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, respos = "http://cran.rstudio.com/"))
      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 ka 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)
  }
}
```

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.

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

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)
```

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

Para esto, debemos agarrar nuestra columna **articles** de nuestro dataframe **periodico.csv** y crear una matriz de transacciones.

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

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

```
sort(itemFrequency(transactions, type = "absolute"), decreasing = TRUE)[1:10]
```

```
## deportes/articulo1 deportes/articulo4 deportes/articulo7
##              21379              21214              21066
## deportes/articulo3 deportes/articulo9 deportes/articulo6
##              19156              19065              19060
## deportes/articulo8 deportes/articulo2 deportes/articulo5
##              17232              17172              16945
## politica/articulo5
##               8834
```

Generador de Curvas ROC

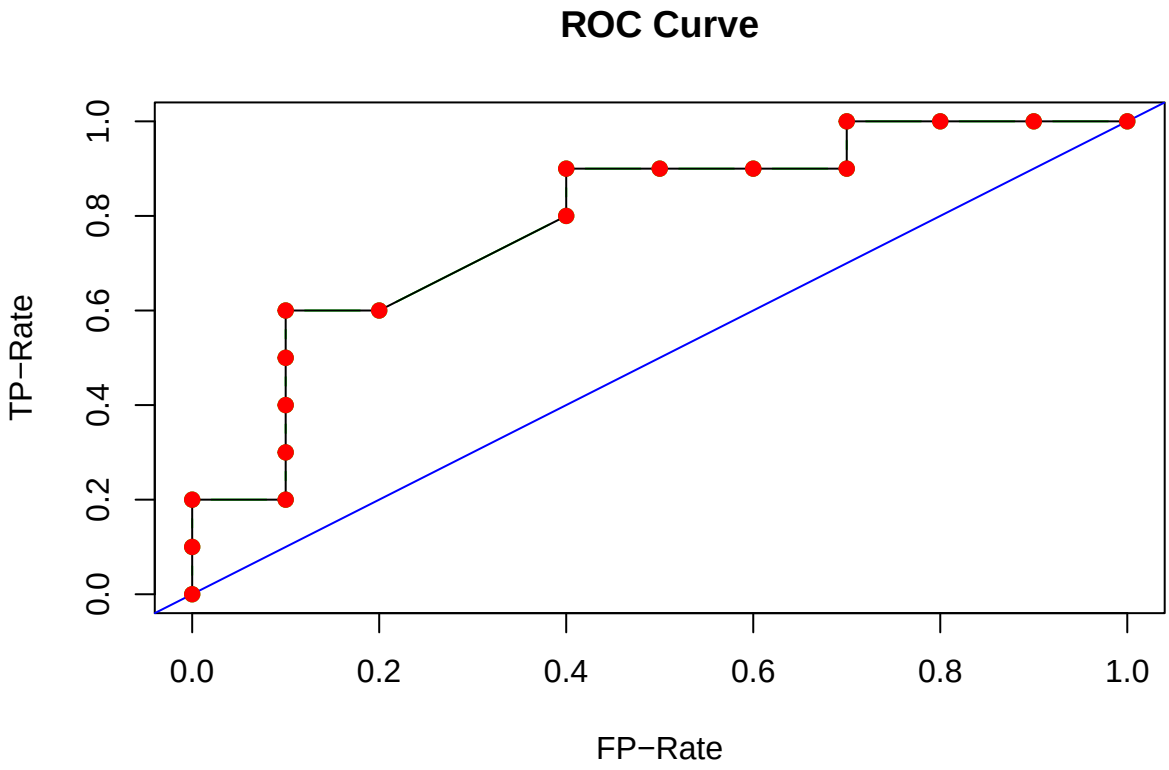
La función para la generación de **Curvas ROC** es la siguiente:

```
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
target = 2
generate_ROC(scores, y, 2)
```


ROC Curve
TP−Rate
FP−Rate
0.0
0.2
0.4
0.6
0.8
1.0