Práctica 2

Fundamentos de la Ciencia de Datos

Samuel Aós Paumard,
Enrique Coronado Barco,
Carmen Martínez Estévez,
Alberto Martínez Ortega

November 17, 2020

1 Ejericio 1

La primera parte consiste en la realización de un ejercicio en clase con ayuda del profesor en el que se va a realizar un análisis de asociación de Datos con R aplicando todos los conceptos vistos en el tema. Se resolverá, utilizando el algoritmo Apriori, el mismo problema que el visto en la descripción teórica del tema. Es decir, para la misma muestra1 que se utilizó para hacer de forma manual el primer ejercicio de asociación, se deberán obtener las asociaciones cuyo soporte sea igual o superior al 50% y cuya confianza sea igual o superior al 80%.

Comenzamos con la carga e instalación de las librerías necesarias. Para el primer ejercicio bastará con la instalación de arules, librería que nos servirá para la realización de la función a priori y que además contiene el conjunto de librerías para el tratamiento de matrices necesario para su realización como Matrix.

```
> install.packages("arules")
> library(arules)
```

Una vez cargadas las librerías, introducimos los datos del problema en forma de matriz, para ello los introducimos en la función Matrix en forma vectorial, seguido de la indicación de filas y columnas y el método de construcción, en este caso filas, continuamos con las cabeceras de filas y columnas para separarlas de los datos y por último denotamos que la matriz se trata de una matriz dispersa con el parámetro *isparse* con valor verdadero.

```
> 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=TRUE,
+         dimnames=list(c("Suceso 1","Suceso 2","Suceso 3",
```

```
+                        "Suceso 4","Suceso 5","Suceso 6"),
+                    c("Pan","Agua","Café","Leche","Narajas")),
+         sparse=TRUE)
> muestra

6 x 5 sparse Matrix of class "dgCMatrix"
         Pan Agua Café Leche Narajas
Suceso 1   1    1    .     1       1
Suceso 2   1    1    1     1       .
Suceso 3   1    1    .     1       .
Suceso 4   1    .    1     1       .
Suceso 5   1    1    .     .       .
Suceso 6   .    .    .     1       .
```

El siguiente paso es convertir las matriz *isparse* en una matriz *ngC*, ya que nos lo exige el paqute. La conversión la realizamos con la función *as*, a la que la pasamos la matriz generada (muestra) y el tipo al que la queremos converitr (nsparseMatrix). La nueva matriz generada se mostrará con puntos donde había un 0 y líneas verticales donde había un 1. A continuación hacemos la traspuesta de la matriz. Se muestran las dos matrices a continuación.

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

6 x 5 sparse Matrix of class "ngCMatrix"
         Pan Agua Café Leche Narajas
Suceso 1   |    |    .     |       |
Suceso 2   |    |    |     |       .
Suceso 3   |    |    .     |       .
Suceso 4   |    .    |     |       .
Suceso 5   |    |    .     .       .
Suceso 6   .    .    .     |       .

> transpmuestrangCMatrix<-t(muestrangCMatrix)
> transpmuestrangCMatrix

5 x 6 sparse Matrix of class "ngCMatrix"
       Suceso 1 Suceso 2 Suceso 3 Suceso 4 Suceso 5 Suceso 6
Pan            |        |        |        |        .
Agua           |        |        |        .        |        .
Café           .        |        .        |        .        .
Leche          |        |        |        |        .        |
Narajas        |        .        .        .        .        .
```

Ahora vamos a determinar las posibles transacciones que podemos realizar, nuevamente con la función *as*. En este caso la matriz que queremos convertir es la última que hemos generado (transpmuestrangCMatrix) y el tipo es transactions. Esta matriz la hemos denominado transacciones. Además con la función *summary* obtenemos más detalles como son los ítems más frecuentes, media, mediana, máximo, mínimo...

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

```
transactions in sparse format with
 6 transactions (rows) and
 5 items (columns)
```

```
> 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    Café Narajas (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   Café

includes extended transaction information - examples:
  itemsetID
1  Suceso 1
2  Suceso 2
3  Suceso 3
```

Para obtener las asociaciones vamos a utilizar el algoritmo *apriori* pasándole la matriz transacciones y los parámetros confianza y soporte. La confianza nos va a decir si se van a producir o no las asociaicones y el soporte lo frecuente que van a ser esas asociaciones. Con la función *inspect*, a la que le pasamos el resultado que devuelve el algoritmo *apriori*, obtendremos las asociaciones.

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

Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen
        0.8    0.1    1 none FALSE            TRUE       5     0.5      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: 3

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[5 item(s), 6 transaction(s)] done [0.00s].
sorting and recoding items ... [3 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 done [0.00s].
writing ... [7 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
```

```
> inspect(asociaciones)
```

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

2 Ejericio 2

La segunda parte consiste en el desarrollo por parte de cada grupo del enunciado y la solución de un ejercicio en el que se realice un análisis con R de asociación introduciendo modificaciones sobre el ejercicio hecho en clase.

Primero tomamos de un fichero txt los datos a representar en la matriz. También merece la pena remarcar la carga de la librearía *arules* en cuyo interior está la función a priori que ejecuta todos los cáculos necesarios para presentarnos las asociaciones en función del porcentaje de soporte y confainza elegidos como mínimos.

```
> library("arules")
> datos<-read.transactions("datos.txt",sep=" ")
```

Podemos observar que los datos aquí representados se corresponden con los reflejados en el txt, habiendo sido formateados para su tratamiento con R. Esto lo hacemos con la función *print* para imprimir y con la función *as* para dar formato. Concretamente usamos una matriz de tipo 'ngCMatrix' que se muestra de la siguiente manera:

```
> print(as(datos,"ngCMatrix"))
```

```
4 x 9 sparse Matrix of class "ngCMatrix"

Alberto | | . | | . . | |
Carmen  | | | | | | | .
Enrique | . . . . . | . |
Samu    . | . . | | | | .
```

A continuación hacemos uso de la función *apriori* que nos proporciona el paquete *arules* anteriormente instalado. Con esto conseguimos devolver el resultado de dicho algoritmo en forma de asociaciones a la variable con ese mismo nombre para posteriormente visualizarlas con la función *inspect()*

```
> asociacion<-apriori(datos,parameter=list(support=0.5,confidence=0.8))

Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen
        0.8    0.1    1 none FALSE            TRUE       5     0.5      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: 4

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[4 item(s), 9 transaction(s)] done [0.00s].
sorting and recoding items ... [3 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 done [0.00s].
writing ... [3 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].

> inspect(asociacion)

    lhs          rhs      support   confidence coverage  lift   count
[1] {}        => {Carmen} 0.8888889 0.8888889  1.0000000 1.0000 8
[2] {Samu}    => {Carmen} 0.5555556 1.0000000  0.5555556 1.1250 5
[3] {Alberto} => {Carmen} 0.5555556 0.8333333  0.6666667 0.9375 5
```

Para la parte 2 de esta mitad hemos obtenido un conjunto de datos a través de un generador de valores aleatorios, pudiendo optar entre el 0 y el 1, desarrollado en Python por nosotros mismos. Estos datos son pasados a un fichero '.csv', desde donde son leídos por el programa en R El código del generador se puede consultar en la misma carpeta de este documento bajo el nombre *generarDatos.py*. Hemos optado por hacer el experimento con nombres de frutas tal y como se puede observar al crear la lista de nombres de los sucesos elementales. Incluimos también las librerías necesarias para llevar a cabo este proyecto. Son las siguientes:

```
> library("Matrix")
> library("arules")
> frutas<- c("Frambuesa","Fresa","Naranja","Sandia","Melon");
```

Después importamos los datos desde el fichero csv mencionado a una variable para mantenerlos en el espacio de trabajo y poder operar con ellos.

```
> datos<-scan("Frutas.csv",sep=";")
```

Después convertimos los datos a una matriz de tamaño (longitud(frutas))x(50). Organizamos la matriz en filas, en lugar de en columnas, para una muestra de datos escasa y le otorgamos a cada elemento de esa fila un Identificador que se corresponde con los nombres aportados en la linea anterior.

```
> matriz_datos<-Matrix(datos,ncol=length(frutas),nrow=50,byrow=T,sparse=T,
+ dimnames=list(c(1:50),frutas));
```

Por último en este paso de formateo de datos, mostramos los resultados para asegurarnos de su correspondencia con el excel.

```
> print(matriz_datos)
```

```
50 x 5 sparse Matrix of class "dgCMatrix"
   Frambuesa Fresa Naranja Sandia Melon
1          .     .       1      1     1
2          .     1       1      1     1
3          .     1       1      1     .
4          .     1       .      1     .
5          .     .       .      1     .
6          .     1       1      1     .
7          .     1       1      .     .
8          1     .       .      .     .
9          .     1       .      .     .
10         .     1       1      .     .
11         1     1       1      .     .
12         1     .       1      1     .
13         .     1       .      1     .
14         1     .       .      .     1
15         1     .       .      1     .
16         1     1       .      .     .
17         1     1       .      1     1
18         1     .       1      1     .
19         1     1       .      1     1
20         1     .       1      1     1
21         1     1       1      .     .
22         .     1       .      1     .
23         .     .       .      1     1
24         1     .       1      1     1
25         .     .       1      .     1
26         .     .       1      1     1
27         1     1       1      1     .
28         .     1       .      .     .
29         .     1       .      1     1
30         .     1       .      1     1
31         .     .       1      .     1
32         .     1       1      .     .
33         1     .       1      1     .
34         .     .       1      .     1
```

```
35         1     .       1      .     1
36         .     .       .      1     1
37         .     .       1      1     1
38         1     .       .      .     .
39         .     .       .      .     1
40         1     .       .      1     .
41         1     1       1      .     .
42         .     .       .      1     .
43         .     .       1      .     .
44         .     .       1      .     1
45         1     1       1      1     1
46         1     .       1      1     .
47         1     1       .      1     1
48         .     1       1      .     1
49         1     .       1      1     1
50         .     .       .      1     .
```

Para continuar con el ejemplo, vamos a realizar un análisis de la misma muestra de datos por medio de dos formas distintas. Lo primero será empezar con el análisis de nuestros datos con el algoritmo apriori, que ya nos es conocido de la parte anterior. Inmediatamente después realizaremos el mismo cálculo con otra función llamada eclat. Dicha función es otra forma de llevar a cabo el algoritmo apriori sobre nuestra muestra y está específicamente recomendado para muestras más pequeñas al estar optimizado para ellas.

Parte 1

```
> asociacion<-apriori(as(t(as(matriz_datos,"nsparseMatrix")),"transactions"),
+ parameter=list(support=0.2,confidence=0.6))

Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen
        0.6    0.1    1 none FALSE            TRUE       5     0.2      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: 10

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[5 item(s), 50 transaction(s)] done [0.00s].
sorting and recoding items ... [5 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 done [0.00s].
writing ... [4 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
```

```
> inspect(asociacion)

    lhs            rhs       support confidence coverage lift     count
[1] {}          => {Sandia}  0.60    0.6000000  1.00     1.000000 30
[2] {Frambuesa} => {Sandia}  0.28    0.6363636  0.44     1.060606 14
[3] {Melon}     => {Naranja} 0.28    0.6086957  0.46     1.086957 14
[4] {Melon}     => {Sandia}  0.30    0.6521739  0.46     1.086957 15
```

Parte 2

```
> asociacion2<-eclat(as(t(as(matriz_datos,"nsparseMatrix")),"transactions"),
+ parameter=list(support=0.2,maxlen=15))

Eclat

parameter specification:
 tidLists support minlen maxlen            target  ext
    FALSE     0.2      1     15 frequent itemsets TRUE

algorithmic control:
 sparse sort verbose
      7   -2    TRUE

Absolute minimum support count: 10

create itemset ...
set transactions ...[5 item(s), 50 transaction(s)] done [0.00s].
sorting and recoding items ... [5 item(s)] done [0.00s].
creating bit matrix ... [5 row(s), 50 column(s)] done [0.00s].
writing  ... [12 set(s)] done [0.00s].
Creating S4 object  ... done [0.00s].

> inspect(asociacion2)

     items               support transIdenticalToItemsets count
[1]  {Fresa,Sandia}      0.26    13                       13
[2]  {Fresa,Naranja}     0.24    12                       12
[3]  {Frambuesa,Sandia}  0.28    14                       14
[4]  {Frambuesa,Naranja} 0.26    13                       13
[5]  {Sandia,Melon}      0.30    15                       15
[6]  {Naranja,Melon}     0.28    14                       14
[7]  {Naranja,Sandia}    0.30    15                       15
[8]  {Sandia}            0.60    30                       30
[9]  {Naranja}           0.56    28                       28
[10] {Melon}             0.46    23                       23
[11] {Frambuesa}         0.44    22                       22
[12] {Fresa}             0.46    23                       23
```

Como podemos observar los parámetros utilizados son:

- En el caso del algoritmo apriori ==> Soporte=0.2 | Confianza=0.6
- En el caso del algoritmo eclat ====> Soporte=0.2 | maxlen=15 (no menos de 15 apariciones del suceso elemental de entre el total de los datos)

Como podemos observar, al no estar la muestra lo suficientemente bien diseñada para que este tipo de apariciones ocurran, sino que se ha diseñado aleatoriamente, obtenemos unos valores muy bajos de soporte y confianza en ambos algoritmos. Esto tiene como consecuencia el haber tenido que usar unos valores tan bajos en ambos casos para poder tener datos que reflejar en esta visualización.