Tratamiento Inteligente de datos

(TID)

Prácticas de la asignatura

2018-2019

En colaboración con:

Participantes

Alejandro Campoy Nieves: alejandroac79@correo.ugr.es

Gema Correa Fernández: gecorrea@correo.ugr.es

Luis Gallego Quero: lgaq94@correo.ugr.es

Jonathan Martín Valera: jmv742@correo.ugr.es

Andrea Morales Garzón: andreamgmg@correo.ugr.es

Índice

Índice de figuras

Índice de cuadros

```
library(magrittr)
library(plyr)
library(RColorBrewer)
library(ggplot2)
library(NLP) # Paquete para minería de datos, permite procesar datos de tipo texto
```

```
##
## Attaching package: 'NLP'
```

```
## The following object is masked from 'package:ggplot2':
##
##     annotate
```

```
library(tm)
library(SnowballC) # Paquete para minería de datos, agrupa aquellos términos que contienen la misma raí
library(wesanderson)
library(devtools)
library(rword2vec) # install_github("mukul13/rword2vec")
library(rlist)
library(quanteda)
```

```
## Package version: 1.3.14
```

```
## Parallel computing: 2 of 4 threads used.
```

```
## See https://quanteda.io for tutorials and examples.
```

```
##
## Attaching package: 'quanteda'
```

```
## The following objects are masked from 'package:tm':
##
##     as.DocumentTermMatrix, stopwords
```

```
## The following object is masked from 'package:utils':
##
##     View
```

```
library(tseries)
library(wordcloud)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:plyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ROCR)
```

```
## Loading required package: gplots

##
## Attaching package: 'gplots'

## The following object is masked from 'package:wordcloud':
##
##     textplot

## The following object is masked from 'package:stats':
##
##     lowess
```

Descripción de los paquetes necesarios

A continuación, se describen los paquetes necesarios para el desarollo del proyecto:

- `tm` : Paquete específico para minería de datos, permite procesar datos de tipo texto. Se puede instalar usando : *install.packages("tm").*
- `SnowballC` : Paquete adicional para minería de datos, implementa un algoritmo que permite reducir el número de términos con lo que trabajar, es decir, agrupa aquellos términos que contienen la misma raíz. El paquete soporta los siguientes idiomas: alemán, danés, español, finlandés, francés, húngaro, inglés, italiano, noruego, portugués, rumano, ruso, sueco y turco. Se puede instalar usando : *install.packages("SnowballC").*
- `wordcloud` : Paquete para crear gráficas de nubes de palabras, permitiendo visualizar las diferencias y similitudes entre documentos. Se puede instalar usando : *install.packages("wordcloud").*
- `arules` : Paquete que proporciona la infraestructura para representar, manipular y analizar datos y patrones de transacción (conjuntos de elementos frecuentes y reglas de asociación). Se puede instalar usando : *install.packages("arules").*
- `arulesViz` : Paquete que extiende el paquete 'arules' con varias técnicas de visualización para reglas de asociación y conjuntos de elementos. El paquete también incluye varias visualizaciones interactivas para la exploración de reglas. Se puede instalar usando : *install.packages("arulesViz").*
- `devtools` : Paquete que contiene una colección de herramientas de desarrollo de paquetes, usando conjuntamento con `rword2vec`, para obtener la agrupación de sinónimos. Se puede instalar usando : *install.packages("devtools").*
- `rword2vec` : Paquete que toma un corpus de texto como entrada y produce los vectores de palabra como salida, usado especialmente para obtener las distancias que existen entre un término y los términos semejantes en el texto de formación (aprende la representación vectorial de las palabras). Se puede instalar usando : *install_github("mukul13/rword2vec").*
- `magrittr` : Paquete ...
- `plyr` : Paquete ...
- `ggplot2` : Paquete ...
- `ROCR` : Paquete ...
- `RColorBrewer` : Paquete ...
- `NLP` : Paquete ...
- `wesanderson` : Paquete ...
- `devtools` : Paquete ...

- `rlist` : Paquete ...
- `quanteda` : Paquete ...
- `tseries` : Paquete ...
- `wordcloud` : Paquete ...
- `dplyr` : Paquete ...

1. Comprender el problema a resolver

El *dataset* **Drug Review Dataset**, proporcionado por *UCI Machine Learning Repository*, contiene una exhaustiva base de datos de medicamentos específicos, en la cual, el conjunto de datos muestra revisiones de pacientes sobre medicamentos específicos para unas condiciones particulares. Dichas revisiones se encuentran desglosadas en función del tema que se esté tratando: beneficios, efectos secundarios y comentarios generales. De igual modo, se dispone de una calificación de satisfacción general, es decir, de una calificación en base a los efectos secundarios del medicamento y de otra en base a la efectividad del mismo.

En este proyecto nos centraremos en el **análisis y experiencia qué tienen los usuarios con ciertos tipos de medicamentos**, para la realización y aplicación de las técnicas explicadas a lo largo del curso. Para ello, se proponen los siguientes objetivos principales:

- Realizar un análisis de sentimientos a partir de la experiencia de dichos usuarios en el uso de ciertos medicamentos, como por ejemplo ver la efectividad del medicamento cuánto está relacionado con los efectos secundarios o beneficios del mismo.
- Compatibilizar dicho modelo de datos con otros conjuntos de datos aportados en **Drugs.com**.

Las características de este conjunto de datos vienen descritas en la siguiente tabla 1:

Características del Data Set	Multivariable, texto
Características de los atributos	Entero
Tareas asociadas	Clasificación, regresión, clustering
Número de instancias	4143
Número de atributos	8
Valores vacíos	N/A
Área	N/A
Fecha de donación	10/02/2018
Veces visualizado	11047

Cuadro 1: Información del conjunto de datos

Los datos se dividen en un conjunto train (75 %) y otro conjunto test (25 %) y se almacenan en dos archivos *.tsv* (tab-separated-values), respectivamente. Los atributos que tenemos en este dataset son:

1. **urlDrugName** (categorical): nombre del medicamento/fármaco
2. **rating** (numerical): clasificación o puntuación del 1 a 10 del medicamento según el paciente
3. **effectiveness** (categorical): clasificación de la efectividad del medicamento según el paciente (5 posibles valores)
4. **sideEffects** (categorical): clasificación de los efectos secundarios del medicamento según el paciente (5 posibles valores)
5. **condition** (categorical): nombre de la condición (diagnóstico)
6. **benefitsReview** (text): opinión del paciente sobre los beneficios
7. **sideEffectsReview** (text): opinión del paciente sobre los efectos secundarios
8. **commentsReview** (text): comentario general del paciente

2. Preprocesamiento de datos

En este apartado, pondremos los datos a punto para la aplicación de diversas técnicas. Por tanto, para poder analizar dicho *dataset* y realizar el preprocesamiento al mismo, lo primero que se va hacer es leer el conjunto de datos *train* y *test*. Una vez leído el conjunto de datos, se procederán a aplicar las siguientes técnicas con el fin de limpiar los datos:

1. Lectura del dataset

2. Eliminación de columnas que información irrelevante para el análisis
3. Eliminación de filas que no proporcionan información alguna
4. Cuantificación de variables
5. Representación gráfica de los datos
6. Creación del corpus
7. Representación gráfica de las frecuencias
8. Correlación
9. Eliminar signos de puntuación
10. Conversión de las mayúsculas en minúsculas
11. Eliminación de Stopwords
12. Agrupación de sinónimos
13. TF-IDF
14. Stemming
15. Borrar espacios en blanco innecesarios
16. Valores perdidos
17. Term Document Matrix
18. Sparsity
19.
20.
21.
22.
23.

2.1. Lectura del dataset

A continuación, mediante la función `read.table` procedemos a la lectura de los datos.

2.1.1. Lectura de datos train

Se va a procedeer a la lectura del conjunto de datos de entrenamiento.

```
# Lectura de datos train
datos_train <- read.table("datos/drugLibTrain_raw.tsv", sep="\t", comment.char="",
                          quote = "\"", header=TRUE)
```

Disponemos de una matriz de 3107 filas x 9 columnas, asimismo vamos a ver un ejemplo de como distribuida la información, en donde para la fila tercera encontramos la siguiente información:

X	urlDrugName	rating	effectiveness	sideEffects	condition
1146	ponstel	10	Highly Effective	No Side Effects	menstrual cramps

Cuadro 2: Información contenida en una fila del conjunto de entrenamiento I

benefitsReview	sideEffectsReview	commentsReview
I was used to having cramps so badly that they would leave me balled up in bed for at least 2 days. The Ponstel doesn't take the pain away completely, but takes the edge off so much that normal activities were possible. Definitely a miracle medication!!	Heavier bleeding and clotting than normal.	I took 2 pills at the onset of my menstrual cramps and then every 8-12 hours took 1 pill as needed for about 3-4 days until cramps were over. If cramps are bad, make sure to take every 8 hours on the dot because the medication stops working suddenly and unfortunately takes about an hour to an hour and a half to kick back in.. if cramps are only moderate, taking every 12 hours is okay.

Cuadro 3: Información contenida en una fila del conjunto de entrenamiento II

De las tablas 2 y 3 podemos extraer que el medicamento **ponstel** con identificador **1146**, tiene la máxima puntuación por parte del paciente (**rating = 10**), el cual tiene un alto nivel de efectividad (**Highly Effective**) sin efectos secundarios (**No Side Effects**), usado para dolores menstruales (**menstrual cramps**), en donde el paciente dice que de estar tumbado en la cama con dolores ha pasado a poder realizar las actividades sin ningún impedimento. Además, asegura que tomar este medicamento le ha supuesto un sangrado más abundante y coagulación de lo normal. La dosis del medicamento oscila entre una píldora cada 8-12 horas durante 3-4 días.

2.1.2. Lectura de datos test

Se va a procedeer a la lectura del conjunto de datos de prueba.

```
# Lectura de datos test
datos_test <- read.table("./datos/drugLibTest_raw.tsv", sep="\t", comment.char="",
                         quote = "\"", header=TRUE)
```

Disponemos una matriz de 1036 filas x 9 columnas, asimismo vamos a ver un ejemplo de como distribuida la información, en donde para la fila primera encontramos la siguiente información:

De las tablas 4 y 5 podemos extraer que el medicamento **biaxin** con identificador **1366**, tiene una puntuación de 9 por parte del paciente (**rating = 9**), el cual tiene un nive considerable de efectividad (**Considerably Effective**) con efectos secundarios leves (**Mild Side Effects**), usado para la infección sinusal (**sinus infection**), en donde el paciente dice que no está muy seguro de si el antibiótico ha destruido las bacterias que causan su infección sinusal. Además, asegura que tomar este medicamento le da algo de dolor de espalda y algunas náuseas. El paciente tomó los antibióticos durante 14 días y la infección sinusal desapareció al sexto día.

X	urlDrugName	rating	effectiveness	sideEffects	condition
1366	biaxin	9	Considerably Effective	Mild Side Effects	sinus infection

Cuadro 4: Información contenida en una fila del conjunto de prueba I

benefitsReview	sideEffectsReview	commentsReview
The antibiotic may have destroyed bacteria causing my sinus infection. But it may also have been caused by a virus, so its hard to say.	Some back pain, some nauseau.	Took the antibiotics for 14 days. Sinus infection was gone after the 6th day.

Cuadro 5: Información contenida en una fila del conjunto de prueba II

La representación del documento se llevará a cabo utilizando palabras, después de un debido filtrado para minimizar la dimensión del espacio de trabajo.

2.2. Procesamiento de los datos

Dado que la representación total del documento puede tener una alta dimensión, se va a procedeer a construir un corpus, necesario para la aplicación de métodos de limpieza y esructuración del texto de entrada e identificación de un subconjunto simplificado de las características del documento con el fin de poder ser representado en un análisis posterior.

2.2.1. Eliminar columnas

El primer paso que vamos a realizar es la **eliminación de columnas**, las cuales contienen información irrelevante para nuestro análisis.

Eliminar columna ID

Al conjunto de datos utilizado se le ha añadido de forma automática una novena columna, que representa un ID para cada uno de los datos con los que estamos trabajando. Como este ID no nos aporta información alguna, hemos decidido quitarla directamente del *dataframe*. Esta columna se corresponde con la primera columna, por lo cuál, debemos eliminar la columna que se corresponde con la posición 1. Los cambios que hacemos en el *dataset* deben modificarse tanto en el conjunto de test como el de train, para que los resultados sean consistentes.

```
datos_train = datos_train[-1] # Eliminar columna para el ID en el train
datos_test = datos_test[-1] # Eliminar columna para el ID en el test
```

Eliminar columna de commentsReview

Consideramos que la información contenida en *commentsReview* no es de nuestro interés. En este atributo se almacena texto, en el cual los consumidores de los medicamentos suelen poner en la mayoría de casos la frecuencia o la dosis con la que consumen la misma. En otros casos menos frecuentes, se establecen comentarios más arbitrarios en el que se muestran sus sensaciones o información sin relevancia. Incluso en algunos casos este campo aparece vacío. Es por eso, que hemos decidido eliminar la columna, tanto para el conjunto test como el train.

```
datos_train = datos_train[-8] # Eliminar columna para el commentsReview en el train
datos_test = datos_test[-8] # Eliminar columna para el commentsReview en el test
```

2.2.2. Eliminar filas

Además de eliminar las columnas innecesarias, se han localizado tres filas que aportan información irrelevante a nuestro análisis, las cuales estaban perjudicando la realización del preprocesamiento.

- Se elimina la fila 387, porque no dispone de información alguna ni en *benefitsReview* y *sideEffectsReview*.
- Se elimina la fila 928, porque en la columna *condition* dispone de un carácter raro.
- Se elimina la fila 3105, porque no tienen información en *benefitsReview* (“—”).

```
datos_train = datos_train[-c(387,928, 3105),] # Eliminar filas en el train
datos_test = datos_test[-c(387,928, 3105),] # Eliminar filas en el test
```

2.2.3. Eliminar elementos repetidos

No se contabiliza elimar medicamentos repetidos, debido a que hacen faltan las opiniones. Por ejemplo vamos a extraer el medicamentos que más veces se repite en el conjunto test.

```
# duplicated(datos_test$urlDrugName)

#df <- data.frame(datos_test$urlDrugName)
#new_df <- aggregate(datos_test$urlDrugName, df, length)
#colnames(new_df)[2]<-"Repeticiones"
#new <- new_df[order(new_df$Repeticiones, decreasing = TRUE),]

#df %>%
#  group_by(datos_test$urlDrugName) %>%       #Luego indico que quiero un grupo por cada valor unico en
#  tally()
```

```
#new[1,]
```

```
#datos_test[datos_test$urlDrugName == "paxil",]
```

2.2.4. Cuantificación de variables

Para poder analizar y trabajar más fácilmente con la información de *sideEffects* y *effectiveness*, se va a realizar una conversión de dichas columnas a forma cuantitativa, es decir, vamos asignar una etiqueta numérica a cada valor pertinente, tanto para para *train* como *test*.

A continuación, vamos a cuantificar la columna de *sideEffects*, para ello se añade una nueva columna a nuestro conjunto de datos denominada *sideEffectsNumber* que nos clasifica los posibles valores de la columna *sideEffects* en un rango numérico, comprendido entre 1 y 5. Dicha columna hace referencia a la clasificación de los efectos secundarios del medicamento según el paciente, en donde la etiqueta con valor 1 hará referencia a que no haya ningún efecto secundario y la etiqueta con valor 5 a que tiene efectos secundarios extremadamente graves:

- Extremely Severe Side Effects (efectos secundarios extremadamente graves) : 5
- Severe Side Effects (efectos secundarios graves): 4
- Moderate Side Effects (efectos secundarios moderados) : 3
- Mild Side Effects (efectos secundarios leves) : 2
- No Side Effects (sin efectos secundarios) : 1

```
# Datos Train
datos_train$sideEffectsNumber[datos_train$sideEffects=="Extremely Severe Side Effects"]<-5
datos_train$sideEffectsNumber[datos_train$sideEffects=="Severe Side Effects"]<-4
datos_train$sideEffectsNumber[datos_train$sideEffects=="Moderate Side Effects"] <- 3
datos_train$sideEffectsNumber[datos_train$sideEffects=="Mild Side Effects"]<- 2
datos_train$sideEffectsNumber[datos_train$sideEffects=="No Side Effects"]<- 1

# Datos Test
datos_test$sideEffectsNumber[datos_test$sideEffects=="Extremely Severe Side Effects"]<-5
datos_test$sideEffectsNumber[datos_test$sideEffects=="Severe Side Effects"]<-4
datos_test$sideEffectsNumber[datos_test$sideEffects=="Moderate Side Effects"]<-3
datos_test$sideEffectsNumber[datos_test$sideEffects=="Mild Side Effects"]<-2
datos_test$sideEffectsNumber[datos_test$sideEffects=="No Side Effects"]<-1
```

Podemos comprobar que se ha creado la nueva columna *sideEffectsNumber*, y que se han añadido los cambios comentados anteriormente.

```
head(datos_train$sideEffectsNumber, 10)
```

```
##  [1] 2 4 1 2 4 4 2 1 1 5
```

Volvemos a aplicar el mismo procedimiento para la columna de *effectiveness*, creándonos para ello una

columna denominada *effectivenessNumber*. Dicha columna, hace referencia a la clasificación de la efectividad del medicamento según el paciente, en donde la etiqueta con valor 1 hace referencia a que el medicamente es ineficaz y la etiqueta con valor 5 a que el medicamente es altamente eficaz:

- Highly Effective (altamente efectivo): 5
- Considerably Effective (considerablemente efectivo) : 4
- Moderately Effective (moderadamente efectivo) : 3
- Marginally Effective (marginalmente efectivo) : 2
- Ineffective (ineficaz) : 1

```
# Datos de entrenamiento
datos_train$effectivenessNumber[datos_train$effectiveness=="Highly Effective"]<-5
datos_train$effectivenessNumber[datos_train$effectiveness=="Considerably Effective"]<-4
datos_train$effectivenessNumber[datos_train$effectiveness=="Moderately Effective"]<-3
datos_train$effectivenessNumber[datos_train$effectiveness=="Marginally Effective"]<-2
datos_train$effectivenessNumber[datos_train$effectiveness=="Ineffective"]<- 1

# Datos de test
datos_test$effectivenessNumber[datos_test$effectiveness=="Highly Effective"]<-5
datos_test$effectivenessNumber[datos_test$effectiveness=="Considerably Effective"]<-4
datos_test$effectivenessNumber[datos_test$effectiveness=="Moderately Effective"]<-3
datos_test$effectivenessNumber[datos_test$effectiveness=="Marginally Effective"]<-2
datos_test$effectivenessNumber[datos_test$effectiveness=="Ineffective"]<-1
```

Comprobamos que se ha creado la nueva columna *effectivenessNumber*, y que se han añadido los nuevos cambios.

```
head(datos_train$effectivenessNumber, 10)
```

```
##  [1] 5 5 5 2 2 1 5 4 5 1
```

2.2.5. Cálculo de rating ponderado

En esta sección se va a realizar una valoración general de la medicina, teniendo en cuenta los efectos secundarios y efectividad de ésta. Para ello, se va a realizar una ponderación entre estos dos valores, considerando en este caso una ponderación del 70 % para los efectos secundarios y un 30 % para la efectividad de la medicina.

El motivo de esta ponderación es que se considera que es peor tener efectos secundarios severos en una medicina, a que sea más efectiva o no.

Tras haber cuantificado las variables que miden la efectividad y efectos secundarios de una medicina, se ha añadido una nueva columna llamada "weightedRating", cuyo resultado se podrá utilizar como valoración general de la medicina, e incluso poder compararlo con las propias valoraciones de los usuarios.

```
# Ponderaciones
sideEffectstWeight <- 0.7
effectivenessWeight <- 0.3

# Recorremos el dataframe
for (i in 1:length(datos_train[[1]])){

  # Obtenemos el valor de sideEffect
  sideEffectNumber <- datos_train$sideEffectsNumber[i]

  # Obtenemos el valor de efectiveness
  effectivenessRating <- datos_train$effectivenessNumber[i]
```

```
  # Inicializamos la variable
  sideEffectRating <- 0
  # Convertimos el valor de sideEffect a la misma escala que efectiveness, ya que
  # sideEffect == 1 significa que no tiene efectos secundarios (lo cual es bueno),
  # y efecctiveness == 1 significa que no es efectivo (lo cual no es bueno). Para
  # ello realizamos la siguiente conversión:
  if(sideEffectNumber == 1)
     sideEffectRating <- 5
  else if(sideEffectNumber == 2)
    sideEffectRating <- 4
  else if(sideEffectNumber == 3)
    sideEffectRating <- 3
  else if(sideEffectNumber == 4)
    sideEffectRating <- 2
  else if(sideEffectNumber == 5)
    sideEffectRating <- 1
  # Obtenemos el resultado ponderado en tipo float
  floatResult <- effectivenessRating * effectivenessWeight + sideEffectRating * sideEffectstWeight

  # Convertimos el resultado a valor entero.
  integerResult <- as.integer(floatResult)

  # Calculamos la parte decimal y redondeamos
  if(floatResult - integerResult < 0.5)
    result <- integerResult
  else
    result <- integerResult+1

  # Añadimos el resultado obtenido y lo multiplicamos por dos para pasarlo a escala (1-10)
  datos_train$weightedRating[i] <- result * 2
}
```

Comprobamos que se ha creado la nueva columna *weightedRating*, y que se han añadido los nuevos cambios.

```
head(datos_train$weightedRating, 10)
```

```
##  [1]  8  6 10  6  4  4  8 10 10  2
```

Ahora hacemos lo mismo en test:

```
# Ponderaciones
sideEffectstWeight <- 0.7
effectivenessWeight <- 0.3

# Recorremos el dataframe
for (i in 1:length(datos_test[[1]])){

  # Obtenemos el valor de sideEffect
  sideEffectNumber <- datos_test$sideEffectsNumber[i]

  # Obtenemos el valor de efectiveness
  effectivenessRating <- datos_test$effectivenessNumber[i]

  # Inicializamos la variable
  sideEffectRating <- 0
  # Convertimos el valor de sideEffect a la misma escala que efectiveness,
```

```
  # ya que sideEffect == 1 significa que no tiene efectos secundarios (lo cual es bueno),
  # y efecctiveness == 1 significa que no es efectivo (lo cual no es bueno). Para ello
  # realizamos la siguiente conversión:
  if(sideEffectNumber == 1)
     sideEffectRating <- 5
  else if(sideEffectNumber == 2)
    sideEffectRating <- 4
  else if(sideEffectNumber == 3)
    sideEffectRating <- 3
  else if(sideEffectNumber == 4)
    sideEffectRating <- 2
  else if(sideEffectNumber == 5)
    sideEffectRating <- 1
  # Obtenemos el resultado ponderado en tipo float
  floatResult <- effectivenessRating * effectivenessWeight + sideEffectRating * sideEffectstWeight

  # Convertimos el resultado a valor entero.
  integerResult <- as.integer(floatResult)

  # Calculamos la parte decimal y redondeamos
  if(floatResult - integerResult < 0.5)
    result <- integerResult
  else
    result <- integerResult+1

  # Añadimos el resultado obtenido y lo multiplicamos por dos para pasarlo a escala (1-10)
  datos_test$weightedRating[i] <- result * 2
}
```

Comprobamos que se ha creado la nueva columna *weightedRating*, y que se han añadido los nuevos cambios.

```
head(datos_train$weightedRating, 10)
```

```
##  [1]  8  6 10  6  4  4  8 10 10  2
```

2.2.5. Convertir a etiquetas 0 y 1 la columna rating

La columna rating, está entre 0 y 10, pero en algunos casos, cuando queramos aplicar técnicas, deberemos tener una variable binaria, por eso, los numeros que están entre 1 a 4 son 0 y los que estan entre 5 y 10 son 1.

Lo hacemos en train

```
# Recorremos el dataframe
for (i in 1:length(datos_train[[1]])){

  # Obtenemos el valor de rating
  rating <- datos_train$rating[i]

  if (datos_train$rating[i] < 5) # 1 a 2 - no favorable
    result <- 0
  else if (datos_train$rating[i] >= 5) # 3 a 5 - favorable
    result <- 1

  datos_train$ratingLabel[i]  <- result
}
```

Ahora en test

```
# Recorremos el dataframe
for (i in 1:length(datos_test[[1]])){

  # Obtenemos el valor de rating
  rating <- datos_test$rating[i]

  if (datos_test$rating[i] < 5) # 1 a 2 - no favorable
    result <- 0
  else if (datos_test$rating[i] >= 5) # 3 a 5 - favorable
    result <- 1

  datos_test$ratingLabel[i]  <- result
}
```

2.2.6. Cambiar el orden de la columna sideEffectsNumber

Cuando queremos comparar la columna sideEffectsNumber con effectivenesNumber, ambas siguen distinto orden:

Ya que el 5 en sideEffectsNumber es efectos secundarios extremadamente graves y el 5 en effectivenesNumber es altamente efectivo, es por eso que se ha decidido cambiar el orden de sideEffectsNumber.

Para train

```
# Recorremos el dataframe
for (i in 1:length(datos_train[[1]])){

  # Obtenemos el valor de sideEffect
  sideEffectNumber <- datos_train$sideEffectsNumber[i]

  # Inicializamos la variable
  sideEffectRating <- 0
  # Convertimos el valor de sideEffect a la misma escala que efectiveness, ya que
  # sideEffect == 1 significa que no tiene efectos secundarios (lo cual es bueno),
  # y efecctiveness == 1 significa que no es efectivo (lo cual no es bueno). Para ello
  # realizamos la siguiente conversión:
  if(sideEffectNumber == 1)
     sideEffectRating <- 5
  else if(sideEffectNumber == 2)
    sideEffectRating <- 4
  else if(sideEffectNumber == 3)
    sideEffectRating <- 3
  else if(sideEffectNumber == 4)
    sideEffectRating <- 2
  else if(sideEffectNumber == 5)
    sideEffectRating <- 1

  # Añadimos el resultado obtenido y lo multiplicamos por dos para pasarlo a escala (1-10)
  datos_train$sideEffectsInverse[i] <- sideEffectRating
}
```

Para test

```r
# Recorremos el dataframe
for (i in 1:length(datos_test[[1]])){

  # Obtenemos el valor de sideEffect
  sideEffectNumber <- datos_test$sideEffectsNumber[i]

  # Inicializamos la variable
  sideEffectRating <- 0
  # Convertimos el valor de sideEffect a la misma escala que efectiveness, ya que
  # sideEffect == 1 significa que no tiene efectos secundarios (lo cual es bueno),
  # y efecctiveness == 1 significa que no es efectivo (lo cual no es bueno). Para
  # ello realizamos la siguiente conversión:
  if(sideEffectNumber == 1)
     sideEffectRating <- 5
  else if(sideEffectNumber == 2)
    sideEffectRating <- 4
  else if(sideEffectNumber == 3)
    sideEffectRating <- 3
  else if(sideEffectNumber == 4)
    sideEffectRating <- 2
  else if(sideEffectNumber == 5)
    sideEffectRating <- 1

  # Añadimos el resultado obtenido y lo multiplicamos por dos para pasarlo a escala (1-10)
  datos_test$sideEffectsInverse[i] <- sideEffectRating
}
```

2.2.7. Representación gráfica de los datos

Antes de comenzar con el análisis exploratorio, vamos realizar distintas gráfica de barras con el fin de comprender mejor los datos, antes del procesamiento. Primero realizaremos un gráfico de barras de las clasificaciones del medicamente por parte del paciente.

En la figura 1, se observa como más del 50 % de los medicamentos obtienen una nota superior 6 por parte del paciente. Esto nos sugiere que el paciente, tiene una buena opinión sobre un alto porcentaje de los medicamentos, lo que puede dar lugar a opiniones más positivas.

Por otro lado, vamos a mostrar gráficamente los efectos secundarios del medicamento según el paciente. En donde el 1 significa que el medicamento tiene pocos efectos secundarios y el 5 que tiene muchos efectos secundarios. Como se aprecia en la figura 3, un alto porcentaje de los medicamentos no tiene efectos secundarios, de acuerdo a las valoraciones de los pacientes.

Por último, vamos a visualizar gráficamente, la efectividad del medicamentoe según del paciente. En donde el 1 significa que el medicamento tiene poca efectividad y el 5 que tiene mucha efectividad. De acuerdo a la figura 3 y sabiendo que los pacientes aseguran que los medicamentos tienen pocos efectos secundarios, no es de extrañar que también tengan una alta efectividad.

Acabamos de comprobar, como las valoraciones de los medicamentos por parte de los pacientes son mayormente positivas.

Por tanto, una vez comprendidos los datos y eliminadas las columnas anteriores y modificadas las necesarias, ya podemos continuar con el procesamiento de los datos. Para ello, lo primero tenemos que hacer es cargar la librería que procesa los datos de tipo texto en R, para la construcción y manipulación del corpus. La librería más conocida se llama **tm**, aunque también haremos uso del paquete **SnowballC** para realizar el *Stemming*. Si no tenemos instaladas las librerías:

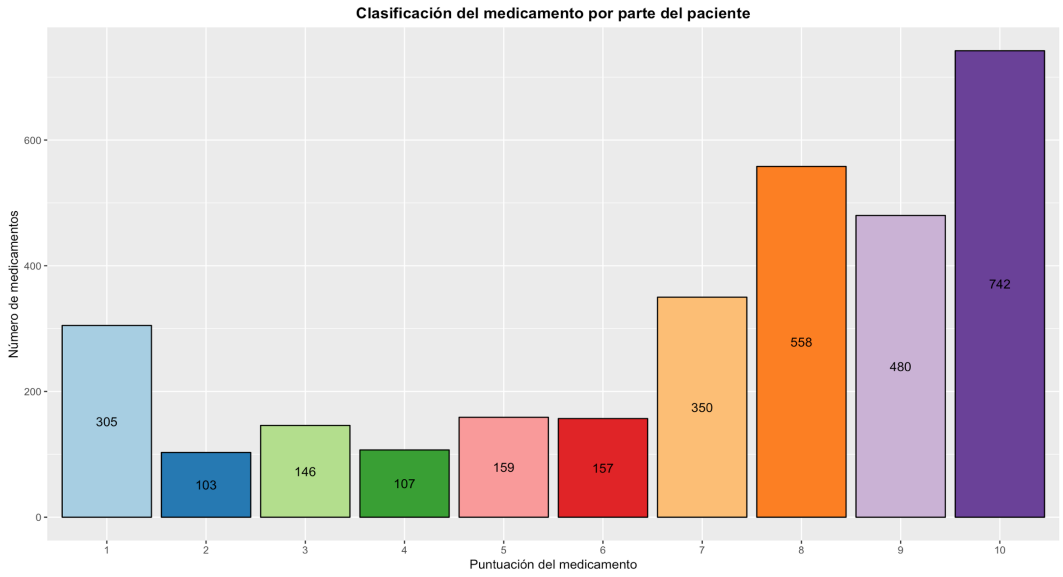


Figura 1: Clasificación del medicamento por parte del paciente

2.2.8. Creación del corpus

Para poder obtener la estructura con la que vamos a procesar nuestra información, debemos obtener un vector con documentos. En nuestro caso, cada uno de los documentos se corresponde con una opinión sobre un fármaco (*benefitsReview*) y los efectos que tiene (*sideEffectsReview*). Para ello, primero debemos de construir un vector con todas los opiniones del *dataframe* y convertir cada elemento del vector al formato de documento. Podemos usar la función *VectorSource* para hacer esta conversión. Se deberán realizar todas las modificaciones tanto para el conjunto train como test.

```
# Datos train

# Nos quedamos con la única columna del dataset que nos interesa.
# Necesitamos obtenerla en forma de vector, y no como un dataframe de una columna,
# por lo que usamos as.vector para hacer la conversión
benefits_train_review_data = as.vector(datos_train$benefitsReview)
effects_train_review_data = as.vector(datos_train$sideEffectsReview)

# Lo convertimos en la estructura de documento, y lo guardamos ya en el corpus
# que lo vamos a utilizar
benefits_train_corpus = (VectorSource(benefits_train_review_data))
effects_train_corpus = (VectorSource(effects_train_review_data))

# Creamos el propio corpus
benefits_train_corpus <- Corpus(benefits_train_corpus)
effects_train_corpus <- Corpus(effects_train_corpus)
```

```
# Datos test

# Nos quedamos con la única columna del dataset que nos interesa.
# Necesitamos obtenerla en forma de vector, y no como un dataframe de una columna,
# por lo que usamos as.vector para hacer la conversión
benefits_test_review_data = as.vector(datos_test$benefitsReview)
effects_test_review_data = as.vector(datos_test$sideEffectsReview)
```

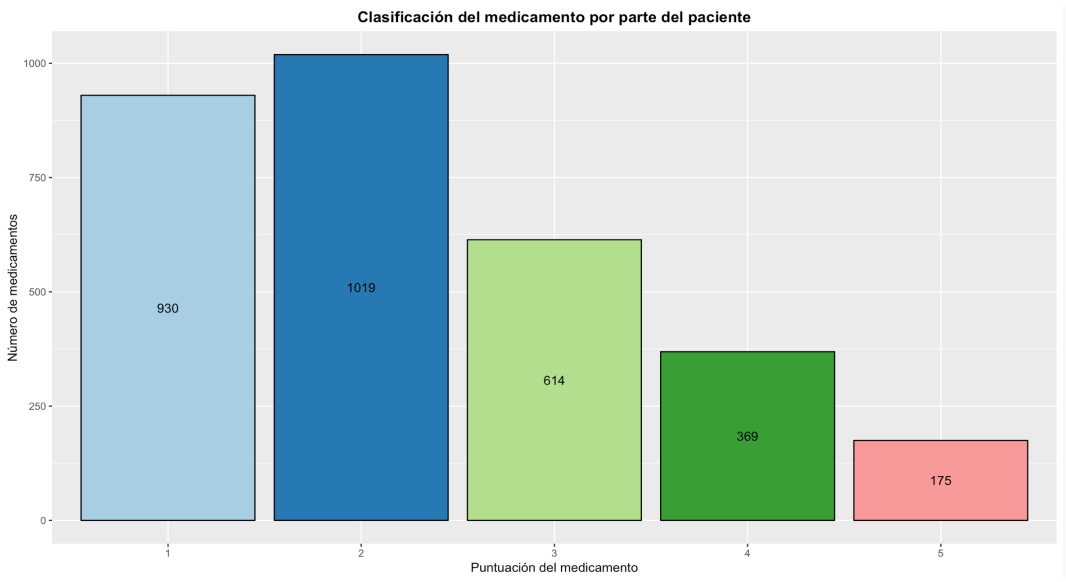


Figura 2: Clasificación de los efectos secundarios del medicamento según el paciente

```
# Lo convertimos en la estructura de documento, y lo guardamos ya en el corpus
# que lo vamos a utilizar
benefits_test_corpus = (VectorSource(benefits_test_review_data))
effects_test_corpus = (VectorSource(effects_test_review_data))

# Creamos el propio corpus
benefits_test_corpus <- Corpus(benefits_test_corpus)
effects_test_corpus <- Corpus(effects_test_corpus)
```

Podemos ver que funciona accediendo a uno cualquiera, de la forma `inspect(benefits_train_corpus[4])`:

O de la forma `benefits_train_corpus[[4]]$content`:

Y si nos fijamos en el contenido, vemos que tiene signos de puntuación y exclamación.

2.2.9. Representación gráfica de las frecuencias

Una vez creado el corpus y antes de aplicar las técnicas, vamos a visualizar las frecuencias, para las dos columnas (*benefitsReview* y *sideEffectsReview*) de textos a las que vamos aplicar el preprocesamiento. Para ello, debemos calcular la matriz de términos y obtener los términos con mayor frecuencia.

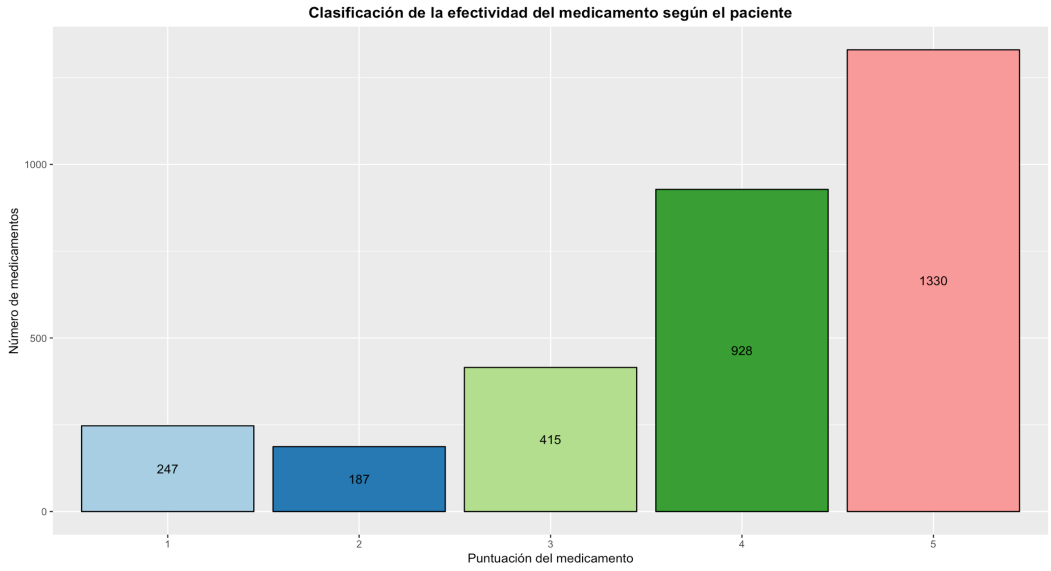
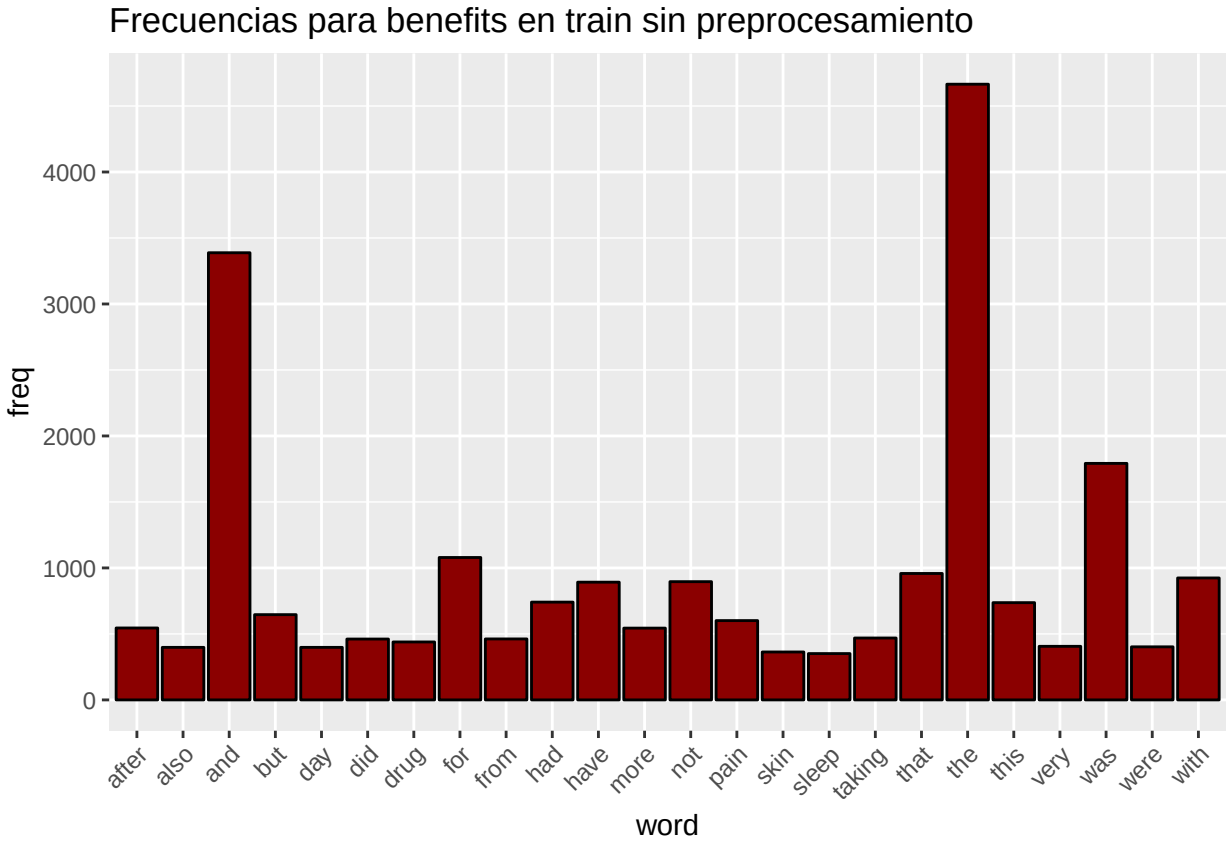


Figura 3: Clasificación de la efectividad del medicamento según el paciente

```
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 1

[1] The acid reflux went away    months  just   days     drug. The heartburn started   soon  I stopped taking . So I began treatment . 6
months passed  I stopped taking . The heartburn came back,  seemed worse even. The doctor said I  try another 6 month treatment. I ,
exact thing happened. This went    three years. I asked    curing  reflux. The doctor quite frankly told      cure,   "treatment
symptoms". I  told  I  probably      rest   life.
```

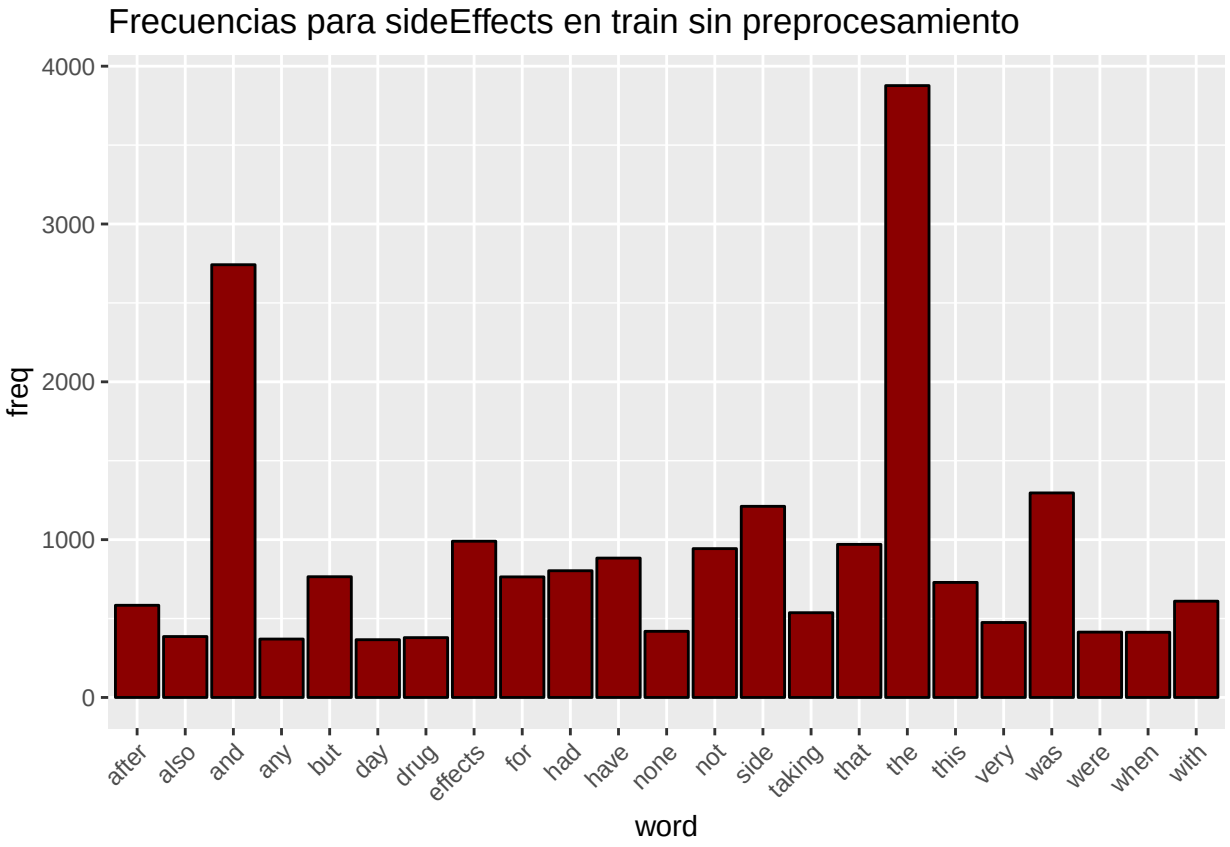
Figura 4: Contenido de benefits_train_corpus I



```
[1] "The acid reflux went away    months  just   days     drug. The heartburn started   soon  I stopped taking . So I began treatment . 6
months passed  I stopped taking . The heartburn came back,  seemed worse even. The doctor said I  try another 6 month treatment. I ,
exact thing happened. This went    three years. I asked    curing  reflux. The doctor quite frankly told      cure,   \"treatment
symptoms\". I  told  I  probably      rest   life."
```

Figura 5: Contenido de benefits_train_corpus II

A continuación, mostramos los términos para la columna *sideEffects*, realizando el mismo procedimiento que antes.



2.2.10. Correlación

Una forma de medir la distancia es calcular la correlación entre un término y todos los demás de la matriz. Como en este caso, estamos usando variables textuales, no se recomienda hacer la correlación. Ya que está pensada para variables cuantitativas, y nosotgros disponemos de variables categóricas. Por otro lado, la correlación está relacionada con el análisis de componentes principales (PCA), por tanto, tampoco sería posible la realización de dicha técnica.

2.2.11. Eliminar signos de puntuación

Como hemos podido ver en el documento que se ha mostrado por pantalla, en él se aprecia el uso de signos de puntuación y exclamación. En un principio, no tiene sentido en *Data Mining* contemplar los signos de puntuación, ya que no nos van a aportar información. Por ello, los quitamos, como se puede ver a continuación. Con `tm_map(corpus, removePunctuation)`, se eliminan los símbolos: ! " $ % & ‘() * + , - . / : ; < = > ? @ [ ] ^ _’ { | } ~

```
# Una vez que tenemos el corpus creado, continuamos con el procesamiento para datos train
benefits_train_corpus <- tm_map(benefits_train_corpus,
                                content_transformer(removePunctuation))
effects_train_corpus <- tm_map(effects_train_corpus,
                               content_transformer(removePunctuation))

# Una vez que tenemos el corpus creado, continuamos con el procesamiento para datos test
benefits_test_corpus <- tm_map(benefits_test_corpus,
                               content_transformer(removePunctuation))
effects_test_corpus <- tm_map(effects_test_corpus,
                              content_transformer(removePunctuation))
```

Si volvemos a mostrar la opinión número cuatro, vemos como todos los signos han desaparecido. De hecho, podemos inspeccionar el corpus, y se ve como todos los signos de puntuación, exclamación y derivados ya no están.

```
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 1

[1] The acid reflux went away for a few months after just a few days of being on the drug The heartburn started again as soon as I stopped
taking it So I began treatment again 6 months passed and I stopped taking it The heartburn came back and seemed worse even The doctor said
I should try another 6 month treatment I did and the same exact thing happened This went on for about three years I asked why this wasnt
curing my reflux The doctor quite frankly told me that it wasnt a cure but a treatment for the symptoms I was told that I would probably
be on it for the rest of my life
```

Figura 6: Contenido de benefits_train_corpus con inspect(benefits_train_corpus[4])

Ocurre lo mismo con el comentario de efectos número siete.

```
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 1

[1] a few experiences of nausiea heavy moodswings on the days I do not take it decreased appetite and some negative affect on my shortterm
memory
```

Figura 7: Contenido de effects_train_corpus con inspect(effects_train_corpus[7])

2.2.12. Conversión de las mayúsculas en minúsculas

Para poder hacer uso de los términos por igual, debemos convertir las mayúsculas en minúsculas. Ya que normalmente se convierte en minúsculas todas las letras para que los comienzos de oración no sean tratados de manera diferente por los algoritmos.

```
# Datos train
benefits_train_corpus <- tm_map(benefits_train_corpus, content_transformer(tolower))
effects_train_corpus <- tm_map(effects_train_corpus, content_transformer(tolower))

# Datos test
benefits_test_corpus <- tm_map(benefits_test_corpus, content_transformer(tolower))
effects_test_corpus <- tm_map(effects_test_corpus, content_transformer(tolower))
```

2.2.13. Eliminación de Stopwords

En cualquier idioma, hay palabras que son tan comunes o muy utilizadas que no aportan información relevante, a dichas palabras se las conoce como *stopwords* o palabras *stop*. Por ejemplo, en español, las palabras “la”,

```
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 1

[1] the acid reflux went away for a few months after just a few days of being on the drug the heartburn started again as soon as i stopped
taking it so i began treatment again 6 months passed and i stopped taking it the heartburn came back and seemed worse even the doctor said
i should try another 6 month treatment i did and the same exact thing happened this went on for about three years i asked why this wasnt
curing my reflux the doctor quite frankly told me that it wasnt a cure but a treatment for the symptoms i was told that i would probably
be on it for the rest of my life
```

Figura 8: inspect(benefits_train_corpus[4])

```
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 1

[1] a few experiences of nausiea heavy moodswings on the days i do not take it decreased appetite and some negative affect on my shortterm
memory
```

Figura 9: inspect(effects_train_corpus[7])

"a", "en", "de" son ejemplos de *stopwords*. Este tipo de palabras debemos de suprimirlas de nuestro corpus. Como, en nuestro caso, el contenido del corpus está en inglés, debemos especificar el idioma correcto para que nos elimine del corpus las palabras adecuadas en dicho idioma.

```
# Datos train
benefits_train_corpus <- tm_map(benefits_train_corpus, content_transformer(removeWords),
                                stopwords("english"))
effects_train_corpus <- tm_map(effects_train_corpus, content_transformer(removeWords),
                               stopwords("english"))

# Datos test
benefits_test_corpus <- tm_map(benefits_test_corpus, content_transformer(removeWords),
                               stopwords("english"))
effects_test_corpus <- tm_map(effects_test_corpus, content_transformer(removeWords),
                              stopwords("english"))
```

```
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 1

[1]  acid reflux went away    months  just   days     drug  heartburn started   soon   stopped taking    began treatment  6 months passed
stopped taking   heartburn came back  seemed worse even  doctor said   try another 6 month treatment      exact thing happened  went
three years  asked   wasnt curing  reflux  doctor quite frankly told    wasnt  cure   treatment   symptoms   told    probably      rest
life
```

Figura 10: inspect(benefits_train_corpus[4])

Ahora ya hemos eliminado las stopwords de forma correcta.

2.2.14. Agrupación de sinónimos

Con el fin de disminuir la dimensión del espacio a trabajar, se pueden identificar palabras distintas con el mismo significado y reemplazarlas por una sola palabra. Para ello se toman los sinónimos de dicha palabra. Dentro de las librerías que podemos usar para agrupar sinónimos, destacamos dos: `wordnet` y `rword2vec`. Sin embargo, por su sencillez se va hacer uso de `rword2vec`. Previamente, se obtendrán que palabras son las que mayor frecuencia presentan en nuestro texto, para ello nos quedamos con las 100 más representativas tanto para *benefitsReview* como *sideEffectsReview* del conjunto train y test:

```
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 1

[1]   experiences  nausiea heavy moodswings   days    take  decreased appetite   negative affect   shortterm memory
```

Figura 11: inspect(effects_train_corpus[7])

```r
# Columna benefitsReview del conjunto train

# Obtenemos su matriz de términos
matrix_train_benefits_corpus <- TermDocumentMatrix(benefits_train_corpus)
# No tenemos los datos en la matriz que buscamos, sino en un vector
# por tanto, lo convertimos en matriz
matrix_train_benefits_corpus <- as.matrix(matrix_train_benefits_corpus)
# Sumamos las filas para obtener la frecuencia de una palabra en benefitsReview
matrix_train_benefits_corpus <- rowSums(matrix_train_benefits_corpus)
# Ordenamos de mayor a menor los términos
terms_frecuency_benefits_train_corpus <- sort(matrix_train_benefits_corpus, decreasing = TRUE)
terms_frecuency_benefits_train_corpus <- terms_frecuency_benefits_train_corpus[1:length(benefits_train_
# terms_frecuency_benefits_train_corpus
```

```r
# Columna benefitsReview del conjunto test

# Obtenemos su matriz de términos
matrix_test_benefits_corpus <- TermDocumentMatrix(benefits_test_corpus)
# No tenemos los datos en la matriz que buscamos, sino en un vector
# por tanto, lo convertimos en matriz
matrix_test_benefits_corpus <- as.matrix(matrix_test_benefits_corpus)
# Sumamos las filas para obtener la frecuencia de una palabra en benefitsReview
matrix_test_benefits_corpus <- rowSums(matrix_test_benefits_corpus)
# Ordenamos de mayor a menor los términos
terms_frecuency_benefits_test_corpus <- sort(matrix_test_benefits_corpus, decreasing = TRUE)
terms_frecuency_benefits_test_corpus <- terms_frecuency_benefits_test_corpus[1:length(benefits_test_cor
# terms_frecuency_benefits_test_corpus
```

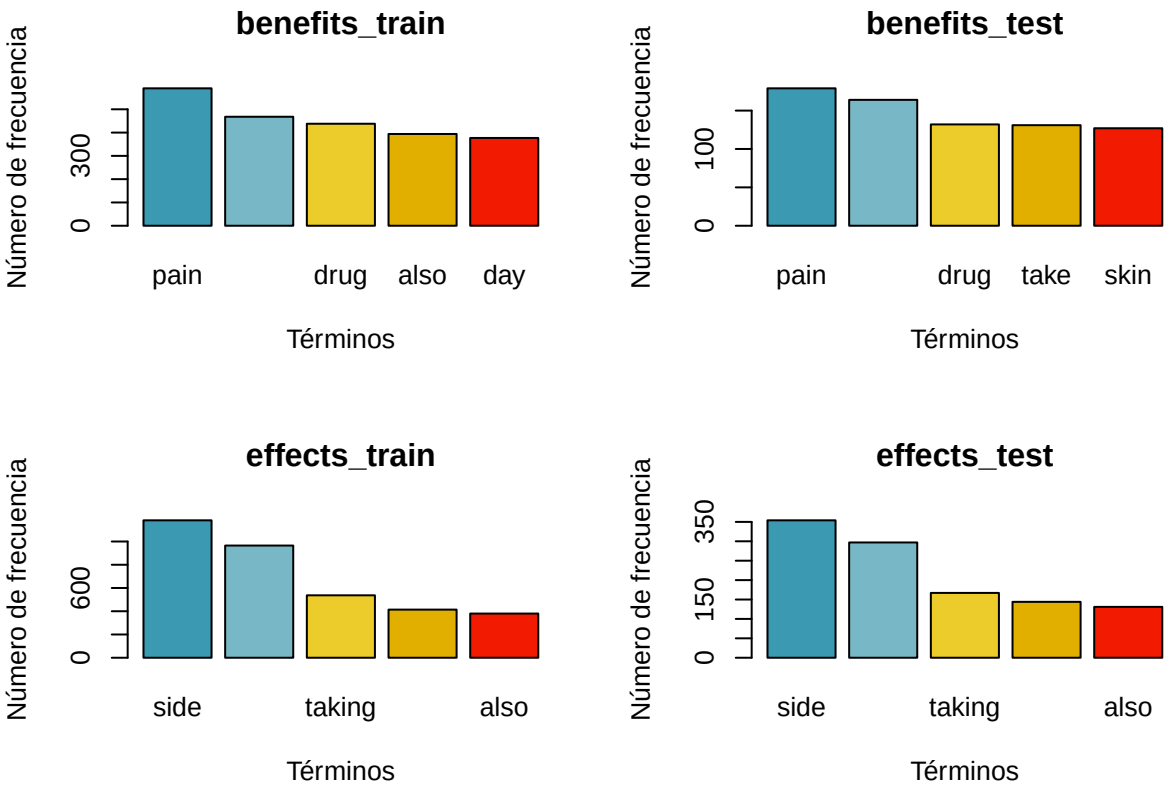
```r
# Columna sideEffectsReview del conjunto train

# Obtenemos su matriz de términos
matrix_train_effects_corpus <- TermDocumentMatrix(effects_train_corpus)
# No tenemos los datos en la matriz que buscamos, sino en un vector
# por tanto, lo convertimos en matriz
matrix_train_effects_corpus <- as.matrix(matrix_train_effects_corpus)
# Sumamos las filas para obtener la frecuencia de una palabra en benefitsReview
matrix_train_effects_corpus <- rowSums(matrix_train_effects_corpus)
# Ordenamos de mayor a menor los términos y nos quedamos con lso 100 primeros
terms_frecuency_effects_train_corpus <- sort(matrix_train_effects_corpus, decreasing = TRUE)
terms_frecuency_effects_train_corpus <- terms_frecuency_effects_train_corpus[1:length(effects_train_cor
# terms_frecuency_effects_train_corpus
```

```r
# Columna benefitsReview del conjunto test

# Obtenemos su matriz de términos
matrix_test_effects_corpus <- TermDocumentMatrix(effects_test_corpus)
# No tenemos los datos en la matriz que buscamos, sino en un vector
```

```
# por tanto, lo convertimos en matriz
matrix_test_effects_corpus <- as.matrix(matrix_test_effects_corpus)
# Sumamos las filas para obtener la frecuencia de una palabra en benefitsReview
matrix_test_effects_corpus <- rowSums(matrix_test_effects_corpus)
# Ordenamos de mayor a menor los términos
terms_frecuency_effects_test_corpus <- sort(matrix_test_effects_corpus, decreasing = TRUE)
terms_frecuency_effects_test_corpus <- terms_frecuency_effects_test_corpus[1:length(effects_test_corpus)
# terms_frecuency_effects_test_corpus
```

Y visualizamos dichos términos gráficamente:

```
par(mfrow=c(2,2))

graph_terms_frecuency_benefits_train_corpus <- as.matrix(terms_frecuency_benefits_train_corpus)
barplot(graph_terms_frecuency_benefits_train_corpus[1:5,],  xlab="Términos", ylab="Número de frecuencia"
        col=wes_palette(n=5, name="Zissou1"))
title(main = list("benefits_train", font = 2))

graph_terms_frecuency_benefits_test_corpus <- as.matrix(terms_frecuency_benefits_test_corpus)
barplot(graph_terms_frecuency_benefits_test_corpus[1:5,],  xlab="Términos", ylab="Número de frecuencia"
        col=wes_palette(n=5, name="Zissou1"))
title(main = list("benefits_test", font = 2))

graph_terms_frecuency_effects_train_corpus <- as.matrix(terms_frecuency_effects_train_corpus)
barplot(graph_terms_frecuency_effects_train_corpus[1:5,],  xlab="Términos", ylab="Número de frecuencia"
        col=wes_palette(n=5, name="Zissou1"))
title(main = list("effects_train", font = 2))

graph_terms_frecuency_effects_test_corpus <- as.matrix(terms_frecuency_effects_test_corpus)
barplot(graph_terms_frecuency_effects_test_corpus[1:5,],  xlab="Términos", ylab="Número de frecuencia",
        col=wes_palette(n=5, name="Zissou1"))
title(main = list("effects_test", font = 2))
```



Una vez que tenemos los términos con mayor frecuencia en nuestra columna *benefitsReview* y su frecuencia asociada, pasamos a matriz dichos datos, con el fin de obtener solo las palabras y descartar su frecuencia.

```
# Columna benefitsReview del conjunto train

# Convertimos a matriz "terms_frecuency_benefits_corpus_100"
terms_frecuency_benefits_train_corpus <- as.matrix(terms_frecuency_benefits_train_corpus)
# terms_frecuency_benefits_train_corpus

# Me quedo solo con los términos
terms_benefits_train_corpus <- rownames(terms_frecuency_benefits_train_corpus)
# terms_benefits_train_corpus
```

```
# Columna benefitsReview del conjunto test

# Convertimos a matriz "terms_frecuency_benefits_corpus_100"
terms_frecuency_benefits_test_corpus <- as.matrix(terms_frecuency_benefits_test_corpus)
# terms_frecuency_benefits_test_corpus

# Me quedo solo con los términos
terms_benefits_test_corpus <- rownames(terms_frecuency_benefits_test_corpus)
# terms_benefits_test_corpus
```

```
# Columna sideEffectsReview del conjunto train

# Convertimos a matriz "terms_frecuency_benefits_corpus_100"
terms_frecuency_effects_train_corpus <- as.matrix(terms_frecuency_effects_train_corpus)
# terms_frecuency_effects_train_corpus

# Me quedo solo con los términos
```

```
terms_effects_train_corpus <- rownames(terms_frecuency_effects_train_corpus)
# terms_effects_train_corpus
```

```
# Columna sideEffectsReview del conjunto test

# Convertimos a matriz "terms_frecuency_benefits_corpus_100"
terms_frecuency_effects_test_corpus <- as.matrix(terms_frecuency_effects_test_corpus)
# terms_frecuency_effects_test_corpus

# Me quedo solo con los términos
terms_effects_test_corpus <- rownames(terms_frecuency_effects_test_corpus)
# terms_effects_test_corpus
```

Como ya sabemos las palabras a usar, es decir, los términos que más se repite, procedemos a la agrupación por sinónimos. En donde, mediante la función *distance(...)* de la libería *rword2vec*, obtendremos todas palabras más similares de nuestro conjunto, en nuestro caso nos vamos a quedar con las 2 primeras:

```
# http://mccormickml.com/2016/04/12/googles-pretrained-word2vec-model-in-python/
# https://github.com/mukul13/rword2vec
# http://www.rpubs.com/mukul13/rword2vec
```

Una vez, que tenemos todas las palabras con los 2 términos más similares, procedemos a sustituir todos esos términos por el término general, es decir:

```
# Obtenemos el tercer término -> "drug"
terms_benefits_train_corpus[3]
```

```
## [1] "drug"
```

```
# Vamos a sustituir "pain" por sus dos palabras más similares
dist_terms_benefits_train_corpus_new[[3]]
```

```
## [1] medication medicine
## Levels: medication medicine
```

Por último, ya solo nos queda hacer el reemplazamiento, para ello se usará la función *gsub(...)* sobre el corpus (benefits_corpus). Para sustituir las palabras en el texto, se ha uso de la función `gsub(pattern, replacement, x, ignore.case = FALSE, perl = FALSE, fixed = FALSE, useBytes = FALSE)`.

```
# Para la columna benefitsReview del conjunto train

#dist_terms_benefits_train_corpus_new <- list.remove(dist_terms_benefits_train_corpus_new, c(446, 506))
# dist_terms_benefits_train_corpus_new <- list.remove(dist_terms_benefits_train_corpus_new, c(447, 508,
# dist_terms_benefits_train_corpus <- list.remove(dist_terms_benefits_train_corpus, c(1049, 1077, 1103,

#View(dist_terms_benefits_train_corpus)

#for (i in 1:640) # iteramos sobre los terminos, hasta el 1797, porque si vemos el fichero, ya no hay m
#  for (j in 1:2) # iteramos sobre los sinónimos, en este caso solo tenemos 2
#    benefits_train_corpus_new <- tm_map(benefits_train_corpus, content_transformer(gsub),
#                                        pattern = tolower(as.character(dist_terms_benefits_train_corpus_new[
#                                        replacement = as.character(terms_benefits_train_corpus[i]))

# guardar "benefits_train_corpus_new"
#saveRDS(benefits_train_corpus_new, file = "datos/ficheros-sinonimos/benefits/benefits_train_corpus_new
benefits_train_corpus_new_load <- readRDS(file = "datos/ficheros-sinonimos/benefits/benefits_train_corpu
```

```
# Comprobamos que efectivamente se han producido cambios, por ejemplo al revisar el término "medication
write.table(benefits_train_corpus$content, "datos/ficheros-sinonimos/benefits/benefitsTrainSinSinonimos
write.table(benefits_train_corpus_new_load$content, "datos/ficheros-sinonimos/benefits/benefitsTrainCon
```

```
# Para la columna benefitsReview del conjunto test

# dist_terms_benefits_test_corpus_new <- list.remove(dist_terms_benefits_test_corpus_new, c(171, 470))

#for (i in 1:450) # iteramos sobre los terminos
#  for (j in 1:2) # iteramos sobre los sinónimos, en este caso solo tenemos 2
#    benefits_test_corpus_new <- tm_map(benefits_test_corpus, content_transformer(gsub),
#                                pattern = tolower(as.character(dist_terms_benefits_test_corpus_new[[
#                                replacement = as.character(terms_benefits_test_corpus[i]))

# guardar "benefits_train_corpus_new"
#saveRDS(benefits_test_corpus_new, file = "datos/ficheros-sinonimos/benefits/benefits_test_corpus_new.r
benefits_test_corpus_new_load <- readRDS(file = "datos/ficheros-sinonimos/benefits/benefits_test_corpus

# Comprobamos que efectivamente se han producido cambios, por ejemplo al revisar el término "medication
write.table(benefits_test_corpus$content, "datos/ficheros-sinonimos/benefits/benefitsTestSinSinonimos.t
write.table(benefits_test_corpus_new_load$content, "datos/ficheros-sinonimos/benefits/benefitsTestConSi
```

```
# Para la columna sideEffectsReview del conjunto train

#dist_terms_effects_train_corpus_new <- list.remove(dist_terms_effects_train_corpus_new, c(318, 504, 65

#for (i in 1:710) # iteramos sobre los terminos, hasta el 1539, porque si vemos el fichero, ya no hay m
#  for (j in 1:2) # iteramos sobre los sinónimos, en este caso solo tenemos 2
#    effects_train_corpus_new <- tm_map(effects_train_corpus, content_transformer(gsub),
#                                pattern = tolower(as.character(dist_terms_effects_train_corpus_new[[
#                                replacement = as.character(terms_effects_train_corpus[i]))

# guardar "effects_train_corpus_new"
# saveRDS(effects_train_corpus_new, file = "effects_train_corpus_new.rds")
effects_train_corpus_new_load <- readRDS(file = "datos/ficheros-sinonimos/effects/effects_train_corpus_

# Comprobamos que efectivamente se han producido cambios, por ejemplo al revisar el término "medication
write.table(effects_train_corpus$content, "datos/ficheros-sinonimos/effects/effectsTrainSinSinonimos.tx
write.table(effects_train_corpus_new_load$content, "datos/ficheros-sinonimos/effects/effectsTrainConSin
```

```
# Para la columna sideEffectsReview del conjunto test


#for (i in 1:450) # iteramos sobre los terminos
#  for (j in 1:2) # iteramos sobre los sinónimos, en este caso solo tenemos 2
#    effects_test_corpus_new <- tm_map(effects_test_corpus, content_transformer(gsub),
#                                pattern = tolower(as.character(dist_terms_effects_test_corpus_new[[i
#                                replacement = as.character(terms_effects_test_corpus[i]))

# guardar "effects_train_corpus_new"
#saveRDS(effects_test_corpus_new, file = "datos/ficheros-sinonimos/effects/effects_test_corpus_new.rds"
effects_test_corpus_new_load <- readRDS(file = "datos/ficheros-sinonimos/effects/effects_test_corpus_ne

# Comprobamos que efectivamente se han producido cambios, por ejemplo al revisar el término "medication
```

```
write.table(effects_test_corpus$content, "datos/ficheros-sinonimos/effects/effectsTestSinSinonimos.txt")
write.table(effects_test_corpus_new_load$content, "datos/ficheros-sinonimos/effects/effectsTestConSinoni
```

2.2.15. TF-IDF

Para estudiar la importancia de los términos de un documento en particular, en lugar de utilizar la frecuencia de cada uno de los términos directamente, se pueden utilizar diferentes ponderaciones denominadas TF-IDF (Term Frequency-Inverse Document Frecuency).

Estas ponderaciones se calculan como el producto de dos medidas, la frecuencia de aparición del término (tf) y la frecuencia inversa del documento (idf). La fórmula matemática para esta métrica es la siguiente:

```
# ARREGLAR FORMULA DA ERROR
# $$ tfidf(t, d, D) = tf(t, d) × idf(t, D)$$
```

donde t es el término, d denota cada documento, D el espacio total de documentos y tfidf es el peso asignado a ese término en el documento correspondiente.

La combinación de los valores de tf e idf da una métrica que permite saber cómo de únicas son las palabras de un documento. La ponderación asigna un alto peso a un término si se produce con frecuencia en ese documento, pero rara vez en la colección completa. Sin embargo, si el término ocurre pocas veces en el documento, o aparece prácticamente en todos ellos, disminuye el peso asignado por la ponderación tfidf.

El peso aumenta proporcionalmente al número de veces que una palabra aparece en el documento, pero es compensada por la frecuencia de la palabra en la colección de documentos, lo que permite filtrar las palabras más comunes.

```
# Convertimos las columnas de review a una tabla en el que cada columna será una palabra del comentario
dataframe_benefits_train<-data.frame(text=unlist(sapply(benefits_train_corpus_new_load, `[`)), stringsA
dataframe_effects_train<-data.frame(text=unlist(sapply(effects_train_corpus_new_load, `[`)), stringsAsFa

dataframe_benefits_test<-data.frame(text=unlist(sapply(benefits_test_corpus_new_load, `[`)), stringsAsFa
dataframe_effects_test<-data.frame(text=unlist(sapply(effects_test_corpus_new_load, `[`)), stringsAsFac

# Crea un vector vacío
vector_benefits_train<- c()
vector_effects_train <- c()

vector_benefits_test<- c()
vector_effects_test <- c()

# Convertimos el dataframe a vector en train
for (i in 1:length(dataframe_benefits_train[[1]])){
  vector_benefits_train[i] <- dataframe_benefits_train[[1]][i]
  vector_effects_train[i] <- dataframe_effects_train[[1]][i]
}

# Convertimos el dataframe a vector en test
for (i in 1:length(dataframe_benefits_test[[1]])){
  vector_benefits_test[i] <- dataframe_benefits_test[[1]][i]
  vector_effects_test[i] <- dataframe_effects_test[[1]][i]
}
```

Frecuencia del término

La primera parte de la fórmula tf (t, d) es simplemente calcular el número de veces que aparece cada palabra en cada documento.

Frecuencia inversa del documento

Durante el cálculo de la frecuencia del término se considera que todos los términos tienen igual importancia, no obstante, se conocen casos en los que ciertos términos pueden aparecer muchas veces pero tienen poca importancia. Esta segunda parte de la fórmula completa el análisis de evaluación de los términos y actúa como corrector de tf.

Usando la matriz de frecuencia de términos, el peso idf se puede calcular de la siguiente forma:

```
# Calculamos los pesos asociados a cada término
terms_benefits_train <- ( idf_benefits_train <- log( ncol(tf_benefits_train) / ( 1 + rowSums(tf_benefits
terms_effects_train <- ( idf_effects_train <- log( ncol(tf_effects_train) / ( 1 + rowSums(tf_effects_tra

terms_benefits_test <- ( idf_benefits_test <- log( ncol(tf_benefits_test) / ( 1 + rowSums(tf_benefits_te
terms_effects_test <- ( idf_effects_test <- log( ncol(tf_effects_test) / ( 1 + rowSums(tf_effects_test

terms_benefits_train[1:5] # Muestra los pesos asociados a cada término
```

```
##      agents       alone  congestive dysfunction      failur
##    6.941835    5.044715    6.654153    6.941835    7.347300
```

Ahora que tenemos nuestra matriz con el término frecuencia y el peso idf, estamos listos para calcular el peso total de tf-idf. Para hacer esta multiplicación de matrices, también tendremos que transformar el vector idf en una matriz diagonal. Ambos cálculos se muestran a continuación.

Hay que recordar que en la sección tf (frecuencia del término), estamos representando cada término como el número de veces que aparecieron en el documento.El principal problema para esta representación es que creará un sesgo hacia documentos largos, ya que un término dado tiene más posibilidades de aparecer en documentos más largos, lo que los hace parecer más importantes de lo que realmente son.

Por lo tanto, el enfoque para resolver este problema es la buena normalización:

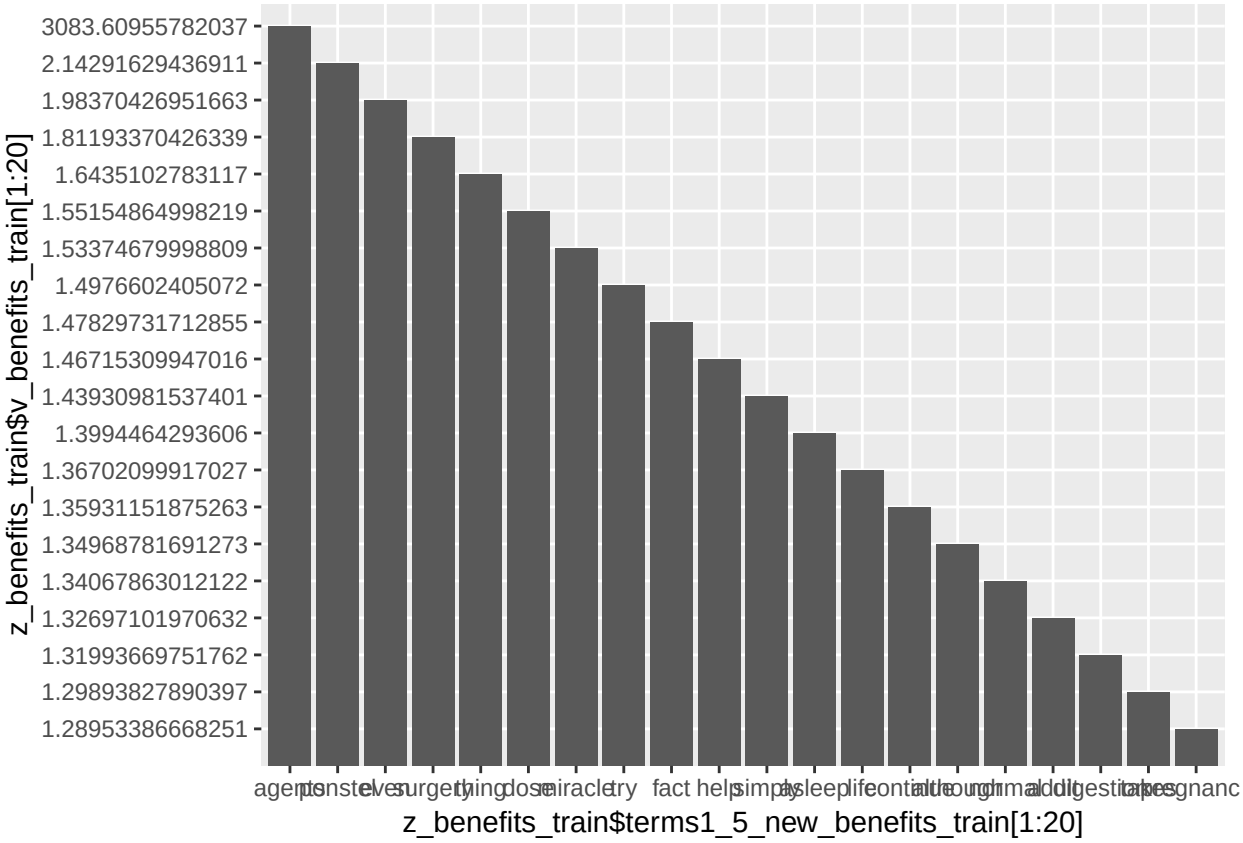
```
# Normalización
t_benefits_train <- tf_idf_benefits_train_new / sqrt( rowSums( tf_idf_benefits_train_new^2 ) )
t_effects_train <- tf_idf_effects_train_new / sqrt( rowSums( tf_idf_effects_train_new^2 ) )

t_benefits_test <- tf_idf_benefits_test_new / sqrt( rowSums( tf_idf_benefits_test_new^2 ) )
t_effects_test <- tf_idf_effects_test_new / sqrt( rowSums( tf_idf_effects_test_new^2 ) )


# Creamos un vector vacío
v_benefits_train <- c()
v_effects_train <- c()

v_benefits_test <- c()
v_effects_test <- c()

# Realiza la suma de todos los pesos de los términos y los almacena en un vector
for (i in 1:length(dataframe_benefits_train[[1]])){
  v_benefits_train[i] <- sum(t_benefits_train[,i])
  v_effects_train[i] <- sum(t_effects_train[,i])
}

for (i in 1:length(dataframe_benefits_test[[1]])){
  v_benefits_test[i] <- sum(t_benefits_test[,i])
```

```
  v_effects_test[i] <- sum(t_effects_test[,i])
}


# Convertimos a matriz
terms1_benefits_train <- as.matrix(terms_benefits_train)
terms1_effects_train <- as.matrix(terms_effects_train)

terms1_benefits_test <- as.matrix(terms_benefits_test)
terms1_effects_test <- as.matrix(terms_effects_test)


# Me quedo solo con los términos
terms1_benefits_train <- rownames(terms1_benefits_train)
terms1_effects_train <- rownames(terms1_effects_train)

terms1_benefits_test <- rownames(terms1_benefits_test)
terms1_effects_test <- rownames(terms1_effects_test)


# Me quedo solo con los términos
terms1_5_benefits_train <- terms1_benefits_train[1:length(dataframe_benefits_train[[1]])]
terms1_5_effects_train <- terms1_effects_train[1:length(dataframe_effects_train[[1]])]

terms1_5_benefits_test <- terms1_benefits_test[1:length(dataframe_benefits_test[[1]])]
terms1_5_effects_test <- terms1_effects_test[1:length(dataframe_effects_test[[1]])]


# Unimos la tabla de términos y valores en x
x_benefits_train <- cbind(terms1_5_benefits_train,v_benefits_train)
x_effects_train <- cbind(terms1_5_effects_train,v_effects_train)

x_benefits_test <- cbind(terms1_5_benefits_test,v_benefits_test)
x_effects_test <- cbind(terms1_5_effects_test,v_effects_test)


# Convertimos dicha tabla a dataframe
x1_benefits_train <- as.data.frame(x_benefits_train)
x1_effects_train <- as.data.frame(x_effects_train)

x1_benefits_test <- as.data.frame(x_benefits_test)
x1_effects_test <- as.data.frame(x_effects_test)


# Guardamos en z los valores de x1 ordenados
z_benefits_train <- x1_benefits_train[order(v_benefits_train, decreasing = TRUE),]
z_effects_train <- x1_effects_train[order(v_effects_train, decreasing = TRUE),]

z_benefits_test <- x1_benefits_test[order(v_benefits_test, decreasing = TRUE),]
z_effects_test <- x1_effects_test[order(v_effects_test, decreasing = TRUE),]


# Ordenamos los datos, respetando el orden de los niveles de los factores. Referencia :https://rstudio-
```

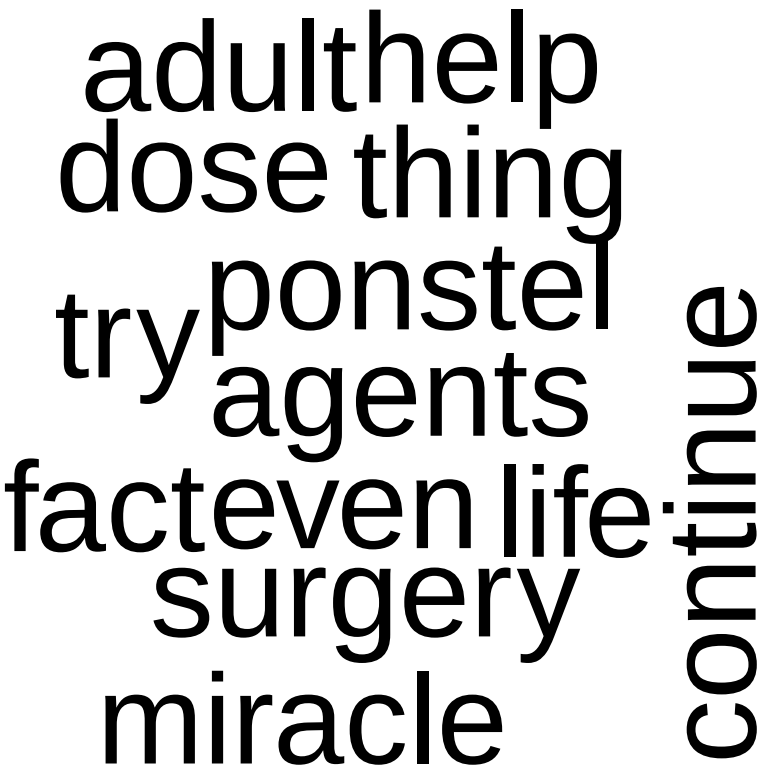
```
z_benefits_train$terms1_5_new_benefits_train <- factor(z_benefits_train$terms1_5_benefits_train, levels

# Nos quedamos con los primeros 20 valores
z_benefits_train$terms1_5_new_benefits_train[1:20]
```

```
##  [1] agents    ponstel   even      surgery   thing     dose      miracle
##  [8] try       fact      help      simply    asleep    life      continue
## [15] although  normal    adult     digestion takes     pregnancy
## 3104 Levels: agents ponstel even surgery thing dose miracle try ... aunt
```

```
# Graficamos los resultados
ggplot() +
 geom_bar(data=z_benefits_train[1:20,],aes(x=z_benefits_train$terms1_5_new_benefits_train[1:20], y=z_ben
```




```
# INTENTAR ARREGLAR
# wordlcoud ver como arreglar
# SIGUE SIN SALIR

z_benefits_train$v2 <- as.numeric(z_benefits_train$v_benefits_train)

wordcloud(
 words = z_benefits_train$terms1_5_benefits_train,
 freq = z_benefits_train$v2,
 max.words = 20,
 random.order = F,
 colorPalette="Dark2"
 )
```



2.2.16. Stemming

El siguiente paso consiste en reducir el número de palabras totales con las que estamos trabajando. En este caso, se trata de reducir aquellas que no nos aportan nada relevante a lo que ya tenemos. En la columna con la que estamos trabajando en este dataframe, se repite una gran cantidad de veces la palabra "benefit", al igual que "benefits".

Sin embargo, realizar el análisis de nuestros datos con ambas palabras no tiene gran relevancia, ya que una no aporta nada respecto a la otra. Este es un ejemplo del tipo de casos que se nos dan en nuestro dataset. Igual ocurre con "reduce" y "reduced", por ejemplo. Este tipo de situaciones son las que intentamos corregir con este paso. Vamos a ver un ejemplo de este suceso, que se da por ejemplo en los siguientes valores del corpus (y en muchos más).

```
inspect(benefits_train_corpus_new_load[183])
```

```
## <<SimpleCorpus>>
## Metadata:  corpus specific: 1, document level (indexed): 0
## Content:  documents: 1
##
## [1]  treatment benefits   temporary    made sneezing  watery eyes diminish    address  issue  respira
```

```
inspect(benefits_train_corpus_new_load[213])
```

```
## <<SimpleCorpus>>
## Metadata:  corpus specific: 1, document level (indexed): 0
## Content:  documents: 1
##
## [1] overall   ease mantally    true benefit  felt
```

A continuación, aplicamos el proceso de stemming mediante la siguiente orden:

```
benefits_train_corpus_new_load <- tm_map(benefits_train_corpus_new_load, stemDocument)
effects_train_corpus_new_load <- tm_map(effects_train_corpus_new_load, stemDocument)

benefits_test_corpus_new_load <- tm_map(benefits_test_corpus_new_load, stemDocument)
effects_test_corpus_new_load <- tm_map(effects_test_corpus_new_load, stemDocument)
```

Si ahora volvemos a mostrar el contenido de dichas opiniones, podemos ver que el stemming se ha hecho efectivo: donde ponía *benefits*, ahora pone *benefit*, como se puede comprobar si volvemos a mostrar dichos elementos del corpus. De hecho, si nos fijamos, no solo esta palabra ha resultado modificada, sino que se han resumido muchas más palabras en comparación a como teníamos los documentos en el momento previo a la aplicación del método *Stem*. Desde este momento, ya tenemos nuestro conjunto reducido a nivel de concepto.

```
inspect(benefits_train_corpus_new_load[183])
```

```
## <<SimpleCorpus>>
## Metadata:  corpus specific: 1, document level (indexed): 0
## Content:  documents: 1
##
## [1] treatment benefit temporari made sneez wateri eye diminish address issu respiratori difficulti f
```

```
inspect(benefits_train_corpus_new_load[213])
```

```
## <<SimpleCorpus>>
## Metadata:  corpus specific: 1, document level (indexed): 0
## Content:  documents: 1
##
## [1] overal eas mantal true benefit felt
```

2.2.17. Valores perdidos

Tras el proceso de limpieza anterior en un volumen tan grande de datos cabe esperar que algún documento (tuit) estuviera formado por tan solo palabras vacías, enlaces o combinaciones de estos, es por ello, que por medio de filtrado básico de R se obtienen aquellos que no contienen ninguna palabra y se elimina del conjunto del dataset para evitar problemas en los procesos posteriores.

```
# https://github.com/joseangeldiazg/twitter-text-mining/blob/master/ner.R
# Localizamos posibles valores perdidos que se hayan generado tras el proceso de limpieza
which(benefits_train_corpus_new_load$content=="")
```

```
## integer(0)
```

```
which(effects_train_corpus_new_load$content==" ")
```

```
## integer(0)
```

```
which(benefits_test_corpus_new_load$content=="  ")
```

```
## integer(0)
```

```
which(effects_test_corpus_new_load$content=="   ")
```

```
## integer(0)
```

```
# Vemos que no hay muchos vacios
benefits_train_corpus_new_load<-benefits_train_corpus_new_load[which(benefits_train_corpus_new_load$cont
effects_train_corpus_new_load<-effects_train_corpus_new_load[which(effects_train_corpus_new_load$conten
benefits_test_corpus_new_load<-benefits_test_corpus_new_load[which(benefits_test_corpus_new_load$conten
effects_test_corpus_new_load<-effects_test_corpus_new_load[which(effects_test_corpus_new_load$content!=
```

2.2.18. Borrar espacios en blanco innecesarios

Hasta el momento hemos hecho distintos cambios en el texto de nuestro dataset. No solo hemos modificado algunas palabras, sino que también hemos borrado otras muchas. Por ello, es adecuado asegurarnos que no hay más espacios en blanco que los que separan las palabras del texto. Para asegurarnos de ello, podemos ejecutar la siguiente orden, que se encarga de suprimir los espacios en blanco sobrantes.

```
benefits_train_corpus_new_load <- tm_map(benefits_train_corpus_new_load, stripWhitespace)
effects_train_corpus_new_load <- tm_map(effects_train_corpus_new_load, stripWhitespace)

benefits_test_corpus_new_load <- tm_map(benefits_test_corpus_new_load, stripWhitespace)
effects_test_corpus_new_load <- tm_map(effects_test_corpus_new_load, stripWhitespace)
```

2.2.19. Sparsity

También puede resultar muy útil eliminar los términos que aparecen en muy pocos documentos antes de proceder a la clasificación. El motivo principal es la factibilidad computacional, ya que este proceso reduce drásticamente el tamaño de la matriz sin perder información significativa. Además puede eliminar errores en los datos, como podrían ser palabras mal escritas. En el caso del clasificador la escasez es del 100 %, como se puede ver en la salida anterior de R.

```
## ARREGLAR FORMULA DA ERROR
# $$ df(t) > N · (1 != sparse)$$
```

Para suprimir estos términos, denominados escasos, se utiliza el comando removeSparseTerms(). Éste conserva todos los términos que cumplen

siendo df la frecuencia de documentos del término t y N el número de vectores. El parámetro sparse toma valores entre 0 y 1. En este caso, el umbral de escasez es 0,97, se toman los términos que aparecen en más del 3 % de documentos.

```
dtm <- DocumentTermMatrix(benefits_train_corpus_new_load)
inspect(dtm)
```

```
## <<DocumentTermMatrix (documents: 3104, terms: 5835)>>
## Non-/sparse entries: 52316/18059524
## Sparsity           : 100%
## Maximal term length: 33
## Weighting          : term frequency (tf)
## Sample             :
##       Terms
## Docs   day drug effect feel help medic pain take time work
##   1049   4    0      0    0    0     0    5    1    1    1
##   1189   4    0      0    0    0     0    5    1    1    1
##   1279   0    0      1    3    2     0    0    1    1    1
##   1824   1    0      0    3    1     0    0    1    2    0
##   2504   4    1      3    0    1     4    0    2    0    3
##   317    7    2      0    1    1     1    0    3    4    0
##   339    1    1      5    1    2     6    0    2    0    1
##   462    4    1      3    0    1     4    0    2    0    3
##   565    3    2      2    3    0     0    6    4    0    1
##   665    3    4      0    3    0     0    0    1    1    0
```

```
dtm <- removeSparseTerms(dtm, sparse=0.999)
inspect(dtm)
```

```
## <<DocumentTermMatrix (documents: 3104, terms: 1702)>>
## Non-/sparse entries: 46811/5236197
## Sparsity           : 99%
## Maximal term length: 17
## Weighting          : term frequency (tf)
## Sample             :
##       Terms
## Docs   day drug effect feel help medic pain take time work
##   1049   4    0      0    0    0     0    5    1    1    1
##   1189   4    0      0    0    0     0    5    1    1    1
##   1279   0    0      1    3    2     0    0    1    1    1
##   1824   1    0      0    3    1     0    0    1    2    0
##   2504   4    1      3    0    1     4    0    2    0    3
##   317    7    2      0    1    1     1    0    3    4    0
##   339    1    1      5    1    2     6    0    2    0    1
##   462    4    1      3    0    1     4    0    2    0    3
##   565    3    2      2    3    0     0    6    4    0    1
##   665    3    4      0    3    0     0    0    1    1    0
```

Se observa como de x términos pasamos a x términos, ni un 5 % del total, por tanto se ha considerado despreciar aplicar esta técnica. Por tanto, ya tenemos nuestros datos listos.

2.2.20. Convertir a dataframe

```
benefits_train_dataframe <- data.frame(text=unlist(sapply(benefits_train_corpus_new_load, `[`)), strings
effects_train_dataframe <- data.frame(text=unlist(sapply(effects_train_corpus_new_load, `[`)), stringsAs

benefits_test_dataframe <- data.frame(text=unlist(sapply(benefits_test_corpus_new_load, `[`)), stringsAs
effects_test_dataframe <- data.frame(text=unlist(sapply(effects_test_corpus_new_load, `[`)), stringsAsFa

# Añadimos las dos filas train al conjunto
library(data.table)
```

```
##
## Attaching package: 'data.table'
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
datos_train_preprocesado = data.frame(datos_train, benefits_train_dataframe)
setnames(datos_train_preprocesado, "text", "benefits_preprocesado")
datos_train_preprocesado = data.frame(datos_train_preprocesado, effects_train_dataframe)
setnames(datos_train_preprocesado, "text", "effects_preprocesado")

# Añadimos las dos filas train al conjunto
library(data.table)
datos_test_preprocesado = data.frame(datos_test, benefits_test_dataframe)
setnames(datos_test_preprocesado, "text", "benefits_preprocesado")
datos_test_preprocesado = data.frame(datos_test_preprocesado, effects_test_dataframe)
setnames(datos_test_preprocesado, "text", "effects_preprocesado")

# Guardamos en ficheros
library(data.table)
```

```
write.csv(datos_train_preprocesado, file = "datos/datos_train_preprocesado.csv", row.names = FALSE) # g
#View(datos_train_preprocesado)

write.csv(datos_test_preprocesado, file = "datos/datos_test_preprocesado.csv", row.names = FALSE) # gua
#View(datos_test_preprocesado)

d1 <- read.csv(file="datos/datos_train_preprocesado.csv")
```

2.2.21. Term Document Matrix

Ahora vamos a mapear nuestro corpus creando una matriz de términos, donde las filas corresponden a los documentos y las columnas a los términos. Para ello usaremos la función TermDocumentMatrix:

```
matrix_corpus <- TermDocumentMatrix(benefits_train_corpus_new_load)
```

Podemos observar que tenemos 5838 términos, esto quiere decir que tenemos 5838 palabras diferentes en nuestro Corpus. Obtengamos la *frecuencia de las palabras*:

```
class(matrix_corpus)
```

```
## [1] "TermDocumentMatrix"    "simple_triplet_matrix"
```

Como podemos ver, actualmente aún no tenemos nuestros datos en la matriz que buscamos, sino en un vector, por tanto:

```
matrix_corpus <- as.matrix(matrix_corpus)
class(matrix_corpus)
```

```
## [1] "matrix"
```

```
dim(matrix_corpus)
```

```
## [1] 5835 3104
```

Con este método, hemos obtenido la ocurrencia de las palabras que tenemos en nuestro dataset para cada uno de los documentos/comentarios. Esta matriz tiene 5838 columnas, que representa la totalidad de palabras diferentes que hay en los comentarios de la columna *benefitsReview*, y 3107 filas, donde cada una representa un comentario. Por tanto, en la fila iésima la matriz, tendremos la ocurrencia de las palabras en *benefitsReview* que existen en el comentario i.

```
# Sumamos las filas
suma_matrix_corpus <- rowSums(matrix_corpus)
head(suma_matrix_corpus,5)
```

```
##    agent     alon  congest dysfunct   failur
##        2       19       19        2        7
```

```
# Ordenanamos de mayor a menor y muestra los 10 primeros
ordena_mayor_matrix_corpus <- sort(suma_matrix_corpus, decreasing = TRUE)
head(ordena_mayor_matrix_corpus,10)
```

```
##   take effect   pain    day   help   drug   feel   time   work  medic
##    789    682    642    626    524    498    479    450    404    399
```

```
copia_ordena_mayor = ordena_mayor_matrix_corpus # Para graficos (evitando data.frame)

# Ordenanamos de menor a mayor y muestra los 10 primeros
```

```
ordena_menor_matrix_corpus <- sort(suma_matrix_corpus, decreasing = FALSE)
head(ordena_menor_matrix_corpus,10)
```

```
##         mangag          overt    ventricular            con           pros
##              1              1              1              1              1
##        ponstel          frank       valerian allergiesirrit          dryer
##              1              1              1              1              1
```

```
# Transformamos a objeto data.frame, con dos columnas (palabra, frec), para posteriormente graficarlo.
ordena_mayor_matrix_corpus <- data.frame(palabra = names(ordena_mayor_matrix_corpus), frec = ordena_mayo
```

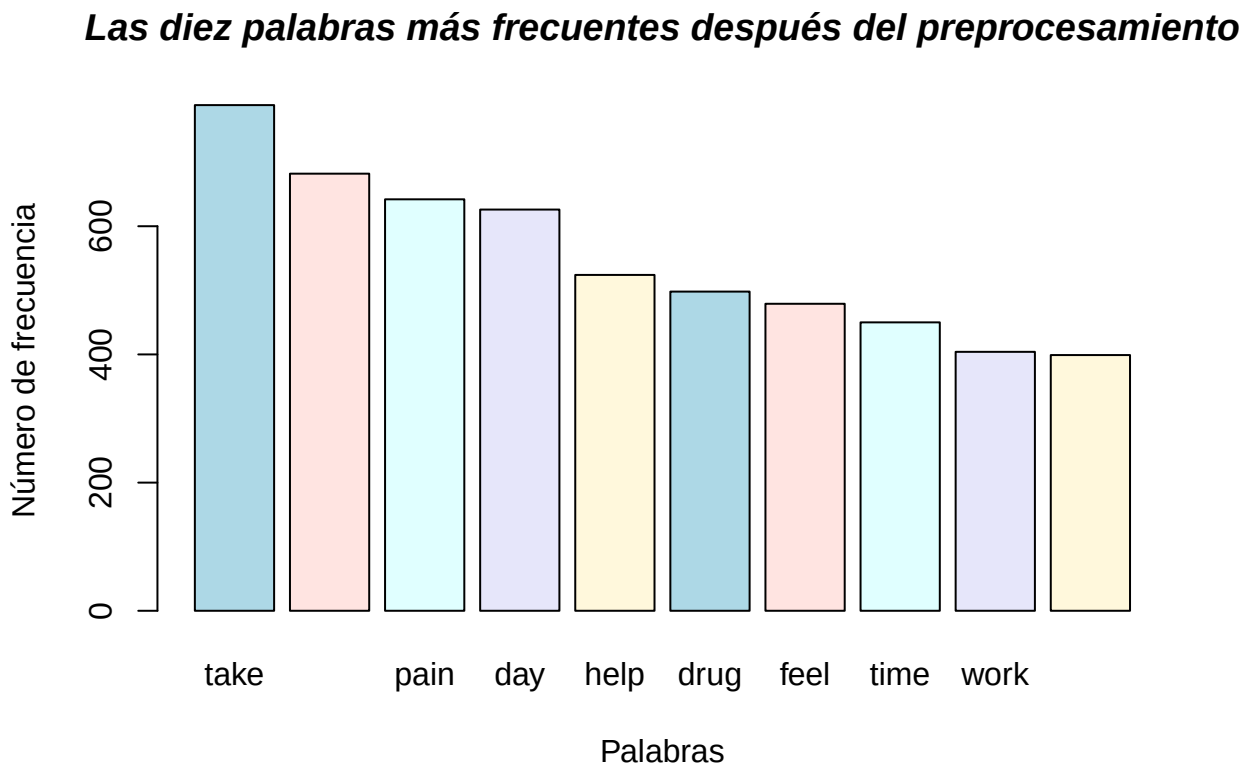
Mostramos las más frecuentes:

```
ordena_mayor_matrix_corpus[1:20,]
```

```
##             palabra frec
## take           take  789
## effect       effect  682
## pain           pain  642
## day             day  626
## help           help  524
## drug           drug  498
## feel           feel  479
## time           time  450
## work           work  404
## medic         medic  399
## also           also  394
## use             use  381
## sleep         sleep  381
## reduc         reduc  381
## year           year  369
## get             get  368
## skin           skin  358
## treatment treatment  357
## benefit     benefit  353
## depress     depress  349
```

Y obtenemos la *gráfica*:

```
copia_ordena_mayor <- as.matrix(copia_ordena_mayor)
barplot(copia_ordena_mayor[1:10,],  xlab="Palabras", ylab="Número de frecuencia",
        col = c("lightblue", "mistyrose", "lightcyan",
                "lavender", "cornsilk"))
title(main = list("Las diez palabras más frecuentes después del preprocesamiento", font = 4))
```



2.2.22. Nube de palabras

```
# instalar paquete worldcloud

#wordcloud(
#  words = ordena_mayor_matrix_corpus$palabra,
#  freq = ordena_mayor_matrix_corpus$frec,
#  max.words = 80,
#  random.order = F,
#  colors=brewer.pal(name = "Dark2", n = 8)
#  )


# cogemos el fichero ya preprprocesado

d1 <- read.csv(file="datos/datos_train_preprocesado.csv")

d1$items <- as.character(d1$benefits_preprocesado) # ordena_mayor_matrix_corpus

library(tidytext)
library(wordcloud)
library(dplyr)

cloud_negative_positive <- function(data){

  drugstext <- unnest_tokens(data, word, items)

  binarytextscore <- get_sentiments(lexicon = "bing")
```

```
  drugscloudbinary <- drugstext %>%
    inner_join(binarytextscore, by = "word") %>%
    count(word, sentiment) %>%
    mutate(color = ifelse(sentiment == "positive", "darkgreen", "red"))

  drugscloudbinary
}

res <- cloud_negative_positive(d1)
wordcloud(res$word, res$n, random.order = FALSE, colors = res$color, ordered.colors = TRUE)
```


```
# sin preprocesaro ls datos

datos_train <- read.table("datos/drugLibTrain_raw.tsv", sep="\t", comment.char="",quote = "\"", header=
datos_train$items <- as.character(datos_train$benefitsReview)
res <- cloud_negative_positive(datos_train)
wordcloud(res$word, res$n, random.order = FALSE, colors = res$color, ordered.colors = TRUE)
```


pain
depression
benefits
helped
like
better
symptoms
anxiety
infection
work
relief
improved
effective
cleared
headaches
well
severe
benefit
patient
panic
worked
improvement
problems
enough
good
clear
cold
allergies
allergy
loss
sore
tired
best
worse
happy
problem
inflammation
swelling
risk
smoke
cancer
fast
terrible
hurt
headache
chronic
painful
bad
nice
heal
slowed
upset
restless
proven
safe
hard
unfortunately
cure
lack
stuck
gain
worth
eased
sever
lighter
failure
rapid
failed
sensation
smoother
worst
irritable
elimination
drastically
clean
loose
beautiful
hopelessness
irritated
smooth
tingling
bleeding
helpful
depressed
wrinkles
fall
great
hot
fat
lose
flare
lost
dull
ease
sick
calm
suicide
shortness
swollen
obsessive
progress
stiff
freak
jitters
loved
marginal
remission
restored
poor
ineffective
sad
excellent
fatigued
firmer
consistent
ignore
sadness
missed
burning
suffering
syndrome
uncomfortable
tremendously
dust
concerned
wonders
lazy
calming
confidence
debilitating
wonderful
concern
calmness
quicker
pale
weakness
convenient
badly
limited
soft
split
restful
successful
lethargic
emergency
stresses
fantastic
darker
intense
thicker
allergic
healthy
risks
suicidal
stronger
manic
disorder
cramping
anxious
itching
important
bumps
energetic
vivid
boost
glow
manageable
useless
warned
greasy
willing
crash
dead
soreness
miss
tension
fail
positive
died
worry
irritation
available
charm
tolerable
strong
infected
stability
adverse
clears
issue
unable
fever
instantly
awful
dark
warm
thank
proper
smoke
bloated
inability
angry
fairly
risk
swelling
fair
hurt
terrible
effectiveness
beneficial
clearly
success
rough
aching
congestion
slow
motivated
saver
drain
problem
boil
crazy
awesome
anger
falling
free
love
fell
lie
burn
virus
led
pretty
faint
recovery
difficult
fatigue
perfect
rash
scar
attack
scared
recommend
injury
clarity
miracle
leading
refreshed
significant
worried
gained
temper
illness
sensitive
limits
miserable
vomiting
stable
confident
doubt
recommended
symptom
difficulty
remarkable
uneven
exhausted
broken
cry
appropriate
useful
addicted
quiet
tough
drastic
productive
liked
discontinued
fun
relaxed
suffered
attacks
stress
break
effectively
easier
fine
clearer
improve
negative
properly
infections
damage
stiffness
top
nervous
pains
faster
enjoy
numb
horrible
comfortable
wonder
nervousness
harsh
irritating
dizzy
wrong
happiness
stressful
afraid
hairloss
comfort
excessive
glad
zombie
ready
amazing
happier
thankful
blister
mania
consistently
balanced
severity
worrying
aches
ache
itch
perfectly
weak
variety
softer
brighter
smell
pleased
trouble
dangerous
stuffy
issues
fear
lead
losing
slowly
die
discomfort
work
enough
relief
benefits
helped
like
depression
pain
better
symptoms
anxiety
infection
work