Sistemas de Recomendación y Evaluación de Modelos

Deyban Pérez

May 13, 2016

Abstract

Los **Sistemas de Recomendación** son aquellos que se basan en datos de transacciones de usuarios para generar reglas que permitan asociar la presencia de **items** con la presencia de otro, tienen una estructura de **antecedende => consecuencia** y el objetivo es buscar antecedentes que generen consecuentes con algún valor de **soporte mínimo** y **confianza mínima**. A su vez, en esta oportunidad hablaremos un poco de las **Curvas ROC** cómo herramienta para la clasificación de los modelos, esta técnica se basa en la **tasa de verdaderos-positivos** (eje y) y en la **tasa de falsos -positivos** (eje x) para generar una curva en dos dimensiones que permita evaluar un modelo, cabe destacar que para esto es importante tener la matriz de confusión de la salida del modelo que permita elaborar la curva; la curva ROC tiene ventajas desde el punto de vista que permite la visualización y evalucación rápida dle modelo, tomando como referencia ciertos puntos claves en cuanto al comportamiento de la misma, y nos da una herramienta más para estudiar el modelo y ajustarlo en caso de ser necesario.

Introducción

Se presentan dos escenarios, el primero consta de transacciones realizadas en un periódico digital con respecto a los artículos que los diferentes usuarios ven. Dicho esto se solicitan varias actividades como lo son:

1. Eliminar las transacciones bot.
2. Conocer los **tipos de usuarios** que ingresan a la página, y determinar la proporción de cada uno de los usuarios.
3. Modificar el dataset de tal manera que las transacciones contengan la estructira **contenido/articuloN**.
4. Recomendar un nuevo artículo a un nuevo usuario que haya visto **n** artículos previos basándose en el contedio de esos n artículos previos.
5. Conocer las 10 visitas con mayor tiempo de estadía en la página y las 10 visitas con menor tiempo de estadía en la página.
6. Conocer las 10 transacciones con mayor número de apariciones en el dataset.

La segunda sección consta de elaborar una función de genere el gráfico de la **Curva ROC** dada la siguiente firma de la función:

generate_ROC = functions(scores, real, target) { #Generar curva }

Implementación

A continuación comenzaremos con el análisis e implementación de lo mencionado en el punto anterior.

Definición de Funciones

Esta sección corresponde a todo lo relacionado con el periódico digital, hay algunos pasos comunes que se refieren a la carga del dataset y definición de algunas funciones que realizaremos a continuación:

1. Función instala un paquete pasado como parámetro sólo si este no se encuentra instalado

```
install = function(pkg)
{
  # If is is installed does not install packages
  if (!require(pkg, character.only = TRUE))
  {
    install.packages(pkg)
    if (!require(pkg, character.only = TRUE))
      stop(paste("load failure:", pkg))
  }
}
```

2. Función que retorna un substring de una cadena de la forma **itemX**, donde **X** es nuestro dato de interés que corresponde a un número entero y la palabra **item** queremos desecharla.

```
subString = function(element)
{
  return (substring(element, 5))
}
```

3. Función que retorna la converción de un un elemento X como tipo numérico flotante en doble precisión.

```
changeType = function(element)
{
  return(as.numeric(element))
}
```

4. Función que dado un el número de item (element), calcula el número del artículo, el tipo de clase y lo ensambla en el formato **clase/artículoX**.

```
changeFormat = function(element)
{
  classes = c("deportes", "politica", "variedades", "internacional",
              "nacionales", "sucesos", "comunidad", "negocios", "opinion")

  article = element%%9;
  class = element%/%9 + 1;

  if(article == 0)
  {
    article = 9;
    class = class-1;
  }

  return(paste(classes[class], "/", "articulo",article, sep = ""))
}
```

5. Función que aplica la función **changeFormat** a todos los elementos de la lista de transacciones y luego los agrupa con una "," como deparador.

```
convertFormat = function(element)
{
  items = as.integer(element)
  newItems = changeFormat(items[1])

  if(length(items) == 1)
  {
    return(newItems)

  }else
  {
    for(i in 2:length(items))
    {
      newItems = paste(newItems,",", changeFormat(items[i]), sep = "")
    }

    return(newItems)
  }
}
```

6. Función que que retorna sólo la clase dado el númeor de un item

```
changeFormatAux = function(element)
{
  classes = c("deportes", "politica", "variedades", "internacional",
              "nacionales", "sucesos", "comunidad", "negocios", "opinion")

  article = element%%9;
  class = element%/%9 + 1;

  if(article == 0)
  {
    article = 9;
    class = class-1;
  }

  return(classes[class])
}
```

7. Función que retorna los items únicos de una transacción donde sólo están presente las clases.

```
convertFormatAux = function(element)
{
  items = as.integer(element)
  newItems = changeFormatAux(items[1])
  aux = vector(mode = "character")

  if(length(items) == 1)
  {
    return(newItems)

  }else
```

```
  {
    aux[1] = newItems

    for(i in 2:length(items))
    {
      aux[i] = changeFormatAux(items[i])
    }

    aux = unique(aux)

    return(as.character(aux))
  }
}
```

8. Función que toma los items únicos por clases de las transacciones y los ensambla de la forma: **claseX, claseY, . . . , claseZ**.

```
union = function(element)
{
  newItem = element[1]

  if(length(element) == 1)
  {
    return(element)
  }else
  {
    for(i in 2:length(element))
    {
      newItem = paste(newItem, ",", element[i], sep = "")

    }
    return(newItem)
  }
}
```

9. Función que remueve el postfijo **articuloX** de un elemento de una transacción.

```
removeArticle = function(element)
{
  returnValue = vector(mode = "character")

  for (i in 1:length(element))
  {
    aux = substr(element[i],1,1)

    if(aux == "d")
    {
      returnValue[i] = "deportes"

    }else if(aux == "p")
    {
      returnValue[i] = "politica"
```

```
    }else if(aux == "v")
    {
      returnValue[i] = "variedades"

    }else if(aux == "i")
    {
      returnValue[i] = "internacional"

    }else if(aux == "n")
    {
      returnValue[i] = "nacionales"

    }else if(aux == "s")
    {
      returnValue[i] = "sucesos"

    }else if(aux == "c")
    {
      returnValue[i] = "comunidad"

    }else if(aux == "n")
    {
      returnValue[i] = "negocios"

    }else if(aux == "o")
    {
      returnValue[i] = "opinion"

    }
  }

  return(returnValue)
}
```

Instalando los Paquetes Necesarios

Cómo implementaremos reglas de asociación, utilizaremos el algoritmo **apriori** de la biblioteca **arules** y adicionalmente utilizaremos la biblioteca **arulesViz** para poder generar gráficos, adicionalmente para poder clusterizar usaremos la biblioetca **flexclust**.

```
install("arules")
install("arulesViz")
install("flexclust")
```

Ahora cargamos los paquetes

```
library("arules")
library("arulesViz")
library("flexclust")
```

Periódico

En esta sección irá todo lo correspondiente al estudio de peridódico digital.

Cargar Dataset

```
df_periodico = read.csv("../Data/periodico.csv", sep = ",")
```

Transacciones Bots

El dataset tiene dos columnas refentes al tiempo: **entry** y **exit**, lo que haremos es restar por cada entrada en el dataset los tiempo exit-entry. Si esa resta nos da <= que 20 segundos. Entonces identificaremos esa entrada como un bot.

El número inicial de entradas en el dataset es:

```
nrow(df_periodico)
```

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

Ahora vamos a ver cuantas transacciones bots tenemos, vamos a transformas el formato de la hora de las columnas **entry** y **exit** a un formato más manipulable.

```
entry = as.POSIXct(df_periodico$entry, format = "%Y-%m-%d %H:%M:%S")
exit = as.POSIXct(df_periodico$exit, format = "%Y-%m-%d %H:%M:%S")
```

Ahora vamos a restar las horas.

```
diff = difftime(exit, entry, units = "secs")
```

Ahora en **diff** tenemos un vector con el tiempo total en **segundos** de cada una de las transacciones, y buscaremos aquellas que fueron <= a 20 segundos; pero antes debemos descomponer el conjunto de transacciones para poder tener el número total de transacciones, ya que lo que en verdad estamos buscando es que: **(tiempo_total / transacciones) <= 20**.

```
df_periodico$articles = as.character(df_periodico$articles)
split = substring(df_periodico$articles, 2)
split = strsplit(x = split, split = ",|}")
```

De esta manera en **split** tenemos las transacciones de manera individual cómo se muestra a continuación:

```
split[[1]]
```

```
## [1] "item1"  "item9"  "item63"
```

Ahora podemos contar cuantos artículos vio cada usuario y poder aplicar la fórmula: *(tiempo_total / transacciones) <= 20**.

```
bots = df_periodico$X[(diff/lengths(split[1:nrow(df_periodico)])) <= 20]
```

En bots, tenemos el índice de todas las transacciones que se asumen que son bots, que son un total de:

```
length(bots)
```

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

Ahora eliminaremos a los bots de nuestro dataset inicial:

```
df_periodico = df_periodico[-bots,];
```

Quedándonos un total de:

```
nrow(df_periodico)
```

```
## [1] 124701
```

Transacciones que no son bots.

Modificando el Formato de los Artículos

Lo primero que haremos es cambiar el nombre la columna **5** a **items**:

```
colnames(df_periodico)[5] = "items"
```

Ahora bien, recordemos que en split, teníamos los items individuales de cada usuario de la siguiente manera:

```
split[[1]]
```

```
## [1] "item1"  "item9"  "item63"
```

Ahora lo que haremos es remover el prefijo **item** a cada uno de los items en cada una de las transacciones, pero primero vamos eliminaremos las transacciones bots tambien del conjunto **split**:

```
split = split[-bots]
articles = lapply(split, subString)
```

En este punto en **articles** tenemos los items que acda usuario ha visto de la siguiente manera:

```
articles[[1]]
```

```
## [1] "1"  "9"  "63"
```

Y ahora si podemos organizarlo de la forma deseada haciendo uso de la función definida previamente **convertFormat**:

```
df_periodico$articles = lapply(articles, convertFormat)
```

Ahora, tenemos una nueva columna en nuestro dataset llamada **articles** que tiene la siguiente estructura:

```
df_periodico$articles[1]
```

```
## [[1]]
## [1] "deportes/articulo1,deportes/articulo9,comunidad/articulo9"
```

La vamos a transformar esa columnda a tipo **char**

```
df_periodico$articles = as.character(df_periodico$articles)
```

Y vamos a guardar ese dataset con el nuevo nombre **periodico_arreglado.csv**.

```
write.csv(x = df_periodico, file = "../Data/periodico_arreglado.csv", row.names = FALSE)
```

Generando las Transacciones

A partir de acá necesitamos crear las matrices de transacciones pertinentes para poder realizar los próximos items de las preguntas.

Empecemos:

Primero vamos a también convertir las transacciones en la forma **claseX,claseY,. . . ,claseZ**

```
articlesAux = lapply(articles, convertFormatAux)
articlesAux = lapply(articlesAux, union)
```

Ahora también tenemos los items ordenados de la siguiente forma:

```
articlesAux[1]
```

```
## [[1]]
## [1] "deportes,comunidad"
```

Ahora vamos a generar las matrices de transacciones de la forma con y sin artículos

Con Artículos

```
list = lapply(df_periodico$articles,strsplit, split = ",")
list = lapply(list, unlist)
transactions = as(list, "transactions")
```

Sin Artículos

```
listAux = lapply(articlesAux,strsplit, split = ",")
listAux = lapply(listAux, unlist)
transactionsAux = as(listAux, "transactions")
```

Clasificando a los Usuarios del Sistema

Existen nueve (9) categorías, acá lo que nos interesa es ver que tipo se sección ven los diferentes usuarios, no nos interesa el artículo en específico, para eso utilizaremos la modalidad de las transacciones donde sólo salen las clases. y no sólo eso además sólo utilizaremos las transacciones únicas.

```
uniqueTransactions = unique(listAux)
uniqueTransactions = as(uniqueTransactions, "transactions")
uniqueTransactions = as(uniqueTransactions, "matrix")
```

Ahora si bien vemos las dimensiones de la matriz nos damos cuenta de que hay:

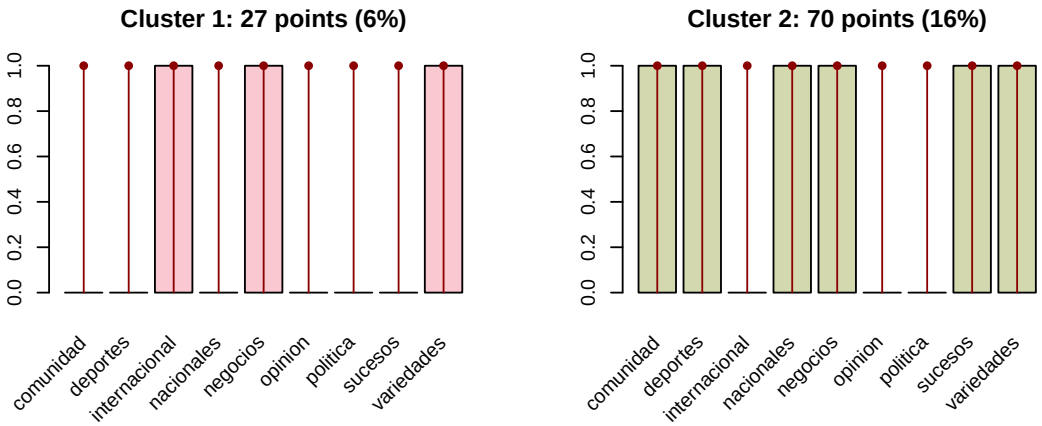
```
dim(uniqueTransactions)
```

```
## [1] 425   9
```

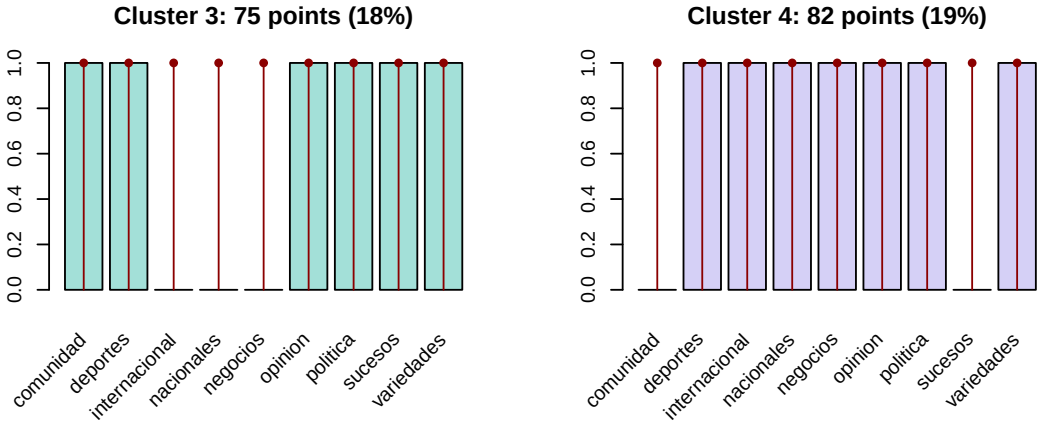
425 filas de **512**, esto es debido a 2^9 es igual a *512* y **9 columnas** que correesponden a las diferentes secciones.

Ahora bien para clusterizar se usará **Kmeans** con el paquete **flexclust** y con el algoritmo de **distancia** que saca la distancia entre conjuntos que es exactamente lo que tenemos acá.

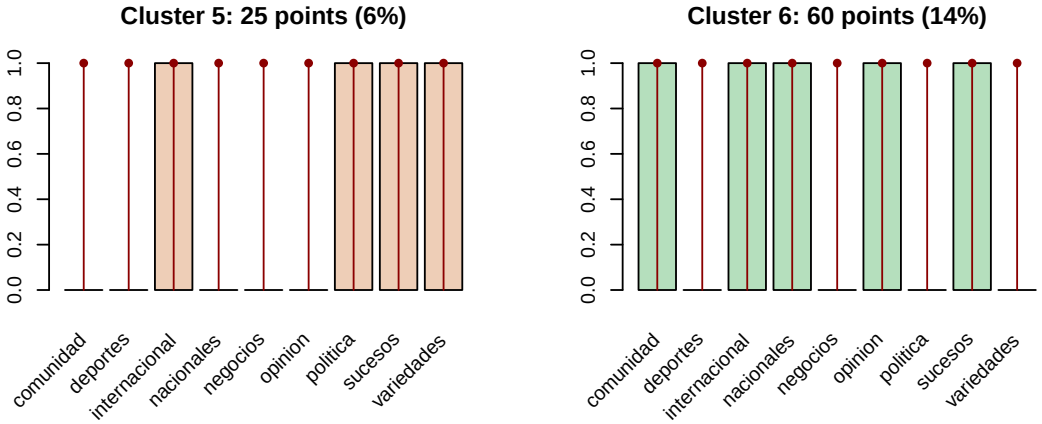
```
set.seed(22)
model = kcca(uniqueTransactions, k=8 ,family = kccaFamily("jaccard"))
barplot(model)
```


Cluster 1: 27 points (6%)
0.0 0.2 0.4 0.6 0.8 1.0
comunidad deportes internacional nacionales negocios opinion politica sucesos variedades

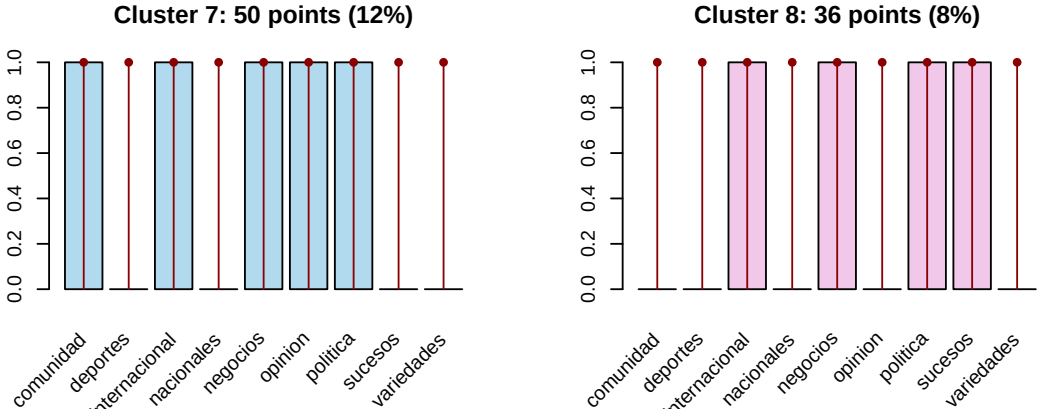
Cluster 2: 70 points (16%)
0.0 0.2 0.4 0.6 0.8 1.0
comunidad deportes internacional nacionales negocios opinion politica sucesos variedades


Cluster 3: 75 points (18%)
0.0 0.2 0.4 0.6 0.8 1.0
comunidad deportes internacional nacionales negocios opinion politica sucesos variedades

Cluster 4: 82 points (19%)
0.0 0.2 0.4 0.6 0.8 1.0
comunidad deportes internacional nacionales negocios opinion politica sucesos variedades


Cluster 5: 25 points (6%)
0.0 0.2 0.4 0.6 0.8 1.0
comunidad deportes internacional nacionales negocios opinion politica sucesos variedades

Cluster 6: 60 points (14%)
0.0 0.2 0.4 0.6 0.8 1.0
comunidad deportes internacional nacionales negocios opinion politica sucesos variedades


Cluster 7: 50 points (12%)
0.0 0.2 0.4 0.6 0.8 1.0
comunidad deportes internacional nacionales negocios opinion politica sucesos variedades

Cluster 8: 36 points (8%)
0.0 0.2 0.4 0.6 0.8 1.0
comunidad deportes internacional nacionales negocios opinion politica sucesos variedades

En este gráfico podemos ver cómo se dividen con respecto a las secciones que ven y al porcentaje que estos representan dentro del conjunto total de las transacciones.

Recomendando un Nuevo Artículo

Ahora recomendaremos un nuevo artículo a un nuevo usuario, para ello aplicaremos el algorito **apriori** para las dos transacciones generadas previamente.

Con Artículos

```
rules1 = apriori(transactions,
                 parameter = list(supp = 0.00004, conf = 0.7, target = "rules"))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen maxlen
##         0.7    0.1    1 none FALSE            TRUE   4e-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: 4
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[81 item(s), 124701 transaction(s)] done [0.03s].
## sorting and recoding items ... [81 item(s)] done [0.01s].
## creating transaction tree ... done [0.12s].
## checking subsets of size 1 2 3 4 5 6 7 done [0.04s].
## writing ... [82 rule(s)] done [0.00s].
## creating S4 object  ... done [0.03s].
```

Acá tenemos las reglas generadas con uno *soporte mínimo* de *0.00004*, lo que corresponde a aproximadamente *6 ocurrencias* de la regla y una *confianza mínima* de *0.7*, esto debido a que queremos reglas con un soporte relativamente pequeño y confianza alta.

Ahora bien podemos ver las reglas generadas haciendo uso del comando:

```
inspect(rules1)
```

Sin Artículos

```
rules2 = apriori(transactionsAux,
                 parameter = list(supp = 0.00002, conf = 0.4, target = "rules"))
```

```
## Apriori
##
## Parameter specification:
```

```
##  confidence minval smax arem  aval originalSupport support minlen maxlen
##         0.4    0.1    1 none FALSE            TRUE   2e-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: 2
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[9 item(s), 124701 transaction(s)] done [0.02s].
## sorting and recoding items ... [9 item(s)] done [0.00s].
## creating transaction tree ... done [0.08s].
## checking subsets of size 1 2 3 4 5 6 done [0.00s].
## writing ... [71 rule(s)] done [0.00s].
## creating S4 object  ... done [0.02s].
```

Acá elegimos un soporte una menor confianza debido a que que este fue el número que provee un conjunto considerable de reglas y que tienen sentido debido a que ahora las dimensiones del conjunto se redujo considerablemente de 81 columnas a 9.

Podemos revisar las reglas generadad haciendo uso del comando:

```
inspect(rules2)
```

Una vez que tenemos esto, el enfoque es el siguente:

Tenemos dos niveles de recomendación, uno **con artículos** donde se procesa la transacción de la forma tradicional **clase/articulo** y otro nivel donde se procesa la transacción de la forma **clase**, la idea es que si no se consigue una regla para la forma específica con artículo, entonces pasa al segundo nivel donde enntonces recomendará por sección exclusivamente.

```
recomendation = function(a)
{
  a2 = unlist(lapply(a, removeArticle))
  a2 = unique(a2)

  if(length(subset(rules1, lhs %ain% a)) != 0)
  {
    return(inspect(rhs(subset(rules1, lhs %ain% a)[1])))
  }else if(length(subset(rules2, lhs %ain% a2)) != 0)
  {
    return(inspect(rhs(subset(rules2, lhs %ain% a2)[1])))
  }
}
```

Si tenemos una nueva transacción del tipo:

```
a = c("deportes/articulo1", "deportes/articulo8")
```

Entonces le aplicamos la función **recomendation**

```
recomendation(a)
```

```
##   items
## 1 {deportes/articulo2}
```

Y esto nos da la regla de la forma específica o global por clases, se invita a revisar el archivo **script.r** que está en el directorio **Source** para crear nuevas transacciones y probar.

Visitas con Mayor y Menor Duración

Basándonos en nuestra variable **diff** que contiene la resta de todos los tiempos de entrada y de salida, le quitamos las entradas bots:

```
diff = diff[-bots]
length(diff)
```

```
## [1] 124701
```

Podemos observar cómo tenemos la misma cantidad de filas que en nuestro dataframe **df_periodico** sin bots. Ahora lo que haremos es ordenar el vector **diff** de manera ascendente y descendente para conocer las transacciones más largas y cortas.

Menor Duración

```
as.character(df_periodico$ID[order(diff)][1:10])
```

```
##  [1] "trans2144"  "trans14600" "trans26914" "trans30571" "trans34895"
##  [6] "trans39574" "trans50391" "trans60586" "trans63057" "trans77010"
```

Mayor Duración

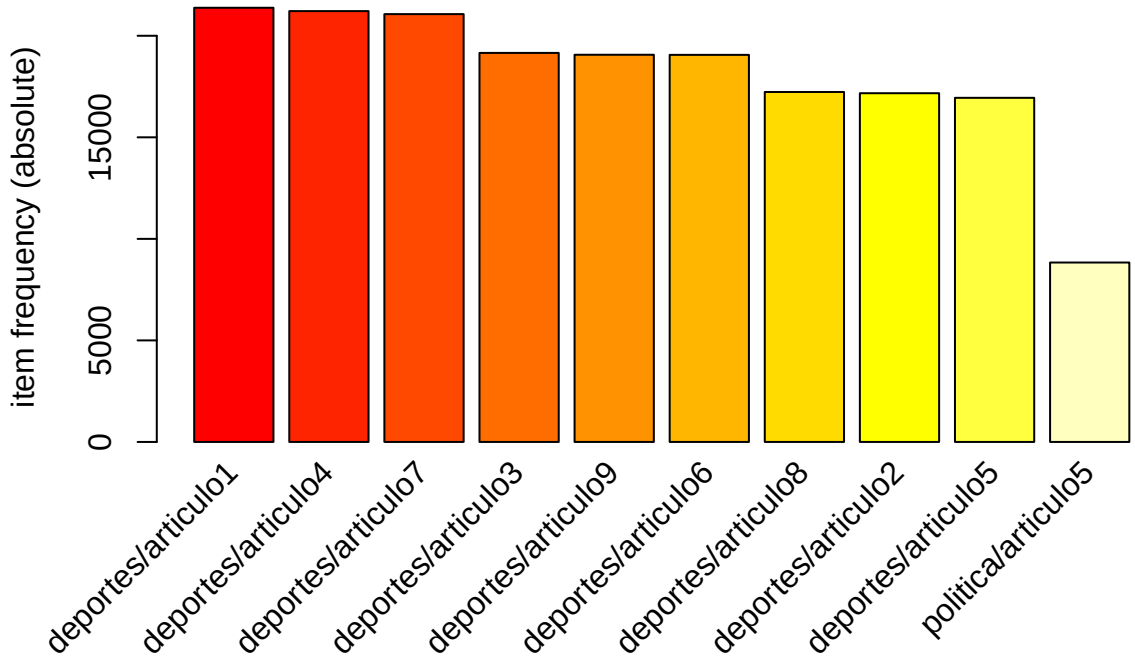
```
as.character(df_periodico$ID[order(diff, decreasing = TRUE)][1:10])
```

```
##  [1] "trans93676" "trans7511"  "trans80516" "trans4579"  "trans7341"
##  [6] "trans66995" "trans1653"  "trans23099" "trans55628" "trans48007"
```

Transacciones con Mayor Número de Apariciones

Ahora bien podemos ver la cantidad de ocurrencias de cada transacción y ordenarla para ver las 10 transacciones más frecuentes:

```
itemFrequencyPlot(transactions, topN = 10, type = "absolute", col = heat.colors(10))
```



Generador de Curvas ROC

La función para la generación de **Curvas ROC** es la siguiente:

```
generate_ROC = function(scores, real, target)
{
  scores = as.numeric(scores)
  newOrder = order(scores, decreasing = TRUE)
  scores = scores[newOrder]
  real = real[newOrder]
  returnTP = vector(mode = "numeric")
  returnFP = vector(mode = "numeric")
  scorePrev = Inf
  FP = 0
  TP = 0
  i = 1
  P = length(real[real == target])
  N = length(real) - P
  index = 1

  while (i <= length(scores))
  {
    if(scores[i] != scorePrev)
    {
      returnTP[index] = TP/P
      returnFP[index] = FP/N
      scorePrev = scores[i]
      index = index +1
    }

    if(real[i] == target)
    {
```

```
      TP = TP + 1
    }else
    {
      FP = FP  +1
    }
    i = i+1
  }

  returnTP[length(returnTP)+1] = TP/P
  returnFP[length(returnFP)+1] = FP/N

  plot(returnFP, returnTP, type = "b", main = "ROC Curve",
       xlab = "FP-Rate", ylab = "TP-Rate", col = "green")
  abline(0,1, col = "blue")
  lines(returnFP,returnTP, col = 1)
  points(returnFP, returnTP, col = 2, pch = 19)
}
```

Esta toma en cuenta los casos simples de sólo 2 clases y los casos de más de dos clase utilizando el enfoque **one vs all**, vamos a probarla con el caso de prueba que mandó el profesor Crema.

```
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
generate_ROC(scores, y, target)
```