patrones para restringir el espacio de búsqueda, por ejemplo, especificando qué *items* pueden o no pueden aparecer. * `control`: lista que permite modificar 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.

Se procede a extraer aquellos *itemsets*, incluidos los formados por un único *item*, que hayan sido comprados al menos 30 veces. En un caso real, este valor sería excesivamente bajo si se tiene en cuenta la cantidad total de transacciones, sin embargo, se emplea 30 para que en los resultados aparezcan un número suficiente de *itemsets* y reglas de asociación que permitan mostrar las posibilidades de análisis que ofrece el paquete `arules`.

```
soporte <- 30 / dim(transacciones)[1]
itemsets <- apriori(data = transacciones,
                    parameter = list(support = soporte,
                                     minlen = 1,
                                     maxlen = 20,
                                     target = "frequent itemset"))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime    support minlen
##          NA    0.1    1 none FALSE            TRUE       5 0.00305033      1
##  maxlen            target  ext
##      20 frequent itemsets TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 30
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [136 item(s)] done [0.00s].
```

```
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 done [0.00s].
## sorting transactions ... done [0.00s].
## writing ... [2226 set(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
summary(itemsets)
```

```
## set of 2226 itemsets
##
## most frequent items:
##       whole milk other vegetables           yogurt  root vegetables
##              556              468              316              251
##       rolls/buns          (Other)
##              241             3536
##
## element (itemset/transaction) length distribution:sizes
##    1    2    3    4    5
##  136 1140  850   98    2
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   2.000   2.412   3.000   5.000
##
## summary of quality measures:
##     support         transIdenticalToItemsets     count
##  Min.   :0.003050   Min.   :0.0000000        Min.   :  30.00
##  1st Qu.:0.003660   1st Qu.:0.0000000        1st Qu.:  36.00
##  Median :0.004779   Median :0.0000000        Median :  47.00
##  Mean   :0.007879   Mean   :0.0001613        Mean   :  77.49
##  3rd Qu.:0.007219   3rd Qu.:0.0001017        3rd Qu.:  71.00
##  Max.   :0.255516   Max.   :0.0265379        Max.   :2513.00
##
## includes transaction ID lists: FALSE
##
## mining info:
##           data ntransactions    support confidence
##  transacciones          9835 0.00305033          1
```

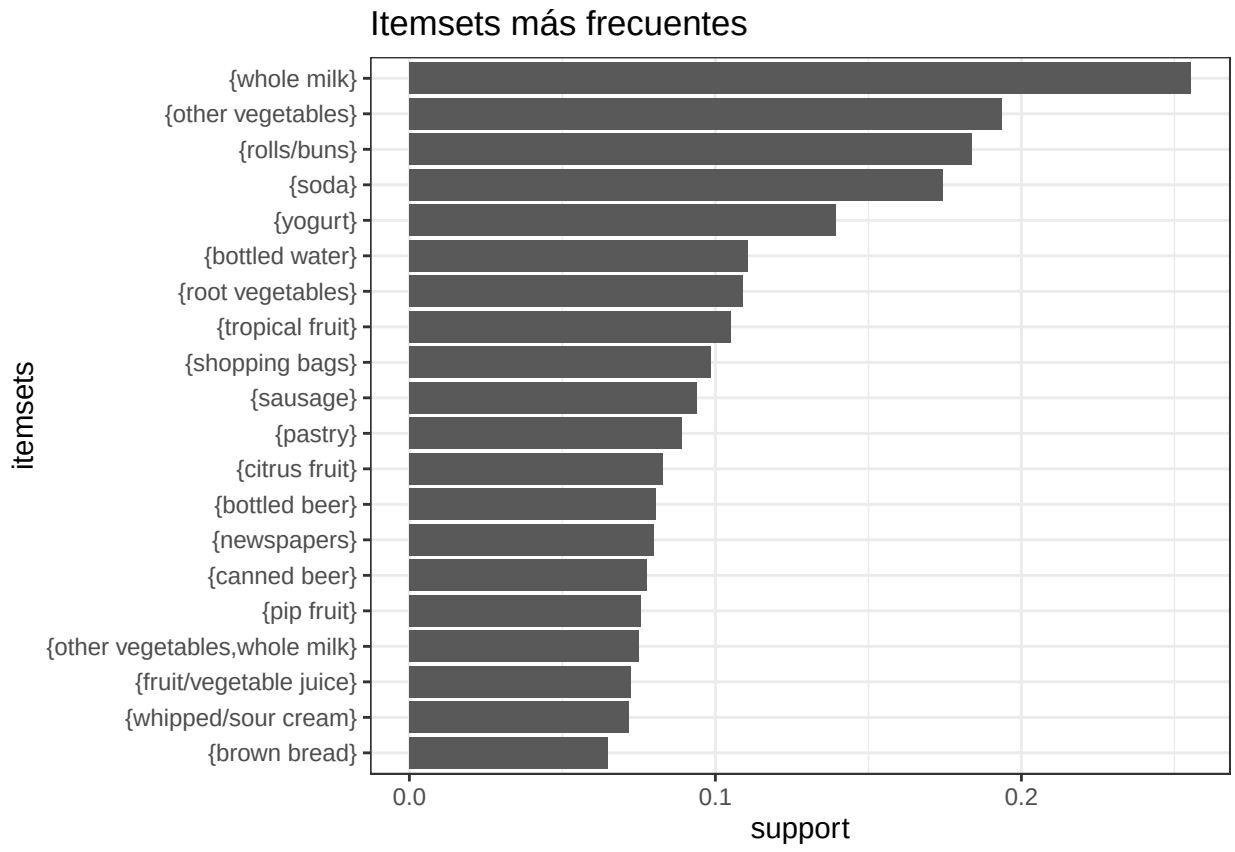
Se han encontrado un total de 2226 *itemsets* frecuentes que superan el soporte mínimo de 0.003908286, la mayoría de ellos (1140) formados por dos *items*. En el siguiente listado se muestran los 20 *itemsets* con mayor soporte que, como cabe esperar, son los formados por items individuales (los *itemsets* de menor tamaño).

```
# Se muestran los top 20 itemsets de mayor a menor soporte
top_20_itemsets <- sort(itemsets, by = "support", decreasing = TRUE)[1:20]
inspect(top_20_itemsets)
```

```
##      items                      support    transIdenticalToItemsets count
## [1]  {whole milk}               0.25551601 0.0125063549             2513
## [2]  {other vegetables}         0.19349263 0.0063040163             1903
## [3]  {rolls/buns}               0.18393493 0.0111845450             1809
## [4]  {soda}                     0.17437722 0.0162684291             1715
## [5]  {yogurt}                   0.13950178 0.0040671073             1372
## [6]  {bottled water}            0.11052364 0.0068124047             1087
## [7]  {root vegetables}          0.10899847 0.0025419420             1072
## [8]  {tropical fruit}           0.10493137 0.0023385867             1032
## [9]  {shopping bags}            0.09852567 0.0048805287              969
```

```
## [10] {sausage}                       0.09395018 0.0024402644            924
## [11] {pastry}                        0.08896797 0.0037620742            875
## [12] {citrus fruit}                  0.08276563 0.0017285206            814
## [13] {bottled beer}                  0.08052872 0.0124046772            792
## [14] {newspapers}                    0.07981698 0.0055922725            785
## [15] {canned beer}                   0.07768175 0.0265378749            764
## [16] {pip fruit}                     0.07564820 0.0028469751            744
## [17] {other vegetables,whole milk}   0.07483477 0.0006100661            736
## [18] {fruit/vegetable juice}         0.07229283 0.0022369090            711
## [19] {whipped/sour cream}            0.07168277 0.0026436197            705
## [20] {brown bread}                   0.06487036 0.0021352313            638
```

```
# Para representarlos con ggplot se convierte a dataframe
as(top_20_itemsets, Class = "data.frame") %>%
  ggplot(aes(x = reorder(items, support), y = support)) +
  geom_col() +
  coord_flip() +
  labs(title = "Itemsets más frecuentes", x = "itemsets") +
  theme_bw()
```



Si se quieren excluir del análisis los *itemsets* formados únicamente por un solo *item*, se puede, o bien aplicar de nuevo la función `apriori()` especificando `minlen = 2`, o filtrar los resultados con la función `size()`.

```
# Se muestran los 20 itemsets más frecuentes formados por más de un item.
inspect(sort(itemsets[size(itemsets) > 1], decreasing = TRUE)[1:20])
```

```
##      items                              support    transIdenticalToItemsets
## [1]  {other vegetables,whole milk}      0.07483477 0.0006100661
```

```
## [2]  {rolls/buns,whole milk}            0.05663447 0.0013218099
## [3]  {whole milk,yogurt}                0.05602440 0.0008134215
## [4]  {root vegetables,whole milk}       0.04890696 0.0003050330
## [5]  {other vegetables,root vegetables} 0.04738180 0.0006100661
## [6]  {other vegetables,yogurt}          0.04341637 0.0004067107
## [7]  {other vegetables,rolls/buns}      0.04260295 0.0008134215
## [8]  {tropical fruit,whole milk}        0.04229792 0.0008134215
## [9]  {soda,whole milk}                  0.04006101 0.0007117438
## [10] {rolls/buns,soda}                  0.03833249 0.0023385867
## [11] {other vegetables,tropical fruit}  0.03589222 0.0005083884
## [12] {bottled water,whole milk}         0.03436706 0.0007117438
## [13] {rolls/buns,yogurt}                0.03436706 0.0010167768
## [14] {pastry,whole milk}                0.03324860 0.0012201322
## [15] {other vegetables,soda}            0.03274021 0.0001016777
## [16] {whipped/sour cream,whole milk}    0.03223183 0.0008134215
## [17] {rolls/buns,sausage}               0.03060498 0.0013218099
## [18] {citrus fruit,whole milk}          0.03050330 0.0003050330
## [19] {pip fruit,whole milk}             0.03009659 0.0004067107
## [20] {domestic eggs,whole milk}         0.02999492 0.0003050330
##      count
## [1]  736
## [2]  557
## [3]  551
## [4]  481
## [5]  466
## [6]  427
## [7]  419
## [8]  416
## [9]  394
## [10] 377
## [11] 353
## [12] 338
## [13] 338
## [14] 327
## [15] 322
## [16] 317
## [17] 301
## [18] 300
## [19] 296
## [20] 295
```

Filtrado de itemsets

Una vez que los *itemsets* frecuentes han sido identificados mediante el algoritmo *Apriori*, pueden ser filtrados con la función `subset()`. Esta función recibe dos argumentos: un objeto `itemset`o `rules`y una condición lógica que tienen que cumplir las reglas/*itemsets* para ser seleccionados. La siguiente tabla muestra los operadores permitidos:

Operador	**Significado**
&	AND
%in%	contiene cualquier de los siguientes elementos
%ain%	contiene todos de los siguientes elementos

Operador	Significado
%pin%	contiene parcialmente los siguientes elementos

Como esta función tiene el mismo nombre que una función del paquete básico de R, para evitar errores, es conveniente especificar el paquete donde se encuentra.

Se procede a identificar aquellos *itemsets* frecuentes que contienen el *item newspapers*.

```
itemsets_filtrado <- arules::subset(itemsets, subset = items %in%   "newspapers")
itemsets_filtrado
```

```
## set of 80 itemsets
```

```
# Se muestran 10 de ellos
inspect(itemsets_filtrado[1:10])
```

```
##      items                                  support     transIdenticalToItemsets
## [1]  {newspapers}                           0.079816980 0.0055922725
## [2]  {meat,newspapers}                      0.003050330 0.0001016777
## [3]  {newspapers,sliced cheese}             0.003152008 0.0000000000
## [4]  {newspapers,UHT-milk}                  0.004270463 0.0001016777
## [5]  {newspapers,oil}                       0.003152008 0.0000000000
## [6]  {newspapers,onions}                    0.003152008 0.0000000000
## [7]  {hygiene articles,newspapers}          0.003050330 0.0000000000
## [8]  {newspapers,sugar}                     0.003152008 0.0000000000
## [9]  {newspapers,waffles}                   0.004168785 0.0001016777
## [10] {long life bakery product,newspapers} 0.003457041 0.0002033554
##      count
## [1]  785
## [2]   30
## [3]   31
## [4]   42
## [5]   31
## [6]   31
## [7]   30
## [8]   31
## [9]   41
## [10]  34
```

Se repite el proceso pero, esta vez, con aquellos *itemsets* que contienen *newspapers* y *whole milk*.

```
itemsets_filtrado <- arules::subset(itemsets,
                                    subset = items %ain% c("newspapers", "whole milk"))
itemsets_filtrado
```

```
## set of 16 itemsets
```

```
# Se muestran 10 de ellos
inspect(itemsets_filtrado[1:10])
```

```
##      items                                 support
## [1]  {newspapers,whole milk}               0.027351296
## [2]  {chocolate,newspapers,whole milk}     0.003152008
## [3]  {brown bread,newspapers,whole milk}   0.004067107
## [4]  {margarine,newspapers,whole milk}     0.003152008
## [5]  {butter,newspapers,whole milk}        0.003152008
## [6]  {newspapers,pastry,whole milk}        0.003863752
```

```
## [7]  {citrus fruit,newspapers,whole milk}   0.003355363
## [8]  {newspapers,sausage,whole milk}        0.003050330
## [9]  {bottled water,newspapers,whole milk}  0.004067107
## [10] {newspapers,tropical fruit,whole milk} 0.005083884
##      transIdenticalToItemsets count
## [1]  0.0008134215             269
## [2]  0.0000000000              31
## [3]  0.0002033554              40
## [4]  0.0000000000              31
## [5]  0.0001016777              31
## [6]  0.0003050330              38
## [7]  0.0000000000              33
## [8]  0.0000000000              30
## [9]  0.0001016777              40
## [10] 0.0001016777              50
```

Puede observarse que muchos *itemsets* están a su vez contenidos en *itemsets* de orden superior, es decir, existen *itemsets* que son subconjuntos de otros. Para identificar cuáles son, o cuales no lo son, se puede emplear la función `is.subset()`. Encontrar los *itemsets* que son subconjuntosd de otros *itemsets* implica comparar todos los pares de *itemsets* y determinar si uno está contenido en el otro. La función `is.subset()` realiza comparaciones entre dos conjuntos de *itemsets* y devuelve una matriz lógica que determina si el *itemset* de la fila está contenido en cada *itemset* de las columnas.

```
# Para encontrar los subsets dentro de un conjunto de itemsets, se compara el
# conjunto de itemsets con sigo mismo.
subsets <- is.subset(x = itemsets, y = itemsets, sparse = FALSE)
```

Para conocer el total de *itemsets* que son subconjuntos de otros *itemsets* se cuenta el número total de `TRUE` en la matriz resultante.

```
# La suma de una matriz lógica devuelve el número de TRUEs
sum(subsets)
```

```
## [1] 11038
```

Reglas de asociación

Para crear las reglas de asociación se sigue el mismo proceso que para obtener *itemsets* frecuentes pero, además de especificar un soporte mínimo, se tiene que establecer una confianza mínima para que una regla se incluya en los resultados. En este caso, se emplea una confianza mínima del 70%.

```
soporte <- 30 / dim(transacciones)[1]
reglas <- apriori(data = transacciones,
                  parameter = list(support = soporte,
                                   confidence = 0.70,
                                   # Se especifica que se creen reglas
                                   target = "rules"))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime    support minlen
##         0.7    0.1    1 none FALSE            TRUE       5 0.00305033      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: 30
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [136 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 done [0.00s].
## writing ... [19 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
summary(reglas)
```

```
## set of 19 rules
##
## rule length distribution (lhs + rhs):sizes
## 3 4 5
## 7 9 3
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.000   3.000   4.000   3.789   4.000   5.000
##
## summary of quality measures:
##     support           confidence        coverage             lift
##  Min.   :0.003050   Min.   :0.7000   Min.   :0.003559   Min.   :2.740
##  1st Qu.:0.003203   1st Qu.:0.7047   1st Qu.:0.004525   1st Qu.:2.758
##  Median :0.003559   Median :0.7164   Median :0.004982   Median :2.804
##  Mean   :0.003767   Mean   :0.7373   Mean   :0.005143   Mean   :3.044
##  3rd Qu.:0.004169   3rd Qu.:0.7500   3rd Qu.:0.005592   3rd Qu.:2.984
##  Max.   :0.005694   Max.   :0.8857   Max.   :0.008134   Max.   :4.578
##      count
##  Min.   :30.00
##  1st Qu.:31.50
##  Median :35.00
##  Mean   :37.05
##  3rd Qu.:41.00
##  Max.   :56.00
##
## mining info:
##           data ntransactions    support confidence
##  transacciones          9835 0.00305033        0.7
```

Se han identificado un total de 19 reglas, la mayoría de ellas formadas por 4 *items* en el antecedente (parte izquierda de la regla).

```
inspect(sort(x = reglas, decreasing = TRUE, by = "confidence"))
```

```
##      lhs                     rhs                   support confidence    coverage     lift count
## [1]  {citrus fruit,
##       root vegetables,
##       tropical fruit,
##       whole milk}         => {other vegetables} 0.003152008  0.8857143 0.003558719 4.577509    31
## [2]  {butter,
```

```
##       root vegetables,
##       yogurt}             => {whole milk}       0.003050330  0.7894737 0.003863752 3.089723    30
## [3]  {citrus fruit,
##       root vegetables,
##       tropical fruit}     => {other vegetables} 0.004473818  0.7857143 0.005693950 4.060694    44
## [4]  {brown bread,
##       other vegetables,
##       root vegetables}    => {whole milk}       0.003152008  0.7750000 0.004067107 3.033078    31
## [5]  {butter,
##       onions}             => {whole milk}       0.003050330  0.7500000 0.004067107 2.935237    30
## [6]  {curd,
##       tropical fruit,
##       yogurt}             => {whole milk}       0.003965430  0.7500000 0.005287239 2.935237    39
## [7]  {curd,
##       domestic eggs}      => {whole milk}       0.004778851  0.7343750 0.006507372 2.874086    47
## [8]  {butter,
##       tropical fruit,
##       yogurt}             => {whole milk}       0.003355363  0.7333333 0.004575496 2.870009    33
## [9]  {root vegetables,
##       tropical fruit,
##       whipped/sour cream} => {other vegetables} 0.003355363  0.7333333 0.004575496 3.789981    33
## [10] {butter,
##       curd}               => {whole milk}       0.004880529  0.7164179 0.006812405 2.803808    48
## [11] {domestic eggs,
##       sugar}              => {whole milk}       0.003558719  0.7142857 0.004982206 2.795464    35
## [12] {other vegetables,
##       root vegetables,
##       tropical fruit,
##       yogurt}             => {whole milk}       0.003558719  0.7142857 0.004982206 2.795464    35
## [13] {baking powder,
##       yogurt}             => {whole milk}       0.003253686  0.7111111 0.004575496 2.783039    32
## [14] {tropical fruit,
##       whipped/sour cream,
##       yogurt}             => {whole milk}       0.004372140  0.7049180 0.006202339 2.758802    43
## [15] {citrus fruit,
##       other vegetables,
##       root vegetables,
##       tropical fruit}     => {whole milk}       0.003152008  0.7045455 0.004473818 2.757344    31
## [16] {butter,
##       pork}               => {whole milk}       0.003863752  0.7037037 0.005490595 2.754049    38
## [17] {butter,
##       coffee}             => {whole milk}       0.003355363  0.7021277 0.004778851 2.747881    33
## [18] {domestic eggs,
##       other vegetables,
##       whipped/sour cream} => {whole milk}       0.003558719  0.7000000 0.005083884 2.739554    35
## [19] {root vegetables,
##       tropical fruit,
##       yogurt}             => {whole milk}       0.005693950  0.7000000 0.008134215 2.739554    56
```

Evaluación de las reglas

Además de la confianza y el soporte, existen otras métricas que permiten cuantificar la calidad de las reglas y la probabilidad de que reflejen relaciones reales. Algunas de las más empleadas son:

- *Lift*: compara la frecuencia observada de una regla con la frecuencia esperada simplemente por azar (si la regla no existe realmente). El valor *lift* de una regla *"si X, entonces Y"* se obtiene como el soporte de la unión de X e Y dividido por el producto del soporte de X por el soporte de Y.
- Cuanto más se aleje el valor de *lift* de 1, más evidencias de que la regla no se debe a un artefacto aleatorio, es decir, mayor la evidencia de que la regla representa un patrón real.
- *Coverage*: es el soporte de la parte izquierda de la regla (antecedente). Se interpreta como la frecuencia con la que el antecedente aparece en el conjunto de transacciones.
- *Fisher exact test*: devuelve el *p-value* asociado a la probabilidad de observar la regla solo por azar.

Con la función `interestMeasure()` se pueden calcular más de 20 métricas distintas para un conjunto de reglas creadas con la función `apriori()`.

```
metricas <- interestMeasure(reglas, measure = c("coverage", "fishersExactTest"),
                            transactions = transacciones)
metricas
```

```
##       coverage fishersExactTest
## 1  0.004575496     1.775138e-10
## 2  0.004067107     7.502990e-11
## 3  0.004982206     1.990017e-11
## 4  0.004778851     1.582670e-10
## 5  0.006812405     2.857678e-15
## 6  0.006507372     1.142131e-15
## 7  0.005490595     5.673757e-12
## 8  0.005287239     1.003422e-13
## 9  0.004067107     8.092894e-12
## 10 0.004575496     2.336708e-11
## 11 0.003863752     7.584014e-12
## 12 0.005083884     5.006888e-11
## 13 0.004575496     5.680562e-15
## 14 0.006202339     2.037822e-13
## 15 0.005693950     1.301061e-21
## 16 0.008134215     7.433712e-17
## 17 0.004473818     5.003096e-10
## 18 0.003558719     1.459542e-18
## 19 0.004982206     1.990017e-11
```

Estas nuevas métricas pueden añadirse al objeto que contiene las reglas.

```
quality(reglas) <- cbind(quality(reglas), metricas)
# inspect(sort(x = reglas, decreasing = TRUE, by = "confidence"))
df_reglas <- as(reglas, Class = "data.frame")
df_reglas %>% as.tibble() %>% arrange(desc(confidence)) %>% head()
```

```
## # A tibble: 6 x 8
##   rules      support confidence coverage  lift count coverage.1 fishersExactTest
##   <chr>        <dbl>      <dbl>    <dbl> <dbl> <int>      <dbl>            <dbl>
## 1 {citrus f~ 0.00315      0.886  0.00356  4.58    31    0.00356         1.46e-18
## 2 {butter,r~ 0.00305      0.789  0.00386  3.09    30    0.00386         7.58e-12
## 3 {citrus f~ 0.00447      0.786  0.00569  4.06    44    0.00569         1.30e-21
## 4 {brown br~ 0.00315      0.775  0.00407  3.03    31    0.00407         8.09e-12
## 5 {butter,o~ 0.00305      0.75   0.00407  2.94    30    0.00407         7.50e-11
## 6 {curd,tro~ 0.00397      0.75   0.00529  2.94    39    0.00529         1.00e-13
```

Filtrado de reglas

Cuando se crean reglas de asociación, pueden ser interesantes únicamente aquellas que contienen un determinado conjunto de *items* en el antecedente o en el consecuente. Con `arules` existen varias formas de seleccionar solo determinadas reglas.

Restringir las reglas que se crean

Es posible restringir los *items* que aparecen en el lado izquierdo y/o derecho de la reglas a la hora de crearlas, por ejemplo, supóngase que solo son de interés reglas que muestren productos que se vendan junto con other vegetables. Esto significa que el *item* other vegetables, debe aparecer en el lado derecho (*rhs*).

```
soporte <- 30 / dim(transacciones)[1]
reglas_vegetables <- apriori(data = transacciones,
                             parameter = list(support = soporte,
                                              confidence = 0.70,
                                              # Se especifica que se creen reglas
                                              target = "rules"),
                             appearance = list(rhs = "other vegetables"))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime    support minlen
##         0.7    0.1    1 none FALSE            TRUE       5 0.00305033      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: 30
##
## set item appearances ...[1 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [136 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 done [0.00s].
## writing ... [3 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
summary(reglas_vegetables)
```

```
## set of 3 rules
##
## rule length distribution (lhs + rhs):sizes
## 4 5
## 2 1
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.000   4.000   4.000   4.333   4.500   5.000
##
## summary of quality measures:
##     support           confidence        coverage             lift
##  Min.   :0.003152   Min.   :0.7333   Min.   :0.003559   Min.   :3.790
```

```
##  1st Qu.:0.003254   1st Qu.:0.7595   1st Qu.:0.004067   1st Qu.:3.925
##  Median :0.003355   Median :0.7857   Median :0.004575   Median :4.061
##  Mean   :0.003660   Mean   :0.8016   Mean   :0.004609   Mean   :4.143
##  3rd Qu.:0.003915   3rd Qu.:0.8357   3rd Qu.:0.005135   3rd Qu.:4.319
##  Max.   :0.004474   Max.   :0.8857   Max.   :0.005694   Max.   :4.578
##      count
##  Min.   :31.0
##  1st Qu.:32.0
##  Median :33.0
##  Mean   :36.0
##  3rd Qu.:38.5
##  Max.   :44.0
##
## mining info:
##           data ntransactions    support confidence
##  transacciones          9835 0.00305033        0.7
```

```
inspect(reglas_vegetables)
```

```
##     lhs                     rhs                    support confidence    coverage     lift count
## [1] {root vegetables,
##      tropical fruit,
##      whipped/sour cream} => {other vegetables} 0.003355363  0.7333333 0.004575496 3.789981    33
## [2] {citrus fruit,
##      root vegetables,
##      tropical fruit}     => {other vegetables} 0.004473818  0.7857143 0.005693950 4.060694    44
## [3] {citrus fruit,
##      root vegetables,
##      tropical fruit,
##      whole milk}         => {other vegetables} 0.003152008  0.8857143 0.003558719 4.577509    31
```

Filtrar reglas creadas

También es posible filtrar las reglas una vez que han sido creadas. Por ejemplo, se procede a filtrar aquellas reglas que contienen other *vegetables* y *citrus fruit* en el antecedente.

```
filtrado_reglas <- subset(x = reglas,
                          subset = lhs %ain% c("other vegetables","citrus fruit"))
inspect(filtrado_reglas)
```

```
##     lhs                   rhs              support confidence    coverage     lift count    coverage
## [1] {citrus fruit,
##      other vegetables,
##      root vegetables,
##      tropical fruit}   => {whole milk} 0.003152008  0.7045455 0.004473818 2.757344    31 0.004473818
```

Filtrar reglas maximales

Un *itemset* es *maximal* si no existe otro *itemset* que sea su superconjunto. Una regla de asociación se define como regla *maximal* si está generada con un *itemset maximal*. Con la función `is.maximal()` se pueden identificar las reglas maximales.

```
reglas_maximales <- reglas[is.maximal(reglas)]
reglas_maximales
```

```
## set of 17 rules
```

Reglas redundantes

Dos reglas son idénticas si tienen el mismo antecedente (parte izquierda) y consecuente (parte derecha). Supóngase ahora que una de estas reglas tiene en su antecedente los mismos *items* que forman el antecedente de la otra, junto con algunos *items* más. La regla más genérica se considera redundante, ya que no aporta información adicional. En concreto, se considera que una regla $X => Y$ es redundante si existe un subconjunto X' tal que existe una regla $X' => Y$ cuyo soporte es mayor.

$X => Y$ es redundante si existe un subconjunto X' tal que: *conf(X' -> Y) >= conf(X -> Y)*

```
reglas_redundantes <- reglas[is.redundant(x = reglas, measure = "confidence")]
reglas_redundantes
```

```
## set of 0 rules
```

Para este ejemplo no se detectan reglas redundantes.

Transacciones que verifican una determinada regla

Una vez identificada una determinada regla, puede ser interesante recuperar todas aquellas transacciones en las que se cumple. A continuación, se recuperan aquellas transacciones para las que se cumple la regla con mayor confianza de entre todas las encontradas.

```
# Se identifica la regla con mayor confianza
as(reglas, "data.frame") %>%
  arrange(desc(confidence)) %>%
  head(1) %>%
  pull(rules)
```

```
## [1] "{citrus fruit,root vegetables,tropical fruit,whole milk} => {other vegetables}"
```

Las transacciones que cumplen esta regla son todas aquellas que contienen los items: *citrus fruit*, *root vegetables*, *tropical fruit*, *whole milk* y *other vegetables*.

```
filtrado_transacciones <- subset(x = transacciones,
                                 subset = items %ain% c("citrus fruit",
                                                        "root vegetables",
                                                        "tropical fruit",
                                                        "whole milk",
                                                        "other vegetables"))
filtrado_transacciones
```

```
## transactions in sparse format with
##  31 transactions (rows) and
##  169 items (columns)
```

```
# Se muestran 3 de las 31 transacciones
inspect(filtrado_transacciones[1:3])
```

```
##     items                   transactionID
## [1] {berries,
##      bottled water,
##      butter,
##      citrus fruit,
##      hygiene articles,
##      napkins,
##      other vegetables,
##      root vegetables,
##      rubbing alcohol,
```

```
##      tropical fruit,
##      whole milk}                    596
## [2] {bottled water,
##      citrus fruit,
##      curd,
##      dessert,
##      frozen meals,
##      frozen vegetables,
##      fruit/vegetable juice,
##      grapes,
##      napkins,
##      other vegetables,
##      pip fruit,
##      root vegetables,
##      specialty chocolate,
##      tropical fruit,
##      UHT-milk,
##      whipped/sour cream,
##      whole milk}                    1122
## [3] {beef,
##      beverages,
##      butter,
##      candles,
##      chicken,
##      citrus fruit,
##      cream cheese,
##      curd,
##      domestic eggs,
##      flour,
##      frankfurter,
##      ham,
##      hard cheese,
##      hygiene articles,
##      liver loaf,
##      margarine,
##      mayonnaise,
##      other vegetables,
##      pasta,
##      roll products,
##      rolls/buns,
##      root vegetables,
##      sausage,
##      skin care,
##      soft cheese,
##      soups,
##      specialty fat,
##      sugar,
##      tropical fruit,
##      whipped/sour cream,
##      whole milk,
##      yogurt}                        1217
```

Trabajo propuesto

Para completar el tutorial, se propone realizar las mismas tareas, pero utilizando el algoritmo `fpgrowth()` del paquete `rCBA` en lugar del algoritmo `apriori()` del paquete `arules`. El objetivo es comparar los tiempos de ejecución de ambos algoritmos, y si se cumple que se obtienen los mismos resultados cuando se utilizan los mismos parámetros en ambos algoritmos.

Comentarios finales

Con esto concluye este tutorial sobre la aplicación de las reglas de asociación al problema del análisis de la cesta de la compra. Si necesita más información sobre la realización de las tareas, póngase en contacto a través de la dirección de correo electrónico: pglez@ujaen.es