UNIDAD 6. Analisis de Asociación; A Priori. Motores de Búsqueda: Pagerank.

Algoritmo Apriori

El algoritmo Apriori (Agrawal y Srikant, 1994) está diseñado para operar sobre bases de datos que contienen "transacciones" o conjuntos de ítems, como por ejemplo la lista de productos comprados por cada cliente en un supermercado, una de las primeras aplicaciones –análisis de la cesta de la compra- de este algoritmo. Dado un valor umbral k, el algoritmo Apriori identifica todos los conjuntos de ítems "frecuentes", aquellos que son subconjuntos de al menos k transacciones en la base de datos, y establece para cada subconjunto una regla que relaciona las variables entre sí, que consiste simplemente en una partición binaria del subconjunto frecuente. Por ejemplo, el algoritmo puede encontrar (subconjunto frecuente) que el 25% de los clientes compran pan, leche, y vino, y que (regla) si un cliente compra pan y leche, la probabilidad de que también compre vino es el 90%.

Todas las variables tienen que ser cualitativas; las variables numéricas tienen que ser recodificadas como factores para que puedan ser utilizadas con este método.

Apriori utiliza un enfoque "bottom up", con el cual los subconjuntos frecuentes son ampliados con un ítem cada vez (generación de candidatos) y posteriormente los grupos candidatos son validados con los datos. Según el lema de clausura hacia abajo (downward clausure) cualquier subconjunto de otro frecuente tambien es frecuente. El algoritmo termina cuando ya no es posible ampliar los subconjuntos previos o candidatos.

Apriori utiliza un árbol de Merkle (Merkle Hash Tree), para almacenar eficientemente los candidatos. Esta estructura consiste en un árbol en el que cada nodo –excepto las hojas, o nodos terminales- está etiquetado con la información resumida (código hash) de los nodos hijos, lo que permite recorrer el árbol de forma rápida y eficaz.

Ejemplo: supongamos una base de datos, formada por 7 transacciones, que para simplificar representaremos por conjuntos o listas de letras:

Transacciones: {A,B,C,D}, {A,B,D}, {A,B}, {B,C,D}, {B,C}, {C,D}, {B,D}

Definimos un valor umbral de confianza o "soporte" k=3: consideraremos por lo tanto que un subconjunto es "frecuente" si aparece al menos 3 veces en la base de datos.

Empezamos (j=1) calculando la frecuencia de cada ítem por separado:

{A} 3 {B} 6 {C} 4 {D} 5

Todos los conjuntos de tamaño j=1 tienen confianza de al menos 3, por lo que todos son frecuentes. El próximo paso es generar la lista de todos los conjuntos de tamaño j=2 con sus frecuencias:

{A,B} 3 {A,C} 1 {A,D} 2 {B,C} 3 {B,D} 4 {C,D} 3

Los pares {A,B}, {B,C}, {B,D} y {C,D} tienen una confianza igual o mayor que 3, por lo que son frecuentes; el resto no lo son. Como {A,C} y {A,D} no son frecuentes, cualquier conjunto que contenga estos pares es también infrecuente (downward clausure). De esta manera se pueden podar los conjuntos a inspeccionar al aumentar sucesivamente el valor de j.

Revisando los conjuntos de ítems de tamaño j=3, observamos que solo hay uno, {B,C,D}, que no contiene ninguno de los dos pares infrecuentes citados, y ese conjunto tiene frecuencia 2; por lo tanto no hay ninguno frecuente de ese tamaño y hemos finalizado la búsqueda de todos los conjuntos de ítems frecuentes en la base de datos. En general solo tienen interés los maximales, por lo que podemos descartar los candidatos que son subconjunto de otros candidatos (en este ejemplo todos los subconjuntos de tamaño 1), y conservamos solamente los 4 frecuentes de tamaño 2 como resultado de la aplicación del algoritmo.

Para la aplicación de Apriori con R utilizaremos la función apriori del paquete arules.

apriori(data, parameter)

"data" es la base de datos de transacciones. Para que pueda ser utilizada por la función apriori debe tener un formato especial, debe ser un objeto de la clase "transactions". Para ello tenemos que transformar el conjunto de datos (por ejemplo Dataset) que tiene el formato habitual de R, es decir un data frame, en un conjunto de transacciones que pueda ser utilizado como data por la función a priori; la transformación se hace del modo siguiente:

transacciones <- as(Dataset, "transactions")

El segundo argumento de la función apriori, parameter, es opcional y puede ser omitido (tiene valores por defecto que pueden ser razonablemente válidos); permite establecer las opciones de configuración del algoritmo. Se trata de una lista con los siguientes elementos (entre paréntesis se muestra el valor por defecto):

support (0.1):	frecuencia relativa –soporte- que define un subconjunto "frecuente".
confidence (0.8):	confianza de la regla: proporción de veces en la que es cierta.
maxlen (10):	número máximo de ítems en los conjuntos frecuentes.
maxtime (5):	tiempo máximo (en segundos) de búsqueda de subconjuntos.

Si reducimos el soporte, o el nivel de confianza de la regla, o aumentamos el número máximo de ítems, el número de subconjuntos y reglas obtenido será en general mayor.

Utilizaremos para el ejemplo con R el conjunto de datos "compra", que podemos leer desde el archivo "compra.RData", en formato de R, que consiste en una muestra de

1000 consumidores de los cuales algunos han comprado un determinado producto (comprador = si), con variables adicionales como sexo, edad, empleo, formación, o estado civil. Para aplicar esta técnica no necesitamos una variable dependiente, ni hacer supuestos de dependencia entre las variables existentes como en otros métodos, sino que dejaremos que el algoritmo explore los datos para establecer aquellas relaciones o dependencias que quizás podríamos después utilizar para formular hipótesis o construir modelos.

```
setwd("C:/CURSO DM")
load("compra.RData")
head(compra)          # escribe los seis primeros casos

library(arules)       # carga el paquete arules, que debe estar instalado previamente
transacciones <- as(compra, "transactions")       # convierte los datos en transacciones
```

A continuación aplicaremos el método con la función apriori. Para limitar el número de reglas en los resultados fijaremos el grado de confianza 0,9 en lugar del valor por defecto 0,8 y el soporte 0,1 (este último es el valor por defecto, por lo que podríamos omitirlo dentro de la función):

```
reglas <- apriori(transacciones, parameter = list(support = 0.1, confidence = 0.80))
reglas.ordenadas <- sort(reglas, by = "confidence")
as(reglas.ordenadas, "data.frame")    # escribe las reglas, en formato conjunto de datos
```

```
                                      rules                              support confidence  lift   count
3                      {sexo=mujer,empleo=inactivo} => {edad=> 55}   0.103  0.9716981 2.752686  103
2                    {comprador=si,empleo=inactivo} => {edad=> 55}   0.101  0.9619048 2.724943  101
1                                  {empleo=inactivo} => {edad=> 55}   0.194  0.9463415 2.680854  194
9  {sexo=hombre,edad=35 -55,estadocivil=casado} => {empleo=ocupado}   0.134  0.9436620 1.483745  134
4               {empleo=inactivo,estadocivil=casado} => {edad=> 55}   0.100  0.9345794 2.647534  100
12   {comprador=no,edad=35 -55,formacion=media} => {empleo=ocupado}   0.140  0.9333333 1.467505  140
10    {sexo=hombre,edad=35 -55,formacion=media} => {empleo=ocupado}   0.134  0.9305556 1.463138  134
7        {comprador=si,sexo=hombre,edad=35 -55} => {empleo=ocupado}   0.107  0.9224138 1.450336  107
5                     {sexo=hombre,edad=35 -55} => {empleo=ocupado}   0.209  0.9126638 1.435006  209
11{comprador=no,edad=35 -55,estadocivil=casado} => {empleo=ocupado}   0.128  0.9078014 1.427361  128
6                  {edad=35 -55,formacion=media} => {empleo=ocupado}   0.252  0.9064748 1.425275  252
8        {comprador=no,sexo=hombre,edad=35 -55} => {empleo=ocupado}   0.102  0.9026549 1.419269  102
```

El algoritmo ha encontrado 12 subconjuntos frecuentes, aquellos con frecuencia o support mayor o igual que 0,1 y con confianza de la regla al menos 0,90. Los subconjuntos y reglas están ordenados por importancia decreciente.

Al aplicar la función a priori obtenemos el objeto de resultado al que hemos llamado "reglas". Con la orden siguiente las ordenamos por importancia: para esto podemos utilizar diferentes criterios, como "confidence" grado de confianza de la regla, o "support" frecuencia del subconjunto. Por último escribimos los resultados.

El primer subconjunto frecuente está fomado por las tres condiciones `{sexo=mujer, empleo=inactivo, edad=> 55},` y la regla establece la edad como resultado de las otras dos condiciones, lo que ocurre con una probabilidad "confidence" de 0,97. Ese subconjunto tiene una frecuencia absoluta "count" de 103 casos, y relativa "support" de 0,103. Lift (implicación, interés) es un estadístico de calidad de la regla que divide la frecuencia del subconjunto (mujeres inactivas con edad >55) por el producto de las frecuencias de los dos lados de la regla (mujeres inactivas; edad>55). Si ambas cosas

son independientes la frecuencia conjunta es igual al producto de las dos frecuencias separadas, por lo que el cociente será igual a 1; un valor de lift próximo a 1 indicará por lo tanto que la regla no sirve, y por el contrario será más eficaz (los dos lados de la regla estarán más correlacionados) cuanto más alto sea el valor de lift. Un valor de lift igual a 3 indica que la parte derecha de la regla ocurre con una frecuencia tres veces mayor de lo que cabría esperar si no estuviera relacionada con la parte izquierda.

Las reglas indican relaciones dentro de nuestro conjunto de datos: Las mujeres inactivas son mayores, los compradores inactivos o los inactivos en general son mayores, los hombres casados con edad entre 35-55 tienen trabajo, etc.

PageRank

PageRank (Brin y Page, 1998) es una marca registrada por Google que ampara a una familia de algoritmos utilizados para medir la relevancia de los documentos indexados por un motor de búsqueda; actualmente es utilizado por el popular buscador de Google para determinar la importancia de una página o documento en la red, y establecer el orden en el que son presentados los resultados de la búsqueda. Fue desarrollado por los fundadores de Google, Larry Page (de quien este algoritmo recibe su nombre) y Sergey Brin, en la Universidad de Stanford, en la que ambos eran estudiantes.

Google interpreta un enlace de una página web A a una página B como un voto (de A para B). Los votos emitidos por las páginas con PageRank elevado valen más, y ayudan a hacer a otras páginas "importantes". El índice PageRank de una página refleja su importancia en Internet.

PageRank ha tomado su modelo del Science Citation Index (SCI) elaborado por Eugene Garfield para el Instituto para la Información Científica (ISI) de Estados Unidos, que pretende resolver la asignación objetiva de mérito científico mediante un factor de impacto basado en el número de citas o referencias bibliográficas en otros trabajos científicos, elemento determinante en muchos países para seleccionar a los investigadores que reciben becas y recursos de investigación.

La fórmula utilizada por PageRank es:

$$\mathrm{PR}(A) = (1-d) + d\sum_{i=1}^{n}\frac{\mathrm{PR}(i)}{C(i)}$$

PR(A) es el PageRank de la página A.

d es un factor de amortiguación, cuyo valor está entre 0 y 1.

PR(i) son los valores de PageRank de cada una de las n páginas que enlazan a A.

C(i) es el número total de enlaces salientes de la página i (sean o no hacia A)

El valor de d suele ser 0,85 y representa la probabilidad de que un usuario continúe pulsando enlaces al navegar por Internet en vez de escribir una url directamente en la barra de direcciones o pulsar uno de sus propios marcadores. Por lo tanto, la probabilidad de que el usuario deje de pulsar enlaces y navegue directamente a cualquier otra web es 1-d. La introducción del factor de amortiguación en la fórmula resta algo de peso a todas las páginas de Internet y consigue que las páginas que no tienen ningún enlace a otra página no salgan especialmente beneficiadas. Si un usuario aterriza en una página sin enlaces, lo que hará después será navegar a cualquier otra página aleatoriamente, lo que equivale a suponer que una página sin enlaces salientes tiene enlaces a todas las páginas de Internet.

El PageRank de una página se define recursivamente y depende del número y PageRank de todas las páginas que la enlazan. Una página que está enlazada por muchas páginas

con un PageRank alto consigue también un PageRank alto. Si no hay enlaces a una página web, no hay apoyo a esa página específica.

Se puede interpretar el índice PageRank PR(A) como la probabilidad de que un usuario de Internet elegido al azar visite esa página. Por lo tanto los valores del índice están siempre entre 0 y 1, y la suma de todos ellos es la unidad.

Los detalles exactos de la versión real de este algoritmo utilizada por Google son desconocidos, ya que Google intenta mantener un cierto grado de confidencialidad para evitar la manipulación, e introduce correcciones como por ejemplo castigar con un peso nulo a los enlaces asociados a operaciones identificadas como spam, o premiar a las páginas que siguen en su elaboración todos los pasos recomendados por Google.

Para su aplicación con R utilizaremos la función page.rank del paquete igraph, que será necesario instalar.

page.rank(grafo)

El argumento principal, único obligatorio, es un grafo. Los datos deben convertirse necesariamente a ese formato.

Utilizaremos para el ejemplo los datos del archivo "conoce.csv", en formato de texto separado por punto y coma, que contienen 100 casos y dos variables, llamadas "ORIGEN" y "DESTINO". Se trata de un conjunto de 20 personas que representamos con letras (A, B, C,…, T). Cada caso representa una relación entre dos personas (ORIGEN conoce a DESTINO).

Leemos los datos, creando un conjunto o data set llamado "Dataset", para lo cual utilizamos la función read.csv2:

```
setwd("C:/CURSO DM")
Dataset <- read.csv2("conoce.csv", header=TRUE,encoding="latin1")
```

```
names(Dataset)    # escribe los nombres de las variables del conjunto Dataset
```

```
 [1] "ORIGEN"  "DESTINO"
```

```
head(Dataset)    #  escribe los primeros 6 casos  (tail escribiría los 6 últimos)
```

```
    ORIGEN DESTINO
1      A       B
2      A       D
3      A       E
4      A       G
5      A       H
6      A       J
```

table(Dataset$ORIGEN) # tabla de frecuencias de la variable ORIGEN

```
A  B  C  D  E  F  G  H  I  J  K  L  M  N  O  P  Q  R  S  T
8  5  6  3  8  7  5  6  6  6  3  4  2  7  5  6  5  2  3  3
```

"A" conoce a 8 personas del grupo, "B" a 5, "C" a 6, y así sucesivamente.

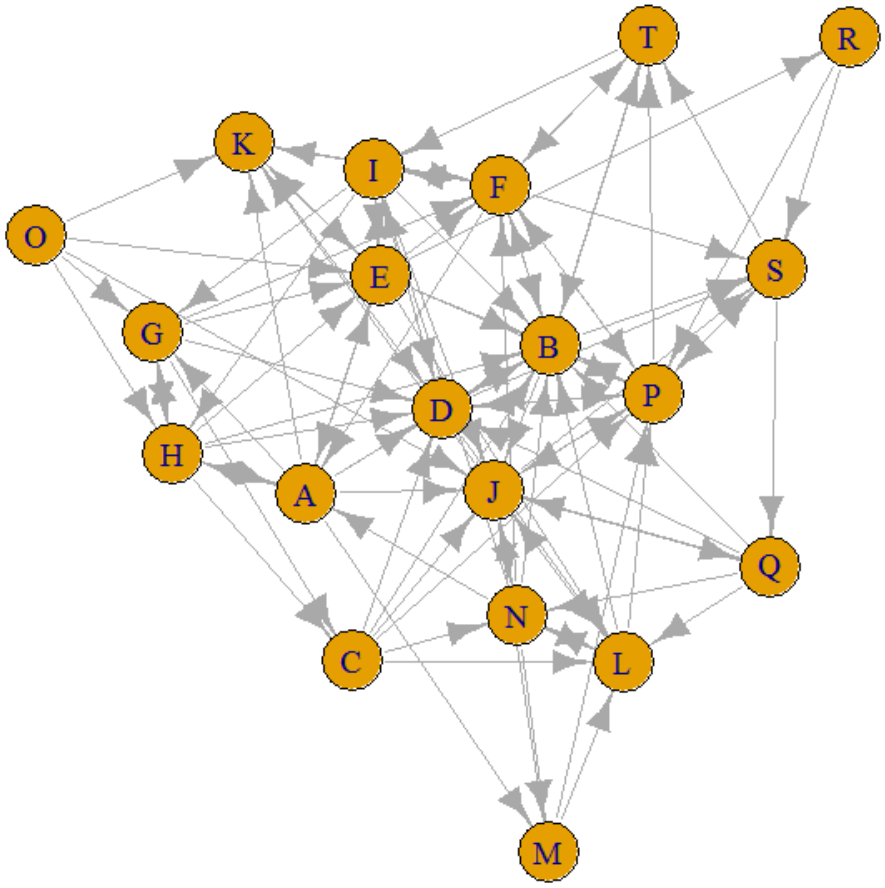
table(Dataset$DESTINO) # tabla de frecuencias de la variable DESTINO

```
 A  B  C  D  E  F  G  H  I  J  K  L  M  N  P  Q  R  S  T
 4 11  2  9  6  7  4  4  4  9  5  6  3  4  9  2  1  6  4
```

"A" es conocida por 4 personas, "B" por 11 (es la frecuencia más alta), "C" solo por 2, y "O", que no aparece, no es conocida por nadie.

Necesitamos transformar nuestro data set en un grafo, que será utilizado dentro de la función page.rank. Para eso utilizaremos la función graph.data.frame del paquete igraph:

```
library(igraph)          #  cargamos el paquete igraph
grafo <- graph.data.frame(Dataset)    # creamos el objeto 'grafo', transformación de Dataset
set.seed(123)
plot(grafo)    #  representamos el grafo
```



El grafo construido permite visualizar todas las relaciones "i conoce a j". Las personas con más flechas salientes conocen a más personas, y las que tienen más flechas entrantes son conocidas por más personas. La representación se realiza en orden aleatorio de los vértices o nodos (su aspecto es diferente cada vez que se ejecuta, aunque es siempre el mismo grafo); fijar el arranque aleatorio con set.seed(123) sirve para obtener siempre el mismo dibujo.

El algoritmo pagerank aplicado a este grafo permitirá construir un índice de conocimiento o importancia de cada componente del grupo. Cada persona tendrá un valor del índice, que será mayor cuanto más conocida sea, especialmente si las personas que la conocen son a su vez muy conocidas.

I <- page.rank(grafo) ; I # calcula el índice PageRank y lo escribe

El ";" en la línea anterior sirve para separar dos órdenes en la misma línea; la primera calcula el índice (objeto I) y la segunda lo escribe ejecutando el nombre del objeto.

```
> I <- page.rank(grafo); I
$vector
         A          B          C          D          E
0.02949676 0.09762971 0.01436672 0.07606231 0.03592066
         F          G          H          I          J
0.08345039 0.02178647 0.02232716 0.04739598 0.07732669
         K          L          M          N          O
0.03681348 0.05836547 0.02647255 0.04022044 0.00750000
         P          Q          R          S          T
0.10910430 0.04310515 0.01131657 0.08700190 0.07433729
```

rev(sort(I$vector)) # escribe los mismos valores en orden decreciente

```
> rev(sort(I$vector))

         P          B          S          F          J
0.10910430 0.09762971 0.08700190 0.08345039 0.07732669
         D          T          L          I          Q
0.07606231 0.07433729 0.05836547 0.04739598 0.04310515
         N          K          E          A          M
0.04022044 0.03681348 0.03592066 0.02949676 0.02647255
         H          G          C          R          O
0.02232716 0.02178647 0.01436672 0.01131657 0.00750000
```

La función "sort" ordena de menor a mayor, y "rev" invierte el orden (de mayor a menor). Ahora vemos en primer lugar los casos o nodos del grafo con mayor valor pagerank (P, B, S, F, …). La persona más conocida, en términos ponderados es P, seguida de B, aunque ésta es conocida directamente por más personas (11 frente a 9 de P como veíamos en las frecuencias relativas). El más relevante (P) no es necesariamente el más conocido (B), sino el que es más conocido por aquellos que a su vez son muy conocidos.

Ahora podemos representar de nuevo el grafo utilizando el índice pagerank para redimensionar el tamaño de los nodos, y visualizar de este modo la distinta importancia de cada miembro del conjunto (los más relevantes tienen mayor tamaño):

```
set.seed(123)
plot(grafo, vertex.size=I$vector*200)
```