Sistemas de Recomendación | Evaluación de Modelos

Josseline Perdomo

Mayo 2016

Introducción

Para esta Tarea 4 se nos dio la labor de realizar los siguientes apartados:

1. Implementar un sistema de recomendación para un periódico digital, recibiendo las transacciones de los usuarios en su portal web.
2. Implementar una función generadora de curvas ROC de la siguiente forma:

```
generate_ROC <- function(scores, real, target){
  # Aqui algoritmo para generar curva
}
```

Con el propósito de realizar un estudio práctico de los **Sistemas de Recomendación** y la **Evaluación de Modelos**.

Sistema de Recomendación

En el primer escenario, un periodico virtual nos da un dataset limpio con la información de las transacciones de su portal y nos solicita las siguientes demandas a resolver:

1. Un análisis exploratorio pertinente para detectar las transacciones bot que los clientes sospechan de su existencia.
2. Modificar el dataset para que los items pertenecientes a las transacciones fueran por su tipo de contenido.
3. Recomendar a un nuevo usuario que ingresa al portal un articulo.
4. Conocer los tipos de usuarios que ingresan al portal.
5. Conocer las 10 visitas con mayor y las 10 con menor tiempo de estadía en el portal.
6. Conocer las 10 transacciones con mayor número de apariciones en el dataset.

Todos estos requerimientos se realizan a continuación:

Instalando los paquetes necesarios

Se implementó una función auxiliar en caso de que algún paquete adicional no esté instalado.

```
include <- function(packages){
    for(pkg in packages){
        # Si ya está instalado, no lo instala.
        if(!require(pkg, character.only = T)){
            install.packages(pkg, repos = "https://cran.rstudio.com", dependencies = T)
            if(!require(pkg, character.only = T))
```

```
                stop(paste("load failure:", pkg))
        }
        library(pkg, character.only = T)
    }
}

include(c("arules", "arulesViz"))
```

Inicializando estructuras base

```
setwd("..") # Directorio padre del proyecto
# Cargando dataset provisto
dataset <- read.csv2("dat/periodico.csv", header = T,
                     sep = ",",
                     colClasses = "character",
                     nrows = 131300,
                     comment.char = "",
                     stringsAsFactors = F)

# Áreas de información
subjects <- c("deportes", "politica",
              "variedades", "internacional",
              "nacionales", "sucesos",
              "comunidad", "negocios",
              "opinion")

head(dataset, n = 5)
```

```
##   X     ID               entry                exit              articles
## 1 1 trans1 2016-05-02 22:39:52 2016-05-02 22:49:08  {item1,item9,item63}
## 2 2 trans2 2016-05-02 17:38:55 2016-05-02 17:53:29   {item1,item2,item3}
## 3 3 trans3 2016-05-01 06:57:57 2016-05-01 07:00:44 {item9,item43,item57}
## 4 4 trans4 2016-05-01 09:10:07 2016-05-01 09:15:16 {item2,item14,item72}
## 5 5 trans5 2016-05-01 00:28:29 2016-05-01 01:01:16              {item11}
```

A pesar de que el dataset se encuentra libre de datos erróneos, realizamos un **preprocesamiento** para la preparación de los datos de acuerdo a los requerimientos a cumplir.

Preprocesamiento

* Transformando *X* de character a integer:

```
dataset$X <- as.integer(dataset$X)
```

* Transformando columnas *entry* y *exit*

Es necesario tranformarlas a **tipo date con formato (YYYY-MM-DD hh-mm-ss)** para poder operar sobre estas columnas.

```
dataset$entry <- strptime(dataset$entry, "%Y-%m-%d %H:%M:%S")
dataset$exit <- strptime(dataset$exit, "%Y-%m-%d %H:%M:%S")
```

*** Creando columna *time***

Diferencia de *entry* - *exit* en segundos.

```
dataset$time <- difftime(dataset$exit, dataset$entry, unit = "secs")
```

*** Definiendo ID de las transacciones**

Hay 2 columnas que pueden ser el id univoco de las transacciones: *ID* y *X*, cada transacción (a pesar de tener los mismos items) son entradas distintas del dataset, por lo que de acuerdo al contexto del problema (Web log) no deben haber entradas repetidas (con mismo id).

Viendo la cantidad de transacciones y cuantas de ellas trasacciones son únicas:

```
rows <- nrow(dataset)
nrow(unique(dataset[,c("entry","exit","articles")]))
```

```
## [1] 131300
```

```
length(unique(dataset$ID)) == rows
```

```
## [1] FALSE
```

```
length(unique(dataset$X)) == rows
```

```
## [1] TRUE
```

Como la cantidad de transacciones usando la columna ID no es igual al total, podemos concluir que la columna *X* es la columna de los id unívocos, por lo que cambiamos su nombre:

```
colnames(dataset)[1] <- "tid"
```

*** Elminando columnas que se van a utilizar más durante la implementación.**

Estas columnas no aportaran información relevante para los venideros requisitos a llevar a cabo.

```
dataset$ID <- NULL
dataset$entry <- NULL
dataset$exit <- NULL
```

Requerimientos

1. Transformando el Dataset

En total, existen 81 items, 9 artículos por cada tema. Utilizando expresiones regulares transformaremos de la notación $item_N$ a $< subject > /articulo_n$:

```
items_id <- seq(1, 81)
items <- sprintf("%s/articulo%d", subjects[(items_id-1)%/%9+1], (items_id-1)%%9+1)
for(i in items_id)  dataset$articles <- gsub(sprintf("item%d(?=[,}])", i),
                                             items[i],
                                             dataset$articles,
                                             perl = T)
# Eliminando {}
dataset$articles <- gsub("\\{|\\}", "", dataset$articles)

head(dataset$articles, n = 5)
```

```
## [1] "deportes/articulo1,deportes/articulo9,comunidad/articulo9"
## [2] "deportes/articulo1,deportes/articulo2,deportes/articulo3"
## [3] "deportes/articulo9,nacionales/articulo7,comunidad/articulo3"
## [4] "deportes/articulo2,politica/articulo5,negocios/articulo9"
## [5] "politica/articulo2"
```

2. Transacciones Bot

El periódico acepta que una transacción no es realizada por un *bot*, cuando una persona dura más de 20 segundos en un artículo. Dado que no podemos asegurar con nigún conocimiento previo que durante el tiempo de la transacción la persona vio por mas de 20 segundos cada artículo, se toma el caso promedio. Por tanto, para verificar que una transacción no proviene de un bot, ésta debe durar al menos 20 segundos por la cantidad de artículos, es decir,

$$x > articles \times 20$$

Para llevar esto acabo, se hizo lo siguiente:

Se crea una lista de transacciones como vectores y se verifica la fórmula anteriormente descrita.

```
transactions <- strsplit(dataset$articles, ",")
tol <- 20 # Cantidad de tiempo en segundos m?nimo para no ser considerado bot
bots <- dataset$time <= (lengths(transactions)*tol)
```

Luego de tener cuales no cumplen la condición, se cuenta las transaciones que son bot en el dataset.

```
nrow(dataset[bots,])
```

```
## [1] 6599
```

El periódico no dice que se tiene que hacer con las transacciones bot, pero se asume que la opción mas viable es eliminarlas para que no perturben el sistema de recomendación que se creará.

```
dataset <- dataset[bots == F,]
transactions <- transactions[bots == F]
```

3. Cargando Transacciones

La lista `transactions` la volvemos del tipo **transactions**, usadada por **arules** para manejar las transacciones.:

```r
# Generando transacciones
names(transactions) <- dataset$tid
transactions <- as(transactions, "transactions")

inspect(tail(transactions, n = 5))
```

```
##   items                     transactionID
## 1 {internacional/articulo8,
##    politica/articulo1}            131296
## 2 {comunidad/articulo3,
##    deportes/articulo1,
##    negocios/articulo3}            131297
## 3 {comunidad/articulo5,
##    comunidad/articulo6,
##    nacionales/articulo3,
##    negocios/articulo2,
##    politica/articulo1}            131298
## 4 {comunidad/articulo7,
##    comunidad/articulo8,
##    nacionales/articulo3,
##    politica/articulo6}            131299
## 5 {comunidad/articulo7,
##    nacionales/articulo6}          131300
```

```r
summary(transactions)
```

```
## transactions as itemMatrix in sparse format with
##  124701 rows (elements/itemsets/transactions) and
##  81 columns (items) and a density of 0.0365946
##
## most frequent items:
## deportes/articulo1 deportes/articulo4 deportes/articulo7
##              21379              21214              21066
## deportes/articulo3 deportes/articulo9            (Other)
##              19156              19065             267754
##
## element (itemset/transaction) length distribution:
## sizes
##     1     2     3     4     5     6     7     8     9    10    11
## 17623 34304 33625 22117 10767  4288  1461   406    86    16     8
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   3.000   2.964   4.000  11.000
##
## includes extended item information - examples:
##               labels
## 1 comunidad/articulo1
## 2 comunidad/articulo2
## 3 comunidad/articulo3
##
## includes extended transaction information - examples:
##   transactionID
## 1             1
```

```
## 2             2
## 3             3
```

4. Tipos de Usuarios

La idea es agrupar a los usuarios del portal de acuerdo al contenido de las transacciones. Para ello consideraremos sólo las transacciones únicas, un modo de simplificar el dataset e incrementar el rendimiento del algoritmo.

```
u_transactions <- unique(transactions)
```

Concretamente, para producir los clusters usaremos **clustering jerárquico con el método ward** sobre una muestra de `u_transactions` con medida de similaridad *Jaccard*, usada comúnmente para datasets binarios sparse (tal y como es la matriz de transacciones), ya que mide la similaridad entre conjuntos finitos.

$$Jaccard(X,Y) = \frac{|X \cap Y|}{|X \cup Y|}, \text{ con } X, Y \in T = \{T1, .., T_n\}$$

```
# Tomando aproximadamente 20% de la muestra
small_sample <- sample(u_transactions, 10000)
# Calculando similaridad entre itemsets
d_jaccard <- dissimilarity(small_sample)
d_jaccard[is.na(d_jaccard)] <- 1 # Eliminando NA
```

Se optó por clustering jerárquico ya que es ideal para clusters rectangulares. De acuerdo a las peticiones del cliente, entonces $k = 8$.

```
# Calculando clusters
hclusters <- hclust(d_jaccard, method = "ward")
```

```
## The "ward" method has been renamed to "ward.D"; note new "ward.D2"
```

```
# Haciendo corte
cutting <- cutree(hclusters, k = 8)
# Dendrograma
pdf(file = "similarity.pdf", width = 150)
plot(hclusters, cex = 0.5)
dev.off()
```

```
## pdf
##   2
```

Ahora, para las restantes transacciones, predeciremos a qué clusters pertenecen.

```
# Descartando transacciones pertenecientes a small_sample
big_sample <- u_transactions[!(u_transactions %in% small_sample)]
# Prediciendo
pred_labels <- predict(small_sample, big_sample, cutting)
```

A continuación, los resultados de los clusters sobre las transacciones:

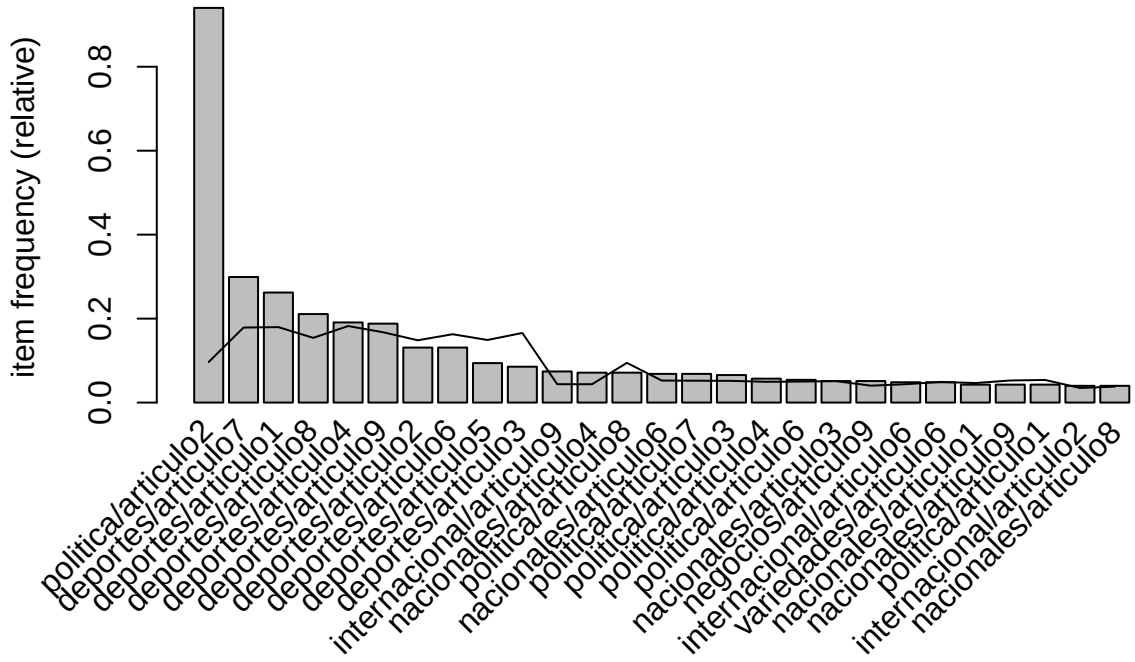
```r
# Resultados
table(cutting)
```

```
## cutting
##    1    2    3    4    5    6    7    8
##  351 5932  849  961  784  357  349  417
```

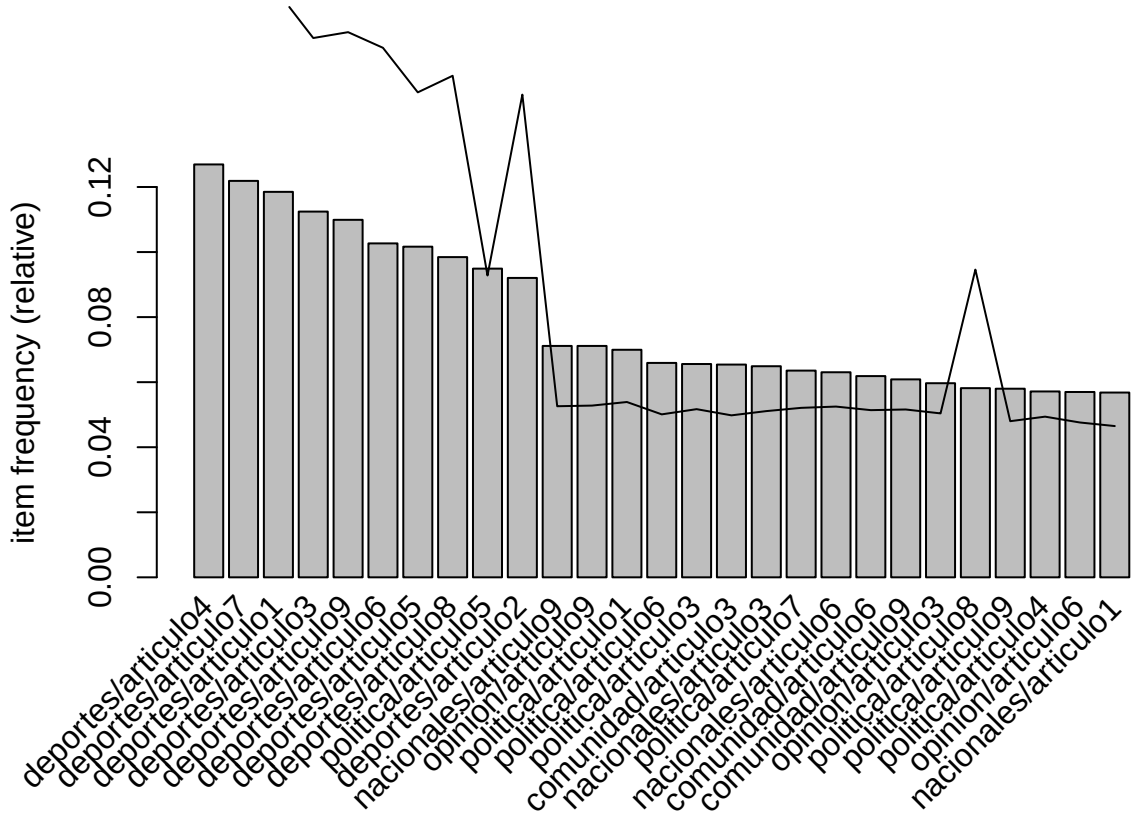
```r
table(pred_labels)
```

```
## pred_labels
##     1     2     3     4     5     6     7     8
##  1537 23078  3469  4498  3579  1558  1483  2048
```

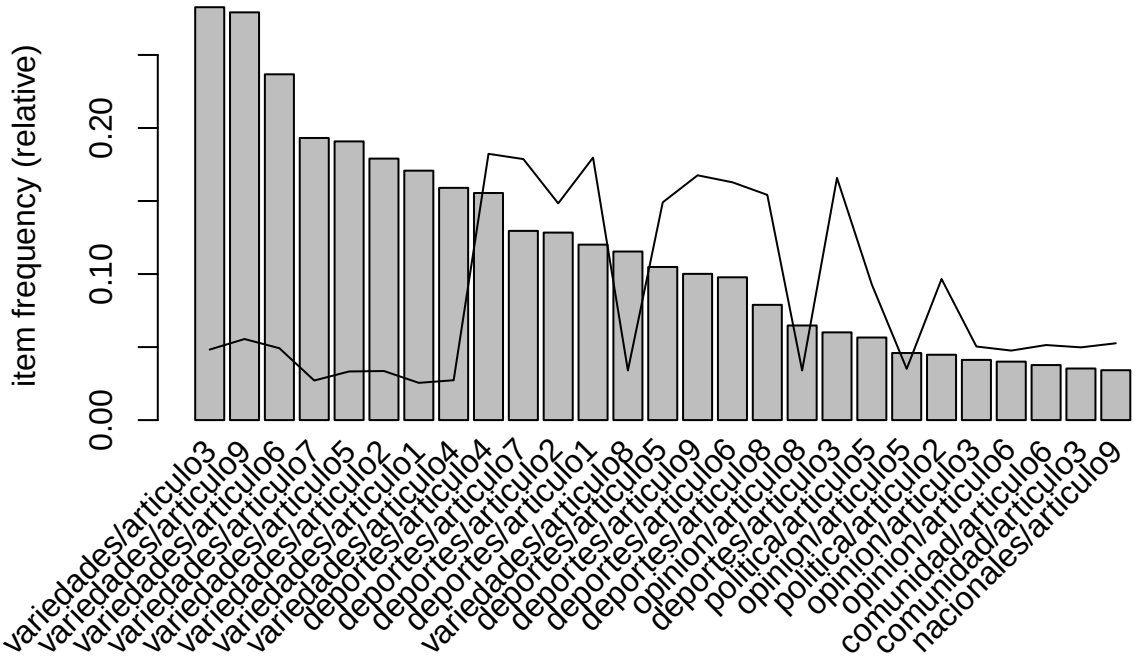
```r
# Items más frecuentes en cada cluster de small_sample
itemFrequencyPlot(small_sample[cutting == 1],
                  population = small_sample,
                  topN = 27)
```



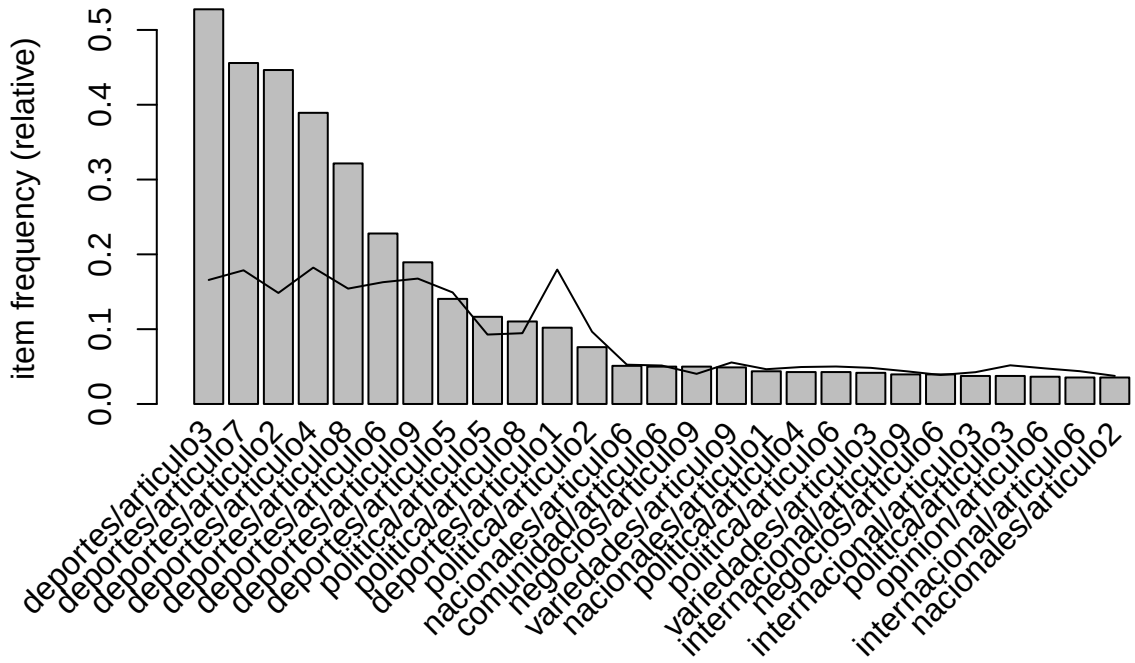

```r
itemFrequencyPlot(small_sample[cutting == 2],
                  population = small_sample,
                  topN = 27)
```



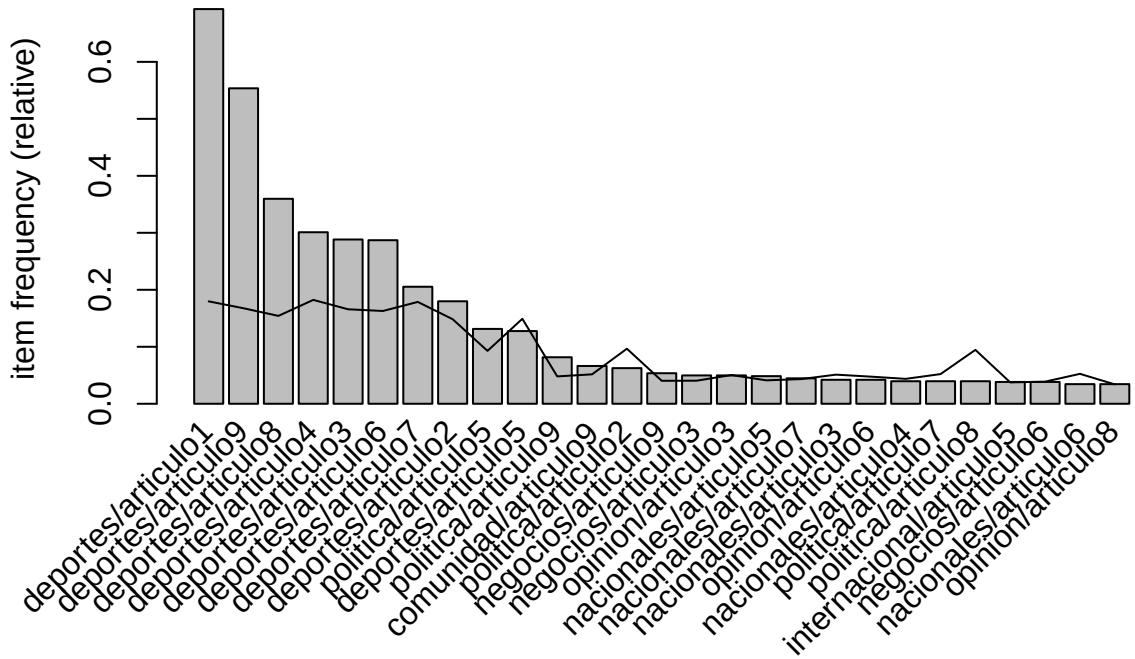

```
itemFrequencyPlot(small_sample[cutting == 3],
                  population = small_sample,
                  topN = 27)
```



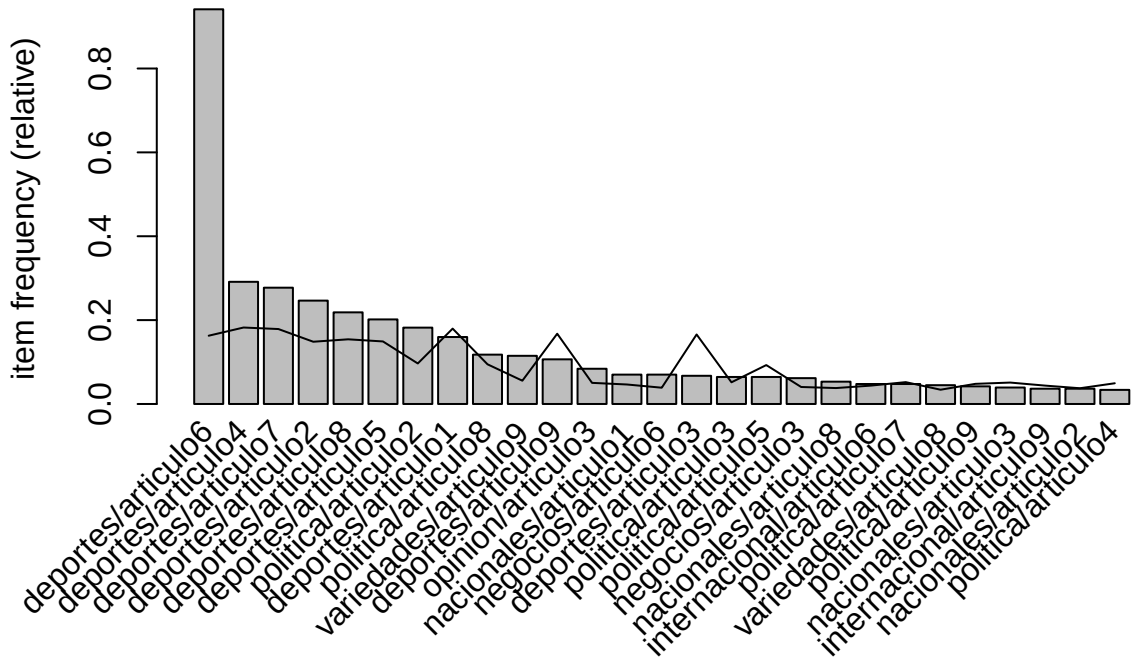
```
itemFrequencyPlot(small_sample[cutting == 4],
                  population = small_sample,
                  topN = 27)
```



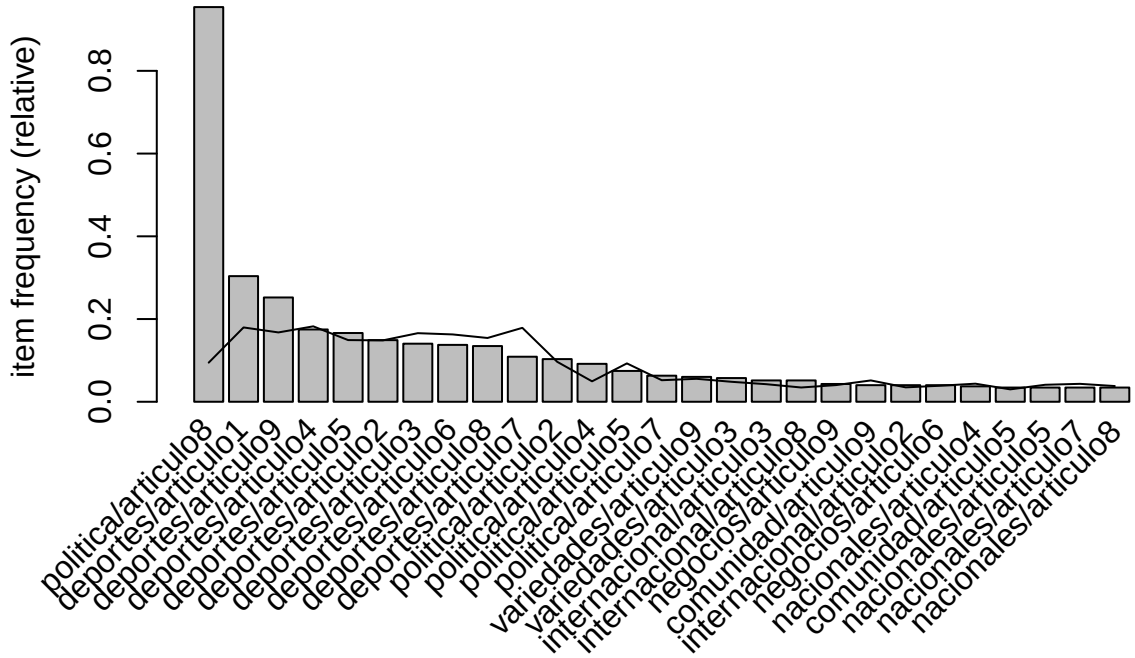

```
itemFrequencyPlot(small_sample[cutting == 5],
                  population = small_sample,
                  topN = 27)
```



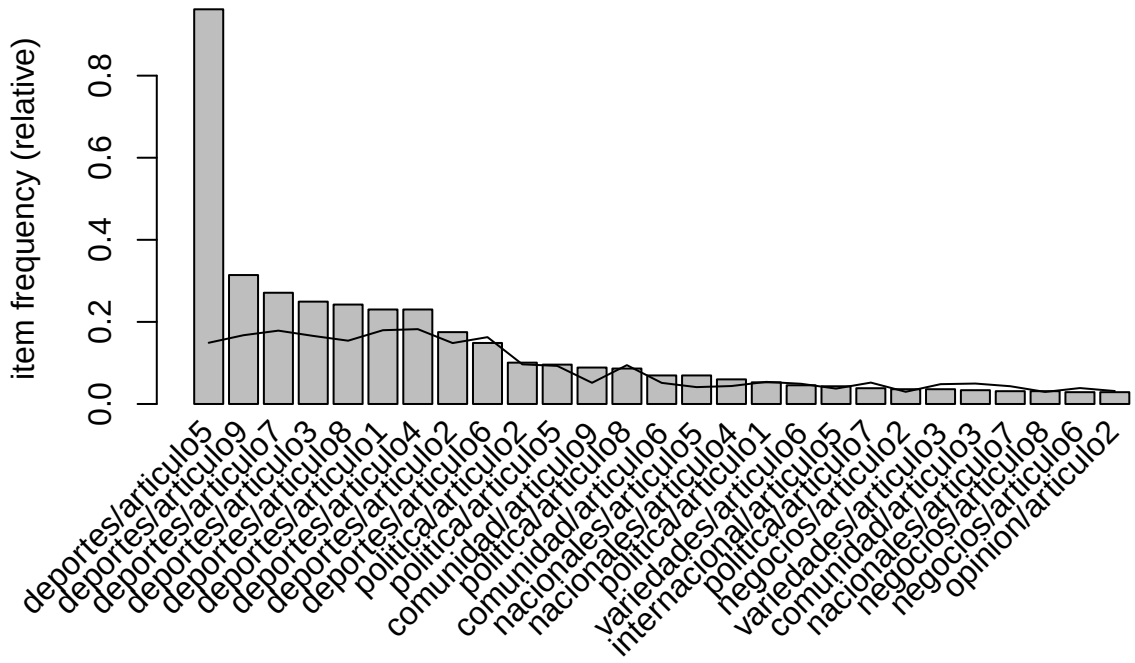
```
itemFrequencyPlot(small_sample[cutting == 6],
                  population = small_sample,
                  topN = 27)
```




```
itemFrequencyPlot(small_sample[cutting == 7],
                  population = small_sample,
                  topN = 27)
```



```
itemFrequencyPlot(small_sample[cutting == 8],
                  population = small_sample,
                  topN = 27)
```



5. Generando Reglas

Para la producción de reglas, hay 2 conceptos importantes a tomar en cuenta: soporte y confianza.

El soporte de un conjunto de items I se define como la proporción de las transacciones en el dataset que contienen a I. Mientras que la confianza de una regla $X \Rightarrow Y$ es la probabilidad condicional de que una transacción contenga el conjunto de items Y dado que contiene el conjunto X. Se requieren reglas de asociación para satisfacer tanto un soporte mínimo y una restricción mínima confianza al mismo tiempo.

Dado que no hubo especificidad con respecto a la mínima frecuencia de los item sets I, queremos tomar la mayor cantidad de transacciones para generar las reglas, por lo que **minsup** debe ser un valor bajo.

```
# Un itemset I debe aparecer al menos 4 veces
sprintf("%.8f", (3 / nrow(transactions)))
```

```
## [1] "0.00002406"
```

Con respecto a la confianza, mientras más alto sea habrá menos reglas, lo que podría no resultar conveniente ya que no generariamos las reglas suficientes como para que el recomendador sea apropiado, sin embargo, tampoco debe ser un valor bajo, por lo que un valor para **minconf** intermedio se adecuaría a nuestro escenario.

```
# Generando reglas
rules <- apriori(transactions,
                 parameter = list(sup = 0.00003,
                                  conf = 0.65,
                                  target = "rules"))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen maxlen
##        0.65    0.1    1 none FALSE            TRUE   3e-05      1     10
##  target   ext
##   rules FALSE
##
## 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 ...[81 item(s), 124701 transaction(s)] done [0.07s].
## sorting and recoding items ... [81 item(s)] done [0.01s].
## creating transaction tree ... done [0.33s].
## checking subsets of size 1 2 3 4 5 6 7 done [0.11s].
## writing ... [482 rule(s)] done [0.01s].
## creating S4 object  ... done [0.09s].
```

```
# Ordenar reglas por confianza descendientemente
rules <- sort(rules, decreasing = T, by = "confidence")

inspect(head(rules, n = 5))
```

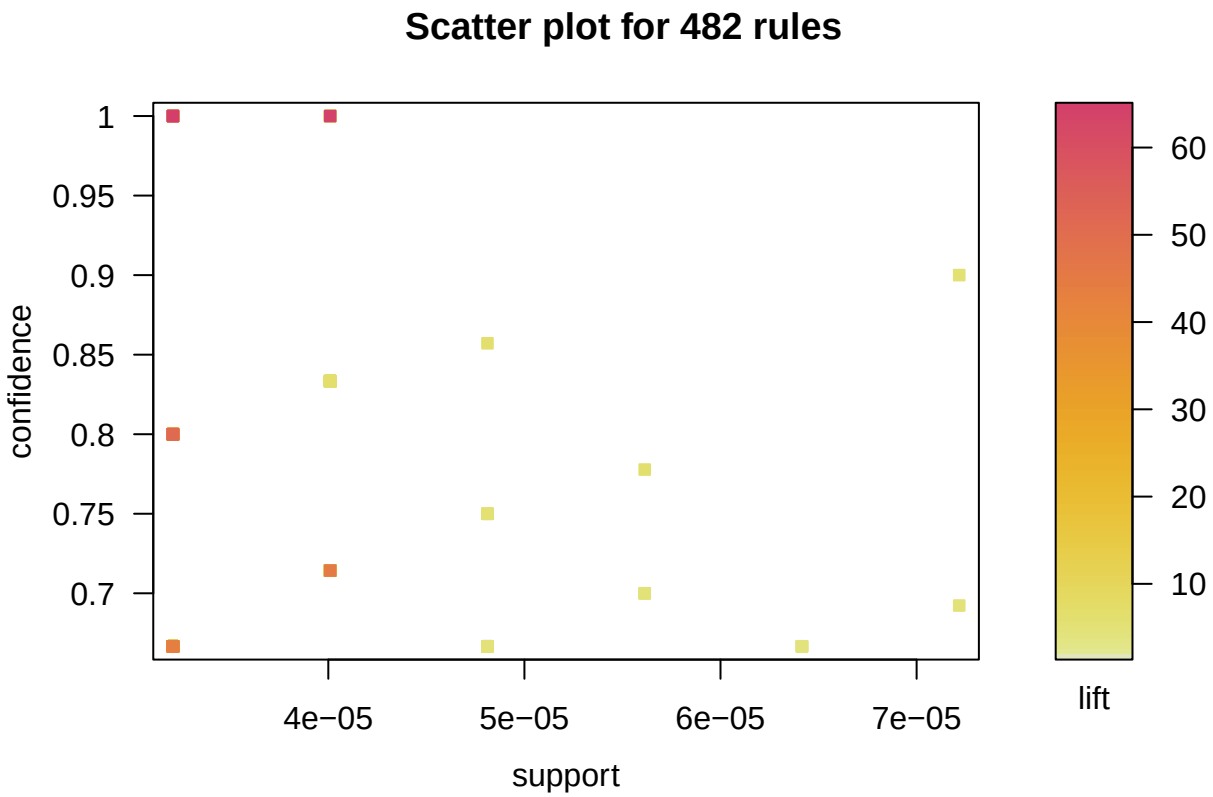
```
##   lhs                       rhs                         support confidence      lift
## 1 {deportes/articulo3,
##    negocios/articulo9,
##    sucesos/articulo2}    => {deportes/articulo1}   3.207673e-05          1  5.832873
## 2 {deportes/articulo8,
##    politica/articulo6,
##    sucesos/articulo5}    => {deportes/articulo2}   3.207673e-05          1  7.261880
## 3 {comunidad/articulo6,
##    variedades/articulo1,
##    variedades/articulo5} => {variedades/articulo7} 4.009591e-05          1 63.818321
## 4 {deportes/articulo6,
##    negocios/articulo2,
##    politica/articulo4}   => {deportes/articulo9}   3.207673e-05          1  6.540834
## 5 {deportes/articulo3,
##    negocios/articulo8,
##    politica/articulo1}   => {deportes/articulo9}   3.207673e-05          1  6.540834
```

```
summary(rules)
```

```
## set of 482 rules
##
## rule length distribution (lhs + rhs):sizes
##   3   4   5   6   7
##   1 208 210  62   1
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
```

```
##   3.000   4.000   5.000   4.697   5.000   7.000
##
## summary of quality measures:
##     support            confidence          lift
##  Min.   :3.208e-05   Min.   :0.6667   Min.   : 3.889
##  1st Qu.:3.208e-05   1st Qu.:0.6667   1st Qu.: 4.228
##  Median :3.208e-05   Median :0.7143   Median : 4.703
##  Mean   :3.444e-05   Mean   :0.7532   Mean   : 6.861
##  3rd Qu.:3.208e-05   3rd Qu.:0.8000   3rd Qu.: 5.234
##  Max.   :7.217e-05   Max.   :1.0000   Max.   :64.813
##
## mining info:
##          data ntransactions support confidence
##  transactions        124701   3e-05       0.65
```

```
plot(rules, data = transactions)
```



6. Recomendando un Artículo

Dado una nueva transacción T_k de tamaño n, recomendar un artículo $n+1$:

```
recommend <- function(itemset, rules){
    suitableRules <- subset(rules, lhs %ain% itemset & !rhs %in% itemset)
    # Si no se encuentra transacción, entonces buscar un subconjunto
    if(length(suitableRules) == 0)
        suitableRules <- subset(rules, lhs %in% itemset & !rhs %in% itemset)
```

```
    return(inspect(suitableRules@rhs[1]))
}

# Evaluando recommend
example <- c("deportes/articulo1",
             "deportes/articulo2",
             "deportes/articulo3")

recommend(example, rules)
```

```
##   items
## 1 {deportes/articulo7}
```

7. Transacciones con Mayor y Menor Tiempo de Estadía en el Portal

Para ello, ordenamos las transacciones en el dataset de acuerdo a la columna *time* descendentemente:

```
by_time <- order(dataset$time, decreasing = T)
```

* Mayor tiempo

```
head(dataset[by_time,c("tid","time")], n = 10)
```

```
##           tid       time
## 93676   93676 12264 secs
## 7511     7511 12234 secs
## 80516   80516 11743 secs
## 122879 122879 11597 secs
## 130641 130641 11508 secs
## 66995   66995 11481 secs
## 111953 111953 11421 secs
## 23099   23099 11419 secs
## 55628   55628 11381 secs
## 48007   48007 11372 secs
```

* Menor tiempo

```
tail(dataset[by_time,c("tid","time")], n = 10)
```

```
##           tid    time
## 39574   39574 21 secs
## 50391   50391 21 secs
## 60586   60586 21 secs
## 63057   63057 21 secs
## 77010   77010 21 secs
## 77795   77795 21 secs
## 88776   88776 21 secs
## 94291   94291 21 secs
## 99027   99027 21 secs
## 125322 125322 21 secs
```

8. Transacciones más frecuentes

Para obtener los itemsets más frecuentes, usaremos `apriori()` pero indicando que deseamos son sólo los items frecuentes.

```
freq_itemsets <- apriori(transactions, parameter = list(sup = 0.00003,
                                                        target = "frequent"))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen maxlen
##          NA    0.1    1 none FALSE            TRUE   3e-05      1     10
##             target   ext
##  frequent itemsets FALSE
##
## 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 ...[81 item(s), 124701 transaction(s)] done [0.07s].
## sorting and recoding items ... [81 item(s)] done [0.01s].
## creating transaction tree ... done [0.34s].
## checking subsets of size 1 2 3 4 5 6 7 done [0.11s].
## writing ... [35720 set(s)] done [0.01s].
## creating S4 object  ... done [0.11s].
```

```
# Obteniendo los 10 itemsets más frecuentes
inspect(head(sort(freq_itemsets, decreasing = T), n = 10))
```

```
##    items                 support
## 81 {deportes/articulo1} 0.17144209
## 79 {deportes/articulo4} 0.17011892
## 80 {deportes/articulo7} 0.16893209
## 78 {deportes/articulo3} 0.15361545
## 76 {deportes/articulo9} 0.15288570
## 77 {deportes/articulo6} 0.15284561
## 74 {deportes/articulo8} 0.13818654
## 75 {deportes/articulo2} 0.13770539
## 73 {deportes/articulo5} 0.13588504
## 72 {politica/articulo5} 0.07084145
```

Evaluación de Modelos: Curva ROC

Este generador fue desarrollado de acuerdo a la explicación del paper *An Introduction to ROC analysis* por *Tom Fawcett*.

Parámetros

1. Los scores por instancia (no necesariamente ordenados).

2. La verdadera clase de las instancias.
3. La clase target. En el caso de que nclass > 2 entonces el enfoque es *1 vs all*.

Algoritmo

```
generate_ROC <- function (scores, real, target){
    data <- data.frame(y = real, score = scores)
    # Ordenando instancias por score descendientemente
    data <- data[with(data, order(score, decreasing = T)),]
    # Matriz de confusión, columna de valores positivos
    p_confusion <- setNames(c(0,0), c("TP", "FP"))
    # Cantidad total de valores de la clase P y N
    target_entries <- data$y == target
    real_v <- setNames(c(length(data$y[target_entries]),
                         length(data$y[!target_entries])),
                       c("P", "N"))
    prev <- Inf
    roc_curve <- data.frame(x = double(), y = numeric(), score = numeric())
    indices <- seq(1, nrow(data))

    for(i in indices){
        if(data$score[i] != prev){
            roc_curve[nrow(roc_curve)+1,] <- c(p_confusion["FP"]/real_v["N"],
                                               p_confusion["TP"]/real_v["P"],prev)
            prev <- data$score[i]
        }
        if(data$y[i] == target)
          p_confusion["TP"] <- p_confusion["TP"] + 1
        else
            p_confusion["FP"] <- p_confusion["FP"] + 1
    }
    # Insertando último punto (1,1)
    roc_curve[nrow(roc_curve)+1,] <- c(p_confusion["FP"]/real_v["N"],
                                       p_confusion["TP"]/real_v["P"],prev)

    return(roc_curve)
}

# Probando función
y <- c(2, 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 1, 2, 1, 1, 1, 2, 1, 1, 1)
scores <- c(0.9, 0.8, 0.7, 0.6, 0.55, 0.54, 0.53, 0.52, 0.5, 0.5, 0.5,
            0.5, 0.38, 0.37, 0.36, 0.35, 0.34, 0.33, 0.30, 0.1)
target <- 2 # Clase considerada positiva

curve <- generate_ROC(scores, y, target)

# Graficando la curva retornada
plot(curve$x, curve$y,
     type ="l",
     lty = 2,
     xlim = c(0, 1.04),
     xlab = "FP-Rate",
```

```
     ylab = "TP-Rate",
     main = "ROC Curve")
abline(0, 1, lty = 2, col = "darkgray")
points(curve$x, curve$y, col = "red", pch = 19)
text(curve$x, curve$y, curve$score, cex = 0.7, pos = 4)
```