R-PL2

Gabriel López, Sergio Sanz, Álvaro Zamorano

18 de octubre de 2019

1. Ejercicio realizado en clase.

Para poder usar el algoritmo **Apriori** y sus reglas de asociación vamos a utilizar el paquete `arules`. Este paquete hay que descargarlo desde la página de CRAN y para instalarlo hay que ejecutar el siguiente código:

```
> install.packages("./arules_1.6-4.zip",repos=NULL)
```

De esta forma, el paquete únicamente está instalado. Para poder usarlo es necesario cargarlo:

```
> library(arules)
```

Los datos a usar en este primer ejercicio se componen de 6 cestas de la compra, en concreto estas son: {Pan, Agua, Leche, Naranjas},{Pan, Agua, Café, Leche}, {Pan, Agua, Leche}, {Pan, Café, Leche}, {Pan, Agua}, {Leche}.

Para introducir estos datos en el algoritmo a usar es necesario crear una matriz con el siguiente aspecto.

Suceso	Pan	Agua	Café	Leche	Naranjas
s1	1	1	0	1	1
s2	1	1	1	1	0
s3	1	1	0	1	0
s4	1	0	1	1	0
s5	1	1	0	0	0
s6	0	0	0	1	0

Esta matriz se introduce en R mediante:

```
> muestra<-Matrix(c(1,1,0,1,1,1,1,1,1,0,1,1,0,1,0,1,0,1,1,0,1,1,0,0,0,0,0,0,1,0),
+ 6,5,byrow=T,dimnames=list(c("suceso1","suceso2","suceso3","suceso4","suceso5",
+ "suceso6"),c("Pan","Agua","Cafe","Leche","Naranjas")),sparse=T)
```

Se necesita convertir la matriz a una matriz dispersa a través de la función **as** la cuál convierte un objeto a una determinada clase, en este caso la clase es

nsparseMatrix. Esta clase lo que hace es cambiar los valores mayores de 0 por un valor binario, con el fin de gastar la menor cantidad de memoria posible ya que solo se almacenan aquellas posiciones no vacias, es decir, las que cuyo valor es distinto de 0.

```
> muestrangCMatrix<-as(muestra,"nsparseMatrix")
```

El siguiente paso a realizar es calcular la **traspuesta** de la última matriz generada.

```
> transpuestangCMatrix<-t(muestrangCMatrix)
```

Antes de aplicar el algoritmo, calculamos y mostramos todas las **transacciones**, es decir, todas las asociaciones que hay en nuestros datos.

```
> transacciones<-as(transpuestangCMatrix,"transactions")
> summary(transacciones)

transactions as itemMatrix in sparse format with
 6 rows (elements/itemsets/transactions) and
 5 columns (items) and a density of 0.5666667

most frequent items:
     Pan    Leche     Agua     Cafe Naranjas  (Other)
       5        5        4        2        1        0

element (itemset/transaction) length distribution:
sizes
1 2 3 4
1 1 2 2

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   2.250   3.000   2.833   3.750   4.000

includes extended item information - examples:
  labels
1    Pan
2   Agua
3   Cafe

includes extended transaction information - examples:
  itemsetID
1   suceso1
2   suceso2
3   suceso3
```

Por último, aplicamos el algoritmo **Apriori** para las asociaciones cuyo soporte sea igual o superior al 50 % y cuya confianza sea igual o mayor que el 80 %.

```
> asociaciones<-apriori(transacciones,parameter=list(support=0.5,confidence=0.8))

> inspect(asociaciones)

    lhs             rhs     support   confidence lift count
[1] {}           => {Leche} 0.8333333 0.8333333  1.00 5
[2] {}           => {Pan}   0.8333333 0.8333333  1.00 5
[3] {Agua}       => {Pan}   0.6666667 1.0000000  1.20 4
[4] {Pan}        => {Agua}  0.6666667 0.8000000  1.20 4
[5] {Leche}      => {Pan}   0.6666667 0.8000000  0.96 4
[6] {Pan}        => {Leche} 0.6666667 0.8000000  0.96 4
[7] {Agua,Leche} => {Pan}   0.5000000 1.0000000  1.20 3
```

2. Parte 2.

2.1. Datos de ventas de coches.

Para el leer el fichero `.txt` hemos creado una función la cuál nos devuelve una lista con las filas de la matriz.

```
> source("leerMatriz.R")
> leerM

function(ruta) {

    data<-read.table(ruta,header=TRUE)
    mat<-as.matrix(data)
    sz<-dim(mat)
    l<-c()

    for (i in 1:sz[1]) {
        for (j in 1:sz[2]){
            l<-c(l,mat[i,j])
        }
    }

    return(l)
}
```

Procedemos a la lectura de dicho fichero.

```
> m<-leerM("2_1.txt")
```

La matriz leida tiene el siguiente aspecto.

Suceso	Xenon	Alarma	Techo	Navegador	Bluetooth	ControlV
s1	1	0	0	1	1	1
s2	1	0	1	0	1	1
s3	1	0	0	1	0	1
s4	1	0	1	1	1	0
s5	1	0	0	0	1	1
s6	0	0	0	1	0	0
s7	1	0	0	0	1	1
s8	0	1	1	0	0	0

Esta matriz se introduce en R mediante:

```
> mCoches<-Matrix(m,8,6,byrow=T,dimnames=list(c("suceso1","suceso2","suceso3",
+ "suceso4","suceso5","suceso6", "suceso7", "suceso8"),
+ c("Xenon", "Alarma", "Techo", "Navegador", "Bluetooth", "ControlV")),sparse=T)
```

Se necesita convertir la matriz a una matriz dispersa a través de la función **as**.

```
> mCochesngC<-as(mCoches,"nsparseMatrix")
```

El siguiente paso a realizar es calcular la **traspuesta** de la última matriz generada.

```
> transpuestangC<-t(mCochesngC)
```

Antes de aplicar el algoritmo, calculamos y mostramos todas las **transacciones**, es decir, todas las asociaciones que hay en nuestros datos.

```
> transac<-as(transpuestangC,"transactions")
> summary(transac)

transactions as itemMatrix in sparse format with
 8 rows (elements/itemsets/transactions) and
 6 columns (items) and a density of 0.5

most frequent items:
    Xenon Bluetooth  ControlV Navegador     Techo   (Other)
        6         5         5         4         3         1

element (itemset/transaction) length distribution:
sizes
1 2 3 4
1 1 3 3

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    2.75    3.00    3.00    4.00    4.00
```

```
includes extended item information - examples:
  labels
1  Xenon
2 Alarma
3  Techo

includes extended transaction information - examples:
  itemsetID
1   suceso1
2   suceso2
3   suceso3
```

Por último, aplicamos el algoritmo **Apriori** para las asociaciones cuyo soporte sea igual o superior al 50 % y cuya confianza sea igual o mayor que el 80 %.

```
> asoc<-apriori(transac,parameter=list(support=0.4,confidence=0.9))

> inspect(asoc)

    lhs                     rhs     support confidence lift     count
[1] {ControlV}           => {Xenon} 0.625   1          1.333333 5
[2] {Bluetooth}          => {Xenon} 0.625   1          1.333333 5
[3] {Bluetooth,ControlV} => {Xenon} 0.500   1          1.333333 4
```

2.2. Desarrollo del grupo.

En primer lugar procedemos a leer los datos que hay en el fichero `.csv` almacenándolos directamente en un objeto de tipo *transactions* ya que es la estructura de almacenamiento empleada por *arules.* Para poder leer estos datos es necesario tener cargada la librería mediante `library(arules)`, como se ha indicado anteriormente.

Las transacciones se leen gracias al uso de la función *read.transactions*.

```
> shopT<-read.transactions("shop.csv", format = "basket",
+                   header = FALSE, sep = ",",
+                   cols = NULL, rm.duplicates = FALSE,
+                   skip = 0)
```

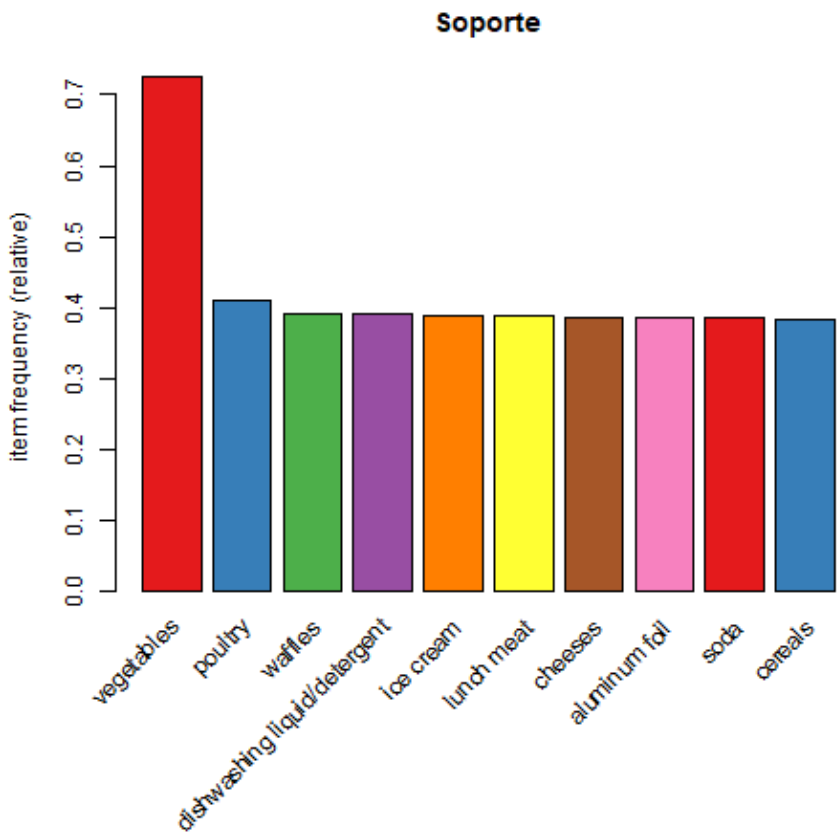
Un ejemplo de estas transacciones es:

```
> inspect(shopT[1])

    items
[1] {all- purpose,
     aluminum foil,
     beef,
     butter,
```

```
dinner rolls,
flour,
ice cream,
laundry detergent,
lunch meat,
mixes,
pork,
sandwich bags,
shampoo,
soap,
soda,
vegetables,
yogurt}
```

Mostramos en un gráfico el **soporte** de los 10 productos más comprados, es decir, en cuantos sucesos aparece cada uno de los productos respecto al total de sucesos.

```
> source("sopImg.R")
> d<-sopImg(shopT,"sopI.png")
```

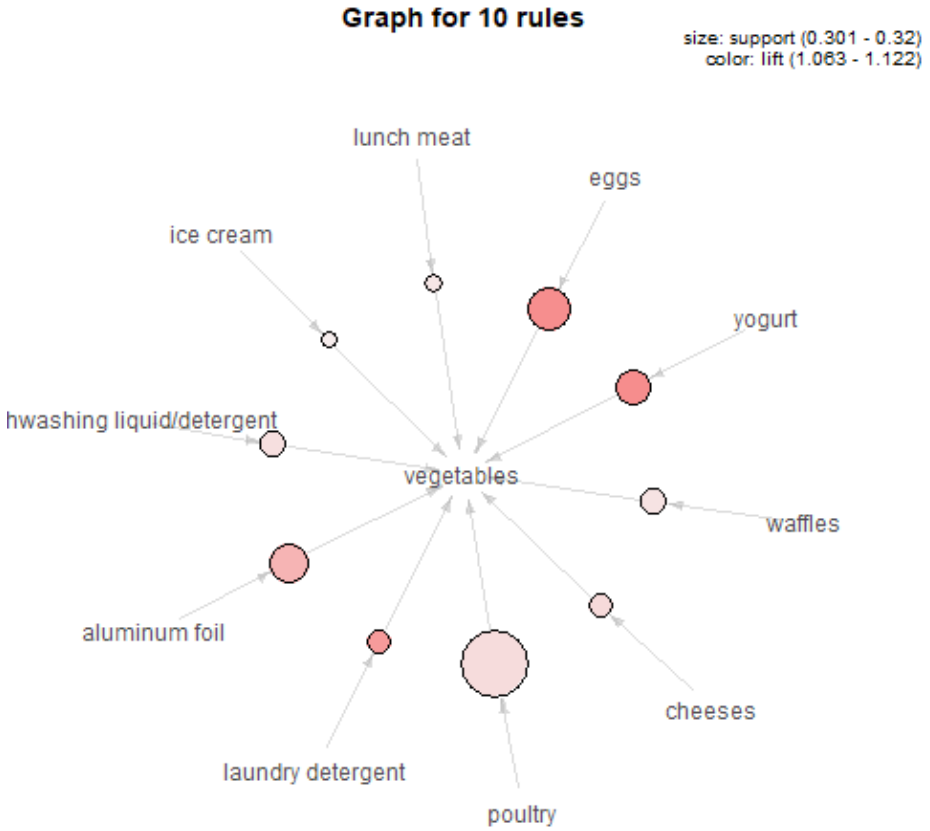


Para usar la función anterior es necesario installar e importar mediante `library` un paquete de R para el uso de la paleta de colores que posteriormente empleará el gráfico. Dicha función tiene el siguiente aspecto:

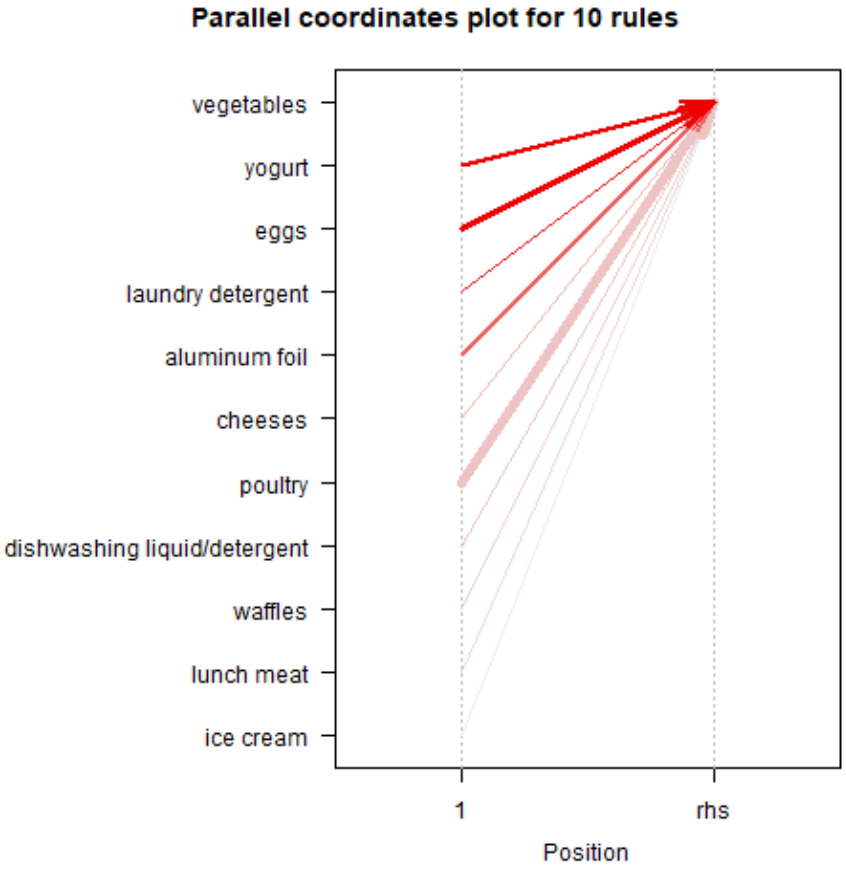
```
> sopImg

function(data, ruta) {

    install.packages("RColorBrewer")
    library(RColorBrewer)

    png(paste("./tmp/",ruta,sep=""))

    itemFrequencyPlot(data, topN=10, col=brewer.pal(8,'Set1'),
        type="relative",main="Soporte")

    dev.off()
}
```

Uso de **eclat:**

```
> is<-eclat(shopT,parameter=list(support=0.3))

> rules<-ruleInduction(is,shopT,confidence=0.7)
> inspect(rules)

     lhs                               rhs          support   confidence lift     itemset
[1]  {laundry detergent}            => {vegetables} 0.3042028 0.8099467  1.114885  1
[2]  {yogurt}                       => {vegetables} 0.3082055 0.8148148  1.121586  2
[3]  {eggs}                         => {vegetables} 0.3108739 0.8146853  1.121408  3
[4]  {ice cream}                    => {vegetables} 0.3008672 0.7722603  1.063010  4
[5]  {lunch meat}                   => {vegetables} 0.3015344 0.7766323  1.069028  5
[6]  {waffles}                      => {vegetables} 0.3048699 0.7785349  1.071647  6
[7]  {cheeses}                      => {vegetables} 0.3035357 0.7844828  1.079834  7
[8]  {aluminum foil}                => {vegetables} 0.3095397 0.8013817  1.103096  8
[9]  {dishwashing liquid/detergent} => {vegetables} 0.3048699 0.7811966  1.075311  9
[10] {poultry}                      => {vegetables} 0.3202135 0.7830343  1.077841 10

> source("graph.R")
> graph(rules,"graphR.png")
```




```
> source("graph2.R")
> graph2(rules,"graphR2.png")
```


Parallel coordinates plot for 10 rules
vegetables
yogurt
eggs
laundry detergent
aluminum foil
cheeses
poultry
dishwashing liquid/detergent
waffles
lunch meat
ice cream
1
rhs
Position