Tarea #4

Andrea Tellería

May 17, 2016

Tarea #4

Un periódico tiene sospechas de que existen bots que están ganando dinero al hacer clicks en artículos con promociones. Es por esto, que se ha pedido a los estudiantes de Minería de Datos, de la UCV del semestre II-2015 que realicen un análisis exploratorio sobre las transacciones a forma de determinar el número de posibles transacciones bot que tiene el periódico dado un set de datos.

1. Carga de Paquetes y Datos

```
######################################################################
# Minería de Datos
# Andrea Telleria - CI: 20.614.114
# Asignacion #4
######################################################################

######################################################################
# Instalación de Paquetes necesarios
######################################################################
# Creamos la función que recibe los paquetes
install = function(pkg){
  #Si ya está instalado, no lo instala.
  if (!require(pkg, character.only = TRUE)) {
    install.packages(pkg)
    if (!require(pkg, character.only = TRUE)) stop(paste("load failure:", pkg))
  }
}
install("foreach")
```

```
## Loading required package: foreach
```

```
## Warning: package 'foreach' was built under R version 3.2.3
```

```
# Seleccionamos los archivos que queremos instalar
archive = c("arules", "arulesViz", "rmarkdown", "ggplot2")
foreach(i = archive) %do% install(i)
```

```
## Loading required package: arules
```

```
## Warning: package 'arules' was built under R version 3.2.5
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'arules'

## The following objects are masked from 'package:base':
##
##     abbreviate, write

## Loading required package: arulesViz

## Warning: package 'arulesViz' was built under R version 3.2.5

## Loading required package: grid

## Loading required package: rmarkdown

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 3.2.3

## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## NULL
```

Primero que nada se procede a cargar los paquetes necesarios ara poder proceder con el estudio, para eso se hace uso de una función, de forma que se maneje la instación (de ser necesaria) de los paquetes.

```
######################################################################
# Entrada de Datos
######################################################################
periodico <- read.csv("periodico.csv")
```

Una vez se tienen los paquetes cargados, se procede a cargar la data como tal. Para esto se hace uso de a función que provee **R**, *read.csv*como pudo verse en el fragmento de código anterior.

2. Anális sobre el data Set

2.1 Bots

```
#####################################################################
# Bots
#####################################################################
# Calculando diferencia entre tiempos
entry <- as.POSIXct(periodico$entry, format='%Y-%m-%d %H:%M:%S')
exit <- as.POSIXct(periodico$exit, format='%Y-%m-%d %H:%M:%S')
diferencia <- mapply('-', exit, entry, SIMPLIFY=T)
diferencia <- as.data.frame(diferencia)
diferencia$id <- periodico$X

# Cuales son las transacciones tipo bots
indx.bot <- diferencia$id[diferencia$diferencia<=0.20]
periodico.bots <- periodico$X[periodico$X %in% indx.bot]

# Data set sin bots
periodico.v2 <- periodico[!(periodico$X %in% periodico.bots),]
```

El periódico mostró su preocupación debido a transacciones de tipo bots (transacciones con las que e gana dinero, haciendo clicks en artículos con promociones). Para descubrir si una transacción es o no, bots. Para saber esto primero se pasó a un tipo fecha ambas *timestamps*, tanto el de entrada como el de salida, tras lo cual se procedió a crear un indice con las transacciones cuya diferencia era menor a los 20 segundos, medida indicada por el periódico.

Una vez se tuvo el indice de bots, se procedió a eliminar aquellas transacciones consideradas bots, guardándose el subconjunto del data set cuyas transacciones no eran de este estilo, en una pariable aparte. En nuestro caso *periodico.v2*.

2.1 Columna de Artículos y Contenidos

El cliente solicitó el modificar el data set de tal forma que la columna de itemN, cambiase a forma de reflejar el tipo de contenido del item. El formato fue: contenido/articuloN.

Ejemplo: {item1, item10, item81} es la transacción {deportes/articulo1, politica/articulo1, opinion/articulo9}.

```
#####################################################################
# Creando la columna articulos
#####################################################################
# Separar transacciones
sep.items = function (str){
  str <- gsub("[^,[:digit:]]", "", str)
  str <- strsplit(str, ",")
  str <- unlist(str)
  str <- str[str!= ""]
  str <- sapply(as.integer(str), def.contenido)
  str <- paste(str, collapse=',')
  return (str)
}

# Escogiendo el nombre del contenido y concatenando el número del articulo
def.contenido = function(numero){
  if(numero < 10)
    return(paste("deportes/articulo", numero, sep=""))
  else if(numero < 20)
```

```
    return(paste("politica/articulo", numero-(9*1), sep=""))
  else if(numero < 30)
    return(paste("variedades/articulo", numero-(9*2), sep=""))
  else if(numero < 40)
    return(paste("internacional/articulo", numero-(9*3), sep=""))
  else if(numero < 50)
    return(paste("nacionales/articulo", numero-(9*4), sep=""))
  else if(numero < 60)
    return(paste("sucesos/articulo", numero-(9*5), sep=""))
  else if(numero < 70)
    return(paste("comunidad/articulo", numero-(9*6), sep=""))
  else if(numero < 80)
    return(paste("negocios/articulo", numero-(9*7), sep=""))
  else
    return(paste("opinion/articulo", numero-(9*8), sep=""))
}

# Creando columna de articulos
articulos <- sapply(periodico$articles, sep.items, USE.NAMES = FALSE)
periodico$articulos <- articulos
```

Para realizar esto se contaron con diversas funciones. La primera *sep.items*, separa los items a forma de tener cada itemN por separado y no como un largo string. Una vez se separaron los items, se envian a una nueva función la cual dependiendo del número añade el contenido en el formato indicado preciamente por el cliente. Para hacer uso de ambas funciones se hace conante uso de la función *sapply*, ya que la manera de las funciones de trabajar es mediante elementos separados del set de datos y no con la lista de todos los articulos al mismo tiempo.

Una vez se formó la nueva cadena, se retornó y guardó en una variable que posteriormente se agregó al data frame con las transacciones del periódico.

2.3 Recomendación

Se desea dado un nuevo usuario, que haya ingresado a N artículos, poder recomendarle un artículo n+1 y así aumentar el compromiso del cliente con su portal web.

```
######################################################################
# Transacciones como objeto
######################################################################
# Separando transaccione
sep.items = function (str){
  str <- gsub("[^,[:alnum:]]", "", str)
  str <- strsplit(str, ",")
  str <- unlist(str)
  str <- str[str!= ""]
  return (str)
}

# Separando las transacciones
periodico.items <- periodico$articles
periodico.items <- sapply(periodico.items, sep.items, USE.NAMES = F)
periodico.items <- unlist(periodico.items)
```

```r
# Función que calcula IDs de las transacciones
indx.tot = function(str){
  str <- gsub("[^,[:alnum:]]", "", str)
  str <- strsplit(str, ",")
  str <- unlist(str)
  str <- str[str!= ""]
  return (length(str))
}

# Generando dataframe con items de transacciones separadas
idx_tot <- sapply(periodico$articles, indx.tot, USE.NAMES = F)
transactions.df <- periodico[rep(row.names(periodico), as.numeric(idx_tot)),]
transactions.df$articles <- periodico.items
transactions.df <- transactions.df[c("X", "articles")]

# Generando objeto transactions
transactions <- as(split(as.vector(transactions.df$articles),
                         as.vector(transactions.df$X)),
                   "transactions")
```

Para realizar esto se aplicó reglas de asociación mediante el paquete *arules*. Antes de poder realizar esto, se tuvo que pasar los datos a un formato tal que pudiese ser leído por *arules*, este tipo de datos se llama *transactions* y para que cada instancia de nuestro data set fuese leída como una *transaction* fue necesario un ligero preprocesamiento.

Como al hacer uso de la función *as* no existían opciones para una entrada en forma de canasta (varios items en la misma transacción), se tuvieron que separar los items, manteniendo el número de la transacción a forma de poder aplicar la función *as* para generar el objeto tipo *transactions*.

```r
######################################################################
# Recomendaciones
######################################################################
# Generando las reglas de asociación
rules = apriori(data = transactions, parameter = list(supp=0.00003, 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   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), 131300 transaction(s)] done [0.04s].
## sorting and recoding items ... [81 item(s)] done [0.01s].
## creating transaction tree ... done [0.11s].
```

```
## checking subsets of size 1 2 3 4 5 6 7 done [0.06s].
## writing ... [326 rule(s)] done [0.00s].
## creating S4 object  ... done [0.02s].
```

```
# Cantidad de transacciones va a tener el antecedente de las reglas
0.00003*nrow(periodico)
```

```
## [1] 3.939
```

Una vez se tuvo el objeto se procedió a aplicar *arules*. Cabe destacar que debido a la cantidad de instancias con las que se contaban para alcanzar un número aceptable de reglas se debió bajar el soporte, como la última línea del frágmento de código anterior lo índica, el número de transacciones con el antecedente de las reflas fue de un apróximado de cuatro (4).

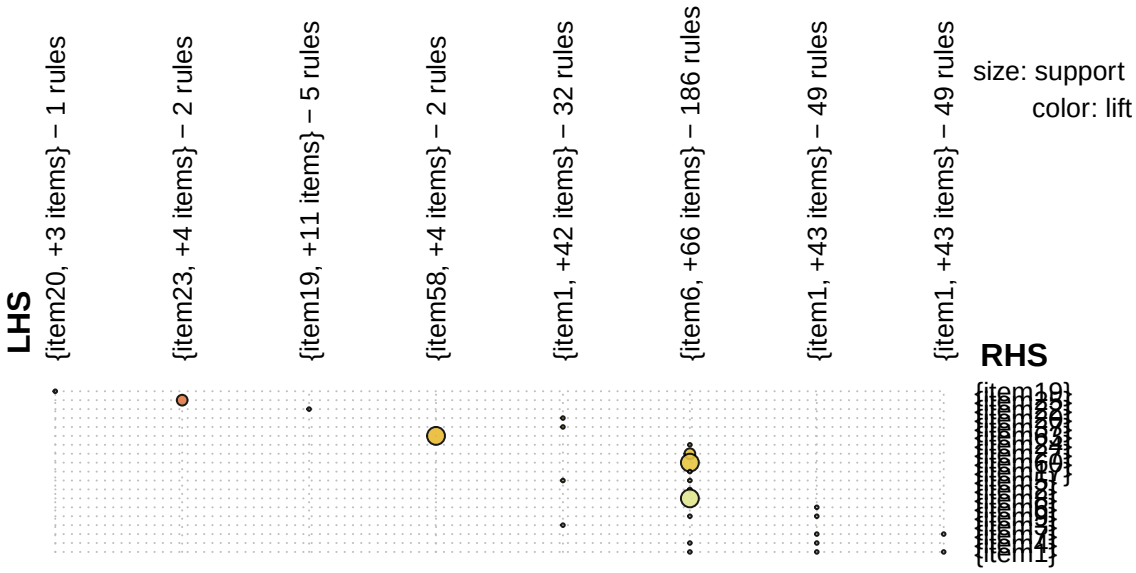
Si se reducía a más de esto el conjunto de reglas iba a ser vacío, al mismo tiempo reducir más el soporte no parecía a mejor idea, ya que la recomendación no tendría la misma seguridad.

Como confianza se decidió por un 70%, si se buscaba más confianza la cantidad de reglas podía llegar a disminuir muy considerablemente, por lo cual después de diversas pruebas se escogieron estos como valores para los parámetros de *arules*.

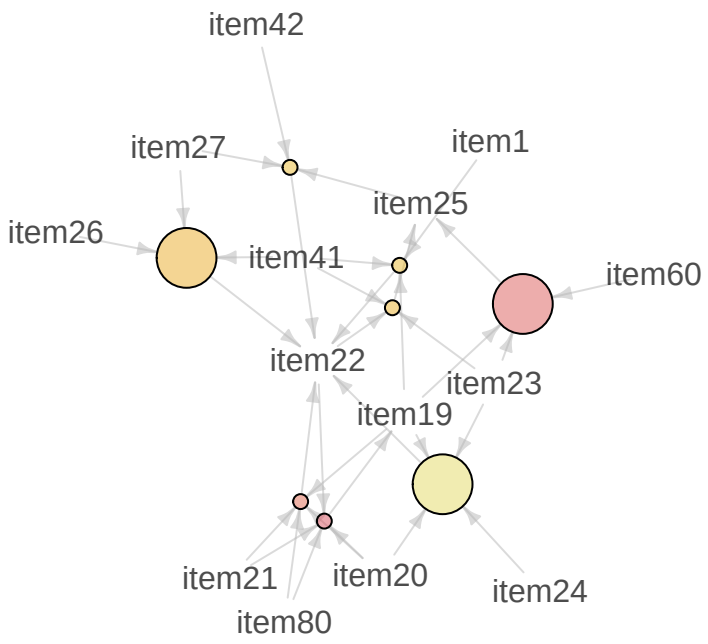
2.4 Tipos de Usuario

Conocer los tipos de usuarios que ingresan a su página (ellos creen que son 8 tipos de usuarios) y tratar de determinar la proporción de cada tipo de usuario.

```
######################################################################
# Tipos de Usuarios
######################################################################
# Graficando agrupaciones de reglas
plot(rules, method="grouped", control=list(k=8))
```

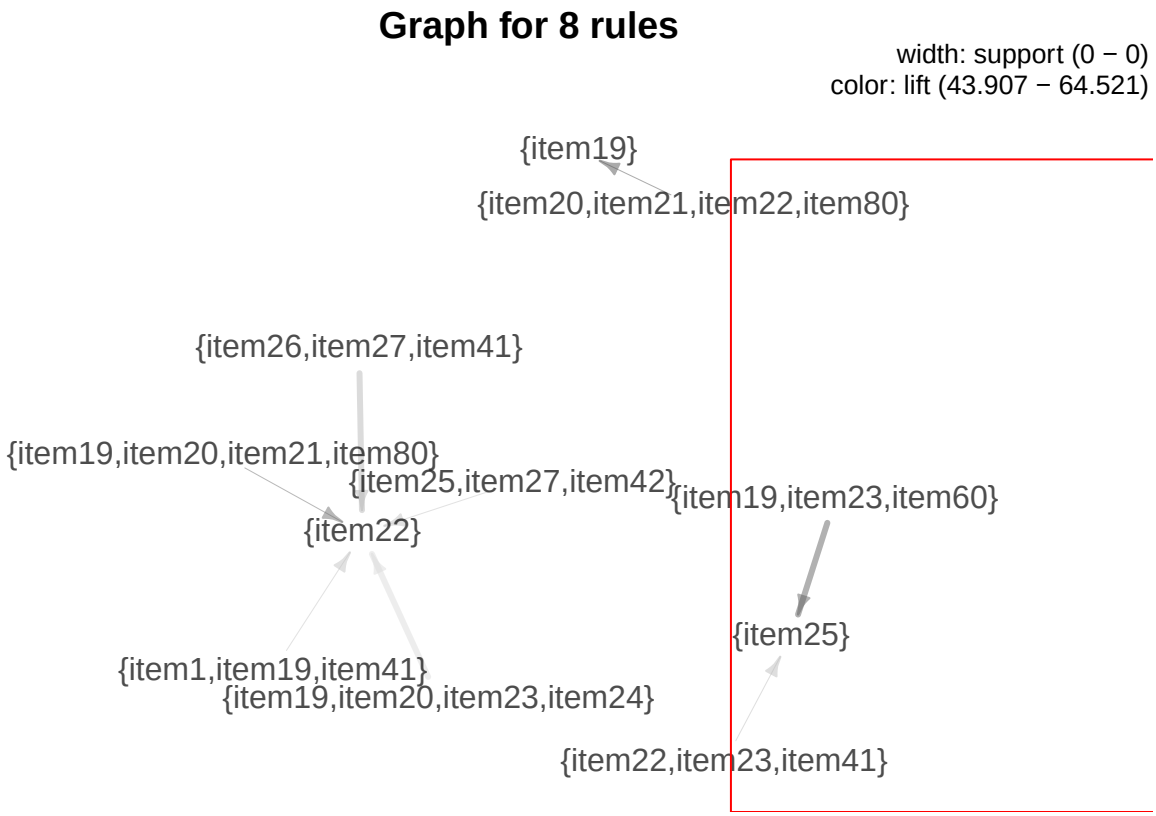



```
# Grafos sobre las asociaciones entre reglas
subrules <- head(sort(rules, by="lift"), 8)
plot(subrules, method="graph")
```




```
plot(subrules, method="graph", control=list(type="itemsets"))

# CLustering Jerárquico de las reglas
d_affinity <- dissimilarity(rules, method = "affinity")
tree <- hclust(d_affinity)
rect.hclust(tree, border="red", k=8)
```



En un comienzo se pensó en la aplicación de algún algoritmo de clustering para atacar esata solicitud del cliente, sin emabargo, debido a la gran cantidad de datos, así como las dificultades de hardware del estudiante, resultó imposible la aplicación de un algritmo de clustering ya fuese usando una matriz de disisimilitud sobre los strings de los items, o usando una distancia de otro tipo.

Así bien se procedieron a usar en vez de las transacciones como tal, las reglas generadas, usando el paquete de *arulesViz* se visualizarn los grupos a los que las reglas pertenecian, indicando siempre la cantidad de estos grupos como los ocho que se sospechan por el cliente.

Se dibujaron además una serie de grafos donde se puede ver la manera en la que los items interactuan y otra en la que son los conjuntos de items los que lo hacen.

FInalmente se aplicó un cluster jerárquico sobre las reglas usando como metodo para la matriz de disimilaridad el *affinity*, el cual nos ndica la afinidad entre reglas.

2.5 Tiempos

El cliente solicitó conocer las 10 visitas con mayor tiempo de estadía en la página y las 10 visitas con menor tiempo de estadía en la página.

```
#####################################################################
# Tiempos
#####################################################################
# Diferencias sin bots
diferencia <- diferencia[!(diferencia$id %in% periodico.bots),]
diferencia <- diferencia[order(diferencia$diferencia, decreasing = T), ]
```

```r
# Mayor tiempo en minutos
head(diferencia, 10)
```

```
##        diferencia     id
## 44837    59.98333  44837
## 22563    59.96667  22563
## 24983    59.96667  24983
## 30834    59.96667  30834
## 51344    59.96667  51344
## 58744    59.96667  58744
## 71471    59.96667  71471
## 91206    59.96667  91206
## 92399    59.96667  92399
## 112279   59.96667 112279
```

```r
# Menor tiempo en minutos
tail(diferencia, 10)
```

```
##        diferencia     id
## 124120          1 124120
## 124186          1 124186
## 125022          1 125022
## 125107          1 125107
## 126497          1 126497
## 126880          1 126880
## 127724          1 127724
## 128785          1 128785
## 130307          1 130307
## 130720          1 130720
```

Para realizar esto se hizo uso de la variable con las diferencias en los tiempos de entrada y salida usada durante el análisis de los bots. Una vez se tuvo dicha variable, se eliminaron las transacciones bots. Luego de lo cual se orderon de forma decreciente.

Con ayuda de las funciones *head* y *tail* se buscaron las transacciones con mayor diferencia y las de menor diferencia.

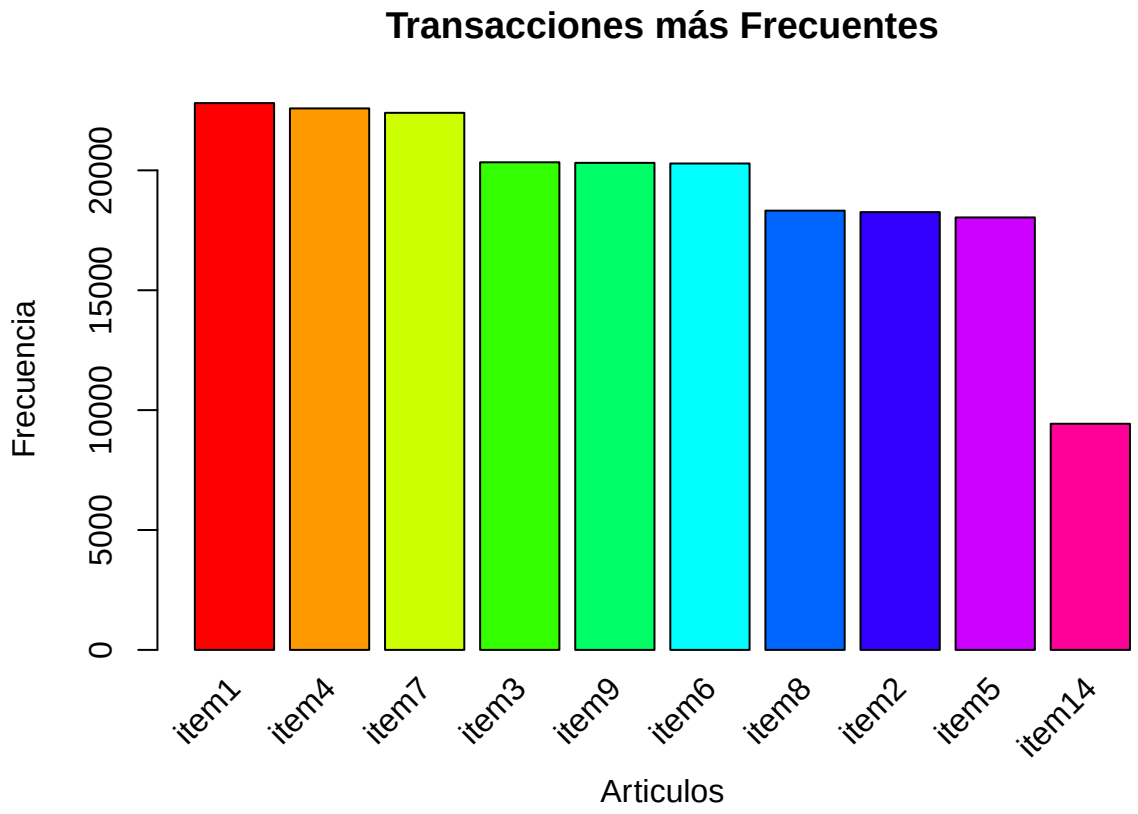
2.6 Apariciones

Similar a la solicitud anterior, el cliente en este caso pidió conocer las 10 transacciones con mayor número de apariciones en el data set.

```r
######################################################################
# Apariciones, frecuencias
######################################################################
# 10 lugares visitados con mayor número de apariciones
head(sort(itemFrequency(transactions, type="absolute"), decreasing = TRUE), n = 10)
```

```
##  item1  item4  item7  item3  item9  item6  item8  item2  item5 item14
##  22805  22584  22400  20337  20313  20286  18320  18259  18036   9431
```

```r
# Gráfica de la frecuencia de los lugares visitados
itemFrequencyPlot(transactions, type = "absolute", topN = 10, col=rainbow(10),
                  main = "Transacciones más Frecuentes", ylab = "Frecuencia",
                  xlab = "Articulos")
```

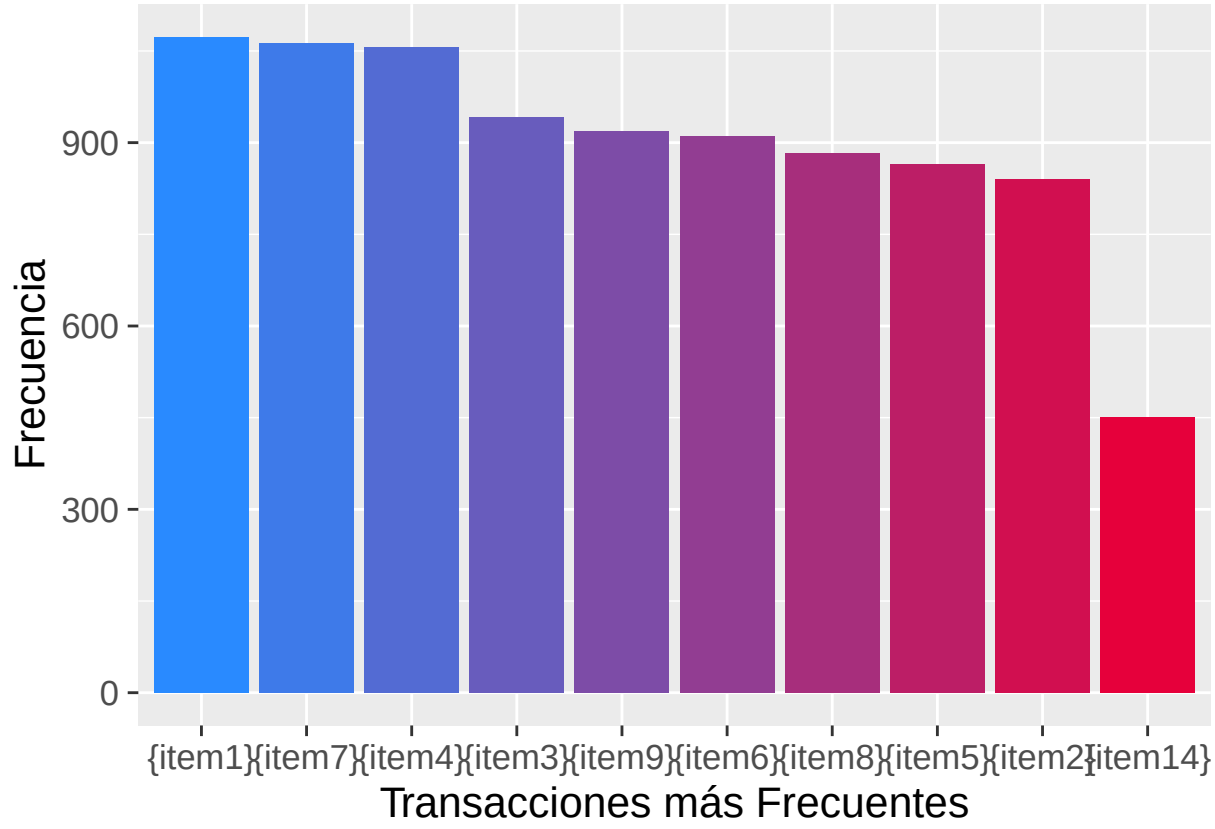



```r
# 10 transacciones con mayor número de aparición
datos <- (sort(tail(sort(table(periodico.v2$articles)), 10), decreasing = T))
datos <- data.frame(Items = names(datos), Count = as.vector(datos), row.names = seq_len(10))
datos
```

```
##       Items Count
## 1   {item1}  1073
## 2   {item7}  1063
## 3   {item4}  1057
## 4   {item3}   942
## 5   {item9}   919
## 6   {item6}   910
## 7   {item8}   883
## 8   {item5}   865
## 9   {item2}   841
## 10 {item14}   450
```

```r
# Gráfica de la frecuencia de las 10 transacciones más populares
ggplot(datos, aes(x=reorder(Items, -Count, order = T), y=Count,
                  fill = reorder(Items, -Count, order = T))) + geom_bar(stat="identity") +
```

```
                scale_fill_manual(values=c("#298aff", "#3e7ae9", "#536bd3",
                                           "#685cbd", "#7d4ca7", "#923d92",
                                           "#a72e7c", "#bc1e66", "#d10f50",
                                           "#e6003b"), guide = F) +
                labs(x="Transacciones más Frecuentes", y="Frecuencia") +
                theme_grey(base_size = 16)
```



Como no se está claro sobre si es sobre cantidad de elementos o el conjunto de artículos de la transacción se realizaron dos gráficas, la primera hace uso de la fnción *itemFrequency* y busca la frecuencia de los elementos en las diferentes transacciones. Ayudando a ser más didáctico los resultados se hizo uso de una gráfica para mostrar los resultados de esa forma también, para esto, se hizo uso de la función *itemFrequencyPlot*.

Posteriormente se hizo la busqueda directo en el data set con las transacciones completas de tal manera de encontrar las más repetidas. A esta variable se la llamó *datos*. Se hizo primero una tabla la cuál fue luego convertida en un objeto de tipo data frame, esto con el fin de facilitar la graficación, la cual fue hecha mediante el paquete *ggplot2*.

3. ROC

3.1 Implementación

se le pide que implemente un graficador de curvas ROC para clasificadores cuya salida sea un score. De esta manera, los parámetros de graficador serán:

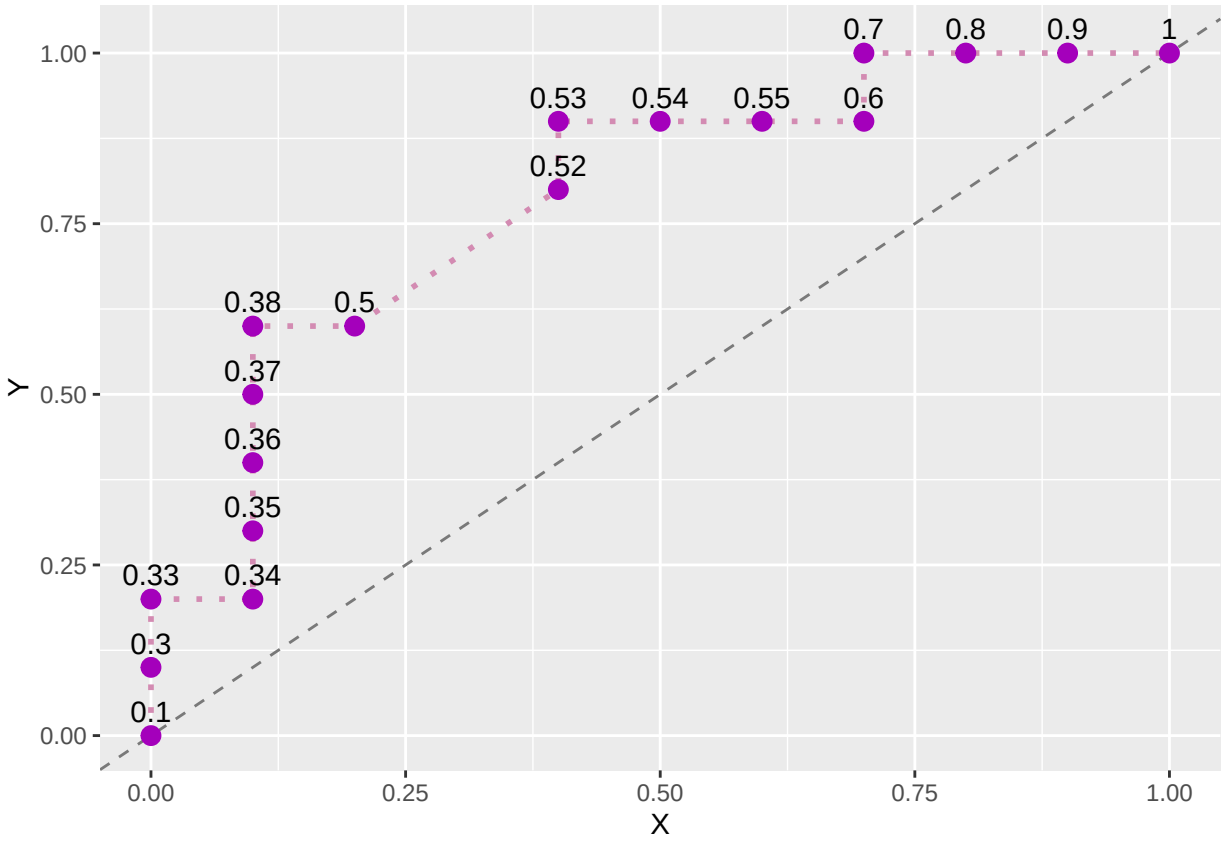
1. Los scores por instancia (no necesariamente ordenados).

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

```
#####################################################################
# ROC
#####################################################################
# Función generadora de Curva ROC
generate_ROC = function(y, scores, target) {
  scores <- scores[order(scores, decreasing = T)]
  y <- y[order(scores, decreasing = T)]
  fp <- tp <- 0
  xy = rbind(rep(0, length(y)+1), rep(0, length(y)+1))
  prev <- -1
  p <- length(y[y==1])
  n <- length(y[y==2])
  for(i in seq_len(length(y))) {
    if (scores[i] != prev) {
      xy[,i] <- c((fp/n),(tp/p))
      prev <- scores[i]
    }else{
      xy[,i] <- c(-1,-1)
    }
    if (y[i] == target)
      tp <- tp + 1
    else
      fp <- fp + 1
  }
  xy[,length(y)+1] <- c((fp/n),(tp/p))
  scores <- scores[order(scores)]
  scores[length(scores)+1] <- 1
  xy <- data.frame(X = xy[1,], Y = xy[2,])
  xy$scores <- scores
  xy <- xy[!(xy$X==-1 & xy$Y==-1),]
  return(ggplot(xy, aes(x=X, y=Y, label=scores)) +
           geom_line(linetype="dotted", size=1, colour="#D38BB2") +
           geom_abline(intercept = 0, slope = 1, linetype="dashed", colour="#797979") +
           geom_point(size=3, colour="#A400BB") +
           geom_text(vjust = 0, nudge_y = 0.02))

  }

# Probando la función generadora de curvas ROC
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(y, scores, target)
```



Para realizar la función se hizo uso del documento facilitado por el grupo de la materia, de esta se consiguió crear una implementación de la curva ROC en el lenguaje R, para la gráfica se hizo uso del paquete *ggplot2* el cual facilitaba el trabajo de crear la gráfica, además de mejorar la estética de la misma.

Recordando que los *scores* pueden no estar ordenados se rocordó el ordenar los elementos acorde a esto al comienzo de la función, asumiendo que los indices de y tenían una relación directa con los de *scores* se ordenaron también lo de y con ayuda del indice de *scores*.

Para tratar con *scores* idénticos se marcaron estas filas a manera de eliminarlas antes del mometno de la graficación para que no afectaran la gráfica.

Como prueba, para mostar la funcionalidad de la función se realizó con un ejemplo, en el cual se puede ver el resultado obtenido.