

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

Descripción de los paquetes necesarios	1
1. Comprender el problema a resolver	3
2. Preprocesamiento de datos	4
2.1. Lectura del dataset	5
2.1.1. Lectura de datos train	5
2.1.2. Lectura de datos test	5
2.2. Preprocesamiento de los datos	6
2.2.1. Eliminar columnas	6
2.2.2. Eliminar filas	7
2.2.3. Eliminar elementos repetidos (medicamentos)	7
2.2.4. Cuantificación de variables	7
2.2.5. Cálculo del rating ponderado	9
2.2.6. Convertir el rating a variable binaria	10
2.2.7. Cambiar el orden para la columna sideEffectsNumber	11
2.2.8. Representación gráfica de los datos	11
2.2.9. Creación del corpus	13
2.2.10. Correlación	14
2.2.11. Representación gráfica de las frecuencias del Corpus	14
2.2.12. Eliminar signos de puntuación	16
2.2.13. Conversión de las mayúsculas en minúsculas	16
2.2.14. Eliminación de Stopwords	17
2.2.15. Agrupación de sinónimos	18
2.2.16. TF-IDF	19
2.2.17. Stemming	21
2.2.18. Valores perdidos	22
2.2.19. Borrar espacios en blanco innecesarios	23
2.2.20. Sparsity	23
2.2.21. Matriz de documentos de los términos	24
2.2.22. Nube de palabras	26
2.2.23. Convertir a dataframe nuestras modificaciones	26
3. Análisis exploratorio de los datos	28
3.1. Valoraciones de los medicamentos por parte de los usuarios.	29
3.2. Efectividad del medicamento	33
3.3. Efectos secundarios del medicamento	37
3.4. Valoración ponderada sobre el medicamento	40
3.5. Correlación sobre las variables	44
4. Análisis de sentimientos	46
5. Reglas de Asociación	49
5.1. Reglas de asociación específicas	51
6. Agrupamiento y Clustering	59
6.1. Introducción	59
6.2. Agrupamiento de los fármacos en base a las puntuaciones obtenidas	59
6.2.1. Establecimiento de una semilla	59
6.2.2. Lectura y adecuación de los datos	59
6.2.3. K-medias	62
6.2.4. K-medoides Clustering	70
6.2.5. Clustering Difuso (FUZZY CLUSTERING)	77

6.2.6. Clustering Jerárquico	79
6.2.7. Agrupamiento jerárquico y K-means	86
6.3. Clustering sobre datos de texto	87
6.3.1. Clustering de Benefits Review	87
6.4. Density based clustering (DBSCAN)	94
6.5. KNN	96
6.6. Árboles de decisión	102
6.6.1. Predicción de effectivenessNumber en función del rating	103
6.6.2. Predicción de sideEffectsNumber en función del rating	108
6.6.3. Predicción de effectivenessNumber en función del weightedRating	110
6.6.4. Conclusión	113
6.6.5. Predicción del rating en función del comentario de beneficios	114
7. Árboles de decisión y Clasificación	116
7.1. Naive Bayes	116
7.2. SVM	121
8. Regresión	123
8.1. Regresión Lineal	124
8.1.1. Regresión Lineal Simple	124
8.1.2. Regresión Lineal Múltiple	132
8.2. Regresión Logística	133
8.2.1. Regresión Logística Simple	133
8.2.2. Regresión Logística Multivariable	135
8.2.3. Regularización en Regresión Logística	136
8.3. Regresión Polinomial	138
9. Conclusiones	141

Índice de figuras

1.	Medicamento "paxil" repetido 20 veces en el conjunto test	7
2.	Clasificación del medicamento por parte del paciente	12
3.	Clasificación de los efectos secundarios del medicamento según el paciente	12
4.	Clasificación de la efectividad del medicamento según el paciente	13
5.	Contenido para benefits_train_corpus I	14
6.	Contenido para benefits_train_corpus II	14
7.	Frecuencia de términos para la columna benefitsReview	15
8.	Frecuencia de términos para la columna sideEffectsReview	15
9.	Contenido de benefits sin signos de puntuación	16
10.	Contenido de effects sin signos de puntuación	16
11.	Contenido de benefits sin mayúsculas	17
12.	Contenido de effects sin mayúsculas	17
13.	Contenido de benefits sin stopwords	17
14.	Contenido de effects sin stopwords	18
15.	Wordcloud con preprocesamiento	27
16.	Wordcloud sin preprocesamiento	28
17.	Ánálisis de sentimientos de los comentarios sobre los beneficios de los medicamentos para obtener propecia.	47
18.	Ánálisis de positivismo de los comentarios sobre los beneficios de los medicamentos para obtener propecia.	48
19.	inspect(head(rules_benefits),100) para visualizar reglas más importantes de los comentarios sobre beneficios.	50
20.	inspect(head(rules_effects),100) para visualizar reglas más importantes de los comentarios sobre efectos secundarios.	50
21.	Mapa de reglas de asociación sobre los comentarios de beneficios.	51
22.	Mapa de regla de asociación sobre los comentarios de efectos secundarios.	52
23.	Ilustración de preprocesamiento específico para reglas de asociación acotada a efectividad. . .	53
24.	Mapa de reglas de asociación con Inneffective en el consecuente en comentarios sobre beneficios. .	57
25.	Mapa de reglas de asociación con Highly-Effective en el consecuente en comentarios sobre efectos secundarios.	58
26.	Matriz de pesos para cada uno de los términos en cada uno de los documentos.	89
27.	Agrupamiento particional por k-means.	92
28.	Agrupamiento jerárquico y muestra de anidamiento con los 100 términos más frecuentes. . . .	93
29.	Agrupamiento particional por densidad.	94
30.	Matriz de confusión y tasa de precisión en modelo KNN con beneficios (k=1).	98
31.	Matriz de confusión y tasa de precisión en modelo KNN con beneficios (k=3).	98
32.	Matriz de confusión y tasa de precisión en modelo KNN con beneficios (k=5).	98
33.	Matriz de confusión y tasa de precisión en modelo KNN con beneficios (k=10).	99
34.	Matriz de confusión y tasa de precisión en modelo KNN con beneficios (k=15).	99
35.	Matriz de confusión y tasa de precisión en modelo KNN con beneficios (k=30).	99
36.	Matriz de confusión y tasa de precisión en modelo KNN con beneficios (k=50).	100
37.	Gráfico del progreso del modelo KNN en función del valor de k.	100
38.	Matriz de confusión y tasa de precisión en modelo KNN con efectos secundarios (k=1). . . .	101
39.	Matriz de confusión y tasa de precisión en modelo KNN con efectos secundarios (k=3). . . .	101
40.	Matriz de confusión y tasa de precisión en modelo KNN con efectos secundarios (k=5). . . .	102
41.	Matriz de confusión y tasa de precisión en modelo KNN con efectos secundarios (k=15). . . .	102
42.	Matriz de confusión y error para la predicción del nivel de efectividad de los medicamentos a partir de comentarios sobre beneficios.	118
43.	Matriz de confusión y error para la predicción del nivel de efectividad de los medicamentos a partir de comentarios sobre efectos secundarios.	119
44.	Matriz de confusión y error para la predicción del ratingLabel de los medicamentos a partir de comentarios sobre beneficios.	120

45.	Matriz de confusión y error para la predicción del ratingLabel de los medicamentos a partir de comentarios sobre efectos secundarios.	120
46.	Salida de SVM para el ratingLabel a partir de los comentarios de texto sobre los beneficios. .	122
47.	Visualización de las columnas a las que aplica regresión	123
48.	Matriz de confusión para clasificador binario.	123

Índice de cuadros

1.	Información del conjunto de datos	3
2.	Información contenida en una fila del conjunto de entrenamiento I	5
3.	Información contenida en una fila del conjunto de entrenamiento II	5
4.	Información contenida en una fila del conjunto de prueba I	6
5.	Información contenida en una fila del conjunto de prueba II	6

Descripción de los paquetes necesarios

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

- **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")`.
- **car** : Paquete que nos proporciona distintas funciones. : `install.packages("car")`.
- **cluster** : Paquete que nos proporciona métodos para el análisis de clusters. : `install.packages("cluster")`.
- **caret** : Paquete para entrenamiento de clasificación y regresión. Se puede instalar usando : `install.packages("caret")`.
- **dbscan** : Paquete que proporciona implementaciones de varios algoritmos basados en densidad de la familia DBSCAN para datos espaciales. Se puede instalar usando : `install.packages("dbscan")`.
- **devtools** : Paquete que contiene una colección de herramientas de desarrollo de paquetes, usando conjuntamiento con **rword2vec**, para obtener la agrupación de sinónimos. Se puede instalar usando : `install.packages("devtools")`.
- **dplyr** : Paquete que agiliza el trabajo con los datos. Se puede instalar usando : `install.packages("dplyr")`.
- **e1071** : Paquete para realizar *fuzzy clustering*, clasificador de *Naive Bayes*... Se puede instalar usando : `install.packages("e1071")`.
- **fclust** : Paquete que nos proporciona algoritmos para la agrupación difusa, índices de validez de clusters y gráficos para la validez de estos. Además de la visualización de los resultados de la agrupación difusa. : `install.packages("fclust")`.
- **factoextra** : Proporciona algunas funciones sencillas para extraer y visualizar la producción de análisis de datos multivariados. : `install.packages("factoextra")`.
- **ggpubr** : Paquete que proporciona algunas funciones de fácil manejo para crear y personalizar gráficos listos para ser usados en **ggplot2**. : `install.packages("ggpubr")`.
- **ggplot2** : Paquete para realizar gráficas. Se puede instalar usando : `install.packages("ggplot2")`.
- **glmnet** : Paquete que nos proporciona procedimientos eficaces para la adaptación de la red para la regresión lineal, los modelos de regresión logística y multinomial, la regresión de Poisson y el modelo de Cox. : `install.packages("glmnet")`.
- **igraph** : Paquete que nos proporciona procesos para gráficos simples y análisis de redes. Puede manejar gráficos grandes muy bien, gráficos aleatorios y gráficos regulares, además de visualización de estos. : `install.packages("igraph")`.
- **MASS** : Paquete que nos proporciona funciones y conjuntos de datos de soporte. : `install.packages("MASS")`.
- **magrittr** : Paquete que proporciona un mecanismo para encadenar comandos con `%>%`. Se puede instalar usando : `install.packages("magrittr")`.
- **NLP** : Paquete con clases básicas y métodos para el procesamiento del lenguaje natural. Se puede instalar usando : `install.packages("NLP")`.
- **ppclust** : Paquete que nos permite la agrupación de particiones de un conjunto de datos en subconjuntos o clusters no superpuestos mediante el uso de los algoritmos de agrupación probabilística basados en prototipos. : `install.packages("ppclust")`.

- **plyr** : Paquete que nos ofrece herramientas para dividir, aplicar y combinar datos. Se puede instalar usando : `install.packages("plyr")`.
- **proxy** : Paquete que proporciona un *framework* para cálculo eficiente. Se puede instalar usando : `install.packages("proxy")`.
- **quanteda** : Paquete para análisis cuantitativo de texto en R, usado para gestionar corpus, crear y manipular tokens y n-gramas, analizar palabras clave, etc, representando visualmente el texto y los análisis de texto. Se puede instalar usando : `install.packages("quanteda")`.
- **rlm** : Paquete que nos proporciona una adaptación robusta de un modelo lineal que puede responder en forma de matriz. : `install.packages("rlm")`.
- **rattle** : Paquete que permite al usuario cargar rápidamente datos desde un archivo CSV, transformarlos y explorarlos, construir y evaluar modelos, y además, exportarlos. : `install.packages("rattle")`.
- **randomForest** : Paquete que nos proporciona clasificación y regresión basada en árboles usando entradas aleatorias. : `install.packages("randomForest")`.
- **RColorBrewer** : Paquete que proporciona paletas de colores. Se puede instalar usando : `install.packages("wordcloud")`.
- **rlist** : Paquete que proporciona un conjunto de funciones para la manipulación de datos en forma de lista. Se puede instalar usando : `install.packages("rlist")`.
- **ROCR** : Paquete que nos permite hacer uso de gráficos ROC (curva ROC), entre otros. Se puede instalar usando : `install.packages("ROCR")`.
- **RTextTools** : Paquete para realizar clasificación automática de textos mediante aprendizaje supervisado. Se puede instalar usando : `install.packages("RTextTools")`.
- **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")`.
- **stargazer** : Paquete que produce código LaTeX, código HTML/CSS y texto ASCII para tablas bien formateadas que contienen resultados del análisis de regresión de varios modelos en paralelo, así como un resumen de estadísticas. : `install.packages("stargazer")`.
- **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")`.
- **tidytext** : Paquete para minería de textos para procesamiento de textos y análisis de sentimientos usando dplyr, ggplot2, y otras herramientas ordenadas. Se puede instalar usando : `install.packages("tidytext")`.
- **tm** : Paquete específico para minería de datos, permite procesar datos de tipo texto. Se puede instalar usando : `install.packages("tm")`.
- **tseries** : Paquete para análisis de series temporales y computación financiera. Se puede instalar usando : `install.packages("tseries")`.
- **wesanderson** : Paquete que proporciona paletas de colores. Se puede instalar usando : `install.packages("wesanderson")`.
- **wordcloud** : Paquete para crear gráficas de nubes de palabras, permitiendo visualizar las diferencias y similitudes entre documentos. Se puede instalar usando : `install.packages("wordcloud")`.

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, valor que representa una puntuación, por parte del paciente, en la que tiene en cuenta todos los factores (tanto positivos como negativos). Además, constará 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.
- Comparar la efectividad o efectos secundarios del medicamento, de acuerdo a la puntuación del medicamento según el paciente, y sacar toda la información que podamos de dicha relación (en caso de existir).
- Obtener dependencias y patrones existentes entre los distintos fármacos, que nos permita conocer aspectos sobre ellos o incluso realizar agrupaciones, como puede ser, en positivos o negativos desde el punto de vista del paciente.
- 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	16759

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 leídos, se procederán a aplicar las siguientes técnicas con el fin de limpiar los datos:

1. *Lectura del dataset a usar.*
2. *Eliminar las columnas* que no aportan información relevante.
3. *Eliminar las filas* que contienen caracteres raros y perjudican al análisis.
4. *Eliminar los medicamentos repetidos* en nuestro dataset.
5. *Cuantificación de variables* para la obtención de las columnas con etiquetas numéricas.
6. *Cálculo del rating ponderado* para la obtención de una valoración general del medicamento.
7. *Conversión de rating a variable binaria*, para ser usada como etiqueta.
8. *Cambiar el orden para la columna sideEffectsNumber*, con el fin de obtener el mismo orden en efectividad y efectos secundarios.
9. *Representación gráfica de los datos.*
10. *Creación del corpus* para las columnas benefitsReview y sideEffectsReview.
11. *Correlación* entre las variables del dataset.
12. *Representación gráfica de las frecuencias del Corpus.*
13. *Eliminar signos de puntuación.*
14. *Conversión de mayúsculas a minúsculas.*
15. *Eliminación de palabras no aportan información relevante (stopwords).*
16. *Agrupación de sinónimos.*
17. Cálculo de la frecuencia de cada término con *TF-IDF*.
18. Reducción de palabras con *stemming*.
19. Obtención y eliminación de *valores perdidos*.
20. *Borrar espacios en blanco innecesarios.*
21. Eliminar los términos que aparecen en muy pocos documentos (*sparsity*).
22. Obtención de la *matriz de términos*.
23. Creación de la *nube de palabras* para las columnas sin y con preprocesamiento.

2.1. Lectura del dataset

A continuación, mediante la función `read.table(...)` procedemos a la lectura de los datos explicados previamente:

2.1.1. Lectura de datos train

Se va a proceder 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 cómo está distribuida la información. Por ejemplo, para la tercera fila 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 los cuadros 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 actividades cotidianas 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 proceder 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 cómo está distribuida la información. Por ejemplo, para la primera fila 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 nivel considerable de efectividad (**Considerably**

Effective) con efectos secundarios leves (**Mild Side Effects**), usado para la infección sinusal (**sinus infection**), 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

Una vez leídos nuestros datos, procedemos a la transformación y preprocesamiento de los mismos. En donde 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. Preprocesamiento de los datos

Dado que la representación total del documento puede tener una alta dimensión, se va a proceder a construir un corpus, necesario para la aplicación de métodos de limpieza y estructuración 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**. En concreto, nos referimos a aquellas que contienen información irrelevante para nuestro análisis.

Eliminar columna ID

Es importante mencionar, que 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 quitarlo directamente del *dataframe*. Esta columna se corresponde con la primera que aparece, con 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 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

Consideraremos 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 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 no aportan información a nuestro análisis. Asimismo, dichas filas perjudicaban la aplicación de técnicas. Al tener un dataset grande, esta modificación se puede llevar a cabo sin problema alguno.

- 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*, tan solo había tres guiones.

```
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 (medicamentos)

Como se puede ver a continuación, existen medicamentos repetidos. Sin embargo, no se ha realizado a priori, debido a que le damos más importancia a las opiniones de los pacientes, con el fin de obtener la efectividad o efectos secundarios del medicamento.

urlDrugName	rating	effectiveness	sideEffects	condition
43 paxil	8	Considerably Effective	Mild Side Effects	depression
70 paxil	1	Ineffective	Severe Side Effects	depression
127 paxil	7	Considerably Effective	Moderate Side Effects	depression
155 paxil	1	Ineffective	Extremely Severe Side Effects	severe depression
180 paxil	1	Ineffective	Extremely Severe Side Effects	ocd
207 paxil	8	Highly Effective	Moderate Side Effects	stress and depression
384 paxil	8	Highly Effective	No Side Effects	anxiety and depression
399 paxil	10	Highly Effective	Mild Side Effects	anxiety
451 paxil	8	Considerably Effective	Mild Side Effects	mild depression
453 paxil	1	Ineffective	Severe Side Effects	depression

Figura 1: Medicamento "paxil" repetido 20 veces en el conjunto test

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 un valor con el que nos sea más sencillo trabajar. En otras palabras, vamos a 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$sideEffects, 5)

## [1] Mild Side Effects    Severe Side Effects  No Side Effects
## [4] Mild Side Effects    Severe Side Effects
## 5 Levels: Extremely Severe Side Effects ... Severe Side Effects

head(datos_train$sideEffectsNumber, 5)

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

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. Asignaremos la etiqueta con valor 1 para indicar que el medicamento es ineficaz, si este es altamente eficaz, le asignaremos la etiqueta con valor 5:

- 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$effectiveness, 5)
## [1] Highly Effective    Highly Effective    Highly Effective
## [4] Marginally Effective Marginally Effective
## 5 Levels: Considerably Effective Highly Effective ... Moderately Effective
head(datos_train$effectivenessNumber, 5)
## [1] 5 5 5 2 2

```

2.2.5. Cálculo del rating ponderado

En este subapartado, se va a realizar una agregación de varias columnas, en donde queremos realizar una valoración general del medicamento. Para ello, se va a realizar una ponderación entre la columna que contiene los efectos secundarios y la efectividad del medicamento. Asimismo, se ha considerado los efectos secundarios del medicamento con mayor importancia, es por eso que se le ha otorgado una ponderación del 70 % frente al 30 % de la efectividad del medicamento.

$$(sideEffects \cdot 07) + (effectiveness \cdot 03)$$

El motivo de esta ponderación es que se considera que es peor tener efectos secundarios severos en un medicamento, que ser efectivo. Dicha agregación se ha añadido a una nueva columna, denominada **weightedRating**, cuyo resultado contiene una valoración general del medicamento. En donde, dicha transformación, puede ser usada para realizar una comparación con las propias valoraciones de los usuarios. Este cambio, nos permitirá trabajar posteriormente de una manera más sencilla y simplificada.

```

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

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

  # Inicializamos la variable
  sideEffectRating <- 0

  # Convertimos el valor de sideEffect a la misma escala que effectiveness, ya que
  # sideEffect == 1 significa que no tiene efectos secundarios (lo cual es bueno),
  # y effectiveness == 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 * 0.7 + sideEffectRating * 0.3
}

```

```

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

Ahora realizamos el mismo procesamiento para el conjunto test. Una vez realizada la modificación, comprobamos que se ha creado la nueva columna *weightedRating*, y además que se han añadido los cambios comentados.

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

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

2.2.6. Convertir el rating a variable binaria

Para la realización de algunas técnicas que se explicarán más adelante, se necesita tener una etiqueta o variable binaria comprendida entre 0 y 1. Es por eso que se ha optado por escoger la columna que contiene la puntuación del medicamento por parte del paciente, y establecerla entre 0 y 1. En donde el 0, tendrá los valores comprendidos entre 1 y 4 y nos indicará que el medicamento no es favorable; y en donde el 1, tendrá los valores comprendidos entre 5 y 10 indicándonos que el medicamento es favorable.

Dichos cambios, serán añadidos a una nueva columna llamada **ratingLabel**. Realizamos dicho cambio, tanto para test como train.

```

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

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

  # Valores comprendidos entre 1 y 4 - no favorable - 0
  if (datos_train$rating[i] < 5) result <- 0
  # Valores comprendidos entre 5 y 10 - favorable - 1
  else if (datos_train$rating[i] >= 5)
    result <- 1

  # Asignamos el resultado a la nueva variable
  datos_train$ratingLabel[i] <- result
}

```

2.2.7. Cambiar el orden para la columna sideEffectsNumber

Al comparar las columnas *sideEffectsNumber* y *effectivenessNumber*, llegamos a la conclusión, de que ambas siguen órdenes distintos, es decir, el valor 5 en *sideEffectsNumber* dice que el medicamento tiene efectos secundarios extremadamente graves, y el valor 5 en *effectivenessNumber* dice que el medicamento es altamente efectivo. Por tanto, se ha considerado, que el valor correspondiente al 5, sea positivo y el valor correspondiente a 1 negativo. Para ello, se ha modificado el orden de la columna *sideEffectsNumber*, en donde, la nueva columna **sideEffectsInverse** tiene:

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

Realizamos dicha modificación, tanto para *train* como *test*. Con ello buscamos, obtener resultados con los que nos resulte más cómodo de trabajar.

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

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

  # Inicializamos la variable
  sideEffectRating <- 0

  # Cambiamos el orden para el el valor de sideEffect
  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
  datos_train$sideEffectsInverse[i] <- sideEffectRating
}

head(datos_train$sideEffectsNumber, 10)
## [1] 2 4 1 2 4 4 2 1 1 5
head(datos_train$sideEffectsInverse, 10)

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

2.2.8. Representación gráfica de los datos

Antes de comenzar con el análisis exploratorio, vamos realizar distintos diagramas de barras con el fin de comprender mejor los datos, previo al procesamiento. Primero realizaremos un diagrama de barras de las clasificaciones del medicamento por parte del paciente.

En la figura 2, se observa cómo 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 4, un alto porcentaje de los medicamentos no tiene efectos secundarios, de acuerdo a las valoraciones de los pacientes.

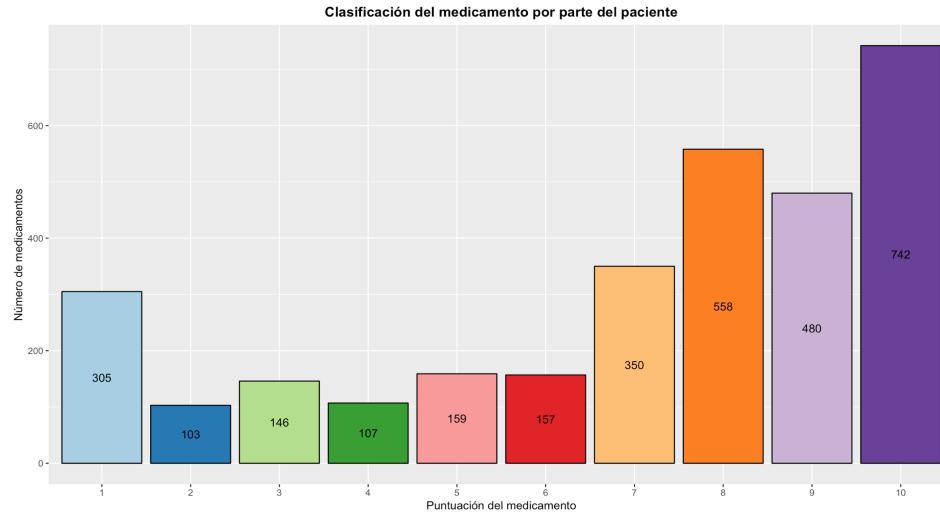


Figura 2: Clasificación del medicamento por parte del paciente

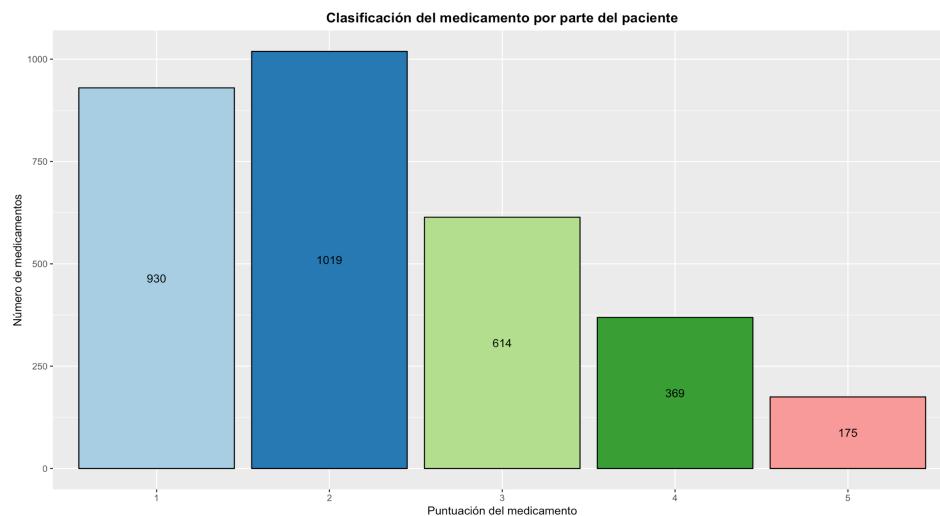


Figura 3: Clasificación de los efectos secundarios del medicamento según el paciente

Por último, vamos a visualizar gráficamente, la efectividad del medicamento según el paciente. En donde el 1 significa que el medicamento tiene poca efectividad y el 5 que tiene mucha efectividad. De acuerdo a la figura 4, es lógica esta observación, sobretodo teniendo en cuenta el hecho de que, para los medicamentos catalogados con pocos efectos secundarios, es de esperar que los pacientes tengan buena opinión de ellos y se traduzca en una alta efectividad.

Acabamos de comprobar, cómo 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

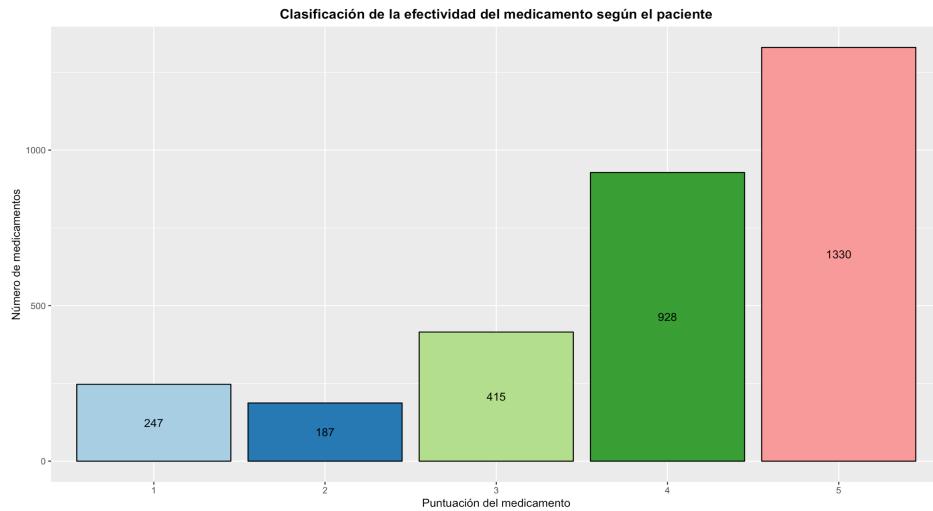


Figura 4: Clasificación de la efectividad del medicamento según el paciente

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*.

2.2.9. 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 las 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)

# 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])`:

```

<<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 5: Contenido para benefits_train_corpus I

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

```

[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 6: Contenido para benefits_train_corpus II

Como aspecto a destacar, si nos fijamos en el contenido, podemos ver que tiene signos de puntuación y exclamación.

2.2.10. Correlación

Una forma de medir la relación existente entre las variables en nuestro dataset, es calcular la correlación entre un término y todos los demás de la matriz. Como en este caso, estamos usando variables textuales, no se aconseja hacer la correlación como tal. Debido a que la correlación indica la fuerza y la dirección de una relación lineal y proporcionalidad entre dos variables estadísticas, ya que está pensada para variables cuantitativas. Por otro lado, sí que disponemos de variables categóricas, pero también se ha despreciado hacer la correlación entre ellas, debido a que no podemos prescindir de las etiquetas, ni agrupar distintos medicamentos en uno solo.

Además, la correlación está relacionada con el análisis de componentes principales (PCA), el cual, es una técnica utilizada para describir un conjunto de datos en términos de nuevas variables no correlacionadas. Por lo que se ha despreciado inicialmente realizar tal método.

2.2.11. Representación gráfica de las frecuencias del Corpus

Una vez creado el corpus y antes de aplicar las técnicas de preprocesamiento, 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.

Como se puede apreciar en la gráfica 7, los términos que más se repiten son *the* y *and*, además de otras preposiciones y conjunciones. Si se observa, las palabras que aparecen en la gráfica, no nos aportan información alguna, es por eso que vamos hacer una transformación a los términos, como la eliminación de los *stopwords*.

A continuación, mostramos los términos para la columna *sideEffects*, realizando el mismo procedimiento que antes. Y como se puede ver en la gráfica 8, obtenemos la misma conclusión, en donde tenemos palabras que no nos aportan nada de información: *very*, *the*, *that*, *and...*

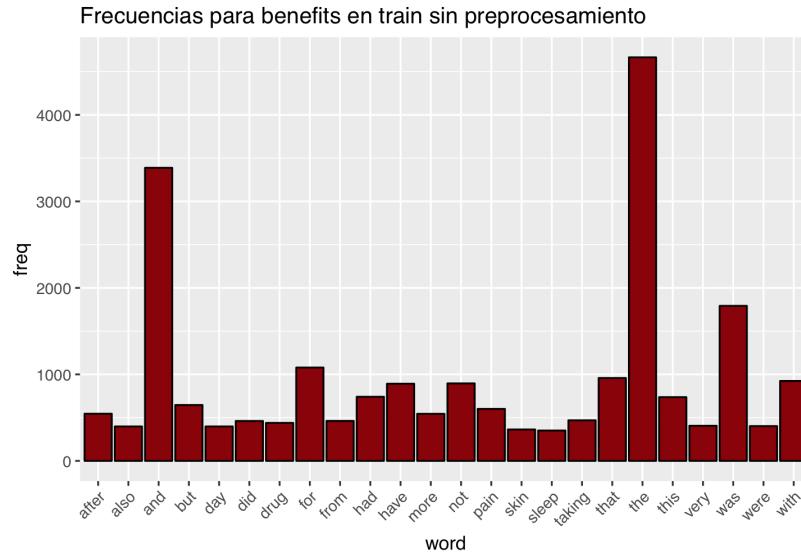


Figura 7: Frecuencia de términos para la columna benefitsReview

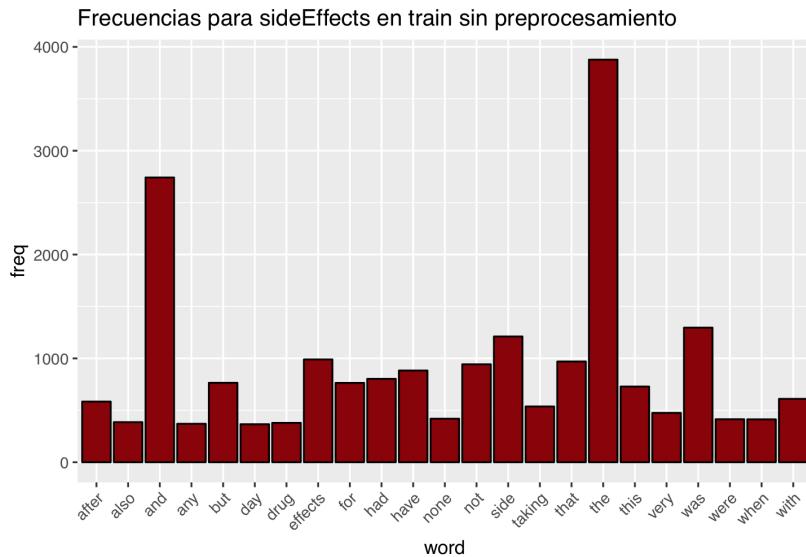


Figura 8: Frecuencia de términos para la columna sideEffectsReview

2.2.12. 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: ! " \$% & '() * + , - . / : ; < = > ? @ [] ^ _ ' { | } ~, tanto para train como test.

```
# 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 9: Contenido de benefits sin signos de puntuación

Ocurre lo mismo con el comentario de efecto número siete.

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

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

Figura 10: Contenido de effects sin signos de puntuación

2.2.13. Conversión de las mayúsculas en minúsculas

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

```
# 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))
```

Si volvemos a mostrar las opiniones, vemos como todas las mayúsculas han desaparecido.

```
<<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 11: Contenido de benefits sin mayúsculas

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

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

Figura 12: Contenido de effects sin mayúsculas

2.2.14. Eliminación de Stopwords

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

```
# 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"))
```

Si volvemos a mostrar las opiniones, vemos como por ejemplo las palabras como *the* o *and*, han desaparecido de nuestro corpus.

```
<<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 13: Contenido de benefits sin stopwords

Ahora ya hemos eliminado las *stopwords* de forma correcta, y pasamos a realizar la agrupación de sinónimos.

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

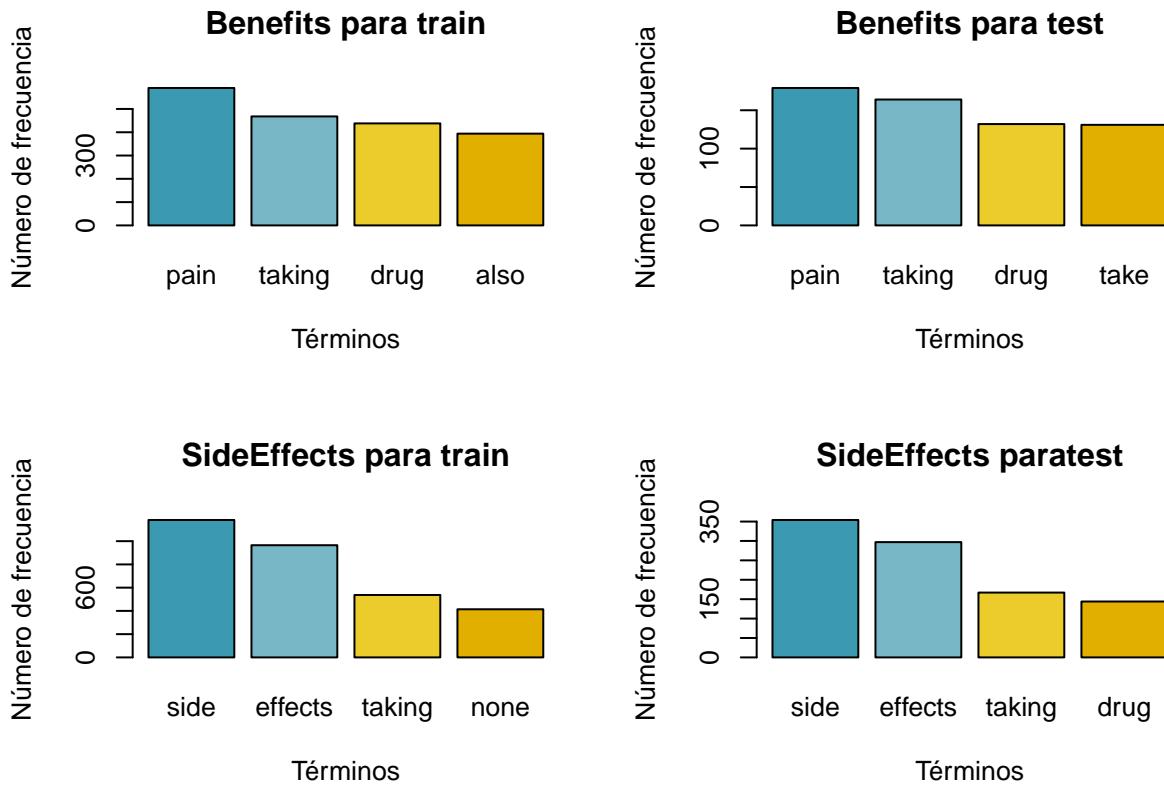
[1] experiences nausia heavy moodswings days take decreased appetite negative affect shortterm memory
```

Figura 14: Contenido de effects sin stopwords

2.2.15. 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 qué palabras son las que mayor frecuencia presentan en nuestro texto, tanto para `benefitsReview` como `sideEffectsReview` del conjunto train y test.

Visualizamos los 4 primeros términos que cumplen estas características para cada columna y conjunto train y test:



Una vez que tenemos los términos con mayor frecuencia en nuestras columnas (`benefitsReview` y `sideEffectsReview`) y su frecuencia asociada, pasamos a matriz dichos datos, con el fin de obtener únicamente las palabras y descartar su frecuencia asociada.

Como ya sabemos las palabras a usar, es decir, los términos que más se repiten, procedemos a la agrupación por sinónimos. En donde, mediante la función `distance(...)` de la librería `rword2vec`, obtendremos todas palabras más similares de nuestro conjunto. En nuestro caso nos vamos a quedar con las 2 primeras, tanto para `benefitsTrainReview` como `sideEffectsReview` del conjunto train y test. A continuación, se muestran los pasos seguidos.

1. Escribir un fichero los datos asociados a dicha columna.

2. Entrenar los datos del fichero, con el fin de obtener los vectores de palabras que nos darán las palabras más similares.
3. Obtener para cada término del documento, la distancia con los términos del ficheros, quedándonos con las de mayor frecuencia.
4. Guardar en un fichero dichas distancias.

Una vez, que tenemos todas las palabras con los 2 términos más similares, procedemos a sustituir todos esos términos por el término general. Veamos un ejemplo sencillo, en donde las palabras “medicine” o “medication”, van a ser sustituidas por “drug”.

```
# 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
```

Se debe tener en cuenta, que los términos más similares han sido creados únicamente para nuestros documentos. Por último, ya solo nos queda hacer el reemplazamiento, para ello se usará la función `gsub(...)` sobre el corpus (`benefits_corpus` y `sideEffectsReview`). Para sustituir las palabras en el texto, se ha hecho uso de la función `gsub(pattern, replacement, x, ignore.case = FALSE, perl = FALSE, fixed = FALSE, useBytes = FALSE)`. A continuación, se describe el proceso seguido:

1. Iteramos sobre los términos del documento.
2. Iteramos sobre las palabras más frecuentes de cada término.
3. Realizamos el reemplazamiento de las palabras más similares por los términos generales, previa conversión a minúsculas.
4. Guardamos los resultados en ficheros.

2.2.16. 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 Frequency*). 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:

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D)$$

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*. **De esta forma, obtenemos una forma de asignar a cada término un score en relación a cuán representativo es dicho término en nuestro conjunto de datos.**

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. Para ello, necesitamos que convertir nuestro corpus a un *dataframe*, en donde cada columna será una palabra del comentario y cada fila un comentario.

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:

1. Creamos el corpus utilizando el vector de string que hemos creado previamente.
2. Generamos la matriz de términos.
3. Convertimos a matriz.

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 en train
terms_benefits_train <- ( idf_benefits_train <- log( ncol(tf_benefits_train)
  / ( 1+rowSums(tf_benefits_train != 0)))))
terms_effects_train <- ( idf_effects_train <- log( ncol(tf_effects_train)
  / ( 1+rowSums(tf_effects_train != 0)))))

# Calculamos los pesos asociados a cada término en test
terms_benefits_test <- ( idf_benefits_test <- log( ncol(tf_benefits_test)
  / ( 1+rowSums(tf_benefits_test != 0)))))
terms_effects_test <- ( idf_effects_test <- log( ncol(tf_effects_test)
  / ( 1+rowSums(tf_effects_test != 0)))))

# Muestra los pesos asociados a cada término (los 5 primeros)
terms_benefits_train[1:5]

##      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.

1. Creamos la matriz diagonal: (`idf_benefits_train <- diag(idf_benefits_train)`)
2. Hacemos la operación para calcular tf_idf como el producto de $td \cdot idf$: `tf_idf_benefits_train<- crossprod(tf_benefits_train, idf_benefits_train)`
3. Guardamos los datos en ficheros, con el fin de reducir el tiempo de procesamiento.

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 normalización:

$$t_benefits_train = \frac{tf_idf_benefits_train_new}{\sqrt{(rowSums(tf_idf_benefits_train_new^2))}}$$

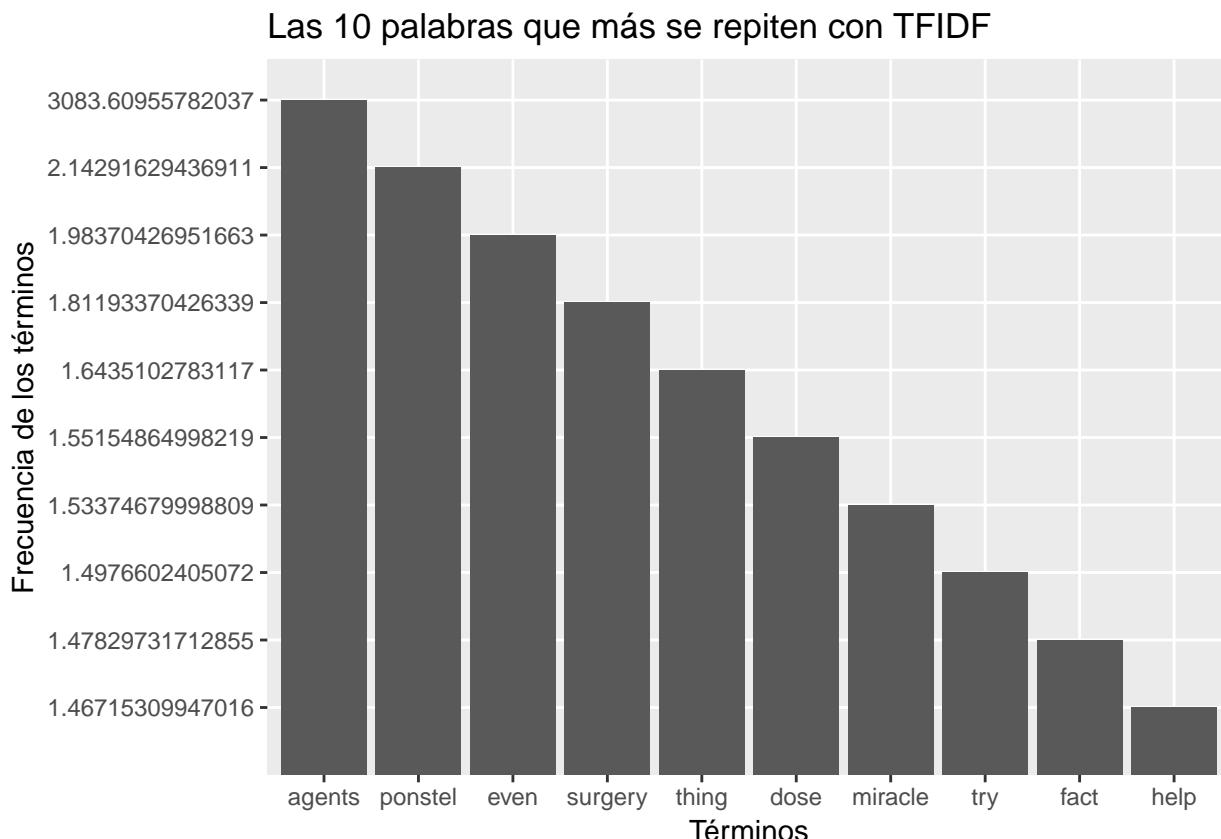
A continuación, vamos a visualizar los términos que más se repiten para el conjunto train de *benefits*:

```
# Graficamos los resultados para benefits train
ggplot() +
  geom_bar(data=z_benefits_train[1:10], ,
```

```

aes(x=z_benefits_train$terms1_5_new_benefits_train[1:10],
y=z_benefits_train$v_benefits_train[1:10]), stat='identity',
position='dodge') +
ggtitle("Las 10 palabras que más se repiten con TFIDF") +
xlab("Términos") +
ylab("Frecuencia de los términos")

```



2.2.17. 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. Por ejemplo, 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 respiratory

```

```
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 función:

```
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 cambio 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
```

2.2.18. Valores perdidos

Tras el proceso de limpieza anterior en un volumen tan grande de datos cabe esperar que algún documento 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$content=="")
which(effects_train_corpus_new_load$content==" ")
which(benefits_test_corpus_new_load$content==" ")
which(effects_test_corpus_new_load$content==" ")

# Vemos que no hay muchos vacíos
benefits_train_corpus_new_load<-benefits_train_corpus_new_load[
  which(benefits_train_corpus_new_load$content!="")]
effects_train_corpus_new_load<-effects_train_corpus_new_load[
```

```

which(effects_train_corpus_new_load$content!=" "))
benefits_test_corpus_new_load<-benefits_test_corpus_new_load[
  which(benefits_test_corpus_new_load$content!=" "))
effects_test_corpus_new_load<-effects_test_corpus_new_load[
  which(effects_test_corpus_new_load$content!=" ")]

```

2.2.19. 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 de 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)

```

2.2.20. 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, o simplemente borrar las palabras que aparezcan una única vez en el dataframe, y que difícilmente nos aportarán información relevante que podamos extrapolar. Para suprimir estos términos, denominados escasos, se utiliza el comando `removeSparseTerms()`.

$$df(t) > N \cdot (1! = sparse)$$

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 0.999. Este umbral elegido significa que se toman los términos que aparecen en más del 1% 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 5835 términos pasamos a 1702 términos, por tanto se ha considerado despreciar aplicar esta técnica ya que consideramos que se ha perdido una gran cantidad de información, y, desde nuestro punto de vista, creemos que no serían suficientes datos como para poder aplicar las técnicas y obtener buenos resultados. Por ello, necesitamos de estos los datos para la aplicación de las técnicas posteriores.

Luego, en función de la técnica que se aplique, si fuera necesario se reducirá la dimensionalidad. En definitiva, ya tenemos nuestros datos listos.

2.2.21. Matriz de documentos de los términos

Ahora vamos a mapear nuestro corpus creando una matriz de términos, donde las filas corresponden a los documentos, las columnas a los términos y cada casilla la ocurrencia de dicho término en dicho documento. 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*:

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

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 *i*ésima 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

# Ordenamos 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)

# Ordenamos 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, freq),
# para posteriormente graficarlo.
ordena_mayor_matrix_corpus <- data.frame(palabra = names(ordena_mayor_matrix_corpus),
                                            freq = ordena_mayor_matrix_corpus)

```

Mostramos las más frecuentes:

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

```

##             palabra freq
## 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:

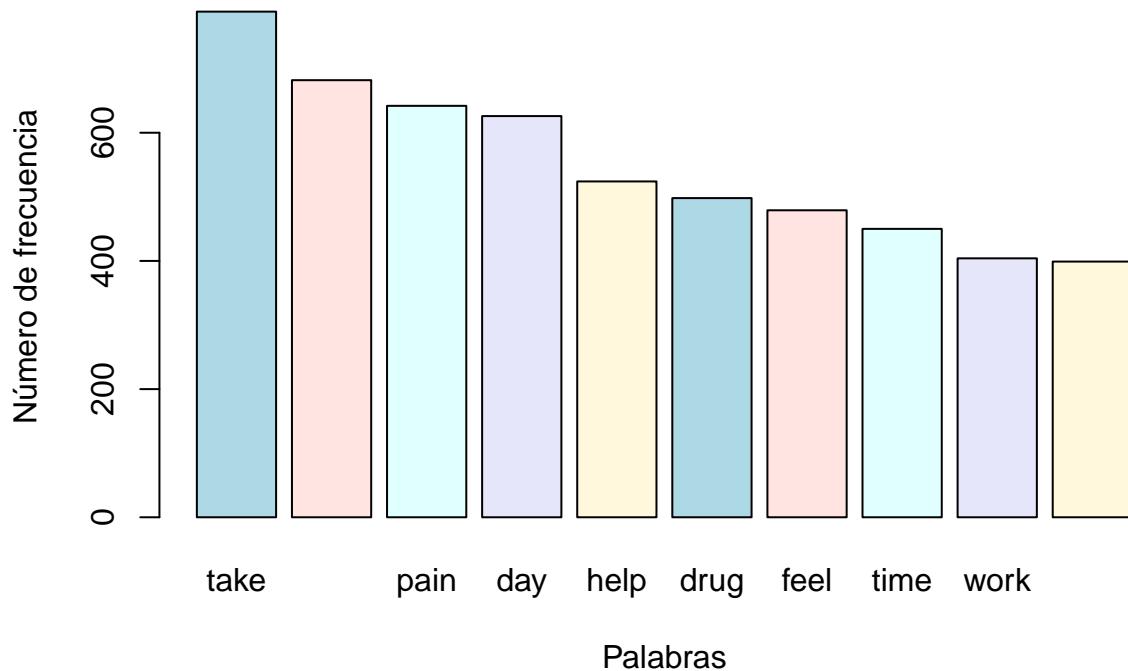
```

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 nueve palabras más frecuentes después del preprocesamiento", font = 4))
```

Las nueve palabras más frecuentes después del preprocesamiento



2.2.22. Nube de palabras

Por último, vamos a visualizar la nube de palabras para benefits preprocesado, tanto al principio del procesamiento como al final. En este caso, en vez de determinar los colores de las palabras en función de la ocurrencia de éstos en el dataset, hemos aplicado la nube de palabras de la forma que vimos en clase: **incluyendo análisis de sentimientos**. Por ello, en primer lugar, identificaremos las palabras a pintar como positivas o negativas, y posteriormente, se generará la nube de palabras con las siguientes características:

- Si la palabra tiene una connotación positiva, aparecerá de color verde.
- Si la palabra tiene una connotación negativa, aparecerá de color rojo.

Este análisis es realmente determinante en nuestro problema, puesto que al fin y al cabo, tenemos comentarios de pacientes, que estarán orientados a una buena y positiva opinión del fármaco o al contrario (una mala, y posiblemente molesta opinión acerca del mismo).

En este caso, visualizando la nube de palabras, obtenemos una cosa clara: principalmente tenemos palabras de connotación negativa, aspecto totalmente lógico puesto que, es más común que un usuario comente algo para quejarse o mostrar su mala experiencia, que para hablar bien del mismo. Además, al tratarse de temas de enfermedades y problemas de salud, es mucho más fácil encontrar palabras relacionadas con el dolor, o la preocupación, que con la alegría u otros aspectos derivados de las connotaciones positivas.

2.2.23. Convertir a dataframe nuestras modificaciones

Por último, con el fin de mejorar los tiempos computacionales y poder hacer uso todos los integrantes del grupo de las columnas preprocesadas, se ha optado por añadir dichos cambios en el dataset y guardarlos en un fichero.



Figura 15: Wordcloud con preprocesamiento

Nota: Cuando se necesite en las sucesivas técnicas, se realizarán las transformaciones necesarias de acuerdo a cada técnica en cuestión.



Figura 16: Wordcloud sin preprocesamiento

3. Análisis exploratorio de los datos

El análisis exploratorio de datos o (EDA) engloba un conjunto de técnicas para poder comprender de manera rápida la naturaleza de una colección de datos o dataset. Se basa principalmente en dos criterios: las **estadísticas de resumen** y la **visualización de datos**.

En primer lugar, vamos a realizar un resumen de nuestros datos utilizando la función **summary()**. Dicha función nos mostrará información relevante para cada una de las columnas del dataset, mostrando información general como valores mínimos, máximos, media, mediana. El resultado que obtenemos al evaluar nuestro dataset es el siguiente:

```
summary(datos_train)
```

```
##      urlDrugName          rating           effectiveness
##    lexapro : 63   Min.   : 1.000 Considerably Effective: 926
##    prozac : 46   1st Qu.: 5.000 Highly Effective     :1330
##    retin-a : 45   Median : 8.000 Ineffective        : 247
##    zoloft : 45   Mean   : 7.008 Marginally Effective : 186
##    paxil  : 38   3rd Qu.: 9.000 Moderately Effective : 415
##    propecia: 38   Max.   :10.000
##    (Other) :2829
##                               sideEffects           condition
##    Extremely Severe Side Effects: 175 depression       : 236
##    Mild Side Effects           :1019 acne            : 165
##    Moderate Side Effects       : 612 anxiety         :  63
##    No Side Effects             : 930 insomnia       :  54
##    Severe Side Effects         : 368 birth control    :  49
##                               high blood pressure:  42
##                               (Other)           :2495
##                               none
##
```

```

##  None
##  NONE
##  None.

##  The treatment benefits were marginal at best. Mood neither improved nor deteriorated, and anxiety was
##  Before the use of vagifem tablets, I had to endure a series of urinary infections after sometimes prolonged
##  (Other)

##          sideEffectsReview sideEffectsNumber effectivenessNumber
##  none                  : 112    Min.   :1.000    Min.   :1.000
##  None                 :  73    1st Qu.:1.000    1st Qu.:3.000
##  None.                :  19    Median :2.000    Median :4.000
##  No side effects.    :   9    Mean    :2.304    Mean    :3.936
##  There were no side effects.:  6    3rd Qu.:3.000    3rd Qu.:5.000
##  no side effects      :   5    Max.    :5.000    Max.    :5.000
##  (Other)              :2880

##  weightedRating     ratingLabel     sideEffectsInverse
##  Min.   : 2.000    Min.   :0.0000    Min.   :1.000
##  1st Qu.: 6.000    1st Qu.:1.0000    1st Qu.:3.000
##  Median : 8.000    Median :1.0000    Median :4.000
##  Mean   : 7.737    Mean   :0.7874    Mean   :3.696
##  3rd Qu.:10.000    3rd Qu.:1.0000    3rd Qu.:5.000
##  Max.   :10.000    Max.   :1.0000    Max.   :5.000

##
##  none
##  lower blood pressur
##  prevent pregnanc
##  treatment benefit margin best mood neither improv deterioran anxiety never signific allevi unsurpris
##  abl work without hassl im free pain right now there need cri anymorn whenev urin
##  believ multipl treatment benefit take tylenol headach tylenol safe inexpens requir physician prescr
##  (Other)

##  effects_preprocesado
##  none      : 214
##  side effect :  51
##  none notic  :  17
##  none awar   :   8
##  notic side effect:   7
##  none can tell :   6
##  (Other)       :2801

```

A continuación se va a realizar un análisis de la información más relevante no textual, como el valor de **rating** de los usuarios, la **efectividad** y los **efectos secundarios** de dicho medicamento y por último, la **valoración ponderada del rating** teniendo en cuenta la proporción entre efectividad y efectos secundarios del medicamento.

3.1. Valoraciones de los medicamentos por parte de los usuarios.

En primer lugar vamos a analizar si el *rating* aportado por los usuarios sobre los medicamentos es, en general, bueno o no. Para ello empezamos obteniendo las frecuencias y porcentaje total de las valoraciones aportadas por los usuarios. Por tanto, se va a calcular la frecuencia de dicho atributo y su porcentaje respecto del total.

```

# Obtener frecuencias del rating
table(datos_train$rating)

```

```
##
```

```

##   1   2   3   4   5   6   7   8   9   10
## 305 102 146 107 158 157 349 558 480 742
# Calculamos el número de documentos
numDocuments <- dim(datos_train)[1]

# Calculamos el porcentaje de cada puntuación respecto del total.
table(datos_train$rating)/numDocuments

##
##           1           2           3           4           5           6
## 0.09826031 0.03286082 0.04703608 0.03447165 0.05090206 0.05057990
##           7           8           9          10
## 0.11243557 0.17976804 0.15463918 0.23904639

```

Como podemos observar, hay una mayoría de valoraciones positivas respecto a las negativas. De hecho el mayor porcentaje (casi el 24 %) tienen la máxima valoración. Podemos comprobar ésto mediante el uso de la moda.

```

# Función para calcular la moda. Se le pasa como parámetro un atributo
calcularModa<-function(var){
  freq.var<-table(var)
  valor<-which(freq.var==max(freq.var)) # Elementos con el valor m
  names(valor)
}

# Obtenemos la moda para el rating
calcularModa(datos_train$rating)

## [1] "10"

```

Como resumen en general del rating, se va a calcular la media y la mediana para calcular la tendencia central para dicha variable. La media es la siguiente:

```

# Media
mean(datos_train$rating)

## [1] 7.008376

```

La mediana es la siguiente:

```

# Mediana
median(datos_train$rating)

## [1] 8

```

El valor medio obtenido es 7 y la mediana es 8. Podemos concluir con dicha información, que en general las valoraciones sobre los medicamentos son bastante positivas, situándose el 50 % de dichas valoraciones en el valor 8.

A continuación, se va a visualizar dicha información gráficamente:

```

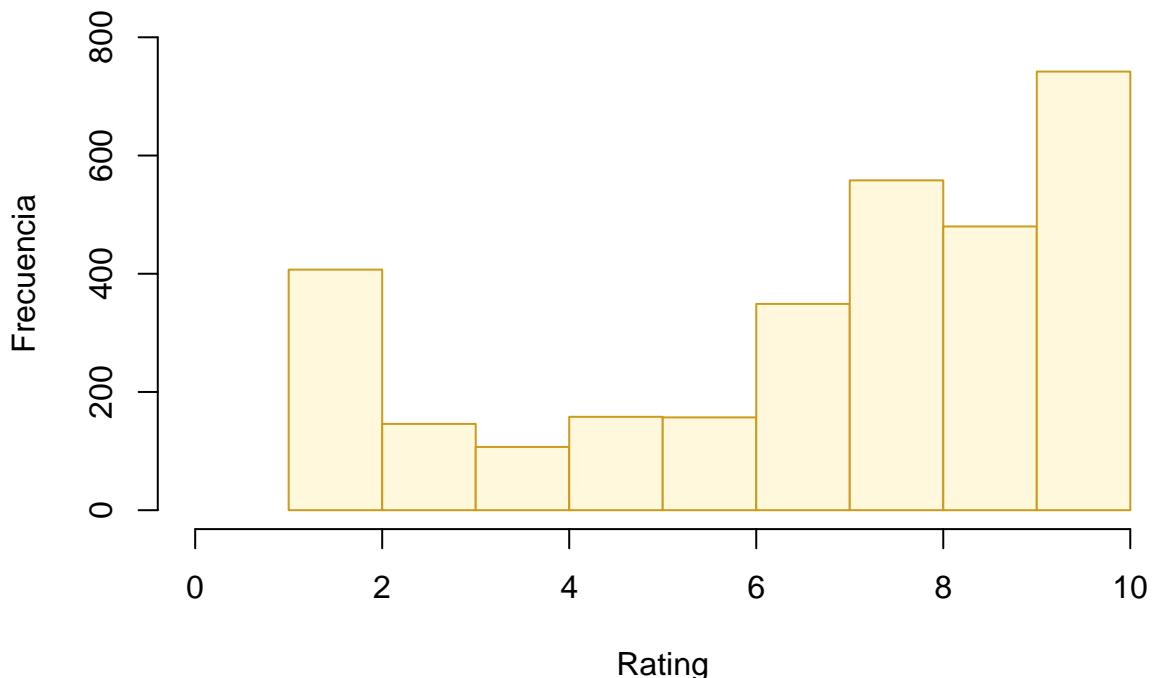
# Histograma de la valoración dada por los usuarios sobre los medicamentos
ratingExploration <- datos_train$rating

hist(ratingExploration,
      main="Rating de los medicamentos",
      xlab="Rating",
      ylab="Frecuencia",
      border="goldenrod3",

```

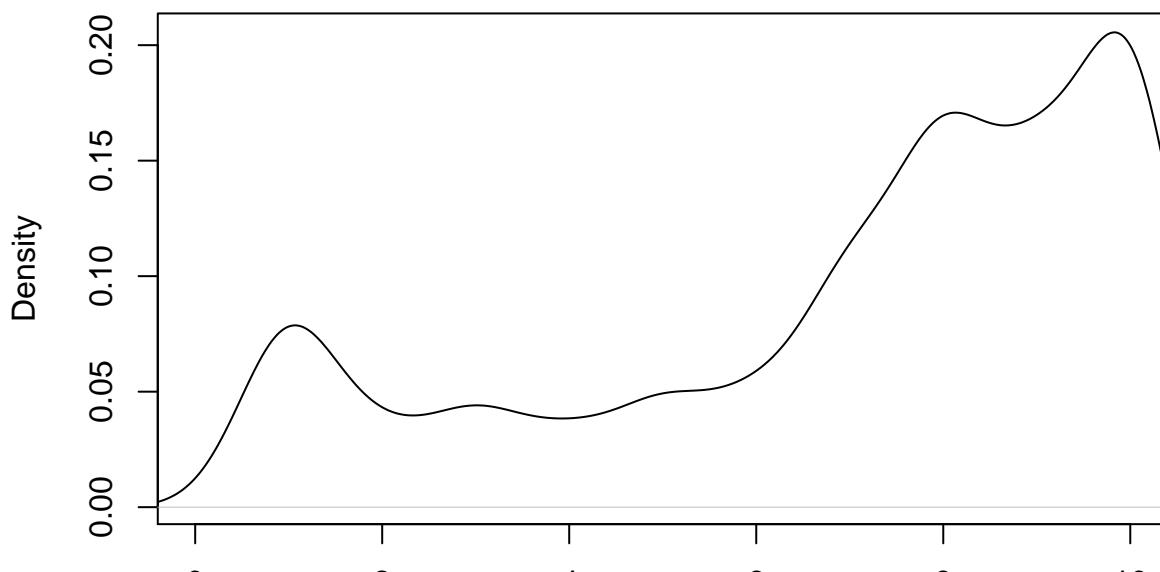
```
xlim=c(0,10),  
ylim=c(0,800),  
col= "cornsilk",  
breaks=10,)
```

Rating de los medicamentos



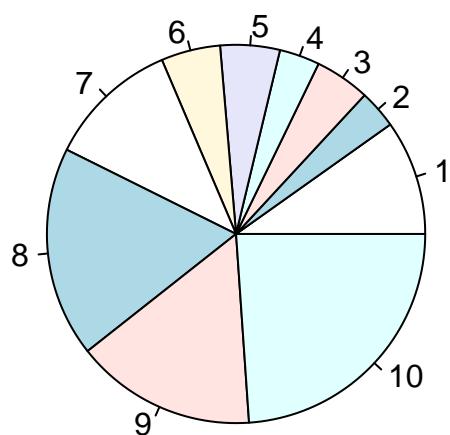
```
# Diagrama de densidad de la valoración dada por los usuarios sobre los medicamentos  
plot(density(ratingExploration),  
     main="Densidad del rating",  
     xlim=c(0,10),  
     )
```

Densidad del rating



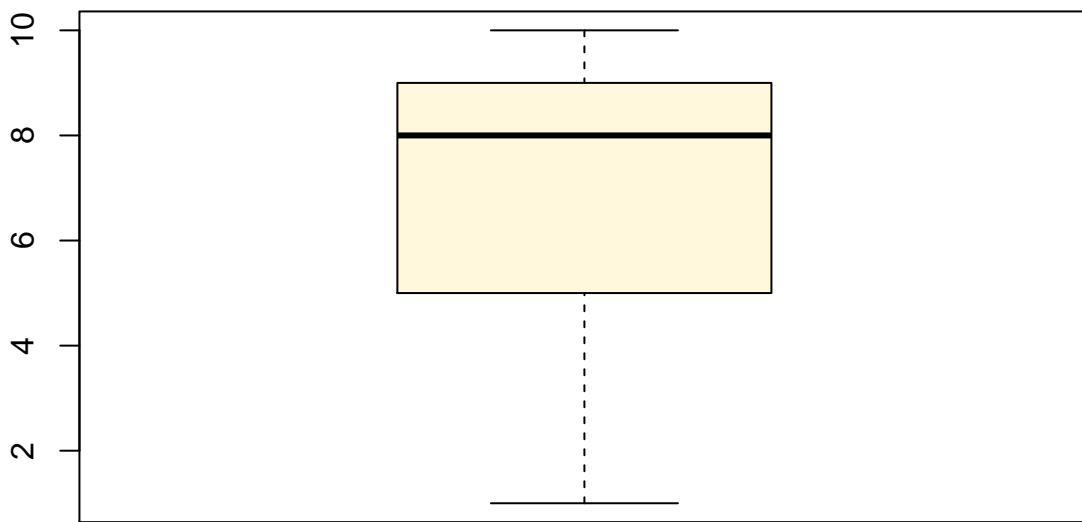
N = 3104 Bandwidth = 0.5294

```
# Diagrama de sectores de las valoraciones dadas por los usuarios  
pie(table(datos_train$rating))
```



```
# Diagrama de cajas sobre las valoraciones dadas por los usuarios  
boxplot(datos_train$rating, main="Rating", col= "cornsilk" )
```

Rating



Como medidas de dispersión, se va a calcular la **desviación típica**:

```
# Desviación típica
sd(datos_train$rating)
```

```
## [1] 2.937406
```

Como se puede observar, la desviación típica nos da un valor de 2.93. Esto quiere decir que los valores no están concentrados en un único valor, sino que la mayoría se sitúan en un intervalo con distancia 3 respecto de la media.

Este valor concuerda, puesto que si observamos el histograma anterior, vemos que la mayoría de las puntuaciones se sitúan entre 5 y 10.

Esto también nos da como **conclusión** que en general las **opiniones** sobre los medicamentos **son buenas**, puesto que la mayor cantidad se sitúan en el intervalo [5,10].

3.2. Efectividad del medicamento

En esta sección, se va a analizar si se consideran que los medicamentos son efectivos o no. Para ello se va a analizar el atributo **effectivenessNumber** (que mide la efectividad del medicamento, siendo 1 menos efectivo y 5 más efectivo).

Empezamos obteniendo las frecuencias y porcentaje total de las anotaciones de efectividad. Para ello se va a calcular la frecuencia de dicho atributo y su porcentaje respecto del total.

```
# Obtener frecuencias del effectivenessNumber
table(datos_train$effectivenessNumber)
```

```
##
##   1    2    3    4    5
## 247 186 415 926 1330
```

```
# Calculamos el número de documentos
numDocuments <- dim(datos_train)[1]
```

```
# Calculamos el porcentaje de cada valor de efectividad respecto del total
table(datos_train$effectivenessNumber)/numDocuments
```

```
##  
##      1      2      3      4      5  
## 0.07957474 0.05992268 0.13369845 0.29832474 0.42847938
```

Como podemos observar, la mayoría de los medicamentos se consideran que son efectivos. De hecho, la mayoría de los medicamentos se consideran altamente efectivos (con un 42%). Podemos ahora comprobar ésto mediante el uso de la moda.

```
# Obtenemos la moda para el effectivenessNumber  
calcularModa(datos_train$effectivenessNumber)
```

```
## [1] "5"
```

Como resumen en general de la efectividad, se va a calcular la media y la mediana para obtener la tendencia central de dicha variable. La media es la siguiente:

```
# Media  
mean(datos_train$effectivenessNumber)
```

```
## [1] 3.936211
```

La mediana es la siguiente:

```
# Mediana  
median(datos_train$effectivenessNumber)
```

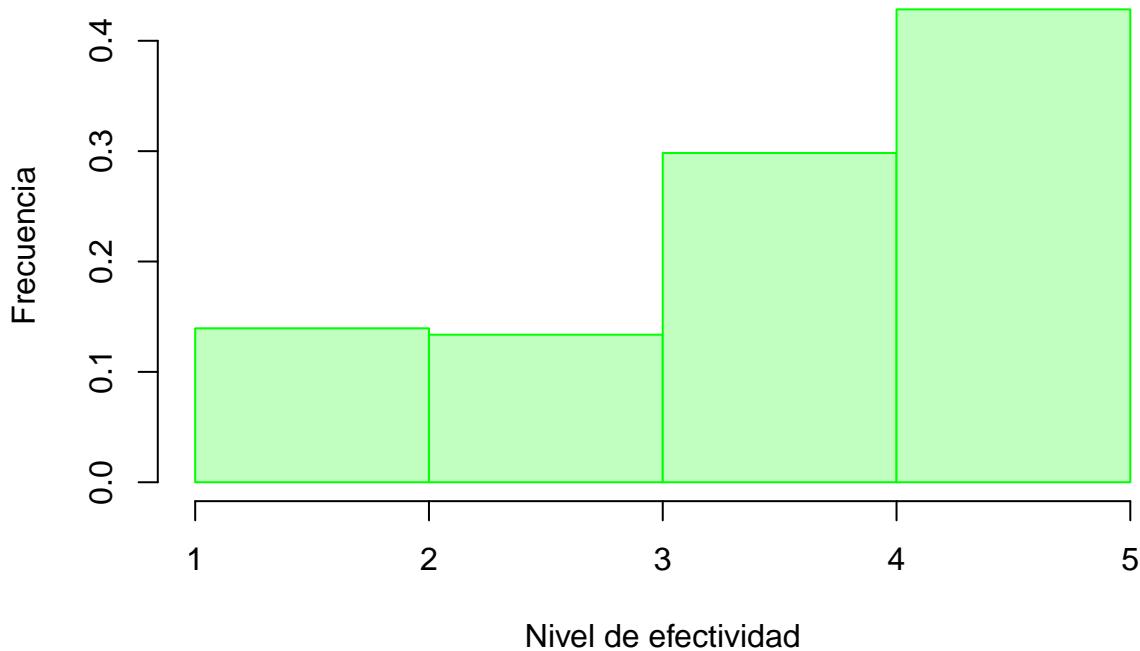
```
## [1] 4
```

El valor medio obtenido es 3.93 sobre 5 y la mediana es 4. Podemos concluir con dicha información, que en general los medicamentos son bastantes efectivos, situándose el 50 % de dichas mediciones sobre el valor 4.

A continuación, se va a visualizar dicha información gráficamente:

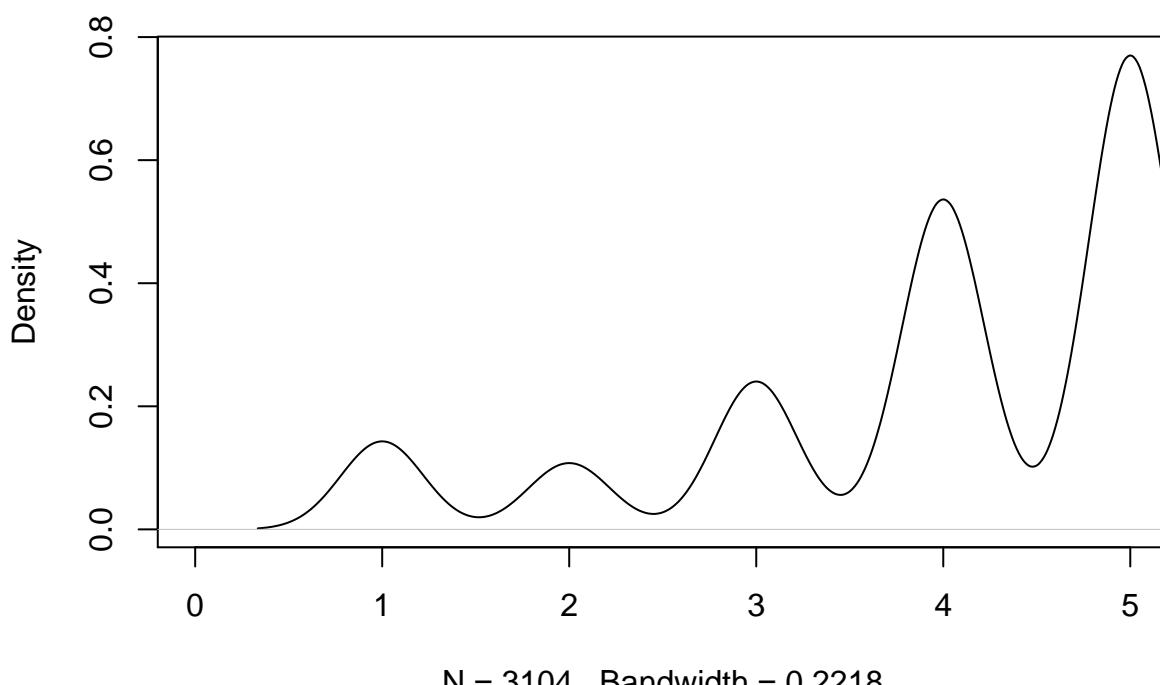
```
# Histograma sobre la efectividad de los medicamentos  
efecctivenessNumberExploration <- datos_train$effectivenessNumber  
  
hist(efecctivenessNumberExploration,  
      main="Efectividad de los medicamentos",  
      xlab="Nivel de efectividad",  
      ylab="Frecuencia",  
      border="green",  
      xlim=c(1,5),  
      col= "darkseagreen1",  
      breaks=5,  
      prob=TRUE  
)
```

Efectividad de los medicamentos

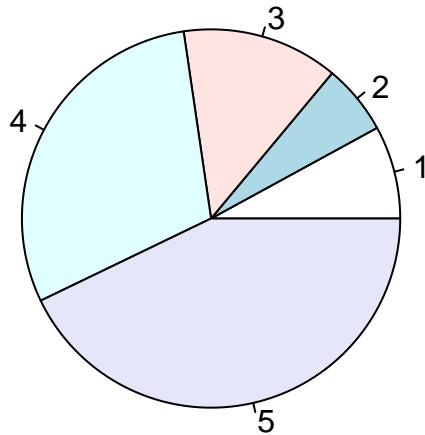


```
# Diagrama de densidad de la efectividad de los medicamentos
plot(density(datos_train$effectivenessNumber),
     main="Densidad de la tasa de efectividad",
     xlim=c(0,5),
     )
```

Densidad de la tasa de efectividad

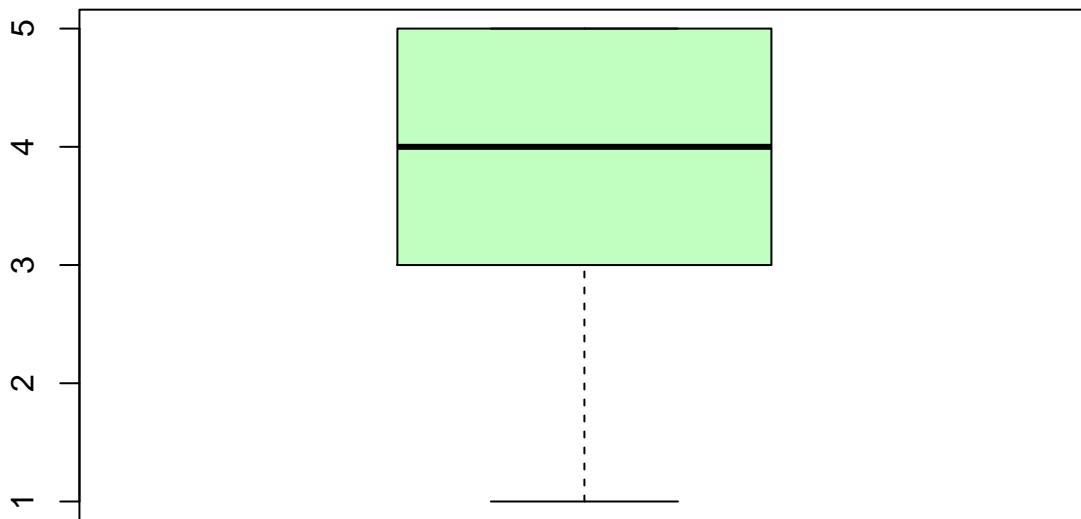


```
# Diagrama de sectores de la efectividad de los medicamentos
pie(table(datos_train$effectivenessNumber))
```



```
# Diagrama de cajas sobre la efectividad de los medicamentos
boxplot(datos_train$effectivenessNumber, main="Tasa de efectividad", col= "darkseagreen1" )
```

Tasa de efectividad



Como medidas de dispersión, se va a calcular la **desviación típica**:

```
# Desviación típica
sd(datos_train$effectivenessNumber)
```

```
## [1] 1.230634
```

Como se puede observar, la desviación típica nos da un valor de 1.23. Esto quiere decir que la mayor parte de los valores se sitúan en un intervalo con una distancia de uno de la media.

Este valor concuerda, puesto que si observamos el histograma anterior, vemos que la mayoría de las puntuaciones se sitúan entre 3 y 5.

Esto también nos da como **conclusión** que en general los medicamentos tienen una tasa bastante **buenas de efectividad** puesto que su tasa se sitúa entre [3,5].

3.3. Efectos secundarios del medicamento

En esta sección, se va a analizar si se consideran que los medicamentos tienen efectos secundarios o no. Para ello se va a analizar el atributo **sideEffectsNumber** (que mide la tasa de efectos secundarios del medicamento, siendo 1 el mínimo de efectos secundarios y 5 el máximo de efectos secundarios).

Empezamos obteniendo las frecuencias y porcentaje total de las anotaciones de efectos secundarios. Para ello se va a calcular la frecuencia de dicho atributo y su porcentaje respecto del total.

```
# Obtener frecuencias del sideEffectsNumber
table(datos_train$sideEffectsNumber)

## 
##      1      2      3      4      5 
## 930 1019  612  368  175

# Calculamos el número de documentos
numDocuments <- dim(datos_train)[1]

# Calculamos el porcentaje de tasa de efectos secundarios respecto del total.
table(datos_train$sideEffectsNumber)/numDocuments

## 
##      1          2          3          4          5 
## 0.29961340 0.32828608 0.19716495 0.11855670 0.05637887
```

Como podemos observar, la mayoría de los medicamentos se consideran que no tienen efectos secundarios severos. De hecho, la mayoría de los medicamentos se sitúan entre sin efectos secundarios (29 %) o que tienen efectos secundarios leves (32 %).

Podemos comprobar ésto mediante el uso de la moda.

```
# Obtenemos la moda para el sideEffectsNumber
calcularModa(datos_train$sideEffectsNumber)

## [1] "2"
```

Como resumen en general, sobre la tasa de efectos secundarios, se va a calcular la media y la mediana para calcular la tendencia central para dicha variable. La media es la siguiente:

```
# Media
mean(datos_train$sideEffectsNumber)

## [1] 2.303802
```

La mediana es la siguiente:

```
# Mediana
median(datos_train$sideEffectsNumber)

## [1] 2
```

El valor medio obtenido es 2.30 sobre 5 y la mediana es 2. Podemos concluir con dicha información, que en general los medicamentos no tienen efectos secundarios o que dichos efectos son leves.

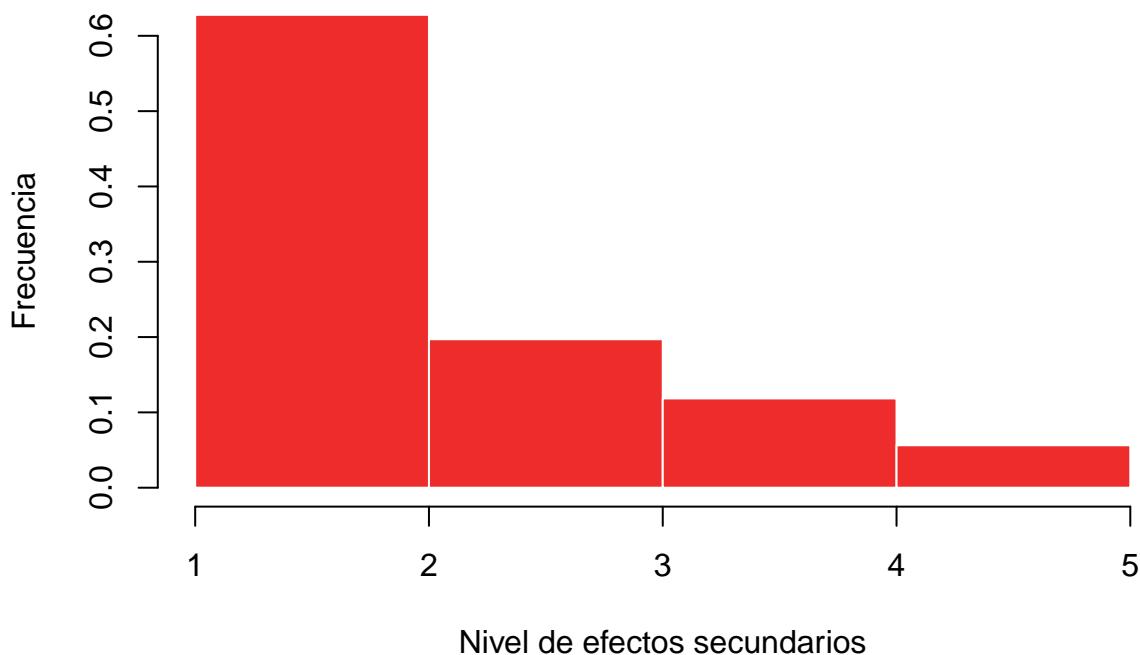
A continuación, se va a visualizar dicha información gráficamente:

```
# Histograma de la tasa de efectos secundarios
sideEffectsNumberExploration <- datos_train$sideEffectsNumber

hist(sideEffectsNumberExploration,
     main="Efectos secundarios de los medicamentos",
```

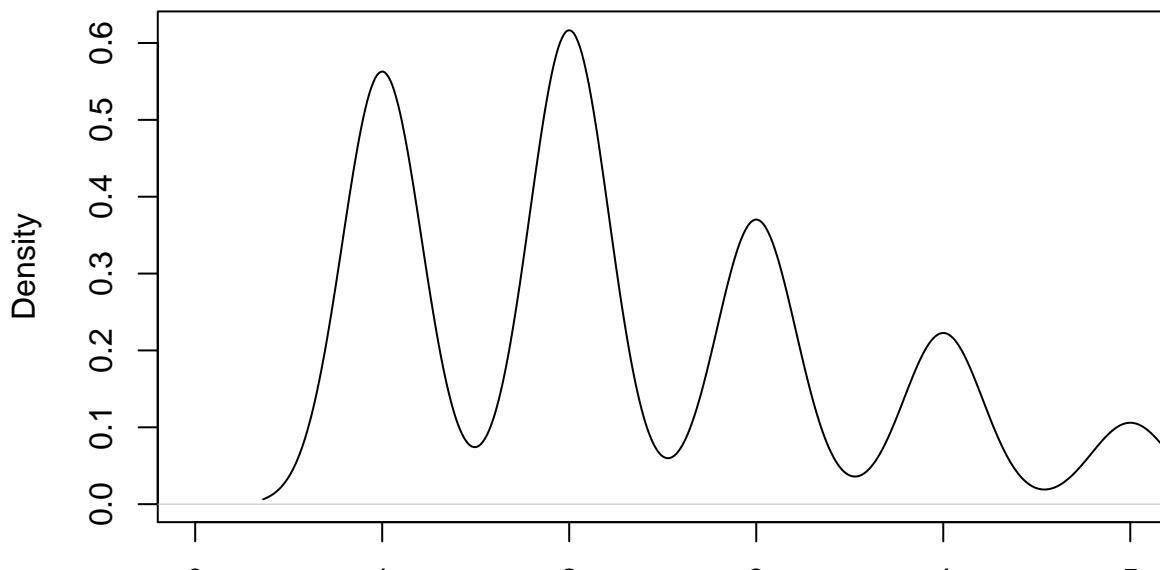
```
xlab="Nivel de efectos secundarios",
ylab="Frecuencia",
border="white",
xlim=c(1,5),
col= "firebrick2",
breaks=5,
prob=TRUE
)
```

Efectos secundarios de los medicamentos



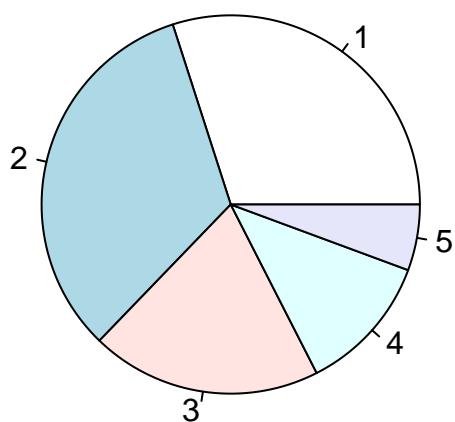
```
# Diagrama de densidad sobre la tasa de efectos secundarios
plot(density(datos_train$sideEffectsNumber),
     main="Densidad de la tasa de efectos secundarios",
     xlim=c(0,5),
     )
```

Densidad de la tasa de efectos secundarios



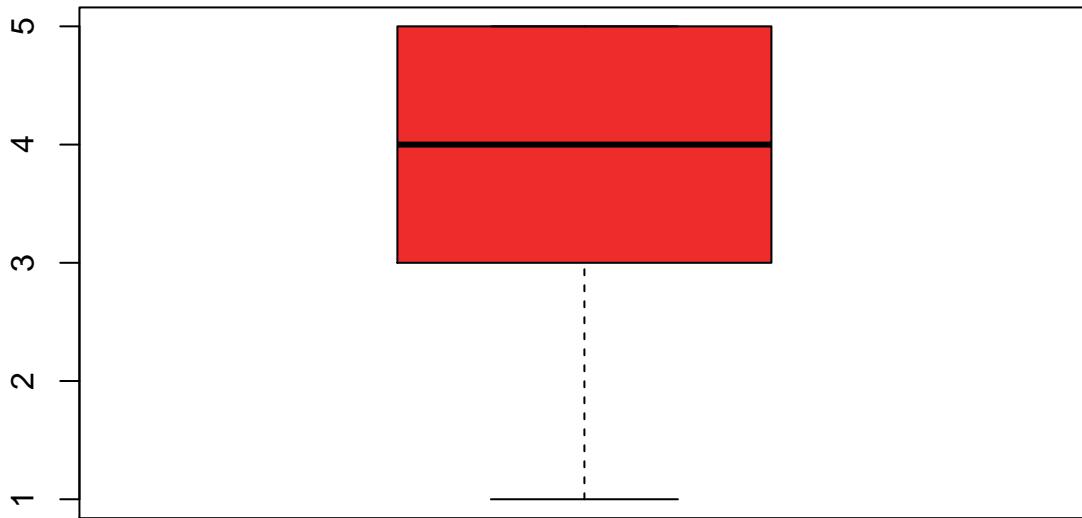
N = 3104 Bandwidth = 0.2122

```
# Diagrama de sectores de los efectos secundarios de los medicamentos  
pie(table(datos_train$sideEffectsNumber))
```



```
# Diagrama de cajas de los efectos secundarios de los medicamentos  
boxplot(datos_train$effectivenessNumber, main="Tasa de efectos secundarios", col= "firebrick2" )
```

Tasa de efectos secundarios



Como medidas de dispersión, se va a calcular la **desviación típica**:

```
# Desviación típica
sd(datos_train$sideEffectsNumber)

## [1] 1.177525
```

Como se puede observar, la desviación típica nos da un valor de 1.17. Esto quiere decir que la mayor parte de los valores se sitúan en un intervalo con una distancia de uno respecto la media. Este valor concuerda, puesto que si observamos el histograma anterior, vemos que la mayoría de las puntuaciones se sitúan entre 1 y 2.

Esto también nos da como **conclusión** que en general los medicamentos **no tienen efectos secundarios o son muy leves**.

3.4. Valoración ponderada sobre el medicamento

En esta sección, se va a analizar si se consideran que los medicamentos son “buenos” o no, teniendo en cuenta la relación entre los beneficios que aporta (efectividad) y las inconvenientes que tiene (efectos secundarios). Para ello se va a analizar el atributo **weightedRating** (que mide dicha relación teniendo en cuenta una tasa de efectividad del 30 % y una tasa de efectos secundarios del 70 %), y siendo 1 peor valorado y 10 mejor valorado.

Empezamos obteniendo las frecuencias y porcentaje total de las anotaciones sobre la puntuación ponderada. Para ello se va a calcular la frecuencia de dicho atributo y su porcentaje respecto del total.

```
# Obtener frecuencias del weightedRating
table(datos_train$weightedRating)

##
##      2      4      6      8     10
## 151  236  494 1212 1011

# Calculamos el número de documentos
numDocuments <- dim(datos_train)[1]

# Calculamos el porcentaje de puntuación ponderada respecto del total.
table(datos_train$weightedRating)/numDocuments
```

```
##  
##          2          4          6          8         10  
## 0.04864691 0.07603093 0.15914948 0.39046392 0.32570876
```

Como podemos observar, la mayoría de los medicamentos se consideran que son generalmente beneficiosos. De hecho, la mayoría de los medicamentos se sitúan con una valoración de 8 sobre 10 teniendo en cuenta la relación beneficio/perjuicio. Podemos comprobar ésto mediante el uso de la moda.

```
# Obtenemos la moda para el weightedRating  
calcularModa(datos_train$weightedRating)
```

```
## [1] "8"
```

Como resumen en general de sobre la tasa de efectos secundarios, se va a calcular la media y la mediana para calcular la tendencia central para dicha variable. La media es la siguiente:

```
# Media  
mean(datos_train$weightedRating)
```

```
## [1] 7.737113
```

La mediana es la siguiente:

```
# Mediana  
median(datos_train$weightedRating)
```

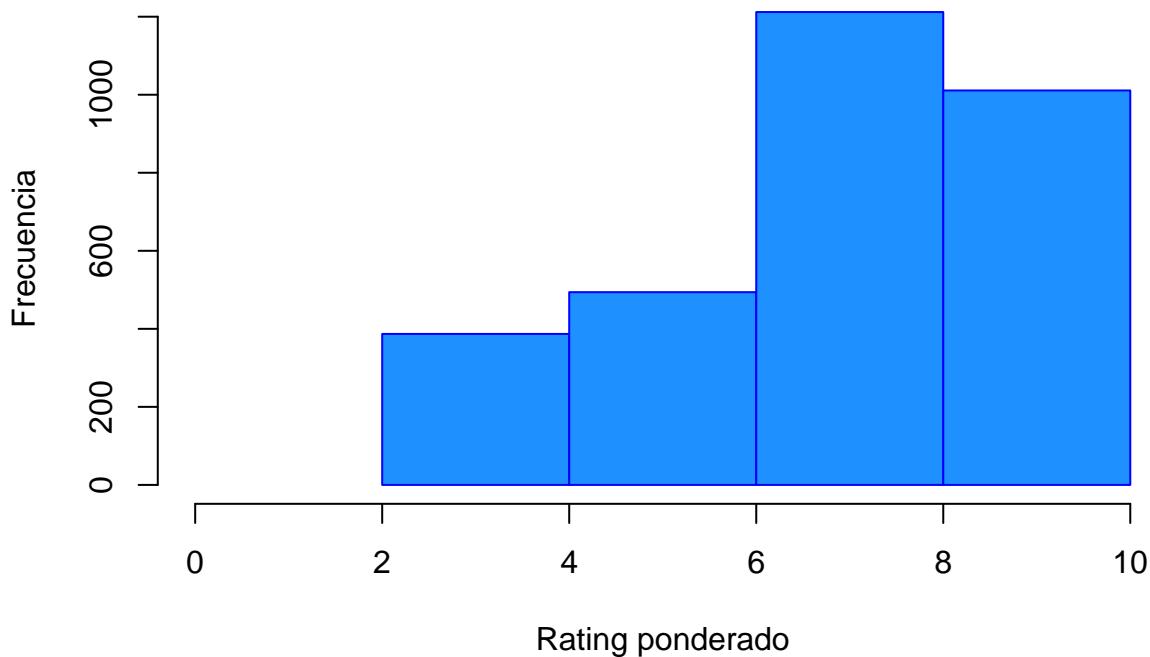
```
## [1] 8
```

El valor medio obtenido es 7.52 sobre 10 y la mediana es 8. Podemos concluir con dicha información que la puntuación general sobre los medicamentos es de **notable**.

A continuación, se va a visualizar dicha información gráficamente:

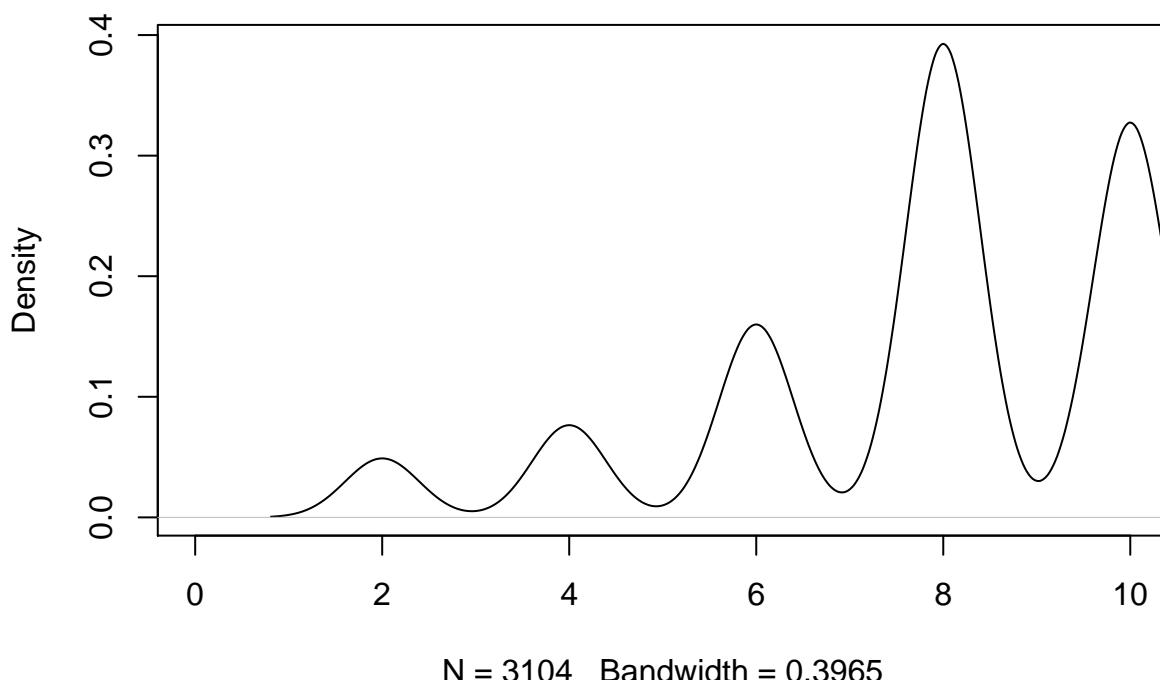
```
# Histograma de valoración ponderada  
weightedRatingExploration <- datos_train$weightedRating  
  
hist(weightedRatingExploration ,  
      main="Valoración ponderada de los medicamentos",  
      xlab="Rating ponderado",  
      ylab="Frecuencia",  
      border="blue",  
      xlim=c(0,10),  
      col= "dodgerblue1",  
      breaks=5  
  
)
```

Valoración ponderada de los medicamentos

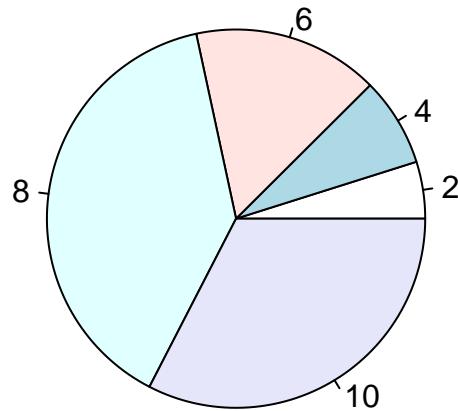


```
# Diagrama de densidad sobre la valoración ponderada
plot(density(datos_train$weightedRating),
      main="Densidad de valoración ponderada",
      xlim=c(0,10),
      )
```

Densidad de valoración ponderada

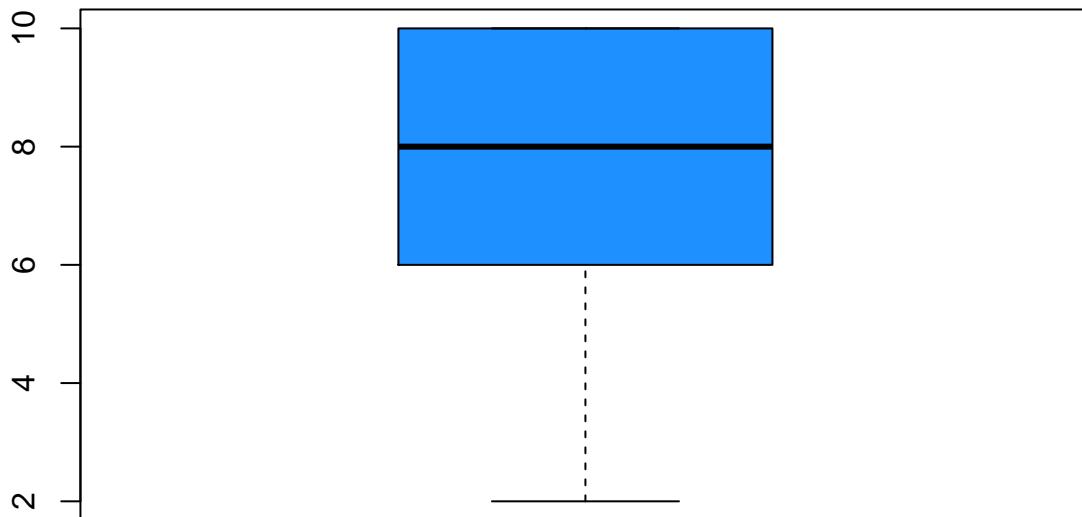


```
# Diagrama de sectores sobre la valoración ponderada
pie(table(datos_train$weightedRating))
```



```
# Diagrama de cajas sobre la valoración ponderada
boxplot(datos_train$weightedRating, main="Valoración ponderada", col= "dodgerblue1" )
```

Valoración ponderada



Como medidas de dispersión, se va a calcular la **desviación típica**:

```
# Desviación típica
sd(datos_train$weightedRating)
```

```
## [1] 2.199924
```

Como se puede observar, la desviación típica nos da un valor de 2.07. Esto quiere decir que la mayor parte de los valores se sitúan en un intervalo con una distancia de dos sobre la media.

Este valor concuerda, puesto que si observamos el histograma anterior, vemos que la mayoría de las puntuaciones se sitúan entre 6 y 10.

Esto también nos da como **conclusión** que en general los medicamentos que forman parte de este dataset **son convenientes tomarlos**.

3.5. Correlación sobre las variables

En esta sección se va a comprobar la correlación que existe entre las variables que miden la efectividad(effectivenessNumber), los efectos secundarios(sideEffectsNumber), la valoración aportada por los usuarios(rating) y la valoración ponderada que se ha realizado sobre el medicamento(weightedRating).

Empezamos calculando la correlación entre la variable que mide la efectividad y los efectos secundarios.

```
# Correlación lineal entre effectivenessNumber y sideEffectsNumber
cor(datos_train[,c(8,9)])
```

```
##                                     sideEffectsNumber effectivenessNumber
## sideEffectsNumber                 1.0000000          -0.3953789
## effectivenessNumber             -0.3953789          1.0000000
```

Podemos observar que cuantos más efectos secundarios tiene, menor es la efectividad del medicamento. Esto puede estar influido por las valoraciones subjetivas del usuario, ya que si ha tenido una mala experiencia (debido a los efectos secundarios) por la ingesta del medicamento, no va a hacer énfasis en los beneficios del medicamento, sino que hará un mayor énfasis en los aspectos negativos.

A continuación vamos a calcular la correlación entre la efectividad y la valoración ponderada del medicamento.

```
# Correlación lineal entre effectivenessNumber y weightedRating
cor(datos_train[,9:10])
```

```
##                                     effectivenessNumber weightedRating
## effectivenessNumber                 1.0000000          0.9237202
## weightedRating                   0.9237202          1.0000000
```

Como se puede observar, cuando el medicamento es más efectivo, la valoración ponderada del medicamento acumenta (como es obvio), y si ahora calculamos la valoración ponderada del medicamento teniendo en cuenta los efectos secundarios.

```
# Correlación lineal entre sideEffectsNumber y weightedRating
cor(datos_train[,c(8,10)])
```

```
##                                     sideEffectsNumber weightedRating
## sideEffectsNumber                 1.0000000          -0.649161
## weightedRating                  -0.649161           1.000000
```

Observamos como si el medicamento tiene una mayor tasa de efectos secundarios, la valoración ponderada disminuye considerablemente (obvio porque el 70% de la valoración ponderada tiene en cuenta los efectos secundarios del medicamento).

Ahora vamos a comprobar la relación que existe entre la valoración dada por el usuario y los efectos secundarios.

```
# Correlación lineal entre rating y sideEffectsNumber
cor(datos_train[,c(2,8)])
```

```
##                                     rating sideEffectsNumber
## rating                         1.0000000          -0.682939
## sideEffectsNumber              -0.682939           1.000000
```

Podemos comprobar cómo si el medicamento tiene una mayor tasa de efectos secundarios, la valoración dada por el usuario disminuye.

```
# Correlación lineal entre rating y effectivenessNumber
cor(datos_train[,c(2,9)])
```

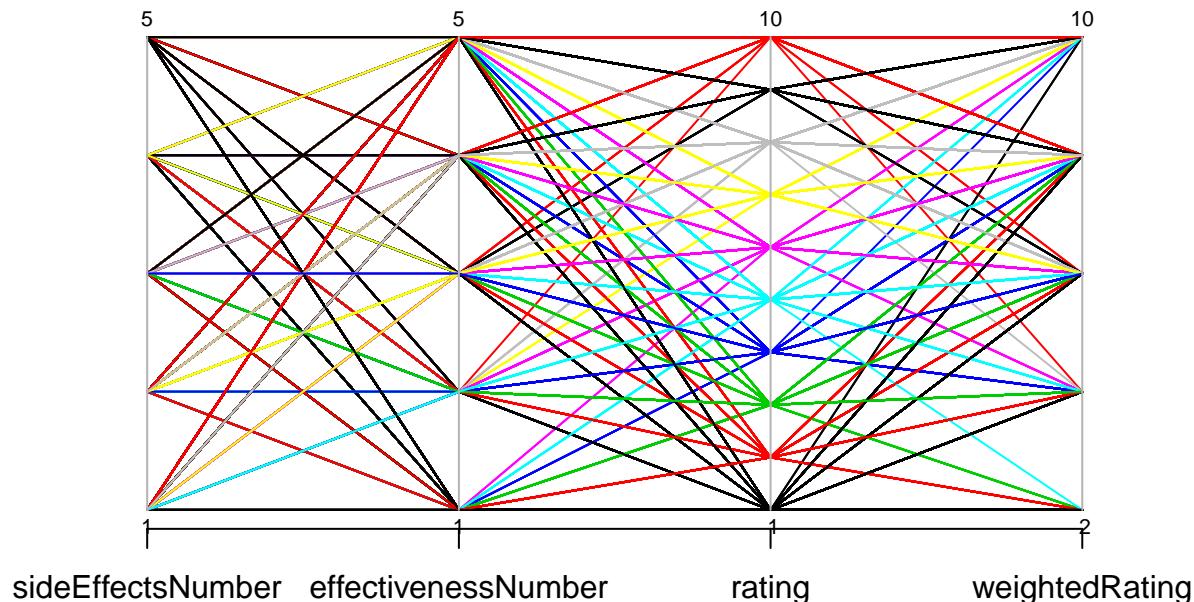
```
##                                     rating effectivenessNumber
## rating                         1.0000000          0.7498171
## effectivenessNumber            0.7498171           1.0000000
```

Y si la efectividad del medicamento es alta, la valoración del usuario se incrementa.

Como observación general, se puede destacar que **la valoración del usuario está condicionada más por la efectividad del medicamento** (relación 1/0.74) que por los efectos secundarios (relación 1/-0.68).

Por último, vamos a observar en el siguiente gráfico cómo se relacionan las variables entre sí en función de sus valores.

```
# Gráfico de coordenadas paralelas
library(MASS)
parcoord(datos_train[,c(8,9,2,10)], col=datos_train$rating, var.label=T)
```



Por ejemplo, podemos destacar que, si el número de efectos secundarios es 1 (no tiene efectos secundarios) y la efectividad del medicamento es 5 (muy efectivo), entonces la valoración del usuario será 10 y la valoración ponderada será también 10.

4. Análisis de sentimientos

En este apartado vamos a realizar un análisis descriptivo de los datos. En esta ocasión, vamos a visualizar cuáles son los sentimientos que expresan las personas en los comentarios que escriben sobre los distintos medicamentos que se encuentran en nuestro dataset. Lo haremos de los comentarios relacionados con los beneficios y efectos de los medicamentos.

La idea que se nos ocurrió consiste en saber cuáles son los medicamentos y condiciones más frecuentes que ocurren, ya que mostrar toda la información puede saturar la visibilidad de los datos.

```
summary(datos_train)

#https://stackoverflow.com/questions/1686569/filter-data-frame-rows-by-a-logical-condition
condiciones = c("depression", "acne", "anxiety", "insomnia", "birth control", "high blood pressure")
medicamentos = c ("lexapro", "prozac", "retin-a", "zoloft", "paxil", "propecia")
```

Gracias al resumen de nuestro dataset sabemos rápidamente cuáles son las drogas y condiciones más comunes. Ahora vamos a hacer un análisis de sentimientos de los comentarios relacionados con estas drogas y condiciones.

```
analisis_sentimientos <- function(comentarios,titulo){

  comentarios = Corpus(VectorSource(comentarios))

  d <- get_nrc_sentiment(comentarios$content)

  #https://medium.com/swlh/exploring-sentiment-analysis-a6b53b026131
  dt <- data.frame(t(d))

  sentimientos <- data.frame(rowSums(dt))

  names(sentimientos)[1] <- "count"
  sentimientos <- cbind("sentiment" = rownames(sentimientos), sentimientos)
  rownames(sentimientos) <- NULL

  qplot(sentiment, data=sentimientos[1:8,], weight=count, geom="bar",fill=sentiment)+
    ggtitle(titulo)
  ggsave(paste(titulo,".png"),path = 'figuras/sentimientos')

  qplot(sentiment, data=sentimientos[9:10,], weight=count, geom="bar",fill=sentiment)+
    ggtitle(titulo)
  ggsave(paste(titulo,"_positivismo.png"),path = 'figuras/sentimientos')

}
```

Básicamente lo que llevamos a cabo en esta función es la generación del Corpus de los comentarios pasados como parámetros (ya preprocesados). El siguiente paso es generar la tabla de sentimientos traspuesta (sino hacemos esto no funciona debido a la disposición de los datos). Luego preparamos una fila en la que aparezcan el nombre de los sentimientos en el mismo orden que la tabla y borramos los nombres de las filas. Esto último se realiza para la correcta visualización de los datos.

Por último se lleva a cabo la representación en forma de gráficas de los comentarios de la gente. Recordemos, están acotadas a conjuntos de comentarios para los fármacos y condiciones más comunes, por separado.

Obtenemos las gráficas mencionadas:

```

for(cadena in condiciones){
  datos <- datos_train[datos_train$condition==cadena ,]
  analisis_sentimientos(datos$benefits_preprocesado,
                        paste("Comentarios_beneficios_de_condición_",cadena))
}

for(cadena in condiciones){
  datos <- datos_train[datos_train$condition==cadena ,]
  analisis_sentimientos(datos$effects_preprocesado,
                        paste("Comentarios_efectos_de_condición_",cadena))
}

for(cadena in medicamentos){
  datos <- datos_train[datos_train$urlDrugName==cadena ,]
  analisis_sentimientos(datos$benefits_preprocesado,
                        paste("Comentarios_beneficio_de_medicamento_",cadena))
}

for(cadena in medicamentos){
  datos <- datos_train[datos_train$urlDrugName==cadena ,]
  analisis_sentimientos(datos$effects_preprocesado,
                        paste("Comentarios_efectos_de_medicamento_",cadena))
}

```

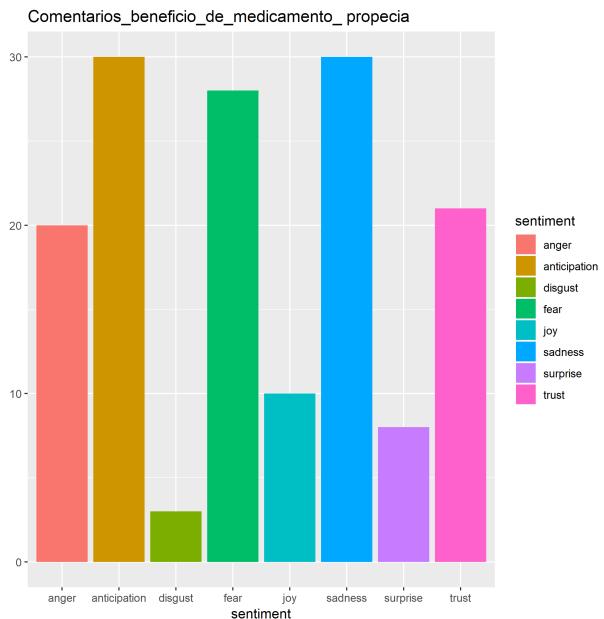


Figura 17: Análisis de sentimientos de los comentarios sobre los beneficios de los medicamentos para obtener propecia.

Hemos visualizado las imágenes resultantes y llegamos a la conclusión de que, en general, hace una buena descripción de los datos. Por ejemplo, para la **propecia** tenemos un altísimo contenido de sentimientos tales como miedo, tristeza y rabia. Mientras que otros sentimientos pasan más desapercibidos. Sin embargo, parece ser que las personas están contentas de forma genérica para el conjunto de medicamentos que la tratan, ya que los comentarios sobre los beneficios de los medicamentos son más positivos que negativos. En cambio, cuando hablamos de los efectos secundarios de este medicamento, el negativismo vence al positivismo; se ve que los efectos secundarios que pueden acarrear este tipo de medicinas no son muy agradables. Los sentimientos de

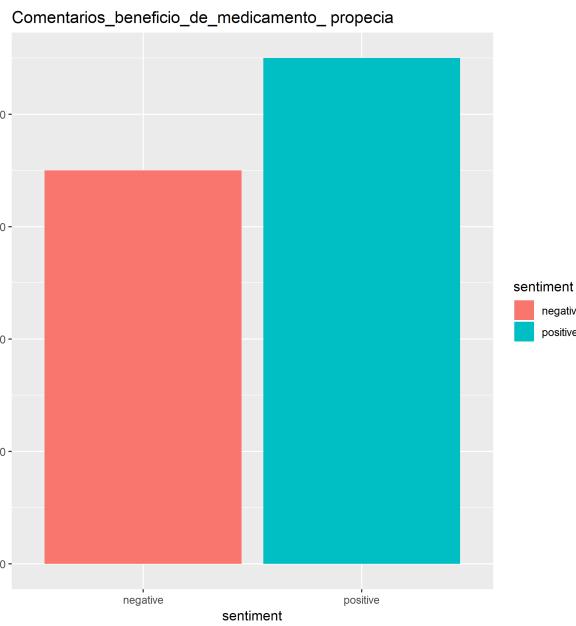


Figura 18: Análisis de positivismo de los comentarios sobre los beneficios de los medicamentos para obtener propecia.

los efectos secundarios de los medicamentos son muy similares a los que hablan de los beneficios, vemos que hay una concordancia de los comentarios de las personas.

Otro dato curioso es que, de forma genérica, hay una mayor presencia de negativismo que positivismo. Hay que darse cuenta de que las personas están escribiendo sobre problemas de salud que tienen, por lo que el único punto positivo que podemos extraer es cuando una persona muestra su felicidad al ver que un cierto medicamento está surtiendo efecto (y aún así es posible que se queje igualmente por los efectos secundarios, por ejemplo).

Además, hay que destacar que los sentimientos están muy entremezclados en los comentarios presentes de nuestro dataset. Los valores del rating vendrán especificados implicitamente dependiendo del peso que le de cada persona a esos sentimientos. En definitiva, nos damos cuenta de que nuestro dataset es bastante subjetivo. Por ejemplo, las personas suelen mostrarse más dispuestas a expresar comentarios para quejarse que para informar de que un medicamento le está funcionando.

Todas estas gráficas pueden apreciarse en el anexo de esta práctica por si se quiere ver el resto de condiciones y medicamentos analizados con esta técnica.

Por último, mencionar que, gracias a estos resultados, hemos podido corroborar lo expuesto tanto en el análisis exploratorio de los datos como en el *word cloud* con sentimientos que mostramos en apartados anteriores.

5. Reglas de Asociación

En esta técnica haremos un análisis descriptivo de los datos de texto ya que, en nuestra opinión, son los que consideramos que nos pueden permitir sacar más provecho de esta técnica.

Buscaremos la relación que existe entre las palabras expuestas en los distintos comentarios sobre los medicamentos y trataremos de asociarlos a distintos conceptos para tener una mejor idea de nuestro dataset. Esto nos permitirá saber de antemano que palabras a priori pueden tener que ver algo con distintos efectos de los medicamentos, y también, obtener información relevante acerca de cuestiones que se nos hayan podido plantear (lo cual se conoce como filtraje de reglas guiado por el usuario).

Vamos a crearnos un corpus por cada uno de los tipos de comentarios que tenemos en cada ítem. Recordar que estos textos ya se encuentran preprocesados, tal y como se puede consultar en secciones anteriores de esta documentación.

```
benefits_corpus = Corpus(VectorSource(datos_train$benefits_preprocesado))
effects_corpus = Corpus(VectorSource(datos_train$effects_preprocesado))
```

Ahora mismo el dataset es de tipo categórico pero nosotros necesitamos tener los datos como una cadena de caracteres para pasos posteriores. Por tanto, todos los datos que se encuentran en estas columnas van a ser transformados. Serán entendidos como palabras diferenciadas. Este paso de diferenciación de las palabras de los documentos, es clave para el funcionamiento de la técnica, puesto que nos permite obtener las distintas palabras que posteriormente formaran los ítem-sets.

Una vez que obtenemos las palabras en tipo *character*, algunas de estas se quedan como palabras vacías. Debemos *limpiar* estos vacíos que se nos han generado, antes de continuar al siguiente paso. Finalmente, una vez llevemos a cabo este último paso, ya podremos obtener una base de datos de tipo transaccional.

```
items_benefits <- strsplit(as.character(benefits_corpus$content), " ")
items_effects <- strsplit(as.character(effects_corpus$content), " ")

# Para eliminar las cadenas vacías
# https://stackoverflow.com/questions/24178854/remove-blanks-from-strsplit-in-r
items_benefits <- lapply(items_benefits, function(x){x[x!=""]})
items_effects <- lapply(items_effects, function(x){x[x!=""]})

transactions_benefits <- as(items_benefits,"transactions")
transactions_effects <- as(items_effects,"transactions")
```

Ya tenemos los datos de textos estructurados en una base de datos transaccional, ahora podemos aplicar la técnica para obtener las reglas de asociación. Vamos a utilizar el algoritmo “a priori” visto en clase. De lo contrario, el coste computacional y de tiempo no sería viable para obtener este conocimiento a partir de los comentarios.

El primer parámetro que encontramos en la función se corresponde con los datos que le proporcionamos y el segundo, un listado de parámetros específicos. En cuanto a estos, el primero es el umbral para el soporte, el segundo el umbral de la confianza, el tercero el target para indicar que buscamos reglas de asociación y en el cuarto indicamos que como mínimo empecemos con ítemset de tamaño 2. De esta forma estamos seleccionando un conjunto más reducido de reglas, que es lo que nos interesa desde el principio para ahorrar costes de procesamiento de los datos, al mismo tiempo que perdemos la mínima calidad posible en las reglas.

Ordenamos por “confidence” a vista de mostrar posteriormente los resultados más importantes. Entonces mostraremos las reglas que tienen más confianza (sino no podemos realizar una buena visualización de los datos), y finalmente visualizaremos el resultado obtenido.

```
rules_benefits <- apriori(transactions_benefits, parameter =
                           list(sup = 0.001, conf = 0.7, target="rules", minlen=2))
rules_effects <- apriori(transactions_effects, parameter =
```

```

list(sup = 0.001, conf = 0.7, target="rules", minlen=2))

rules_benefits <- sort(rules_benefits, decreasing = TRUE, na.last = NA,
                        by = "confidence")
rules_effects <- sort(rules_effects, decreasing = TRUE, na.last = NA,
                        by = "confidence")

detach(package:tm, unload=TRUE)
inspect(head(rules_benefits,100))
inspect(head(rules_effects,100))

```

	lhs `<fctr>	rhs `<fctr>	support `<dbl>	confidence `<dbl>	lift `<dbl>	count `<dbl>
[1]	{uneven}	=> {skin}	0.001288660	1	12.465863	4
[2]	{firmer}	=> {skin}	0.001610825	1	12.465863	5
[3]	{voltaren}	=> {pain}	0.001288660	1	7.201856	4
[4]	{out}	=> {break}	0.001610825	1	134.956522	5
[5]	{puls}	=> {pressur}	0.001288660	1	30.431373	4
[6]	{suppl}	=> {skin}	0.001288660	1	12.465863	4
[7]	{glow}	=> {skin}	0.001288660	1	12.465863	4
[8]	{fungal}	=> {infect}	0.001288660	1	25.442623	4
[9]	{radian}	=> {look}	0.001288660	1	44.985507	4
[10]	{radian}	=> {skin}	0.001288660	1	12.465863	4

Figura 19: inspect(head(rules_benefits),100) para visualizar reglas más importantes de los comentarios sobre beneficios.

	lhs `<fctr>	rhs `<fctr>	support `<dbl>	confidence `<dbl>	lift `<dbl>	count `<dbl>
[1]	{uneven}	=> {skin}	0.001288660	1	12.465863	4
[2]	{firmer}	=> {skin}	0.001610825	1	12.465863	5
[3]	{voltaren}	=> {pain}	0.001288660	1	7.201856	4
[4]	{out}	=> {break}	0.001610825	1	134.956522	5
[5]	{puls}	=> {pressur}	0.001288660	1	30.431373	4
[6]	{suppl}	=> {skin}	0.001288660	1	12.465863	4
[7]	{glow}	=> {skin}	0.001288660	1	12.465863	4
[8]	{fungal}	=> {infect}	0.001288660	1	25.442623	4
[9]	{radian}	=> {look}	0.001288660	1	44.985507	4
[10]	{radian}	=> {skin}	0.001288660	1	12.465863	4

Figura 20: inspect(head(rules_effects),100) para visualizar reglas más importantes de los comentarios sobre efectos secundarios.

```

plot(rules_benefits, method="graph")
title("Reglas de asociación sobre comentarios de beneficios")

```

Al intentar pintar las reglas de asociación, nos dimos cuenta de que plot no puede con tantas reglas al mismo tiempo, por lo que se queda con las 100 primeras. Como las tenemos ordenadas por confianza, no nos resulta un problema, ya que siempre que ocurra esto cogerá las 100 que más nos interesan.

Tal y como se aprecia en el mapa de asociaciones realizado, vemos que a simple vista las asociaciones obtenidas tienen mucho sentido. Por ejemplo, tenemos *attack* junto con *panic* que están relacionados con un lift considerable a *anxiety*. Cuanto más oscuros sean los puntos quieren decir que el lift es más alto. Por tanto, la aparición de una de las palabras favorece la aparición de otra.

Vemos que hay consecuentes en las reglas muy frecuentes, que ocupan el centro o núcleo de las reglas de asociación en el mapa. Estas palabras son principalmente *effect* y *side*. Esto también nos parece prometedor, ya que tiene todo el sentido del mundo que la gente haga comentarios siempre en torno a los efectos que tienen los medicamentos que están probando.

Podríamos poner mil ejemplos más. La palabra *acid* favorece que aparezca la palabra *reflux*. La gente cuando habla de reflujo, se refiere al ácido del estómago que sube de forma accidental por el esófago y causa quemazón.

En definitiva, es una buena forma de tener una vista general sobre lo que hablan las personas en el conjunto de comentarios y tener una idea de que conceptos utilizan más y cómo para expresar los efectos que tienen los medicamentos.

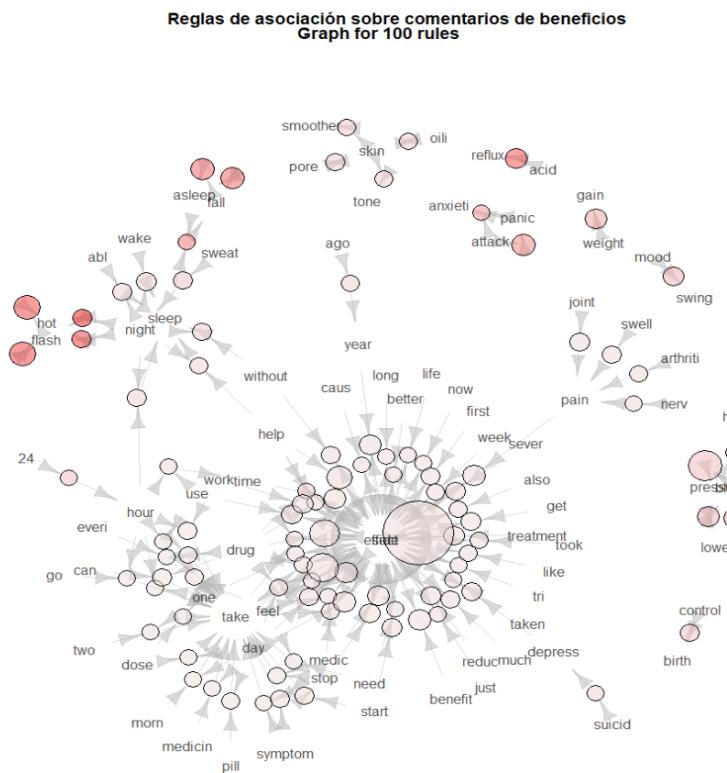


Figura 21: Mapa de reglas de asociación sobre los comentarios de beneficios.

Hicimos lo mismo para los comentarios relacionados con los efectos secundarios.

```
plot(rules_effects, method="graph")
title("Reglas de asociación sobre comentarios de efectos secundarios")
```

En este caso, vemos que no hay demasiadas asociaciones con un valor de lift por encima significativamente del resto. Creemos que se debe a que en este punto la gente escribe una diversidad de tipos de comentarios mucho más grande. Es decir, quizás los comentarios sobre efectos secundarios pueda tener una mayor cantidad de variedad sobre lo que se habla que en los comentarios sobre los beneficios.

No obstante, volvemos a tener los mismos consecuentes más frecuentes. Ocupando una vez más el núcleo del mapa de las reglas de asociación: *side* y *effect*. Tiene sentido ya que en los efectos secundarios vuelven a hablar de este tema de forma general, sólo que desde un punto de vista diferente (por eso la estructura principal del mapa y las reglas de asociación son diferentes).

Hay conceptos interesantes como *nausea*, *headach*, *pain...* Conceptos que están estrechamente relacionados con efectos secundarios.

5.1. Reglas de asociación específicas

Llegados a este punto, quisimos darle un nuevo enfoque a la extracción de reglas de asociación a partir de estos comentarios. En los mapas anteriores, veíamos que había demasiadas reglas enfocadas a una misma cosa, incluso llegando a ser difícil de visualizar en algunos sitios concretos.

Por otro lado, temíamos estar tomando solo las reglas de asociación relacionadas con *effects* y *side* por ser las reglas con una mayor confianza y despreciando otras que podían ser interesantes y que tuvieran consecuentes en otros conceptos.

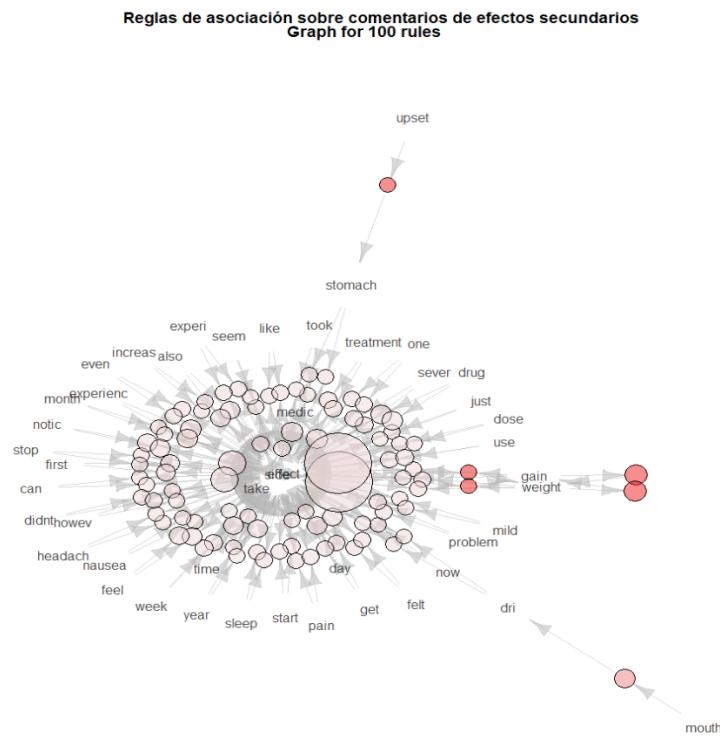


Figura 22: Mapa de regla de asociación sobre los comentarios de efectos secundarios.

Entonces, se nos ocurrió la siguiente idea: ¿Y si enfocamos la selección de reglas en la que el consecuente sea un término concreto que consideremos interesante? En otras palabras, queremos realizar un filtrado de reglas interesantes directamente guiadas por el usuario, tal y como se ha visto en teoría.

Queremos obtener las reglas en la que los consecuentes sean los valores de efectividad de nuestro dataset. Aquí realizamos un “mini-preprocesamiento”, ya que está realizado en apartado anteriores, pero en esta técnica específicamente vimos conveniente realizar estos pequeños cambios y añadirlos en una nueva columna del dataset.

Lo primero que debíamos de hacer es tener la efectividad unida en una sola cadena sin espacios, ya que de lo contrario no sería reconocida como un término a la hora de generar la estructura transaccional posterior.

```
datos_train$effectivenessGuion[datos_train$effectiveness ==  
                                "Highly Effective"] <- "Highly-Effective"  
datos_train$effectivenessGuion[datos_train$effectiveness ==  
                                "Considerably Effective"] <- "Considerably-Effective"  
datos_train$effectivenessGuion[datos_train$effectiveness ==  
                                "Moderately Effective"] <- "Moderately-Effective"  
datos_train$effectivenessGuion[datos_train$effectiveness ==  
                                "Marginally Effective"] <- "Marginally-Effective"  
datos_train$effectivenessGuion[datos_train$effectiveness ==  
                                "Ineffective"] <- "Ineffective"  
  
# Pasamos a factor el effectiveness unido por guiones para  
# concatenarlo con los comentarios  
datos_train$effectivenessGuion = as.factor(datos_train$effectivenessGuion)
```

¿Cómo podemos hacer que estas efectividades sean cosas frecuentes en las reglas en función de los comentarios? Tuvimos una buena idea en este punto. Mientras discutíamos en cómo hacer que el algoritmo a priori pudiera

generar estas reglas, pensamos en coger estas etiquetas de efectividad y ponerlas al final de cada comentario como un término más (fila por fila). De esta forma, cada comentario tendría una palabra final en la que aparece el nivel de efectividad que el usuario reconoce en el medicamento y podemos pasarlo de esa forma directamente al algoritmo.

```
# Juntamos las cadenas en una nueva columna del dataset que combina con los
# comentarios con los beneficios
datos_train$benefits_effectiveness <- with(datos_train,
                                             interaction(benefits_preprocesado, effectivenessGuion), sep=" ")
datos_train$benefits_effectiveness <- gsub("[.]", " ", datos_train$benefits_effectiveness)

# Hacemos lo mismo pero para los comentarios de efectos secundarios
datos_train$effects_effectiveness <- with(datos_train, interaction
                                             (effects_preprocesado, effectivenessGuion), sep=" ")
datos_train$effects_effectiveness <- gsub("[.]", " ", datos_train$benefits_effectiveness)

datos_train$effects_effectiveness[1:10]
```

[1] "slow progress left ventricular dysfunct overt heart failur alon agent manag hypertens mangag congest heart
failur Highly-Effective"
[2] "although type birth control con pros help cramp also effect prevent pregnanc along use condom well Highly-
Effective"
[3] "use cramp bad leav ball bed least 2 day ponstel doesnt take pain away complet take edg much normal activ
possibl definit miracl medic Highly-Effective"
[4] "acid reflux went away month just day drug heartburn start soon stop take began treatment 6 month pass stop
take heartburn came back seem wors even doctor said tri anoth 6 month treatment exact thing happen went three year
ask wasnt cure reflux doctor quit frank told wasnt cure treatment symptom told probabl rest life Marginally-
Effective"
[5] "think lyrical start help pain sideeffect just sever continu Marginally-Effective"
[6] "take propecia year start 20 year age hair continu thin notic signific benefit Ineffective"
[7] "mood notic improv energi experi better sleep digest Highly-Effective"
[8] "although drug origin prescrib depress help sleepless therefor continu take alon still occas problem fall
asleep find can combin melatonin valerian 12 year havent increas elavil dosag Considerably-Effective"
[9] "simpli just work fast without nasti side effect ssri medicin wont go long storis panic attack sought help mani
year fact just work suppos without annoy side effect taken daili stop panic attack start think pattern lead find
can also use need along daili treatment Highly-Effective"
[10] "none noth help allergi just dryer pain version allergi new allergiesirrit develop top origin one stuff danger
side effect mani peopl experienc list packag class action lawsuit need happen Ineffective"

Figura 23: Ilustración de preprocesamiento específico para reglas de asociación acotada a efectividad.

Los pasos siguientes ya son muy parecidos a los explicados anteriormente. Generamos los corpus.

```
# Generamos los corpus con estas nuevas columnas en la que
# añadimos la efectividad como término en cada comentario
benefits_corpus = Corpus(VectorSource(datos_train$benefits_effectiveness))
effects_corpus = Corpus(VectorSource(datos_train$effects_effectiveness))
```

Separamos los términos y eliminamos términos vacíos.

```
#Separamos todas las palabras entre si
benefits_items <- strsplit(as.character(benefits_corpus$content), " ")
effects_items <- strsplit(as.character(effects_corpus$content), " ")
# Para eliminar las cadenas vacías
# https://stackoverflow.com/questions/24178854/remove-blanks-from-strsplit-in-r
benefits_items <- lapply(benefits_items, function(x){x[x!=""]})
effects_items <- lapply(effects_items, function(x){x[x!=""]})
```

Generamos las estructuras transaccionales.

```
#Generamos la estructura de datos transaccional
benefits_transactions <- as(benefits_items,"transactions")
effects_transactions <- as(effects_items,"transactions")
```

Ahora tenemos que aplicar el algoritmo a priori. La estrategia consiste en hacer que los consecuentes frecuentes sean los tipos de efectividad que puede tener los medicamentos; es una forma más de filtrar reglas y no quedarnos con todas (ya que hemos visto en teoría que esto es computacionalmente muy costoso). Por ello, especificamos que los consecuentes de las reglas obtenidas sean cada una de las etiquetas posibles de este campo (con el guion puesto para las palabras compuestas como hicimos anteriormente). Los parámetros han sido fijados para que nos devuelva un conjunto de reglas razonable.

Finalmente, ordenamos las reglas obtenidas por la confianza que tienen de mayor a menor.

```
benefits_rulesInneffective <- apriori (data=benefits_transactions,
                                         parameter=list (supp=0.0007,conf = 0.9, minlen=2),
                                         appearance = list(default="lhs",rhs=c("Ineffective")),
                                         control = list (verbose=F))

benefits_rulesHighlyEffective <- apriori (data=benefits_transactions,
                                             parameter=list (supp=0.0007,conf = 0.9, minlen=2),
                                             appearance = list(default="lhs",
                                                               rhs=c("Highly-Effective")),
                                             control = list (verbose=F))

benefits_rulesConsiderablyEffective <- apriori (data=benefits_transactions,
                                                 parameter=list (supp=0.0007,conf = 0.9, minlen=2),
                                                 appearance = list(default="lhs",
                                                                   rhs=c("Considerably-Effective")),
                                                 control = list (verbose=F))

benefits_rulesModeratelyEffective <- apriori (data=benefits_transactions,
                                                parameter=list (supp=0.0007,conf = 0.9, minlen=2),
                                                appearance = list(default="lhs",
                                                                  rhs=c("Moderately-Effective")),
                                                control = list (verbose=F))

benefits_rulesMarginallyEffective <- apriori (data=benefits_transactions,
                                               parameter=list (supp=0.0007,conf = 0.9, minlen=2),
                                               appearance = list(default="lhs",
                                                                 rhs=c("Marginally-Effective")),
                                               control = list (verbose=F))

benefits_rulesInneffective <-
  sort(benefits_rulesInneffective, decreasing=TRUE, na.last=NA, by="confidence")
benefits_rulesHighlyEffective <-
  sort(benefits_rulesHighlyEffective, decreasing=TRUE, na.last = NA, by="confidence")
benefits_rulesConsiderablyEffective <-
  sort(benefits_rulesConsiderablyEffective, decreasing=TRUE, na.last=NA, by="confidence")
benefits_rulesModeratelyEffective <-
  sort(benefits_rulesModeratelyEffective, decreasing=TRUE, na.last=NA, by="confidence")
benefits_rulesMarginallyEffective <-
  sort(benefits_rulesMarginallyEffective, decreasing=TRUE, na.last=NA, by="confidence")
```

Podemos mostrar las 10 reglas de mayor confianza relacionadas con cada uno de los consecuentes.

```
detach(package:tm, unload=TRUE)
inspect(head(benefits_rulesInneffective,10))
inspect(head(benefits_rulesHighlyEffective,10))
inspect(head(benefits_rulesConsiderablyEffective,10))
inspect(head(benefits_rulesModeratelyEffective,10))
```

```
inspect(head(benefits_rulesMarginallyEffective, 10))
library(tm)
```

Hacemos el mismo proceso pero con los comentarios de los efectos secundarios.

```
effects_rulesInnefective <- apriori (data=effects_transactions,
                                       parameter=list (supp=0.0007, conf = 0.9, minlen=2),
                                       appearance = list(default="lhs", rhs=c("Ineffective")),
                                       control = list (verbose=F))

effects_rulesHighlyEffective <- apriori (data=effects_transactions,
                                           parameter=list (supp=0.0007, conf = 0.9, minlen=2),
                                           appearance = list(default="lhs", rhs=c("Highly-Effective")),
                                           control = list (verbose=F))

effects_rulesConsiderablyEffective <- apriori (data=effects_transactions,
                                                parameter=list (supp=0.0007, conf = 0.9, minlen=2),
                                                appearance = list(default="lhs",
                                                               rhs=c("Considerably-Effective")),
                                                control = list (verbose=F))

effects_rulesModeratelyEffective <- apriori (data=effects_transactions,
                                              parameter=list (supp=0.0007, conf = 0.9, minlen=2),
                                              appearance = list(default="lhs",
                                                               rhs=c("Moderately-Effective")),
                                              control = list (verbose=F))

effects_rulesMarginallyEffective <- apriori (data=effects_transactions,
                                               parameter=list (supp=0.0007, conf = 0.9, minlen=2),
                                               appearance = list(default="lhs",
                                                               rhs=c("Marginally-Effective")),
                                               control = list (verbose=F))

effects_rulesInnefective <- sort(effects_rulesInnefective,
                                   decreasing=TRUE, na.last=NA, by="confidence")
effects_rulesHighlyEffective <- sort(effects_rulesHighlyEffective,
                                      decreasing=TRUE, na.last=NA, by="confidence")
effects_rulesConsiderablyEffective <- sort(effects_rulesConsiderablyEffective,
                                             decreasing=TRUE, na.last = NA, by="confidence")
effects_rulesModeratelyEffective <- sort(effects_rulesModeratelyEffective,
                                           decreasing=TRUE, na.last = NA, by="confidence")
effects_rulesMarginallyEffective <- sort(effects_rulesMarginallyEffective,
                                          decreasing=TRUE, na.last=NA, by="confidence")
```

Podríamos mostrar las 10 reglas con mayor confianza relacionadas con cada uno de los consecuentes si lo deseamos.

```
detach(package:tm, unload=TRUE)
inspect(head(effects_rulesInnefective, 10))
inspect(head(effects_rulesHighlyEffective, 10))
inspect(head(effects_rulesConsiderablyEffective, 10))
inspect(head(effects_rulesModeratelyEffective, 10))
inspect(head(effects_rulesMarginallyEffective, 10))
library(tm)
```

Finalmente podemos extraer todas los mapas de estas reglas de asociación.

```
#Obtenemos las imágenes relacionadas con los comentarios sobre beneficios
# de los medicamentos
png("figuras/asociacion/benefits_Inneffective.png",width=1800,
    height=1700,units="px",pointsize=10,bg="white",res=300)
plot(benefits_rulesInneffective, method="graph")
title("Comentarios beneficios sobre inefectividad")
dev.off()

png("figuras/asociacion/benefits_HighlyEffective.png",width=1800,
    height=1700,units="px",pointsize=10,bg="white",res=300)
plot(benefits_rulesHighlyEffective, method="graph")
title("Comentarios beneficios sobre altamente efectivos")
dev.off()

png("figuras/asociacion/benefits_ConsiderablyEffective.png",width=1800,
    height=1700,units="px",pointsize=10,bg="white",res=300)
plot(benefits_rulesConsiderablyEffective, method="graph")
title("Comentarios beneficios sobre considerablemente efectivos")
dev.off()

png("figuras/asociacion/benefits_ModeratelyEffective.png",width=1800,
    height=1700,units="px",pointsize=10,bg="white",res=300)
plot(benefits_rulesModeratelyEffective, method="graph")
title("Comentarios beneficios sobre moderadamente efectivos")
dev.off()

png("figuras/asociacion/benefits_MarginallyEffective.png",width=1800,
    height=1700,units="px",pointsize=10,bg="white",res=300)
plot(benefits_rulesMarginallyEffective, method="graph")
title("Comentarios beneficios sobre casi inefectivo")
dev.off()
```

Podemos ver que los mapas de reglas salen bastante saturadas igualmente, ya que saca bastantes. Podríamos reducir el número de reglas manipulando los umbrales de soporte y confianza si lo deseamos, aunque llevaría tiempo de computación realizar esas pruebas.

Vamos a comentar por ejemplo la inefectividad, que ha salido más simple y fácil de visualizar:

Como se puede ver en la imagen, los términos que giran en torno de a la inefectividad tienen bastante sentido. Por ejemplo, la palabra *none*. Lo normal es que, cuando un medicamento es inefectivo, las personas realicen comentarios del estilo “no me ha hecho nada” o “no sirve de nada”. Otros como *pain* o *problem* tienen sentido ya que si un medicamento es inefectivo es normal que aparezcan palabras “negativas”.

```
# Obtenemos las imágenes relacionadas con los comentarios
# sobre efectos secundarios de los medicamentos
png("figuras/asociacion/effects_Inneffective.png",width=1800,
    height=1700,units="px",pointsize=10,bg="white",res=300)
plot(effects_rulesInneffective, method="graph")
title("Comentarios efectos secundarios sobre inefectividad")
dev.off()

png("figuras/asociacion/effects_HighlyEffective.png",width=1800,
    height=1700,units="px",pointsize=10,bg="white",res=300)
plot(effects_rulesHighlyEffective, method="graph")
```

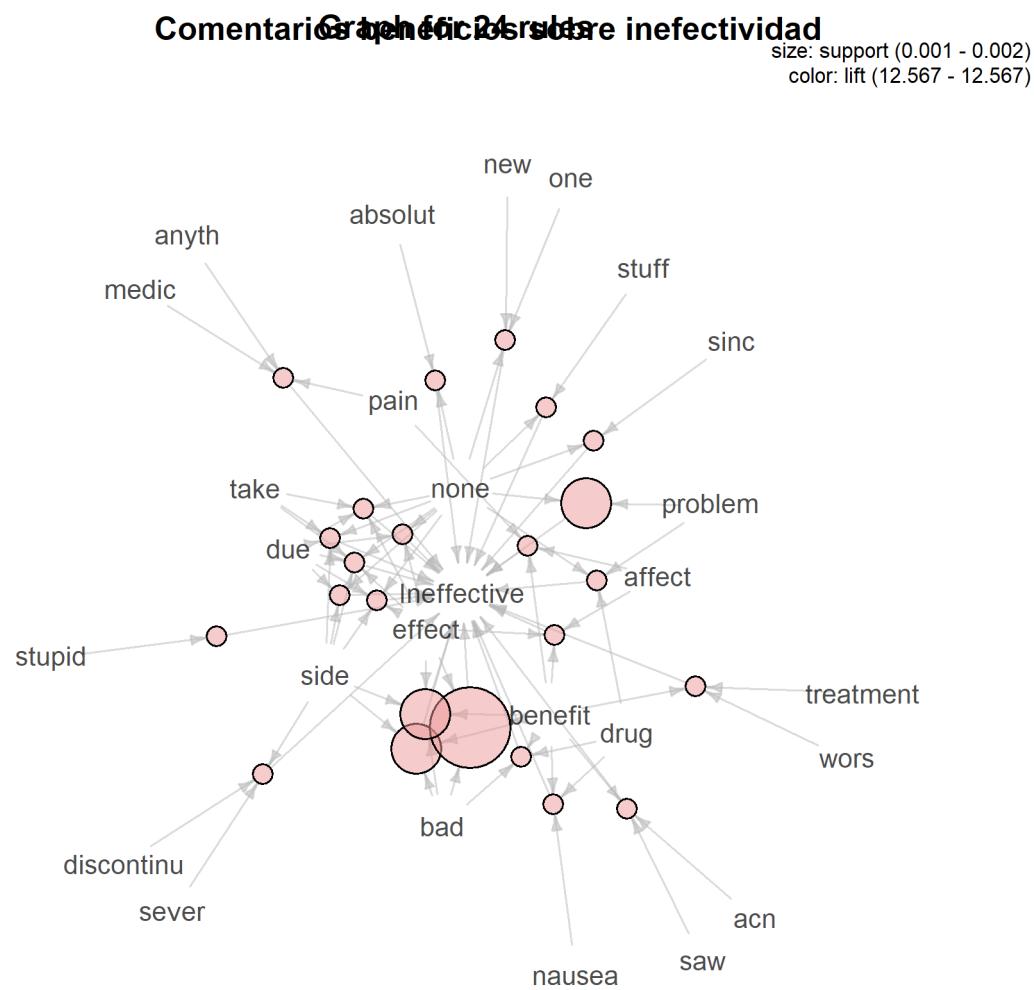


Figura 24: Mapa de reglas de asociación con Inneffective en el consecuente en comentarios sobre beneficios.

```

title("Comentarios efectos secundarios sobre altamente efectivos")
dev.off()

png("figuras/asociacion/effects_ConsiderablyEffective.png",width=1800,
    height=1700,units="px",pointsize=10,bg="white",res=300)
plot(effects_rulesConsiderablyEffective, method="graph")
title("Comentarios efectos secundarios sobre considerablemente efectivos")
dev.off()

png("figuras/asociacion/effects_ModeratelyEffective.png",width=1800,
    height=1700,units="px",pointsize=10,bg="white",res=300)
plot(effects_rulesModeratelyEffective, method="graph")
title("Comentarios efectos secundarios sobre moderadamente efectivos")
dev.off()

```

```

png("figuras/asociacion/effects_MarginallyEffective.png",width=1800,
    height=1700,units="px",pointsize=10,bg="white",res=300)
plot(effects_rulesMarginallyEffective, method="graph")
title("Comentarios efectos secundarios sobre casi inefectivo")
dev.off()

```

Sobre los comentarios de efectos secundarios podemos analizar, por ejemplo, el concepto de altamente efectivo:

Comentarios efectos secundarios sobre altamente efectivos

Graph for 10 rules
size: support (0.003 - 0.007)
color: lift (2.1 - 2.334)

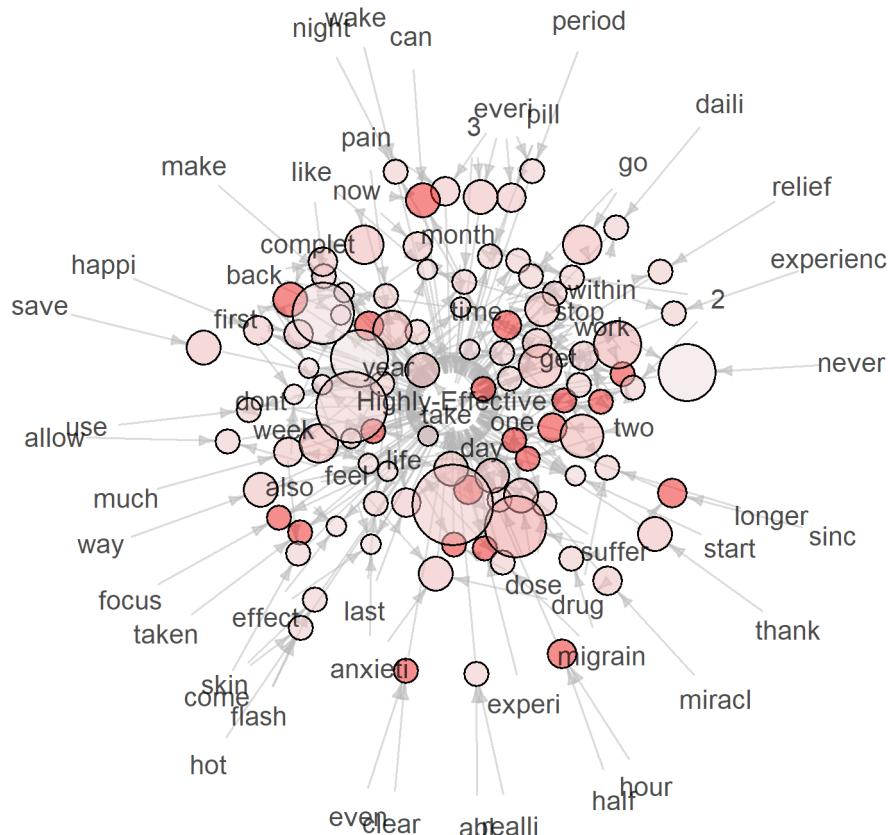


Figura 25: Mapa de reglas de asociación con Highly-Effective en el consecuente en comentarios sobre efectos secundarios.

Como podemos apreciar en la imagen, el mapa de por sí es poco visualizable debido a la cantidad de reglas que aparecen en tan poco espacio. En definitiva, si nos fijamos bien podemos darnos cuenta de que tiene bastante sentido.

6. Agrupamiento y Clustering

Para la realización del Clustering, se debe destacar el uso de este enlace.

6.1. Introducción

En el siguiente capítulo nos enfrentamos al desarrollo de la clasificación no supervisada de patrones en grupos, mas conocido como clustering. Dichos grupos se establecen como particiones en las que las observaciones pertenecientes son similares entre ellas y distintas a las observaciones del resto de particiones.

Debido a la naturaleza de nuestros datos, y la estructuración que hemos hecho para su tratamiento, tenemos tanto datos numéricos (correspondientes a valores de distintas puntuaciones asignadas a los agrupamientos), como datos textuales. Por ello, vamos a dividir esta sección en dos problemas, que se tratarán de forma separada.

- **Agrupamiento de los fármacos en base a las puntuaciones obtenidas.** En este caso, tendremos en cuenta los distintos atributos numéricos que nos dan información acerca de las opiniones del fármaco, y los utilizaremos para realizar dicha clasificación. Estos atributos son *rating*, *effectivenessNumber* y *sideEffectsInverse*.
- **Agrupamiento de los fármacos en base a las opiniones de los usuarios.** Para este apartado, realizaremos, en primer lugar, un agrupamiento en base a *benefitsReview*, y por otra parte, en función de *sideEffectsReview*.

Para los distintos apartados, se estudiarán distintas técnicas de agrupamiento, con la explicación y comparación correspondiente de nuestros resultados, para así poder ver su utilidad y adecuación de la técnica en base a las características de los datos que estamos tratando (puesto que habrá técnicas que, debido a la naturaleza de nuestros datos, sean más apropiadas que otras). En esta sección se verá, como se ha comentado en capítulos previos, que en nuestra base de datos existe un alto factor de subjetividad, que influirá en nuestros resultados.

6.2. Agrupamiento de los fármacos en base a las puntuaciones obtenidas

6.2.1. Establecimiento de una semilla

Como esta técnica tiene un factor de aleatoriedad, lo primero que vamos a hacer es establecer una semilla común, para así poder replicar los resultados en las distintas ejecuciones futuras que queramos realizar.

6.2.2. Lectura y adecuación de los datos

Como proceso anterior al punto en el que nos encontramos, se realizó todo el preprocesado de los datos. Ahora, para su uso en las técnicas de clustering, debemos tomar las columnas de datos que más nos interesan para el análisis.

Como ya se comentó en la introducción de este capítulo, estas serán *rating*, *effectivenessNumber* y *sideEffectsInverse*. Por ello, tras leer los datos, lo primero que vamos a hacer, es crear un nuevo data-frame de la forma específica que necesitamos que estén los datos para esta técnica en concreto. Si consultamos el data-frame original, las variables que necesitamos se corresponde con las columnas número 2, 12 y 19 respectivamente, y estas serán por tanto, las que filtraremos para el nuevo conjunto de datos con el que vamos a trabajar.

```
# Establecemos una semilla para obtener siempre los mismos resultados
set.seed(3)

# Cargamos los datos
datos_train <- read.table("datos/datos_train_preprocesado.csv", sep=",",
```

```

comment.char = "", quote = "\\"", header=TRUE)

datos_test <- read.table("datos/datos_test_preprocesado.csv", sep=",",
                           comment.char = "", quote = "\\"", header=TRUE)

# Nos quedamos con las columnas que nos interesan
# Las columnas que tenemos son: RATING, SIDE_EFFECTS_INVERSE, EFFECTIVENESS_NUMBER
datos_train2 = datos_train[c(2,12,9)]
datos_test2 = datos_test[c(2,12,9)]

```

A continuación, podemos ver la estructura que estamos utilizando para el conjunto de train.

```
head(datos_train2)
```

```

##   rating sideEffectsInverse effectivenessNumber
## 1      4                  4                  5
## 2      1                  2                  5
## 3     10                 5                  5
## 4      3                  4                  2
## 5      2                  2                  2
## 6      1                  2                  1

```

Como ya sabemos, todos aquellos cambios que realicemos en train, tienen que ser replicados para el conjunto de test. Como se puede ver a continuación, se ha realizado este cambio.

```
head(datos_test2)
```

```

##   rating sideEffectsInverse effectivenessNumber
## 1      9                  4                  4
## 2      9                  4                  5
## 3      4                  2                  3
## 4     10                 5                  5
## 5     10                 4                  5
## 6      2                  5                  2

```

Sin embargo, esta selección inicial de columnas no es suficiente para poder empezar a aplicar las técnicas, ya que, a pesar de que nos hemos quedado con los atributos que necesitamos para dicho agrupamiento, esta estructura tiene deficiencias:

- En primer lugar, hemos perdido el nombre de los medicamentos de nuestro conjunto de datos. Con esto queremos decir que, con el dataset que tenemos actualmente, no sabemos el nombre del fármaco que se asocia a cada fila. Esto es importante a la hora de visualizar el agrupamiento, ya que, la única manera de ver qué fármacos aparecen en cada clúster que visualicemos, es asociándole etiquetas. De dicha forma, podremos ver con qué fármaco se corresponde cada item de la gráfica.

Como solución a este problema, vamos a renombrar las filas del dataset, de forma que cada fila tenga como nombre el del fármaco correspondiente. Esto tiene un problema, y es que en un hipotético caso en el que tuviéramos varios comentarios para un mismo fármaco, no podríamos realizar este cambio de nombres, puesto que dos filas de un mismo dataset no pueden tener el mismo nombre. Con esta reflexión, nos surge un nuevo problema, y es que, no queremos que el mismo fármaco salga en distintos agrupamientos. Lo que nos parece más coherente es, que se agrupen los fármacos, de forma que cada uno de los fármacos pertenezca a un clúster. La idea general a la que pretendemos llegar, es a obtener distintas agrupaciones de fármacos en base a sus características. Tenemos dos formas de obtener un dataset donde cada fila sea un fármaco:

- En el caso de que tengamos x items asociados a un mismo fármaco (varios comentarios respecto a un único fármaco), quedarnos únicamente con un item. Esta solución no nos pareció la más adecuada, puesto que **estaríamos perdiendo información**.

- Realizar alguna medida estadística que nos permita agregar dicha información. Como estuvimos observando que las opiniones respecto a los fármacos suelen coincidir cuando hay más de una, esta opción fue la que vimos menos problemática (aunque como ya sabemos, estaríamos perdiendo algo de información igualmente), y por tanto la que utilizaremos en esta sección.

```
# Arreglamos el conjunto de datos para poder trabajar con lo que nos interesa

# Obtenemos en una variable todos los nombres de fármacos que existen en el conjunto de train
nombres_farmacos_train <- unique(datos_train[,1])

# Hacemos lo mismo para test
nombres_farmacos_test <- unique(datos_test[,1])

# Creamos una matriz (vacía) con tantas filas como fármacos haya, y tantas columnas como #atributos queramos utilizar. En este caso son 3 columnas porque necesitamos guardar la #info de "rating", "sideEffectsInverse" y "effectivenessNumber".
datos_procesados_train <- matrix(ncol=3, nrow=length(nombres_farmacos_train))
datos_procesados_test <- matrix(ncol=3, nrow=length(nombres_farmacos_test))

# Primero procesamos TRAIN

# Convertimos la estructura que tiene todos los nombres de los fármacos, en un data-frame, # para poder trabajar con esta información de manera más cómoda
df_farmacos_train = as.data.frame(nombres_farmacos_train)

# Para cada uno de los fármacos existentes, vamos a guardar su información asociada en # la matriz creada anteriormente
for(i in 0:length(nombres_farmacos_train)){

  # Obtenemos todas las filas del dataset que se correspondan con el fármaco número i.
  filas_farmaco_train <- datos_train2[which(datos_train$urlDrugName == nombres_farmacos_train[i]),]

  # Como pueden ser más de una fila, resumimos dicha información.
  mean_rating_train <- mean(filas_farmaco_train$rating)
  mean_side_effect_train <- mean(filas_farmaco_train$sideEffectsInverse)
  mean_effectiveness_train <- mean(filas_farmaco_train$effectivenessNumber)

  # La información resumida es la que asociamos a dicho fármaco, montando la fila de la # manera adecuada. Nótese que redondeamos el número medio obtenido para sideEffects y # effectivenessNumber. Esto es porque necesitamos que sea un número entero (recordemos # que estos dos valores se corresponden con etiquetas).
  datos_procesados_train[i,] <- c(mean_rating_train, round(mean_side_effect_train), round(mean_effectiveness_train))
}

# Procesamos TEST de forma análoga a TRAIN.
df_farmacos_test = as.data.frame(nombres_farmacos_test)

for(i in 0:length(nombres_farmacos_test)){

  filas_farmaco_test <- datos_test2[which(datos_test$urlDrugName == nombres_farmacos_test[i]),]

  mean_rating_test <- mean(filas_farmaco_test$rating)
```

```

mean_side_effect_test <- mean(filas_farmaco_test$sideEffectsInverse)
mean_effectiveness_test <- mean(filas_farmaco_test$effectivenessNumber)

datos_procesados_test[i,] <- c(mean_rating_test, round(mean_side_effect_test), round(mean_effectiveness_test))

# Por último, convertimos las matrices procesadas anteriormente en data-frame
data_train_procesado <- data.frame(datos_procesados_train)
data_test_procesado <- data.frame(datos_procesados_test)

# Una vez creados los data-frame, les asignamos nombres a las filas y columnas de
# los nuevos data-frames.
rownames(data_train_procesado) <- nombres_farmacos_train
rownames(data_test_procesado) <- nombres_farmacos_test
colnames(data_train_procesado) <- c("rating", "sideEffectsInverse", "effectivenessNumber")
colnames(data_test_procesado) <- c("rating", "sideEffectsInverse", "effectivenessNumber")

```

A continuación se puede visualizar parte del nuevo data-frame que hemos creado para el conjunto de train.

```
head(data_train_procesado)
```

	rating	sideEffectsInverse	effectivenessNumber
## enalapril	5.000000	4	4
## ortho-tri-cyclen	7.437500	4	4
## ponstel	10.000000	5	5
## prilosec	7.000000	4	4
## lyrical	5.523810	3	3
## propecia	6.684211	4	4

A continuación se puede visualizar parte del nuevo data-frame que hemos creado para el conjunto de test.

```
head(data_test_procesado)
```

	rating	sideEffectsInverse	effectivenessNumber
## biaxin	5.000	3	4
## lamictal	8.300	4	4
## depakene	4.000	2	3
## sarafem	8.500	4	4
## accutane	7.375	3	4
## carbamazepine	8.000	3	4

Estos dos conjuntos de datos son con los que trabajaremos en el resto de técnicas asociadas a agrupamiento de items, que realizaremos a lo largo de este apartado.

6.2.3. K-medias

La idea básica del presente método es agrupar las observaciones en K clusters distintos, siendo el valor de K preestablecido mediante análisis previos. Entonces, K-medias encuentra los K mejores clusters, siendo estos aquellos cuya varianza interna sea lo más pequeña posible.

Como queremos, al fin y al cabo, agrupar los fármacos en función de los datos, vamos a comenzar por lo más intuitivo posible. La primera pregunta que nos surgió fue, ¿sería posible agrupar los fármacos en dos clústers, de forma que tuvieran un grupo para los fármacos “buenos” (aquellos que tienen una puntuación alta) y otro grupo para los fármacos “malos” (aquellos inefectivos, con efectos secundarios)? En un primer lugar puede parecer, que la agrupación resultante no tiene que ser coherente en cuanto a lo que nosotros buscamos

encontrar, puesto que al fin y al cabo, son datos, y como hemos dicho anteriormente tenemos un gran factor de subjetividad. Pero, observando nuestros datos, nos dimos cuenta de una cosa, y es que, cuando alguien tenía una buena opinión acerca de un fármaco, le asignaba a éste un alto valor de *effectiveness* y un alto bajo valor en *sideEffects*, lo que, a priori, es totalmente lógico e intuitivo.

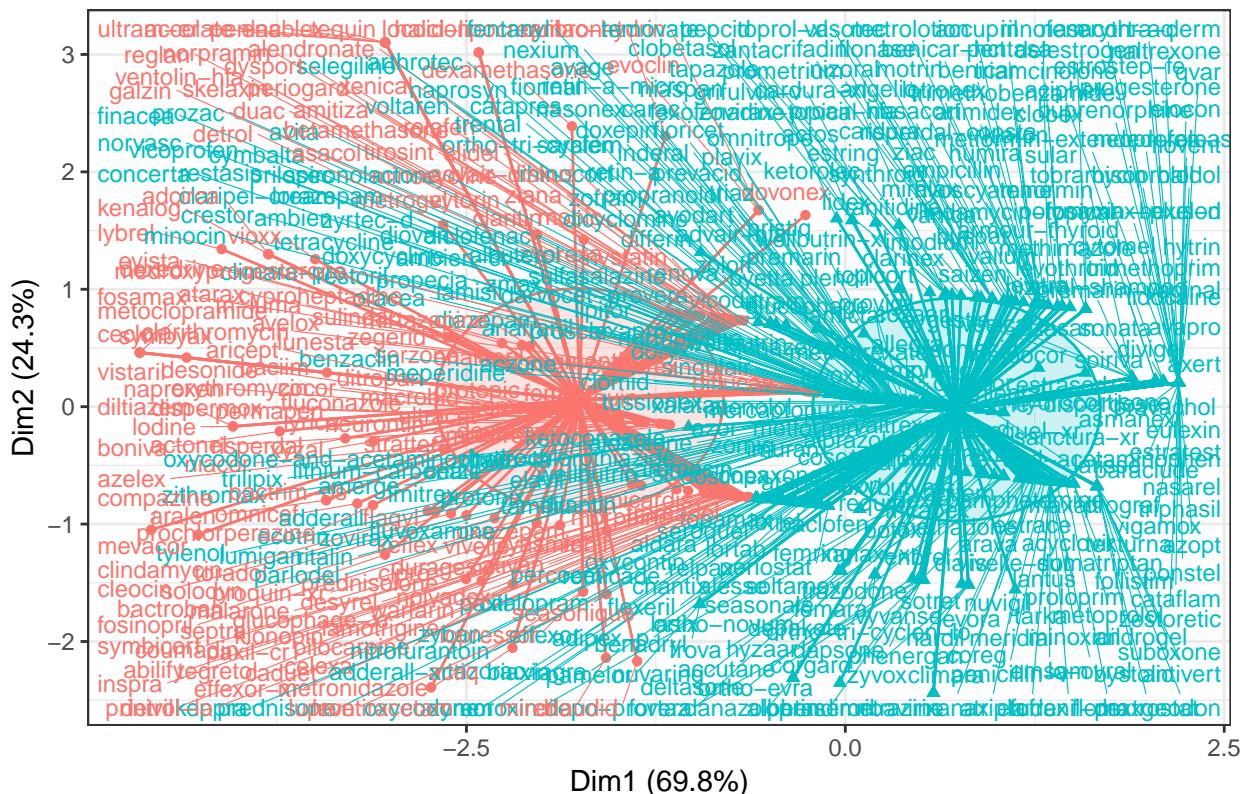
Puesto que los datos que estamos utilizando en el conjunto de train, son efectivamente esos, cabe esperar que sí podamos obtener una clasificación respecto a fármacos cuya opinión es positiva, y fármacos con opinión negativa. Siguiendo esta idea, vamos a probar a establecer un número de cluster igual a 2 ($K=2$), y posteriormente, intentaremos buscar un sentido a las agrupaciones que encontramos, ya que, al tener únicamente dos grupos, nos resultará sencillo.

Para ello, utilizaremos la función `kmeans`, que nos permitirá obtener la agrupación que hemos estado comentando anteriormente. Como parámetros, mencionar, que hemos establecido el número de clústeres a 2 (tal y como hemos expuesto en el párrafo anterior), y que la distancia que utilizaremos en el algoritmo será la distancia *euclídea*. En la siguiente imagen, podemos ver el resultado alcanzado en este caso. Aunque resulta poco visible debido a la gran cantidad de información que hay en ella, podemos ver cómo prácticamente el espacio está dividido en dos clústeres, medianamente bien predefinidos.

```
# ALgoritmo K-means con los datos de test y 2 clusters
res_clustering <- kmeans(data_train_procesado, 2)

# Lo mostramos
fviz_cluster(object = res_clustering, data = data_train_procesado, show.clust.cent = TRUE,
              ellipse.type = "euclid", star.plot = TRUE, repel = TRUE, labelsize = 9) +
  labs(title = "Resultados clustering K-means (con etiquetas)") +
  theme_bw() +
  theme(legend.position = "none")
```

Resultados clustering K-means (con etiquetas)



Intuitivamente, hemos intentado encontrar una relación entre los dos agrupamientos, viendo que, efectivamente,

con los distintos ejemplos que hemos probado hemos podido ver, que los del grupo azul suelen tener una efectividad entre 1 y 3, mientras que los del grupo rojo suelen tener dicho valor entre 3 y 5. Idem para la variable de efectos secundarios, lo que nos lleva a pensar que, realmente, hemos podido obtener una buena clasificación de los fármacos en base a la opinión (positiva o negativa) de los usuarios.

Véase por ejemplo, algunos ejemplos del grupo que se corresponde con el color rojo en la gráfica. Como podemos ver, efectivamente su valor de efectividad (cuarta columna en el dataset), tiene valor bajo. Además, si miramos el valor en efectos secundarios, también es bajo (recordemos que hicimos la inversa a dicho valor, y que cuanto más bajo sea el valor de esa columna, más negativos son sus efectos).

```
train_medicine = bind_cols(data_train_procesado,df_farmacos_train)
ejemplo1 <- train_medicine[which(train_medicine$nombr..._farmacos_train == "baciim"),]
ejemplo2 <- train_medicine[which(train_medicine$nombr..._farmacos_train == "boniva"),]
ejemplo3 <- train_medicine[which(train_medicine$nombr..._farmacos_train == "lodine"),]
ejemplo4 <- train_medicine[which(train_medicine$nombr..._farmacos_train == "mevacor"),]

pruebas_rojo = bind_rows(ejemplo1,ejemplo2,ejemplo3,ejemplo4)

pruebas_rojo

##   rating sideEffectsInverse effectivenessNumber nombr..._farmacos_train
## 1      5              2                  1             baciim
## 2      2              2                  2             boniva
## 3      1              2                  2             lodine
## 4      2              1                  2            mevacor
```

Hemos visto la tendencia. Ahora vamos a calcular, de forma media, el valor asociado a effectivenessNumber en todos los fármacos que forman parte de este primer cluster (el de color azul, el cuál se corresponde con la etiqueta 1). Como se puede ver, el valor obtenido no llega a 3.

```
# Para poder obtener las etiquetas y fármacos asociados del objeto clúster, pasamos a matriz
m = as.matrix(res_clustering$cluster)
x=as.matrix( m[,1] == 1)

# nos quedamos con los nombres de los fármacos que estén en el cluster 1
items_etiqueta1= x[,1]==1

# nos quedamos con los fármacos del dataset que se correspondan con dicho cluster
farmacos_etiqueta1 = data_train_procesado[items_etiqueta1,]

# hacemos la media
mean(farmacos_etiqueta1$effectivenessNumber)
```

```
## [1] 2.972973
```

Si ahora hacemos el mismo proceso para 4 fármacos del conjunto azul, podemos ver que efectivamente, el comportamiento es el esperado. Tenemos altos valores de efectividad, y pocos efectos secundarios.

```
ejemplo1 <- train_medicine[which(train_medicine$nombr..._farmacos_train == "zyvox"),]
ejemplo2 <- train_medicine[which(train_medicine$nombr..._farmacos_train == "bisoprolol"),]
ejemplo3 <- train_medicine[which(train_medicine$nombr..._farmacos_train == "cefzil"),]
ejemplo4 <- train_medicine[which(train_medicine$nombr..._farmacos_train == "benicar"),]

pruebas_azul = bind_rows(ejemplo1,ejemplo2,ejemplo3,ejemplo4)

pruebas_azul

##   rating sideEffectsInverse effectivenessNumber nombr..._farmacos_train
```

```
## 1      9          3      5          zyvox
## 2     10          5      5          bisoprolol
## 3      8          4      4          cefzil
## 4      9          5      4          benicar
```

De nuevo, vamos a consultar el valor medio de efectividad que se consigue si consultamos dicho atributo para todos los fármacos que se han agrupado en el cluster que se corresponde con el color rojo en el gráfico (grupo 2). Para ello, simplemente hacemos la media de todos aquellos fármacos no incluidos en el grupo 1, y, tal y como es de esperar, el resultado está entre 3 y 5, lo cual nos **reafirma al pensar que el agrupamiento se ha hecho en función de fármaco con opinión positiva y fármaco con opinión negativa.**

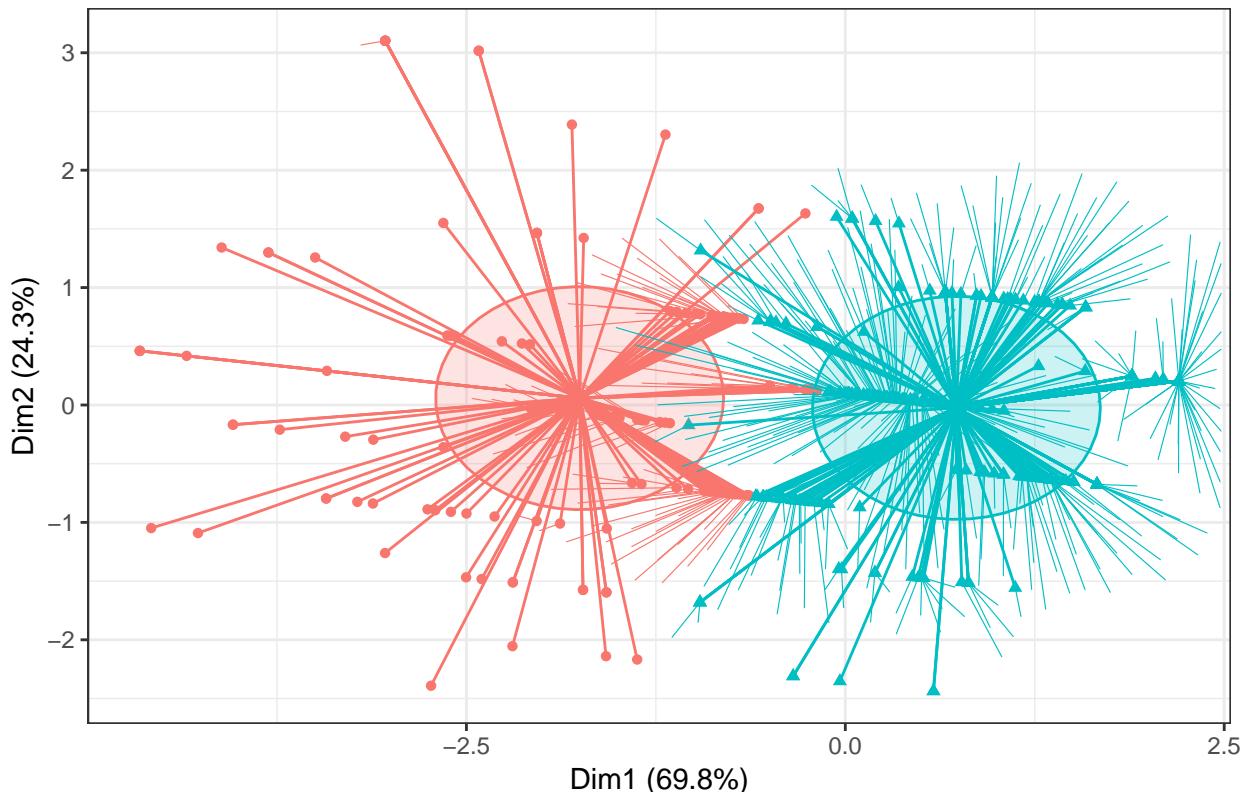
```
farmacos_etiqueta2 = data_train_procesado[-items_etiqueta1,]
mean(farmacos_etiqueta2$effectivenessNumber)
```

```
## [1] 3.894212
```

Como aclaración, debido a la poca visibilidad del clúster, vamos a mostrar el mismo gráfico, pero esta vez sin etiquetas, de forma que podamos ver bien los agrupamientos que se llevan a cabo. Se puede observar este hecho en la siguiente imagen del documento.

```
fviz_cluster(object = res_clustering, data = data_train_procesado, show.clust.cent = TRUE,
             ellipse.type = "euclid", star.plot = TRUE, repel = TRUE, labelsize=0) +
  labs(title = "Resultados clustering K-means (sin etiquetas)") +
  theme_bw() +
  theme(legend.position = "none")
```

Resultados clustering K-means (sin etiquetas)



Sin embargo, tal y como ya se ha dicho, esto es una conclusión, intuitiva, que hemos obtenido nosotros gracias a la gran cantidad de información que nos ha dado el preprocesamiento y estudio exploratorio de los datos.

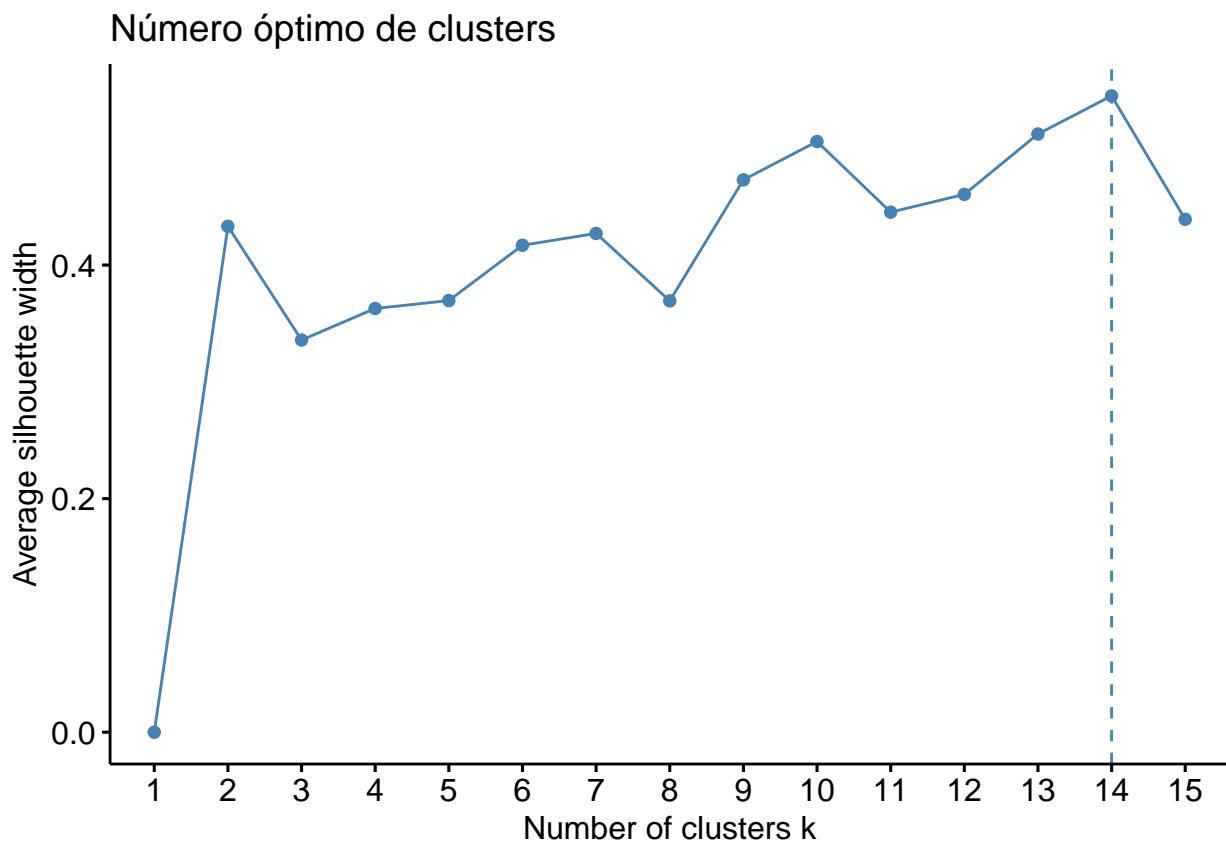
Sin embargo, para validar nuestro modelo, podemos utilizar también algún tipo de medida que nos permita

saber cómo de bueno es el agrupamiento o asignación que se ha llevado a cabo con la técnica. Para llevar a cabo este procedimiento, vamos a utilizar una de las técnicas que hemos visto en la asignatura: el método *Average Silhouette*. Para ello vamos a utilizar lo que se denomina *índice silueta*, el cuál es un coeficiente que compara la distancia de un ítem con el resto de ítems, ya sea del mismo clúster, o del resto de ellos. Se mueve en el rango [-1,1], de forma que, cuánto mayor sea el valor, mayor es su adecuación en el agrupamiento realizado. A su vez, valores negativos llevarán a pensar que se ha llevado a cabo una asignación incorrecta y poco fiable. Este método es muy potente, ya que nos permite evaluar el agrupamiento tanto a nivel de cada uno de los ítems, como a nivel de clúster concreto, o a clústeres totales en el agrupamiento.

Teniendo en cuenta dicho índice, el número de clústeres con el que se obtendrían mejores resultados para el agrupamiento sería aquel **cuya media del índice silueta fuese la mayor posible**. Podemos ver el resultado de esta *media*, para distintos tamaños de clúster en la siguiente imagen. En la siguiente imagen, podemos ver, cómo en este caso, el índice silueta indica que el número óptimo de clúster es 14. Pero sin embargo, tal y como ya sabemos, ese no tiene por qué ser el más adecuado a usar, puesto que, en la práctica, también debemos de tener en cuenta otras cuestiones que hemos visto en la asignatura, como el esfuerzo computacional o el sobreaprendizaje que añadimos a nuestro modelo cuando intentamos precisar demasiado. Al final, nos quedaremos con un número de cluster que, nos permita una mejora sustancial, y a partir del cuál no haya una mejora sustancial.

En dicha gráfica podemos ver que usar dos clusters tiene un valor alto de índice medio de silueta, lo que nos hace pensar que, es un valor que funciona bien y cuyo agrupamiento puede ser fiable (aspecto que ya intuíamos).

```
# datos ya lo tenemos previamente escalado (K-medias)
datos <- scale(data_train_procesado)
fviz_nbclust(x = datos, FUNcluster = kmeans, method = "silhouette", k.max = 15) +
  labs(title = "Número óptimo de clusters")
```

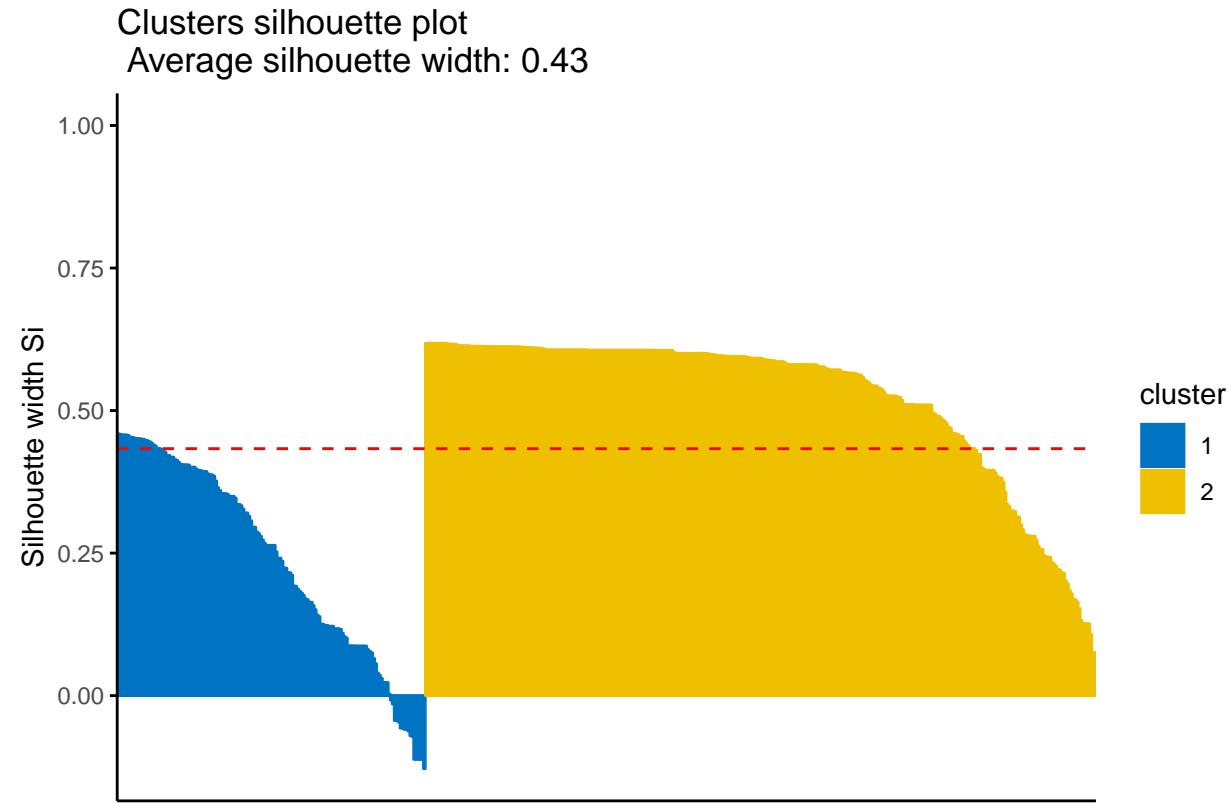


A su vez, podemos realizar una validación interna del clúster haciendo uso del ya mencionado *índice silueta*.

Se puede visualizar el resultado en la siguiente gráfica.

```
datos <- scale(data_train_procesado)
km_clusters <- eclust(x = datos, FUNcluster = "kmeans", k = 2, seed = 123,
                      hc_metric = "euclidean", nstart = 50, graph = FALSE)
fviz_silhouette(sil.obj = km_clusters, print.summary = TRUE, palette = "jco",
                 ggtheme = theme_classic())

##   cluster size ave.sil.width
## 1        1    158      0.24
## 2        2    344      0.52
```



A

su vez, podemos ver la media del índice silueta obtenida para cada uno de los clusters, tal y como se puede ver a continuación. Como podemos ver, la media obtenida para el primer cluster, es próxima a 0, lo que nos puede indicar que los items no están en el grupo correcto. De hecho, si observamos la gráfica anterior, podemos ver que algunos datos rozan el negativo, y que se corresponderán a aquellos más próximos al otro cluster.

```
# Media silhouette por cluster
km_clusters$silinfo$clus.avg.widths
```

```
## [1] 0.2357250 0.5238887
```

En realidad, esto es normal que ocurra, sobretodo en nuestro caso, puesto que:

1. Es normal que algun valor concreto se escape, y no se agrupe bien, sobretodo cuando no hemos usado el valor óptimo de clusters.
2. Hay muchos items cuyo valor de effectiveness es 3. Recordemos que este valor se mueve en el rango [1,3], por lo que, los items cuyo valor sea el valor medio, puede darse el caso de que sufran una mala clasificación. De hecho, podemos filtrar los índices silueta de nuestros items, tal y como se puede ver a continuación. De hecho, en la siguiente salida, podemos estos items, tienen un valor negativo muy muy

muy cercano a 0, lo que hace más evidente dicha conclusión.

```
km_clusters$silinfo$widths %>% filter(sil_width <= 0)
```

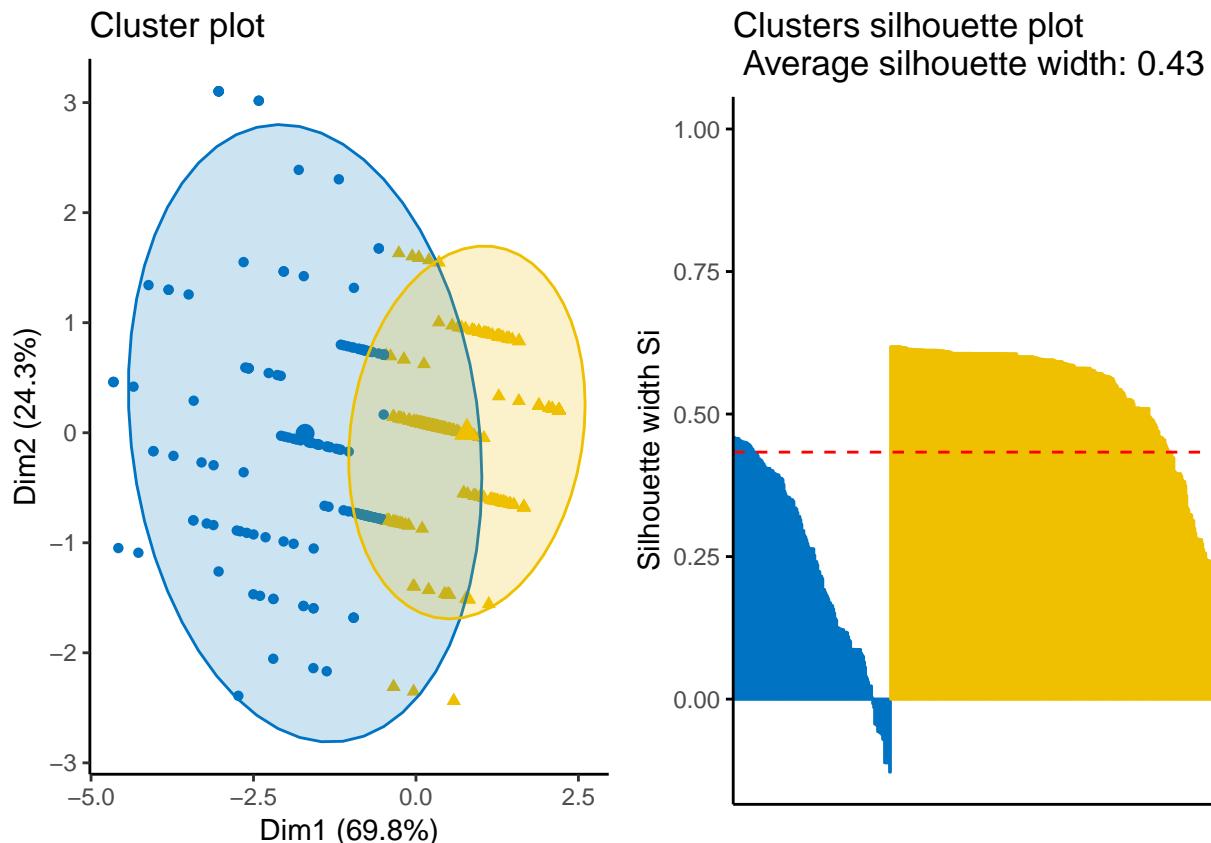
```
##   cluster neighbor   sil_width
## 1       1         2 -0.008108751
## 2       1         2 -0.014884917
## 3       1         2 -0.043886193
## 4       1         2 -0.043886193
## 5       1         2 -0.046451962
## 6       1         2 -0.057008242
## 7       1         2 -0.057008242
## 8       1         2 -0.059306622
## 9       1         2 -0.059306622
## 10      1         2 -0.062150055
## 11      1         2 -0.070412470
## 12      1         2 -0.071749168
## 13      1         2 -0.111808796
## 14      1         2 -0.111808796
## 15      1         2 -0.111808796
## 16      1         2 -0.111808796
## 17      1         2 -0.111808796
## 18      1         2 -0.128184298
```

De hecho, muchas veces, estos valores negativos se corresponden con items que se encuentran en la frontera, pero sobretodo si los clusters se solapan. Tal y como podemos ver en el siguiente conjunto de imágenes, esta podría ser la justificación para lo que nos sucede, explicando así los valores silueta obtenidos.

```
km_clusters <- eclust(x = datos, FUNcluster = "kmeans", k = 2, seed = 123,
                      hc_metric = "euclidean", nstart = 50, graph = FALSE)
p1 <- fviz_cluster(object = km_clusters, geom = "point", ellipse.type = "norm",
                     palette = "jco") +
  theme_classic() + theme(legend.position = "none")

p2 <- fviz_silhouette(sil.obj = km_clusters, print.summary = FALSE,
                      palette = "jco", ggtheme = theme_classic()) +
  theme(legend.position = "none")

ggarrange(p1, p2)
```



Por último para esta técnica, se ha observado que, muchos de los fármacos que estamos tratando en el conjunto de train, vuelven a aparecer en el conjunto de test. Entonces aquí nos surge la siguiente duda: si los agrupamos utilizando como centros los proporcionados con el conjunto de TRAIN, y les asignamos a cada uno de los items en TEST, el clúster al que tiene mínima distancia, ¿nos asignará al grupo correcto?

En base a esta idea, vamos a intentar simular una especie de predicción, para ver que tal agrupa el conjunto de test.

Como ya se ha comentado, mediante el clustering que nos ha generado train, buscamos clasificar cada item de test y comprobaremos el acierto de dicha predicción. Posteriormente, comprobaremos si ese mismo fármaco, ha sido asignado al mismo grupo.

6.2.3.1. Función de predicción

En primer lugar, nos quedaremos con aquellos fármacos que aparecen tanto en Train como en Test (que son los que podremos comprobar). En segundo lugar, buscaremos el cluster que se le ha asociado a dicho fármaco, tanto en el caso de Train como en el de Test. Por último, realizaremos un recuento de en cuántos casos coincide el agrupamiento.

```
test_preds <- predict(res_clustering, data_test_procesado)
table(test_preds)
```

```
## test_preds
##   1   2
## 113 200

datos_test_cluster = cbind(test_preds)
rownames(datos_test_cluster) = rownames(data_test_procesado)
test_result = res_clustering
```

```

test_result$cluster = datos_test_cluster

predicciones = function(cluster_train, cluster_test) {
  count = 0
  count_na = 0
  x = intersect(datos_test$urlDrugName, datos_train$urlDrugName)
  for(farmaco in x){
    indice_train = match(farmaco,df_farmacos_train$nombres_farmacos_train)
    indice_test = match(farmaco,df_farmacos_test$nombres_farmacos_test)

    # cluster que tiene en train dicho fármaco
    c_train = res_clustering$cluster[indice_train]

    # cluster que tiene en test dicho fármaco
    c_test = test_result$cluster[indice_test]

    if ( !is.na(c_train) && !is.na(c_test) && c_train == c_test){
      count = count +1
    }
  }
  count/nrow(df_farmacos_test)
}

```

Tal y como se puede ver a continuación, con el cluster de K-means anterior, obtenemos un 55 % de acierto acerca de si calificamos los fármacos como “positivos” o como “negativos”, los fármacos de test ya agrupados en train (en función de todo lo que se ha comentado anteriormente).

```

## Vamos a comprobar si las predicciones se realizan correctamente.

# Primero buscamos los fármacos que estén en tanto en train como en test, porque
# son con los que podemos ver si funciona o no el clustering. Vamos a ver si coinciden.
x = intersect(datos_test$urlDrugName, datos_train$urlDrugName)

# Lanzamos la función anterior
res = predicciones(res_clustering, test_result)
print(res)

## [1] 0.5527157

```

Acierto del 0.5527157.

NOTA: esta técnica es no supervisada, puesto que realmente no está orientada a predecir un valor concreto, sino a extraer dependencias de los datos. Con este último experimento, simplemente hemos querido ver, si éramos capaces de utilizar dicha información y darle alguna interpretación con sentido.

6.2.4. K-medoides Clustering

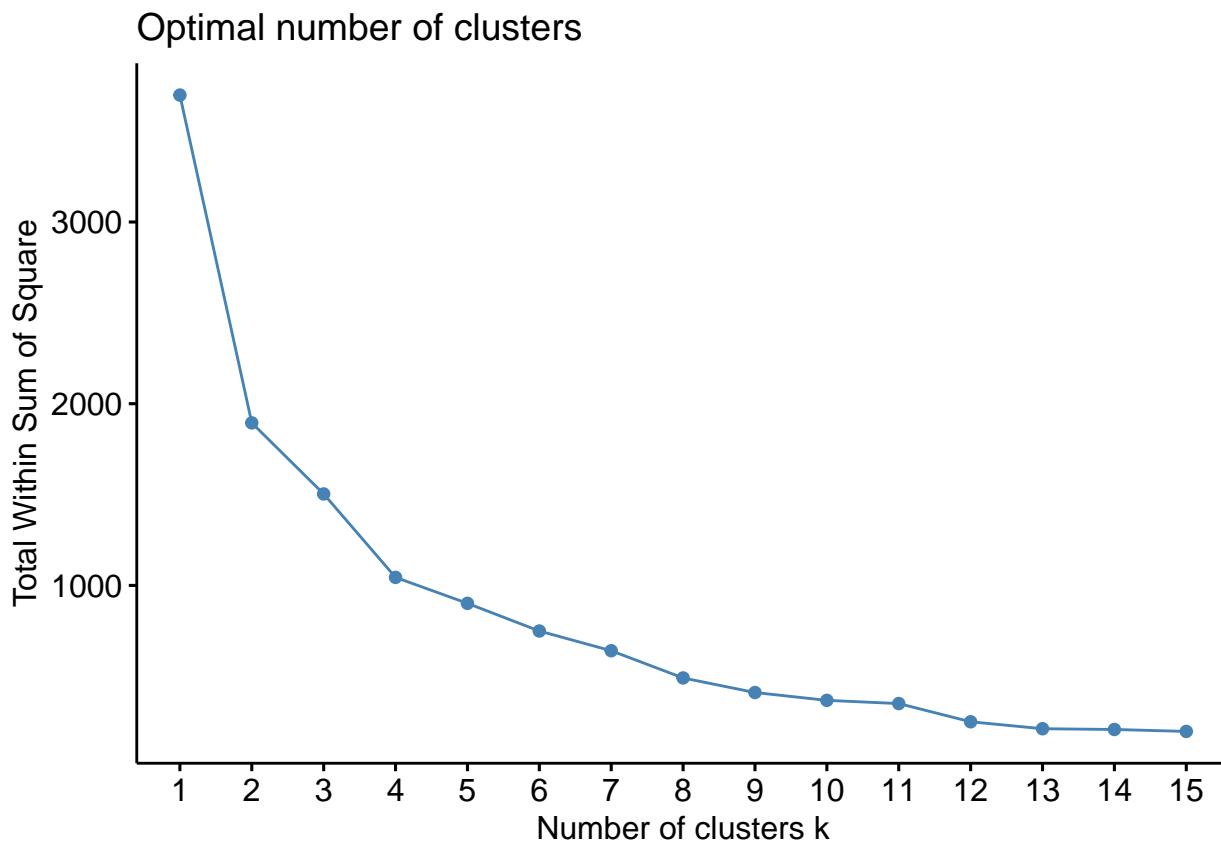
A continuación se va a aplicar una nueva técnica en el que agrupamos las observaciones en K clusters, siendo este valor K preestablecido. En este caso, cada cluster está representado por una observación presente en el cluster, en lugar de un centroide (como hacímos en *K-Means*). Dicho en otras palabras, en este caso, el “centroide” que utilizamos se corresponde con el ítem del cluster cuya distancia a todos los demás es mínima, en lugar de utilizar el promedio del cluster (que no tiene que corresponderse con un ítem del mismo). Para su resolución usaremos el algoritmo PAM (Partitioning Around Medoids). Éste minimiza la suma de las diferencias de cada observación respecto al medioide.

Este método es más robusto que el anterior, entre otras cosas, porque está menos afectado por ruido. De hecho, se suele utilizar cuando se tiene la intuición de que puede haber outliers (hecho que, como hemos mostrado en la técnica anterior, es muy probable que esté ocurriendo en este problema).

En primer lugar, y de igual forma que hemos hecho en el apartado anterior, tenemos que decidir el número de clusters con el que queremos trabajar. Para ello, podemos ver, en la siguiente gráfica, una correspondencia del estudio de la varianza con el número de cluster utilizado. En este caso, utilizaremos la distancia Manhattan, puesto que, como hemos visto en multitud de bibliografía, es la que se suele utilizar con PAM (sobre todo cuando se espera la existencia de puntos *outliers*).

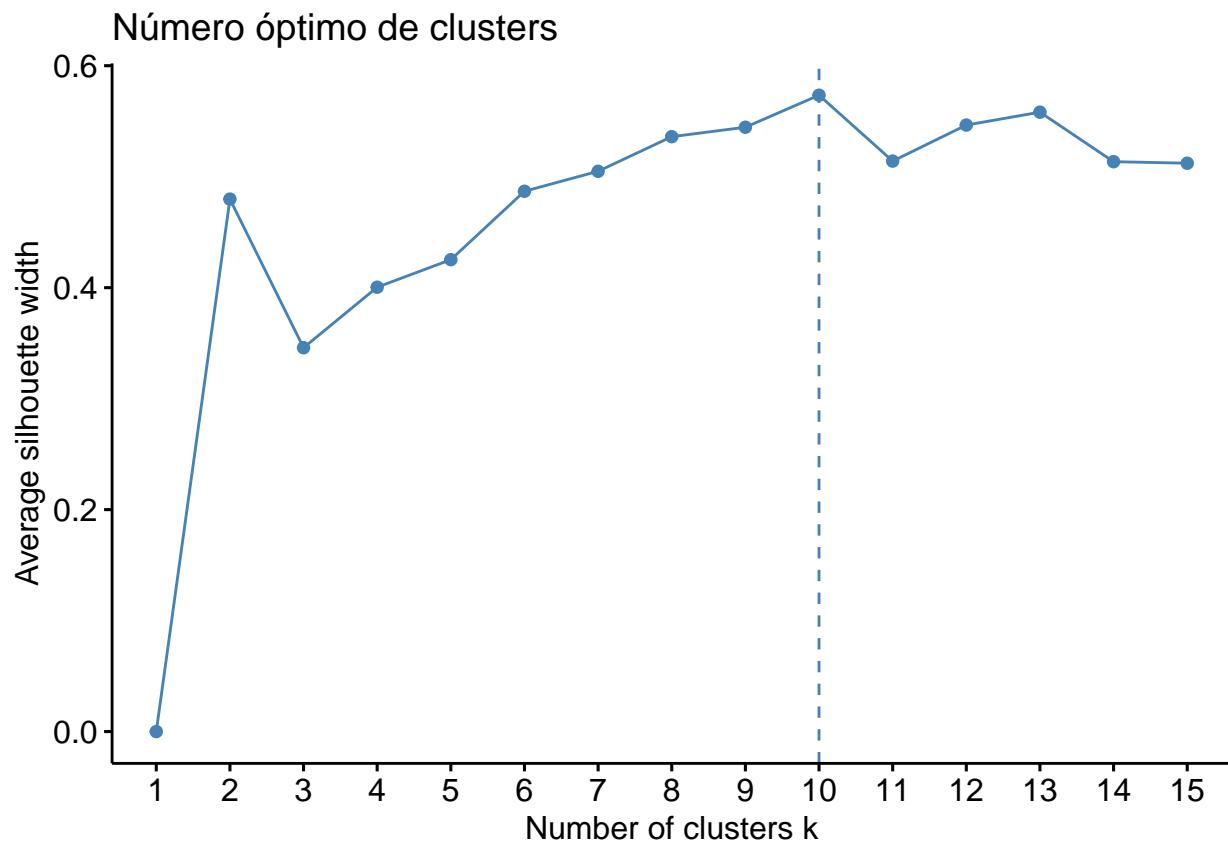
En la siguiente gráfica podemos ver la evaluación para hasta un máximo de 15 clusters. Como hemos visto en clase, llegará un punto desde el cuál no compense añadir más clusters (llegará un punto a partir del cuál la mejora es mínima, mientras que estamos añadiendo coste computacional a una técnica que ya es costosa de por sí. En otras palabras, a partir de un punto, no merece la pena añadir más esfuerzo computacional). En nuestro caso, y teniendo en cuenta la gráfica, hemos estimado que 6 es un número adecuado de agrupaciones.

```
# Identificamos el número óptimo de clusters
fviz_nbclust(x = datos, FUNcluster = pam, method = "wss", k.max = 15,
              diss = dist(datos, method = "manhattan"))
```



Podemos volver a utilizar validación, tal y como hicimos en el apartado anterior, y recurrir de nuevo al índice silueta, para ver si con este método sigue siendo el tamaño 6 sigue siendo aceptable. En otras palabras, buscamos reafirmarnos con este número. Como se puede observar en la gráfica, el número óptimo de clusters es 10, pero siguiendo todas las indicaciones anteriores, 6 sigue siendo un valor decente.

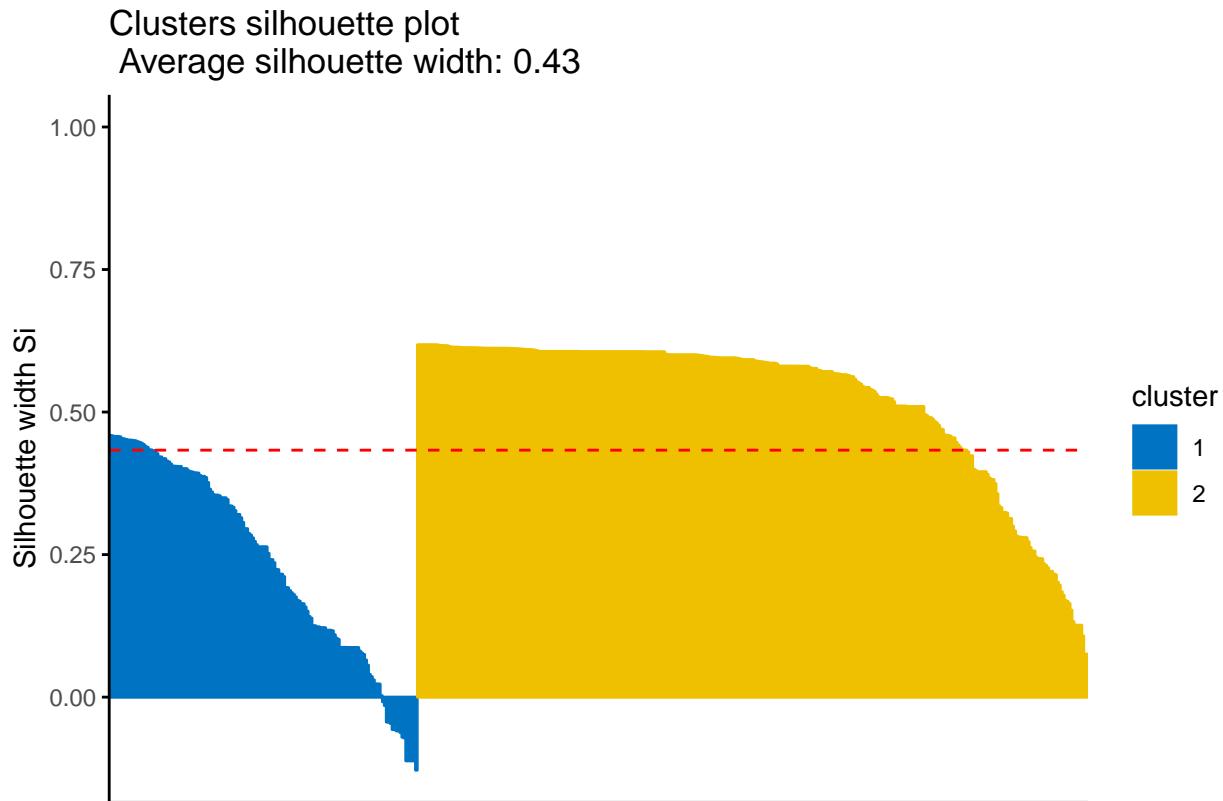
```
datos <- scale(data_train_procesado)
fviz_nbclust(x = datos, FUNcluster = pam, method = "silhouette", k.max = 15,
              diss = dist(datos, method = "manhattan") ) + labs(title = "Número óptimo de clusters")
```



De hecho, si obtenemos gráficamente una representación de los índices silueta para un número de clusters igual a 6, podemos ver que en general, parece funcionar bastante bien, quitando los pequeños valores negativos que tenemos.

```
fviz_silhouette(sil.obj = km_clusters, print.summary = TRUE, palette = "jco",
                 ggtheme = theme_classic())
```

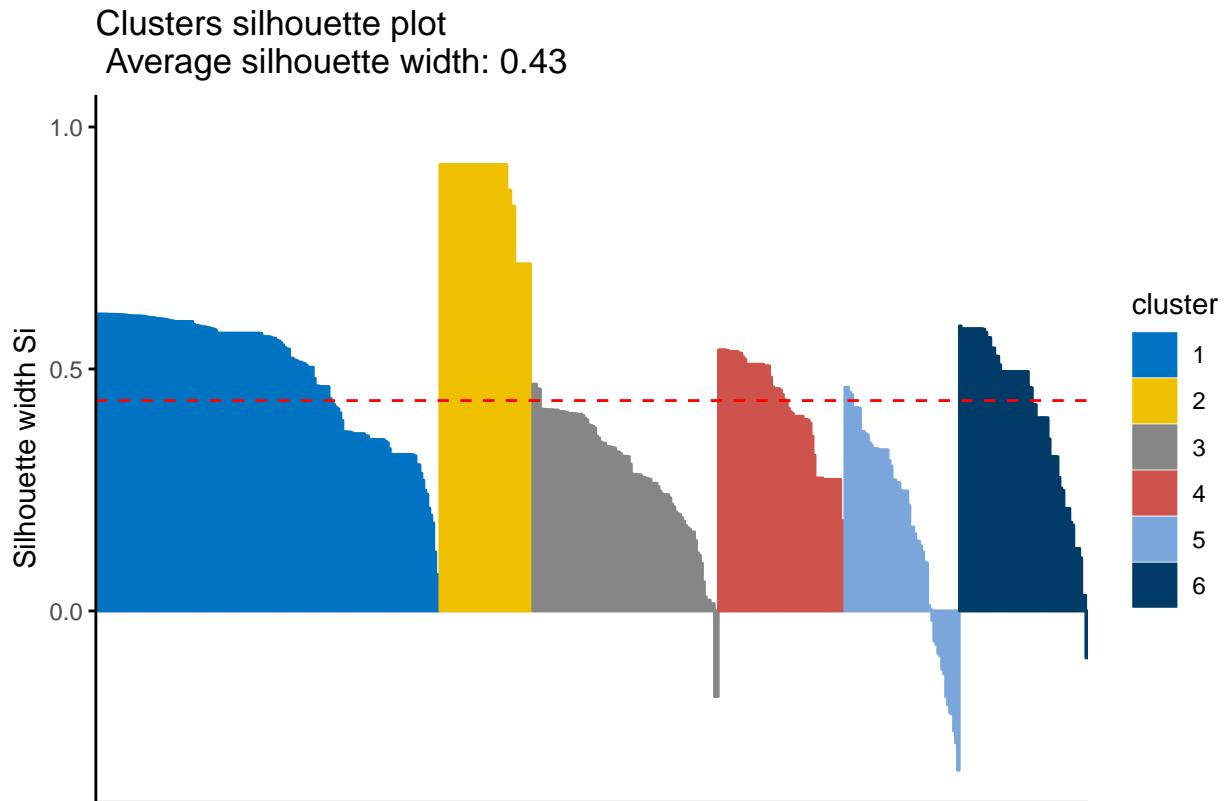
```
##   cluster size ave.sil.width
## 1       1  158        0.24
## 2       2  344        0.52
```



Siembargo, segn hemos podido comprobar, no por aumentar el nmero de clusters hemos conseguido ponerle solucin a este hecho, podemos verlo a continuacin, donde se muestra grficamente el resultado para 10 clusters (supuestamente el mejor valor). Por tanto, como vemos que, en la prctica no hay una mejora sustancial, y que los resultados se asemejan bastante (la media global del ndice silueta no varia), **hemos decidido aplicar esta tcnica con 6 clusteres, donde se obtiene en general, una agrupaci n m s que aceptable**. De hecho, este nmero de cluster nos parece tambin adecuado, puesto que, al final, son muchos factores los que influyen en una opinin, y muchas veces para una persona, un rating 5 exige m s efectividad (effectivenessNumber) que para otra. De nuevo, como podemos ver, la subjetividad intrnseca en nuestro dataset dificulta la interpretaci n y coherencia de los resultados.

```
fviz_silhouette(sil.obj = km_clusters, print.summary = TRUE, palette = "jco",
                 ggtheme = theme_classic())
```

	cluster	size	ave.sil.width
## 1	1	174	0.49
## 2	2	47	0.87
## 3	3	94	0.29
## 4	4	64	0.42
## 5	5	58	0.18
## 6	6	65	0.41

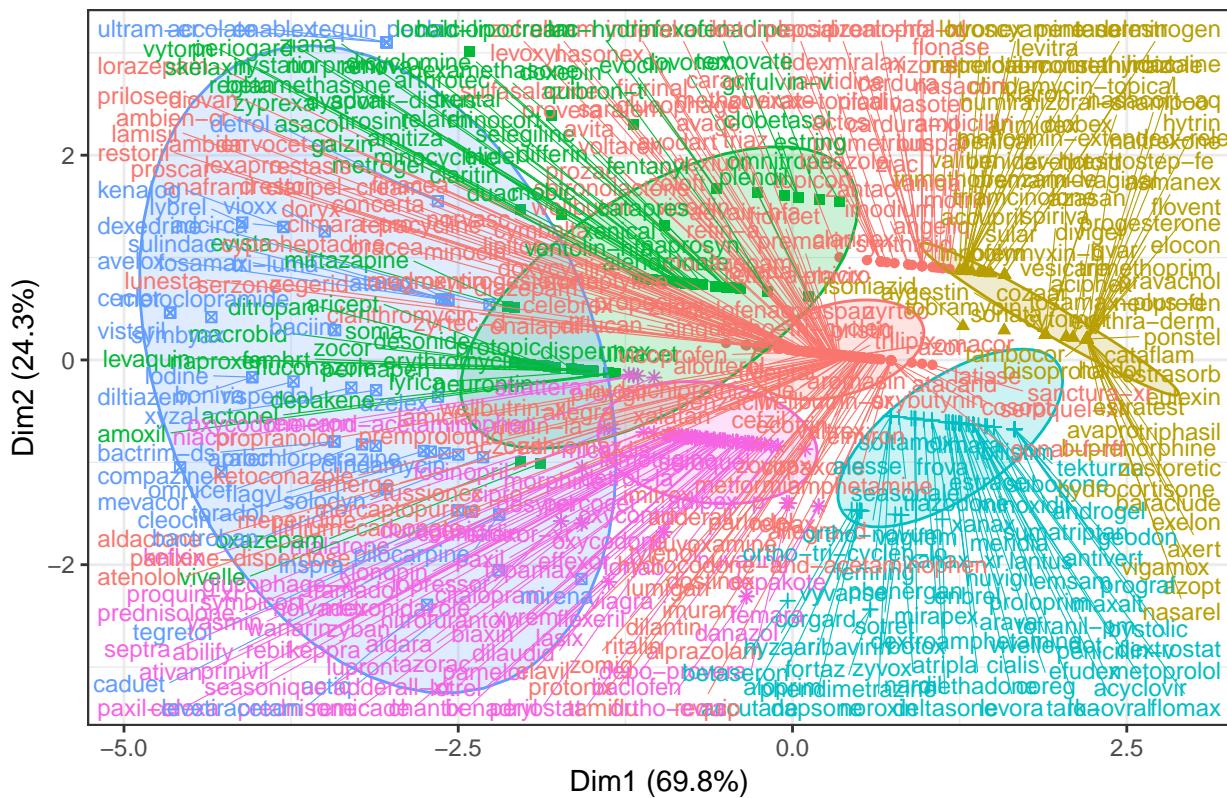


```
# Pintamos gráfica en busca del cluster óptimo
km_clusters <- eclust(x = datos, FUNcluster = "pam", k = 10, seed = 123,
                      hc_metric = "wss", graph = FALSE)
```

A continuación, podemos ver el agrupamiento resultante con los parámetros que se han justificado a lo largo de este apartado. De nuevo, vemos una gráfica en la cuál, de nuevo tenemos tanta información, que puede resultar difícil interpretarla.

```
pam_clusters <- pam(x = datos, k = 6, metric = "manhattan")
fviz_cluster(object = pam_clusters, data = datos, ellipse.type = "t", repel = TRUE,
             labelsize = 9) +
  theme_bw() +
  labs(title = "Resultados clustering PAM (k=6)") +
  theme(legend.position = "none")
```

Resultados clustering PAM (k=6)



Uno de los aspectos más relevantes, es, que tal y como hemos comentado antes, no hay centroides, sino que items del dataset son los que se utilizan como centros en cada uno de los clusters resultantes. Como podemos observar, no hay centroides. De hecho, podemos contrastar este hecho en la siguiente gráfica, la cuál es exactamente la misma que la anterior con la diferencia de que se encuentra destacado en rojo el item que hace de “centroide del cluster”.

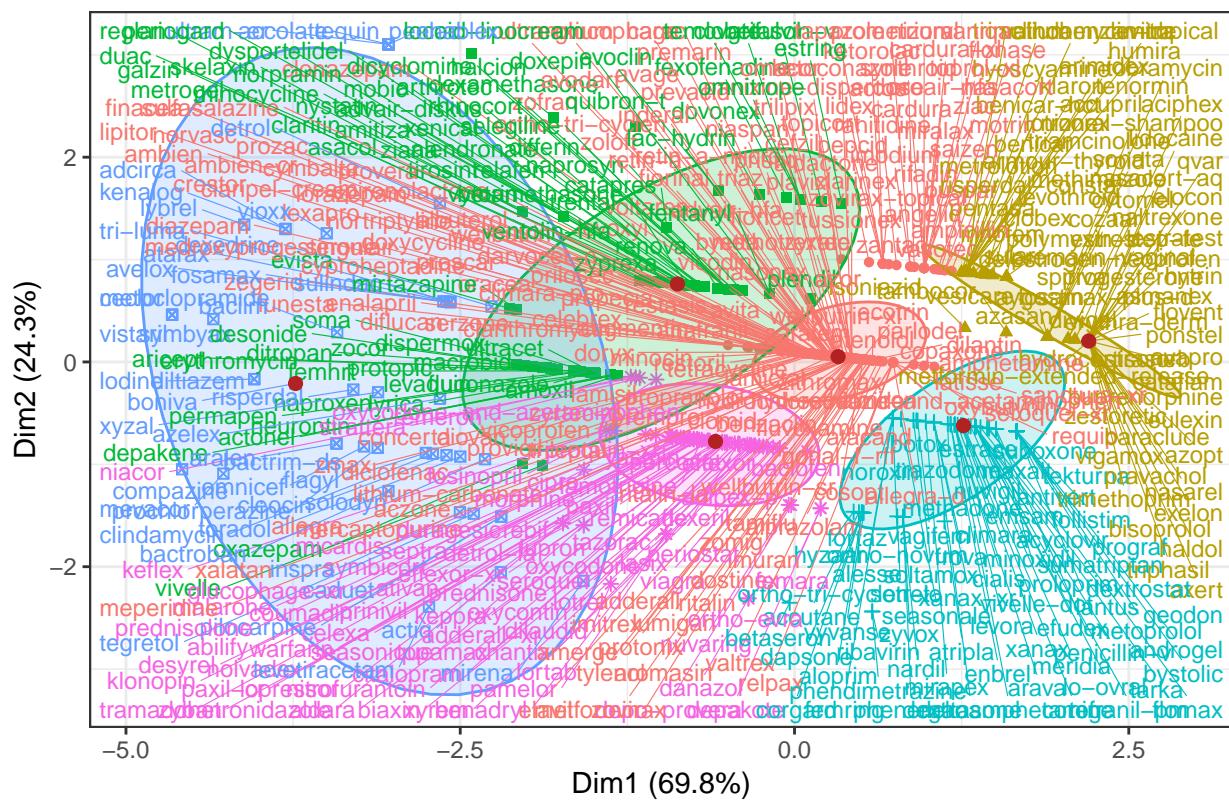
```
# Calculamos el PCA y extraemos las proyecciones almacenadas en el elemento x.
medoids <- prcomp(datos)$x

# Se seleccionan únicamente las proyecciones de las observaciones que son medoids
medoids <- medoids[rownames(pam_clusters$medoids), c("PC1", "PC2")]
medoids <- as.data.frame(medoids)

# Se emplean los mismos nombres que en el objeto ggplot
colnames(medoids) <- c("x", "y")

# Creación del gráfico
fviz_cluster(object = pam_clusters, data = datos, ellipse.type = "t",
             repel = TRUE, labelsize = 9) +
  theme_bw() +
  # Se resaltan las observaciones que actúan como medoids
  geom_point(data = medoids, color = "firebrick", size = 2) +
  labs(title = "Resultados clustering PAM") +
  theme(legend.position = "none")
```

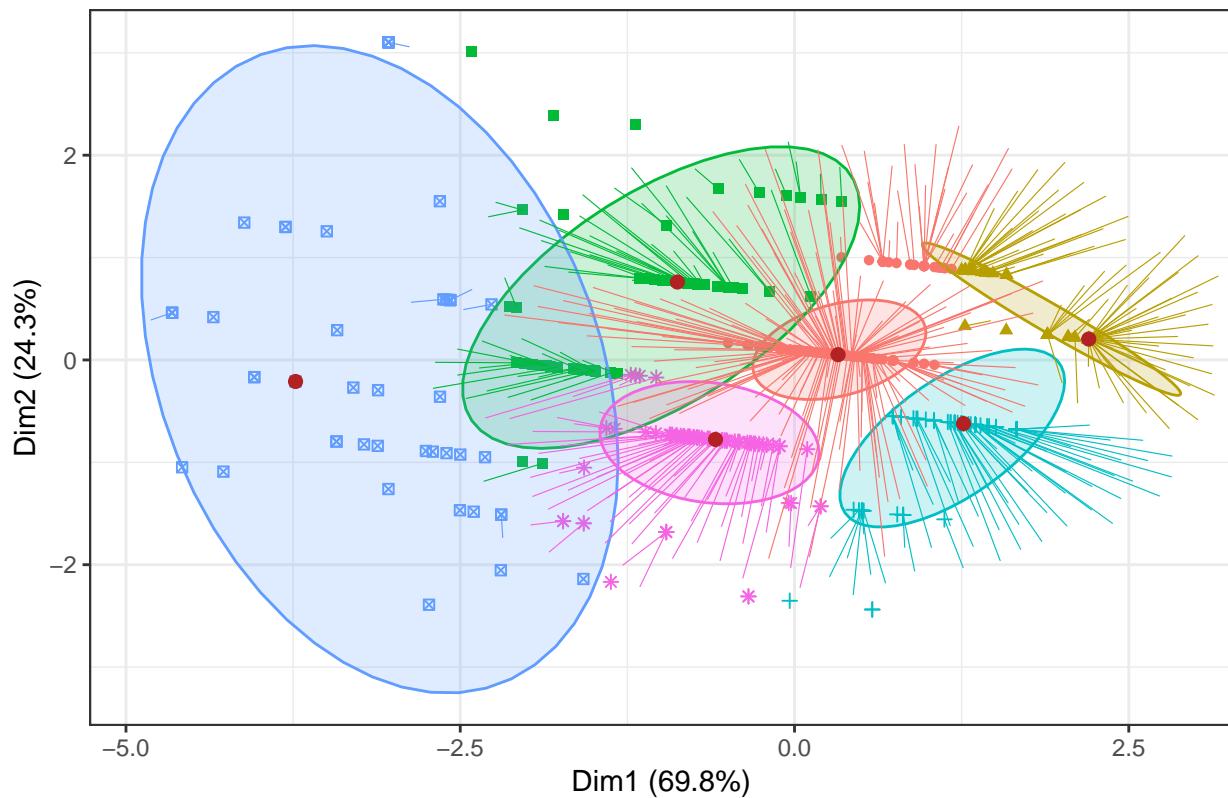
Resultados clustering PAM



Y por último, para ver cada cluster completamente definido, le hemos quitado las etiquetas para su mejor visualización (el agrupamiento no varía).

```
# Creación del gráfico
fviz_cluster(object = pam_clusters, data = datos, ellipse.type = "t",
             repel = TRUE, labelsize = 0) +
  theme_bw() +
  # Se resaltan las observaciones que actúan como medoids
  geom_point(data = medoids, color = "firebrick", size = 2) +
  labs(title = "Resultados clustering PAM") +
  theme(legend.position = "none")
```

Resultados clustering PAM



Realmente, respecto a la técnica anterior, es difícil realizar una comparación, ya que ambas han obtenido la misma media total del índice de silueta, y parecen tener una buena distribución, cohesión y similitud en los agrupamientos. Pero sin embargo, con esta técnica, hemos podido ver, que aunque la media global del índice silueta no varía realmente, sí que **se obtienen mejores resultados a nivel de cluster**, lo que nos hace pensar que esta técnica es adecuada en nuestro problema.

6.2.5. Clustering Difuso (FUZZY CLUSTERING)

En todos los métodos anteriores, se parte de la hipótesis de que el agrupamiento es **exclusivo**. Esto es una buena opción cuando los grupos están muy separados, pero, como hemos podido ver gráficamente en los agrupamientos anteriores, no es nuestro caso, ya que los clusters se solapan, o incluso se podría decir que tienen puntos comunes. Por esta razón, hemos considerado que es una técnica que debe ser contemplada en nuestro trabajo, puesto que se adapta bastante bien a las condiciones de nuestro problema.

En primer lugar, vamos a definir un número de clusters que consideremos adecuado, tal y como hemos venido haciendo para las técnicas anteriores. Para ello, recurrimos a una gráfica donde veamos la relación entre el índice de silueta y el número de clusteres asociado. En este caso, hay una clarísima diferencia entre elegir dos clusteres o más de ellos (ya que el siguiente número de cluster tiene casi la mitad del índice medio de silueta conseguido por el primero mencionado). Por ello, para esta técnica, consideraremos que el valor óptimo de cluster es 2. De nuevo, el índice medio de silueta sigue entorno a 0.4, por lo que, respecto a este aspecto, se encuentra en sintonía con las técnicas de agrupamiento anteriores.

```
# datos ya lo tenemos previamente escalado
datos <- scale(data_train_procesado)
fviz_nbclust(x = datos, FUNcluster = fanny, method = "silhouette", k.max = 15,
             diss = dist(datos, method = "euclidean") ) + labs(title = "Número óptimo de clusters")

## Warning in FUNcluster(x, i, ...): FANNY algorithm has not converged in
```

```

## 'maxit' = 500 iterations

## Warning in FUNcluster(x, i, ...): FANNY algorithm has not converged in
## 'maxit' = 500 iterations

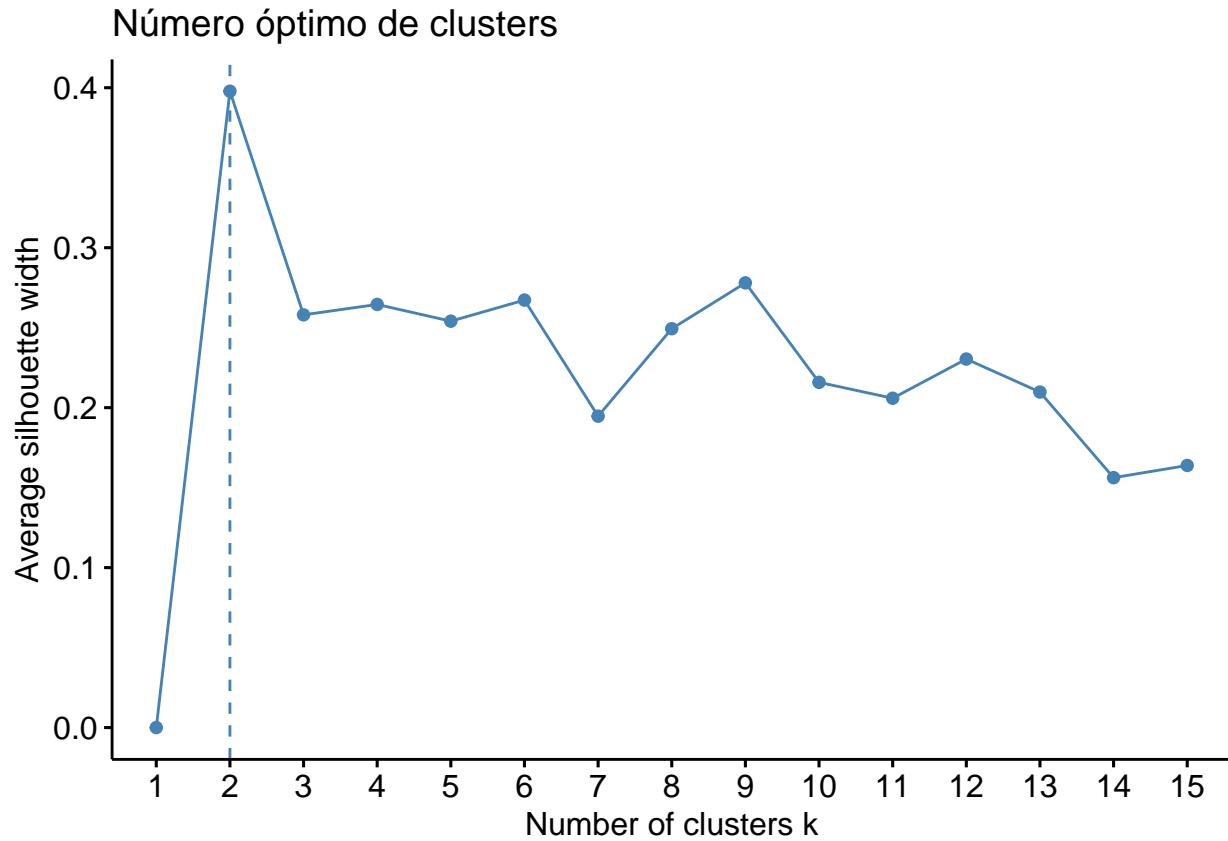
## Warning in FUNcluster(x, i, ...): FANNY algorithm has not converged in
## 'maxit' = 500 iterations

## Warning in FUNcluster(x, i, ...): FANNY algorithm has not converged in
## 'maxit' = 500 iterations

## Warning in FUNcluster(x, i, ...): FANNY algorithm has not converged in
## 'maxit' = 500 iterations

## Warning in FUNcluster(x, i, ...): FANNY algorithm has not converged in
## 'maxit' = 500 iterations

```



```

#
# fuzzy_cluster <- fanny(x = datos, k = 6, metric = "euclidean", stand = FALSE, maxit = 5000)
# # head(fuzzy_cluster$membership)
# # fuzzy_cluster$coeff
# # head(fuzzy_cluster$clustering)
# fviz_cluster(object = fuzzy_cluster, repel = TRUE, ellipse.type = "norm", palette = "jco", labelszie ...

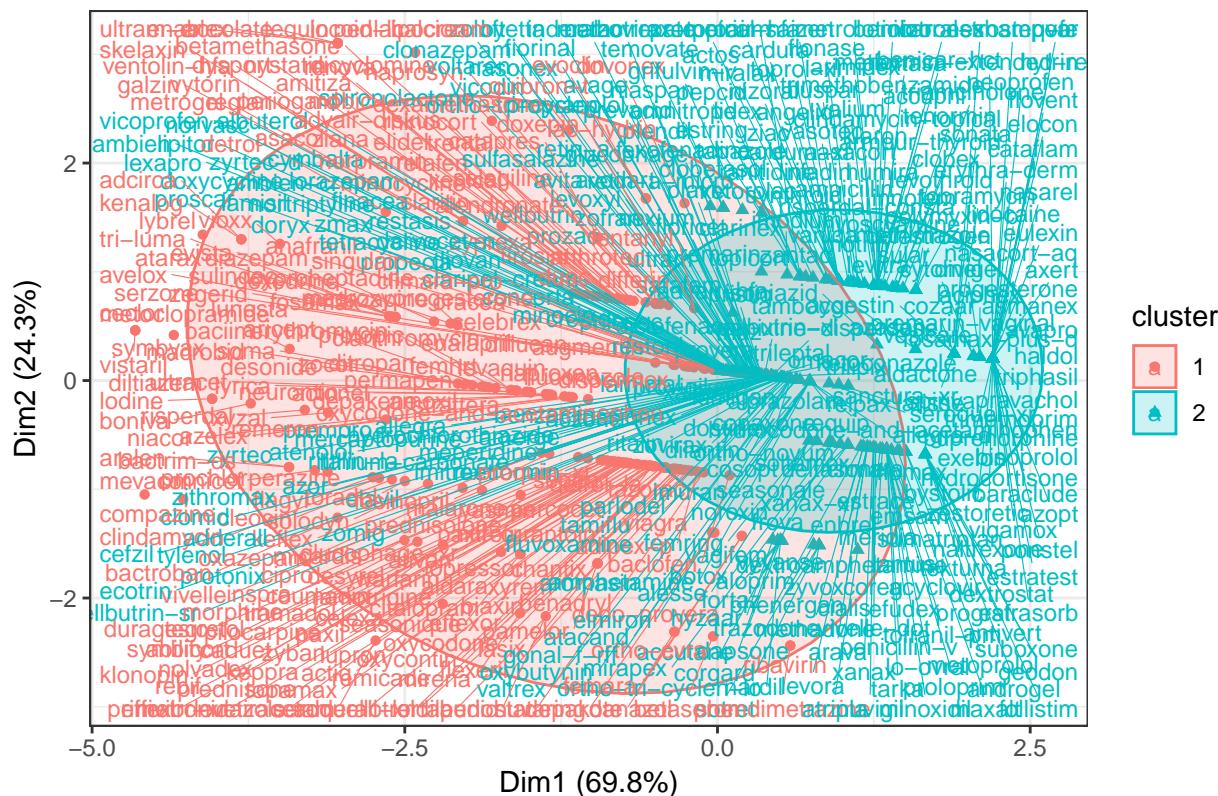
```

A continuación, podemos ver los resultados obtenidos para los parámetros establecidos.

Como dato, podemos ver que en cuanto a resultados, es muy similar al resultado obtenido con Kmeans para el mismo número de clusters, con la pequeña diferencia, de que en este caso, la división del espacio se muestra mucho más clara. Estos resultados, nos siguen pareciendo coherentes, ya que al final, las opiniones derivarán hacia buenas opiniones o malas opiniones de cada fármaco, lo cuál es una división totalmente lógica, tanto desde el punto de vista humano, como de la propia relación que existe entre los atributos que estamos utilizando. También influirán otros valores, no dependientes del fármaco, como puede ser la puntuación que considere cada paciente (que variará en función de cada uno). En este punto del documento, podemos afirmar, que **la subjetividad existente en nuestro dataset está siendo determinante a la hora de definir el agrupamiento.**

```
fuzzy_cluster <- fanny(x = datos, k = 2, metric = "euclidean", stand = FALSE, maxit = 5000)
# head(fuzzy_cluster$membership)
# fuzzy_cluster$coeff
# head(fuzzy_cluster$clustering)
fviz_cluster(object = fuzzy_cluster, repel = TRUE, ellipse.type = "norm",
             pallete = "jco", labelsize = 9) + theme_bw() + labs(title = "Fuzzy Cluster plot")
```

Fuzzy Cluster plot



6.2.6. Clustering Jerárquico

El *clustering jerárquico* es una alternativa a los métodos de clustering particional, que no requiere que se pre-especifique el número de clusters al algoritmo. Tal y como hemos visto en la asignatura, se pueden seguir dos estrategias, aglomerativa o divisiva, en este caso vamos a usar la primera, la cuál es la más común.

Como ya sabemos, en el método aglomerativo el agrupamiento se inicia en la base del árbol, siendo cada observación un cluster individual. Entonces, los cluster se van combinando conforme la estructura crece hasta unirse en única rama central. De forma análoga a como hemos estado llevando a cabo la aplicación de las demás técnicas, en primer lugar, vamos a determinar el número de clusters óptimo. Como podemos ver en

la gráfica, el valor óptimo es 12, a partir del cuál comienza a descender. Sin embargo, la diferencia con un tamaño 10 de clusters es mínima, y computacionalmente nos mejora bastante, por lo que, desde un punto de vista práctico, hemos considerado que es el valor más adecuado en nuestra situación.

De igual forma, como hemos visto en la asignatura, cuando estamos utilizando la distancia de mínimos cuadrados tenemos que tener especial cuidado, debido a que se minimiza la distancia entre los elementos del cluster, por lo que es un valor también a tener en cuenta en este tipo de técnicas.

6.2.6.1. Aplicación de la técnica

En primer lugar, vamos comenzar por ejecutar el algoritmo, y obtener así el dendrograma que se corresponde con nuestro problema. Para ello, simplemente realizamos una llamada a la función `hclust`, especificando el valor de distancia (hemos establecido la distancia euclídea).

```
# Cortar el árbol para generar los clusters
hc_euclidea_completo <- hclust(d = dist(x = datos, method = "euclidean"), method = "complete")
```

6.2.6.2. Elección del número óptimo de clusters

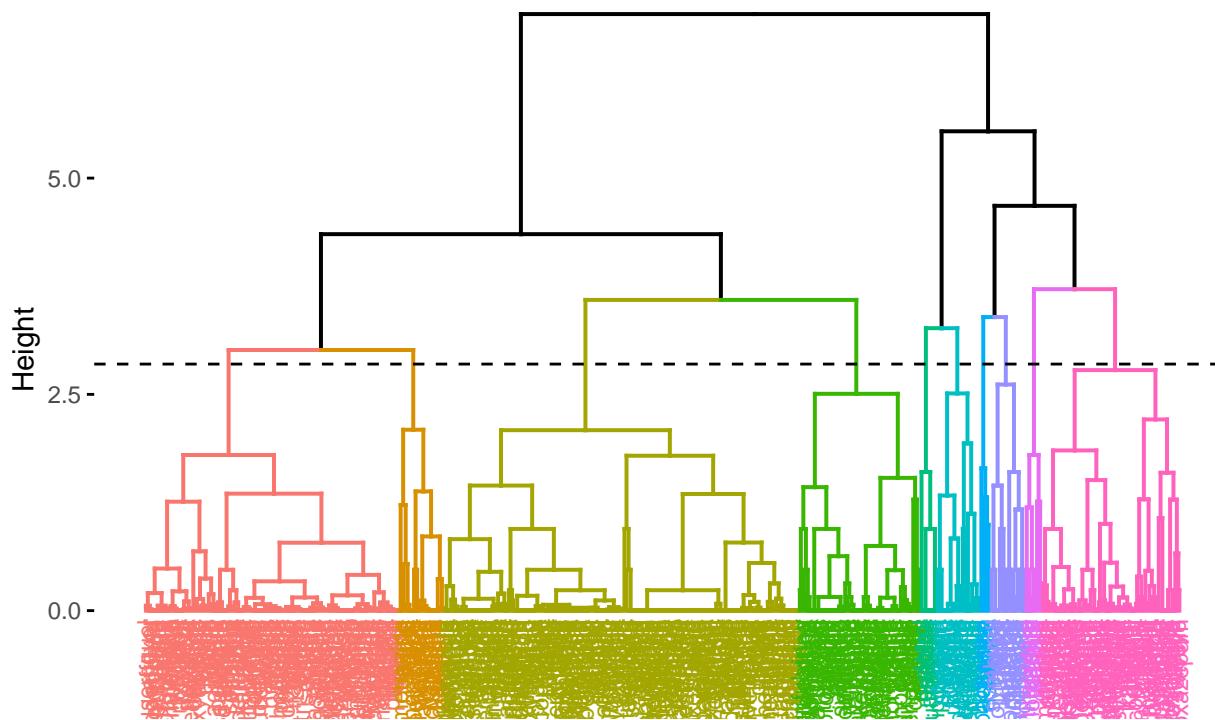
Una de las ventajas que nos da este algoritmo, es que nos permite visualizar, a través del dendrograma, una posible división en clusters, de forma que podemos, “a priori”, intentar dar con un valor de cluster adecuado. De esta forma, podemos hacer uso de las facilidades que nos da visualizar el dendrograma para ver qué cifra es la más adecuada para K.

Para ello, tal y como podemos consultar en la bibliografía, a pesar de ser una estrategia puramente intuitiva, un buen método sería localizar las ramas que se dividen (o unen, según miremos el dendrograma de abajo a arriba o viceversa) más o menos al mismo nivel. En nuestro caso, siguiendo este criterio, hemos decidido que el número más adecuado es 10, puesto que, tal y como se puede ver en la línea mostrada en la gráfica, esta división se corresponde con el corte que deja la unión de las ramas más o menos al mismo nivel.

```
fviz_dend(x = hc_euclidea_completo, k = 10, cex = 0.6) +
  geom_hline(yintercept = 2.85, linetype = "dashed") +
  labs(title = "Hierarchical clustering",
       subtitle = "Distancia euclídea, Lincage complete, K=10")
```

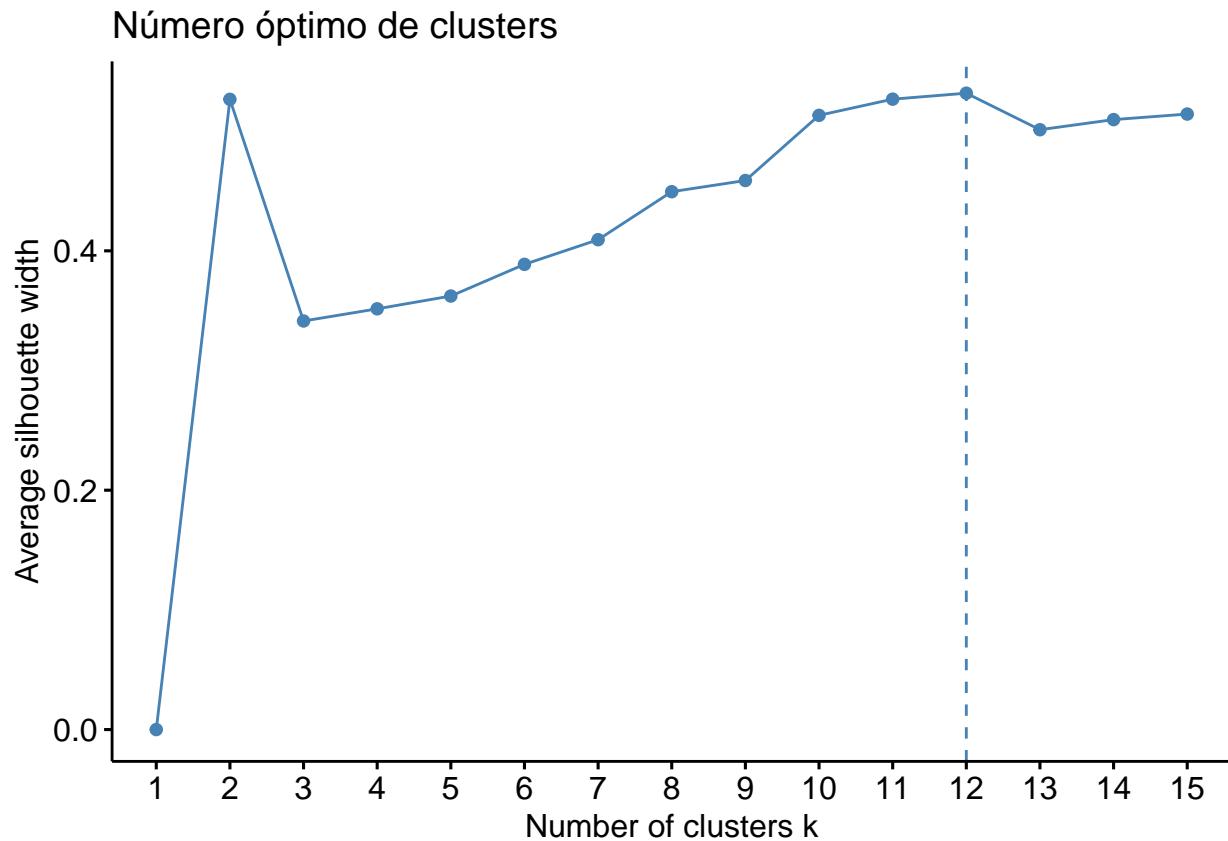
Herarchical clustering

Distancia euclídea, Lincage complete, K=10



Para ver si estamos en lo cierto, podemos obtener, para este tipo de agrupamiento, la gráfica que refleja la relación entre el número de clusteres a utilizar y la media del índice silueta asociada. A continuación, se muestran los resultados para un total de agrupaciones de hasta tamaño 15. Como podemos ver, los resultados obtenidos para un valor de K igual a 10 son muy positivos.

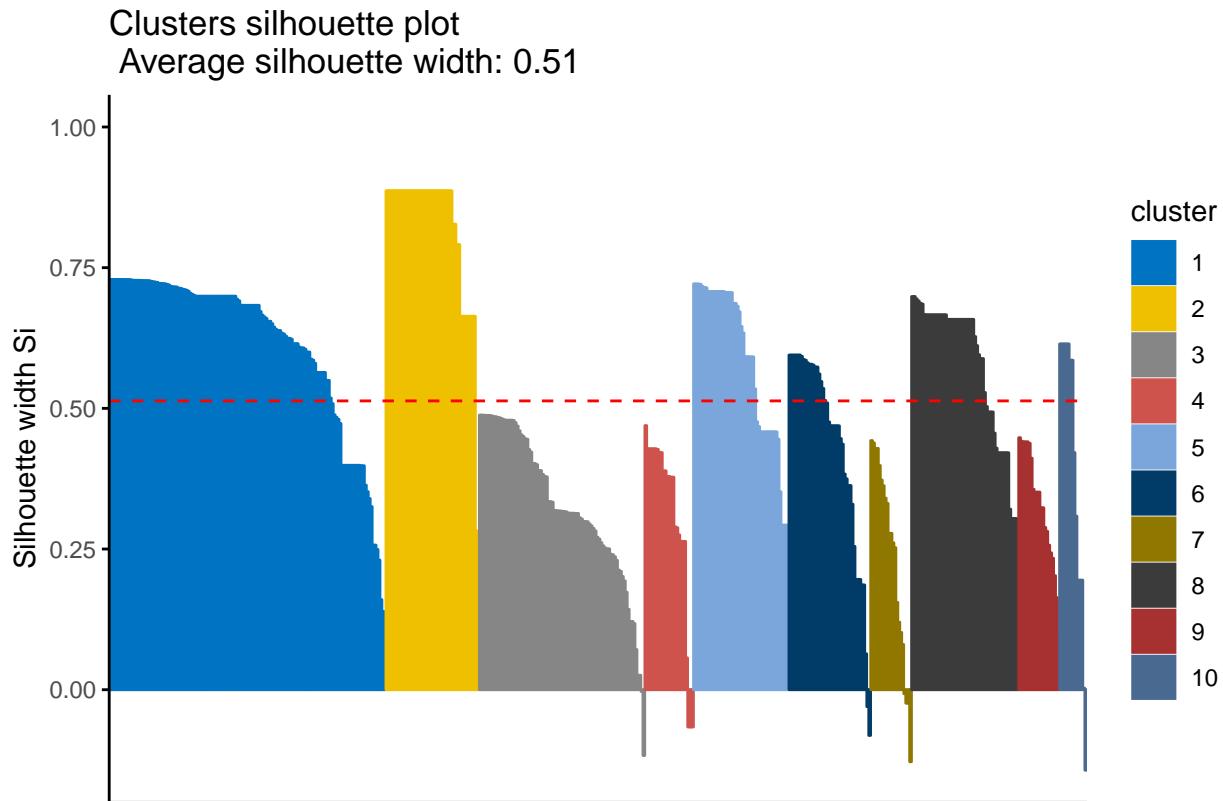
```
# datos ya lo tenemos previamente escalado
datos <- scale(data_train_procesado)
fviz_nbclust(x = datos, FUNcluster = hcut, method = "silhouette", k.max = 15,
             diss = dist(datos, method = "euclidean") ) +labs(title = "Número óptimo de clusters")
```



Ahora vamos a mostrar gráficamente, la media del índice de silueta para el valor de cluster elegido. Como podemos ver, la mayor parte de los grupos supera el valor de 0.50, (lo cual es muy muy positivo, y nos da muy buenas indicaciones de los agrupamientos realizados). Además, de igual forma, vemos cómo muchos de ellos están muy definidos y sin ningún valor negativo (y no como en casos anteriores), lo cuál nos lleva a pensar, que **esta técnica es muy apropiada para nuestro problema**, ya que hace una división de los ítems que aparentemente funciona muy bien, y con la que estamos obteniendo muy buenos resultados.

```
km_clusters <- eclust(x = datos, FUNcluster = "hclust", k = 10, seed = 123,
                      hc_metric = "euclidean", graph = FALSE)
fviz_silhouette(sil.obj = km_clusters, print.summary = TRUE, palette = "jco",
                ggtheme = theme_classic())
```

##	cluster	size	ave.sil.width
## 1	1	142	0.61
## 2	2	48	0.81
## 3	3	85	0.33
## 4	4	25	0.31
## 5	5	49	0.59
## 6	6	42	0.43
## 7	7	21	0.23
## 8	8	55	0.58
## 9	9	21	0.33
## 10	10	14	0.39



Una vez que tenemos el dendrograma, debemos buscar una forma de consultar hasta qué punto la estructura refleja las distancias originales entre observaciones, y por tanto, cómo se comporta el modelo en cuanto a la hora de agrupar los distintos datos del dataset. Para ello usamos el coeficiente de correlación entre las distancias cophenetic de la altura de los nodos y la matriz de distancias original. Tal y como se puede consultar en la documentación, un valor superior al 75 %, se considera buenp. En nuestro caso, nos acercamos con uno de los dos valores, pero con el segundo de los consultados lo superamos, lo que nos lleva a pensar que nuestro método funciona de forma aceptable.

```
# Cuanto más cercano al 1 mejor. Valores superiores al 0.75 se consideran buenos.
# Matriz de distancias euclídeas
mat_dist <- dist(x = datos, method = "euclidean")
# Dendrogramas con linkage complete y average
hc_euclidea_complete <- hclust(d = mat_dist, method = "complete")
hc_euclidea_average <- hclust(d = mat_dist, method = "average")

cor(x = mat_dist, cophenetic(hc_euclidea_complete))

## [1] 0.6432507
cor(x = mat_dist, cophenetic(hc_euclidea_average))

## [1] 0.7773464
```

Como conclusión, nos decantamos, con esta técnica, por un número de clusters igual a 10 en base a todo lo anterior comentado, que se puede resumir en los siguientes puntos:

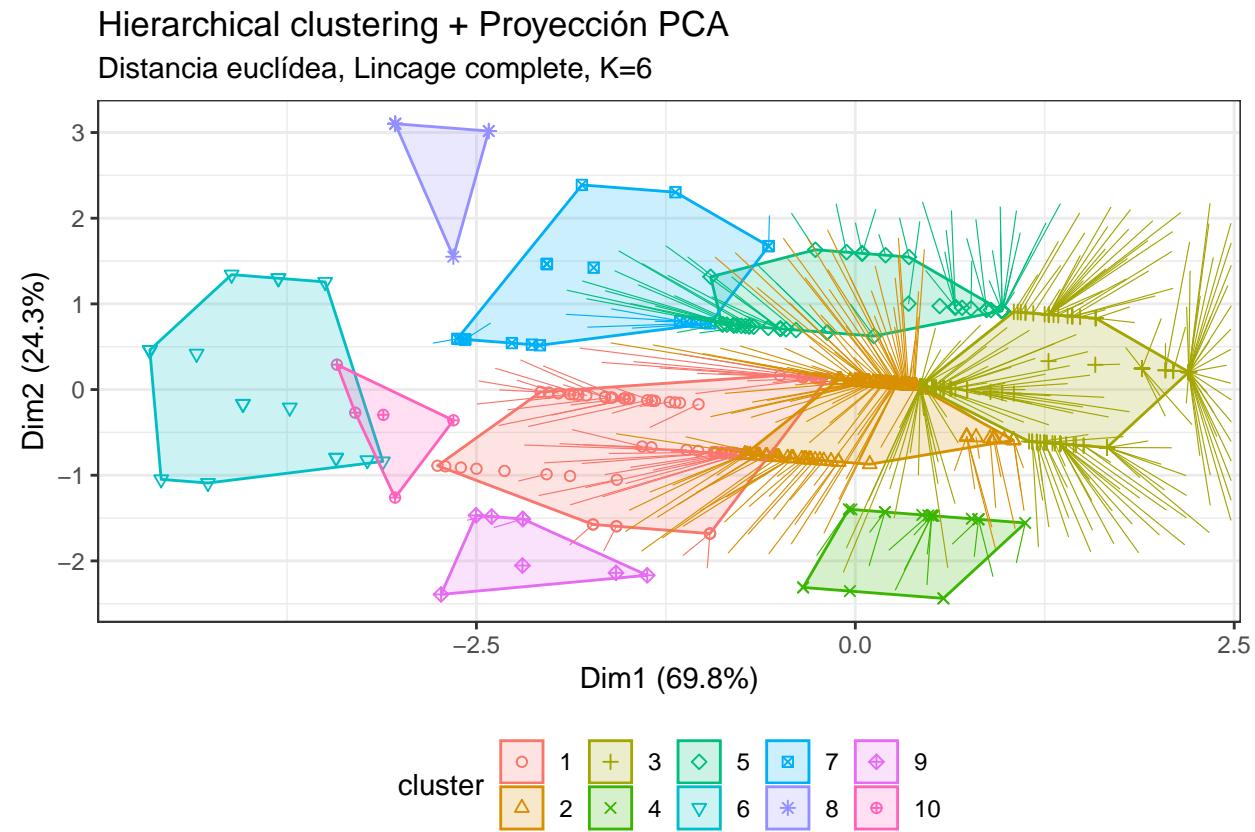
- Teniendo en cuenta el dendrograma, este valor nos permite obtener una división lo más igualada posible en cuanto a nivel del árbol resultante en el dendrograma.
- La media global del índice silueta obtenida, es alta (de hecho, es la más alta alcanzada hasta ahora).
- Realizando una comparación del índice medio de silueta que se puede alcanzar con distintos valores de k, obtenemos muy buenos resultados.

- Los índices silueta obtenidos para el K elegido (que como hemos dicho, es 10), dan lugar a un resultado muy optimista, puesto que se puede observar una división bastante nítida.

Por último, podemos ver una forma (poco frecuente) de representar los clusters, que consiste en combinar los anterior con una reducción de dimensionalidad por PCA. Con ello, podemos conseguir visualizarlo de la misma forma que hemos llevado esta parte a cabo en el resto de técnicas. Primero, se calculan los componentes principales y después se representan empleando las dos primeras componentes. Finalmente, se colorean los clusters mediante elipses.

A continuación, podemos ver los resultados obtenidos (se han eliminado las etiquetas para visualizar mejor los distintos clusteres que hemos obtenido). Como hecho destacable, podemos puntualizar que el bajo acoplamiento que se parecía entre los distintos grupos que se han originado.

```
fviz_cluster(object = list(data=datos, cluster=cutree(hc_euclidea_completo, k=10)),
             ellipse.type = "convex", repel = TRUE, show.clust.cent = FALSE,
             labelsize = 0) +
  labs(title = "Hierarchical clustering + Proyección PCA",
       subtitle = "Distancia euclídea, Lincage complete, K=6") +
  theme_bw() +
  theme(legend.position = "bottom")
```



6.2.6.3. Otras representaciones

Sin embargo, a pesar de que nos hemos remitido a visualizar los resultados de esta técnica mediante dendrogramas y una versión parecida a los agrupamientos particionales, hay muchas más representaciones que se tienen que utilizar. Recordemos, que estamos tratando con técnicas descriptivas, y que al final, para obtener información de los datos y poder estudiarlos de una forma adecuada, es muy útil tener distintas formas de representación.

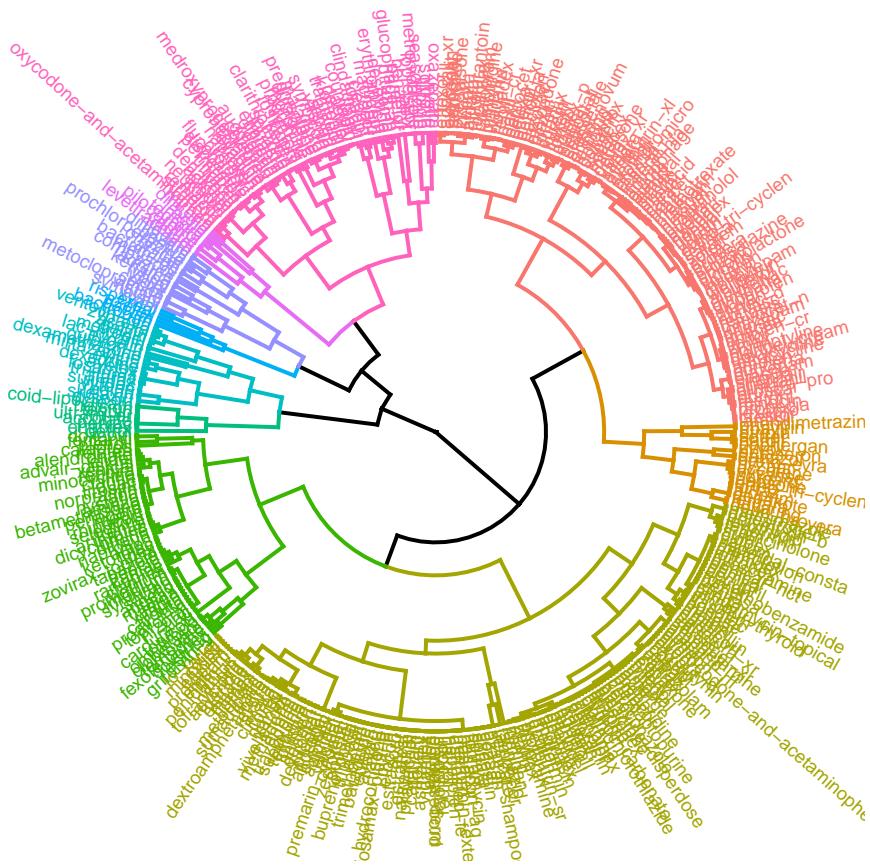
En esta sección, vamos a visualizar los resultados de otras dos formas, las cuales son especialmente utilizadas cuando tenemos demasiados datos, puesto que permiten focalizar la información y centrarnos en distintas áreas de manera específica. A continuación, mostramos una forma de representación cuya finalidad principal cumple con este aspecto. Esta forma de visualización de dendrogramas se conoce como **árbol filogenético**.

```
fviz_dend(x = hc_euclidea_completo,
          k = 10,
          color_labels_by_k = TRUE,
          cex = 0.8,
          type = "phylogenetic",
          repel = TRUE, labelsize=0)
```



También podemos usar representaciones que sigan estructuras circulares, como es el caso de la siguiente visualización, la cual se corresponde con un **dendrograma circular**.

```
set.seed(5665)
fviz_dend(x = hc_euclidea_completo,
          k = 10,
          color_labels_by_k = TRUE,
          cex = 0.5,
          type = "circular")
```



6.2.7. Agrupamiento jerárquico y K-means

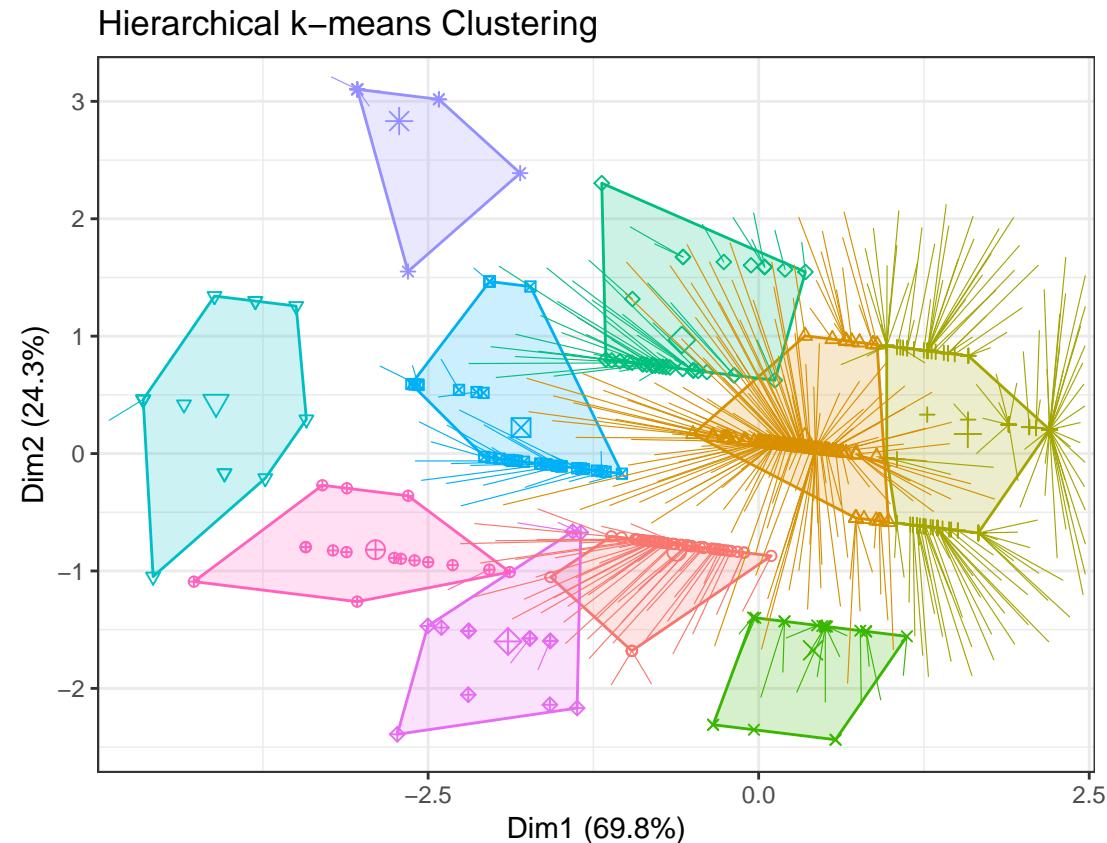
Como se puede ver en el apartado anterior (Clustering Jerárquico), hemos obtenido muy buenos resultados, utilizando para ello medidas de validación como puede ser el *coeficiente de silueta*. Pero podemos utilizar estos resultados para intentar obtener un agrupamiento aún mejor. La idea que se plantea en este apartado, es aplicar la técnica K-means, con la particularidad de que utilizaremos como centroides iniciales los centros calculados con la técnica anterior.

De esta forma, al fin y al cabo, lo que estamos haciendo es proporcionar unos valores iniciales de centroide que ya son buenos de por sí, por lo que con K-means únicamente nos dedicaremos a reajustar dichos valores con el fin de afianzar mejor en cuanto a resultados.

A continuación, podemos ver los resultados obtenidos con dicha técnica. Como aspecto destacable, podemos ver el bajo solapamiento existente en los clusters (a diferencia de otras técnicas ya comentadas anteriormente).

```
# jerárquico con la media
hkmeans_cluster <- hkmeans(x = datos, hc.metric = "euclidean",
                            hc.method = "complete", k = 10)

fviz_cluster(object = hkmeans_cluster, palette = "jco", repel = TRUE, labelsize = 0) +
  theme_bw() + labs(title = "Hierarchical k-means Clustering")
```



Además, podemos destacar otro aspecto, y es que, con esta última representación, hemos conseguido algo no logrado hasta este momento, y es que no quedan puntos representados fuera de los grupos, sino que los puntos quedan correctamente ubicados dentro de ellos, lo cuál, unido al bajo acoplamiento existente en los grupos, nos da un indicativo de buenos resultados.

6.3. Clustering sobre datos de texto

Vamos a ver cómo podemos aplicar técnicas de clustering a las distintas columnas de texto de nuestro dataset. Para ello vamos a explicar en mayor detalle el proceso seguido en la columna Benefits Review preprocesada y luego mostraremos los resultados con los otros textos de una forma más directa.

Esta es un técnica no supervisada, por lo que no sabemos las clases que clasifica realmente, solo agrupa las instancias en grupos sin definir sin tener nada de información previa acerca de la estructura.

6.3.1. Clustering de Benefits Review

Lo primero que debemos hacer es cargar el dataset preprocesado que hemos creado en esta práctica. Formamos el corpus correspondiente a la columna de beneficios que estamos analizando.

```
# Cargamos los datos
datos_train <- read.table("datos/datos_train_preprocesado.csv",
                           sep=",", comment.char="", quote = "\"", header=TRUE)

# Establecemos la semilla
set.seed(3)
corpus = tm::Corpus(tm::VectorSource(datos_train$benefits_preprocesad))
```

Como ya hemos hecho en alguna ocasión a lo largo de esta práctica, vamos a crear una matriz de términos en la que cada fila es un documento o comentario de beneficios y cada columna es un término o palabra que aparece en los textos.

```
tdm <- tm::DocumentTermMatrix(corpus)
tdm.tfidf <- tm::weightTfIdf(tdm)
tfidf.matrix <- as.matrix(tdm.tfidf)
```

Para poder trabajar con técnicas de agrupamiento necesitamos trabajar con valores numéricos continuos. Es por ello que hemos utilizado de nuevo TFIDF (esta vez hemos usado una función de más alto nivel, sin entrar en detalles). Necesitamos tener estos datos como matriz para poder continuar con los siguientes pasos.

```
tfidf.matrix[1:20,1:20]
```

	Terms																	
Docs	agent	alon	congest	dysfunct	failur	heart	hypertens	left	manag	mangag
1	0.6624946	0.4594991	0.4643742	0.6624946	1.126869	0.7208779	0.5634344	0.3946569	0.3946569	0.7249946
2	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
3	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
4	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
5	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
6	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
7	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
8	0.0000000	0.2827687	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
9	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
10	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
11	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
12	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
13	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
14	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
15	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
16	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
17	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
18	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
19	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
20	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000

Figura 26: Matriz de pesos para cada uno de los términos en cada uno de los documentos.

Esta es una muestra de la esquina superior izquierda de la matriz para ver el formato explicado anteriormente. Los valores son pesos normalizados de los términos en cada uno de los comentarios. Estos pesos se basan en la frecuencia en la que aparece cada palabra en cada documento.

En una de las técnicas que veremos más adelante, necesitamos las 100 palabras con mayor peso. La forma que se nos ocurrió de hacerlo consistía en hacer la sumatoria de los pesos por columnas, de esta forma teníamos el peso total de cada palabra para el total de documentos, y después ordenarlas en función del valor. El problema reside en que no podemos perder el número de filas que tenemos (la forma de la matriz), cosa que ocurre si hacemos la sumatoria por columnas. Entonces se nos ocurrió lo siguiente:

```
#Hacemos la sumatoria de columnas para averiguar el peso total de cada término.
v <- c()
for(i in 1:length(tfidf.matrix[,1])){
  v[i] <- sum(tfidf.matrix[,i])
}
# Averiguamos el orden de índices de los pesos de mayor
# a menor y nos quedamos con los 100 mas grandes.
indices <- order(v,decreasing = TRUE)[1:100]

# Nos creamos una submatriz de la original con los términos mas relevantes.
tfidf.matrix100 <- tfidf.matrix[,indices]
```

```
tfidf.matrix100 <- t(tfidf.matrix100)
```

En definitiva, en lugar de ordenar la matriz lo que obtenemos es una lista de los índices en la posición que deberían estar para que estuvieran ordenados sus pesos de mayor a menor. Después solo tenemos que quedarnos con los 100 primeros índices (los 100 términos con mayor peso) y generar una submatriz de la matriz original con las columnas que corresponden a esos índices.

Es importante decir que la técnica que vamos a utilizar después con estos 100 términos necesita la traspuesta de la matriz que nosotros hemos calculado. Tardamos un tiempo en darnos cuenta de que la *y* y la *x* estaban invertidas para la función que usabamos y se debía a este problema.

El siguiente paso sería calcular la matriz de distancias. Este es el proceso que consume más tiempo de todo el desarrollo de clustering por lo que debemos ser pacientes.

```
# Calculamos la distancia de cada término en cada documento
dist.matrix = proxy::dist(tfidf.matrix, method = "cosine")
dist.matrix100 = proxy::dist(tfidf.matrix100, method = "cosine")
```

Hemos visto distintas técnicas en clase de teoría sobre el cálculo de distancias y semejanzas. Como lo que tenemos es un vector de pesos por cada fila (documento), nos pareció muy apropiado calcular la semejanza utilizando el coseno del ángulo que forman dos vectores.

Como resultado, obtenemos una matriz en la que cada valor representa la medida de proximidad del ítem de la fila con el ítem de la columna, por lo que hay mucha información.

Ya tenemos los datos sobre los comentarios de los beneficios de los medicamentos listos para ser utilizados en la técnica de clustering. Principalmente hemos creado tres estrategias para generar los clusters como se puede observar a continuación.

```
#####
### FUNCIONES: QUEREMOS HACER CLUSTERING EN FUNCIÓN DEL NÚMERO DE CLUSTERS QUE DESEEMOS.
#####

clustering.kmeans <- function(centers=5){
  # Calculamos el cluster
  cluster <- kmeans(tfidf.matrix, centers)

  # Para el de 100 términos más frecuentes
  cluster100 <- kmeans(tfidf.matrix100, centers)

  # Pintamos la nube de puntos perteneciente a cada uno de los cluster
  # (cada color es un cluster)
  points <- cmdscale(t(dist.matrix), k = 2)
  palette <- colorspace::diverge_hcl(centers) # Creating a color palette

  plot(points, main = 'K-Means clustering', col = as.factor(cluster$cluster),
       mai = c(0, 0, 0, 0), mar = c(0, 0, 0, 0),
       xaxt = 'n', yaxt = 'n', xlab = '', ylab = '')

}

#####
clustering.hierarchical <- function(centers=5){
  # Calculamos el cluster
```

```

cluster <- hclust(dist.matrix, method = "ward.D2")

# Para el de 100
cluster100 <- hclust(dist.matrix100, method = "ward.D2")

# Pintamos la nube de puntos de cada uno de ellos
points <- cmdscale(dist.matrix, k = 2)
palette <- colorspace::diverge_hcl(centers) # Creating a color palette
previous.par <- par(mfrow=c(1,2), mar = rep(1.5, 4))

plot(points, main = 'Hierarchical clustering',
      col = as.factor(cutree(cluster, k = centers)),
      mai = c(0, 0, 0, 0), mar = c(0, 0, 0, 0),
      xaxt = 'n', yaxt = 'n', xlab = '', ylab = '')

# Para el de 100
plot(cluster100, cex=0.9, hang=-1)
rect.hclust(cluster100, centers, border=rainbow(centers))

}

#-----

clustering.dbscan <- function(centers=5){
  # Calculamos el cluster
  cluster <- dbSCAN::hdbscan(dist.matrix, minPts = 10)

  # Pintamos la nube de puntos de cada uno de ellos
  points <- cmdscale(dist.matrix, k = 2)
  palette <- colorspace::diverge_hcl(centers)

  plot(points, main = 'Density-based clustering',
        col = as.factor(cluster$cluster),
        mai = c(0, 0, 0, 0), mar = c(0, 0, 0, 0),
        xaxt = 'n', yaxt = 'n', xlab = '', ylab = '')
}
#-----#

```

Cada función utilizará por defecto 5 grupos en los que clasificar los diferentes términos. Si se desea cambiar este hecho es posible simplemente especificando el número de centros por medio de un argumento al llamar la función. Vamos a comentar el proceso de cada uno de ellos y los datos obtenidos.

```
# Primera técnica  
clustering.kmeans()
```

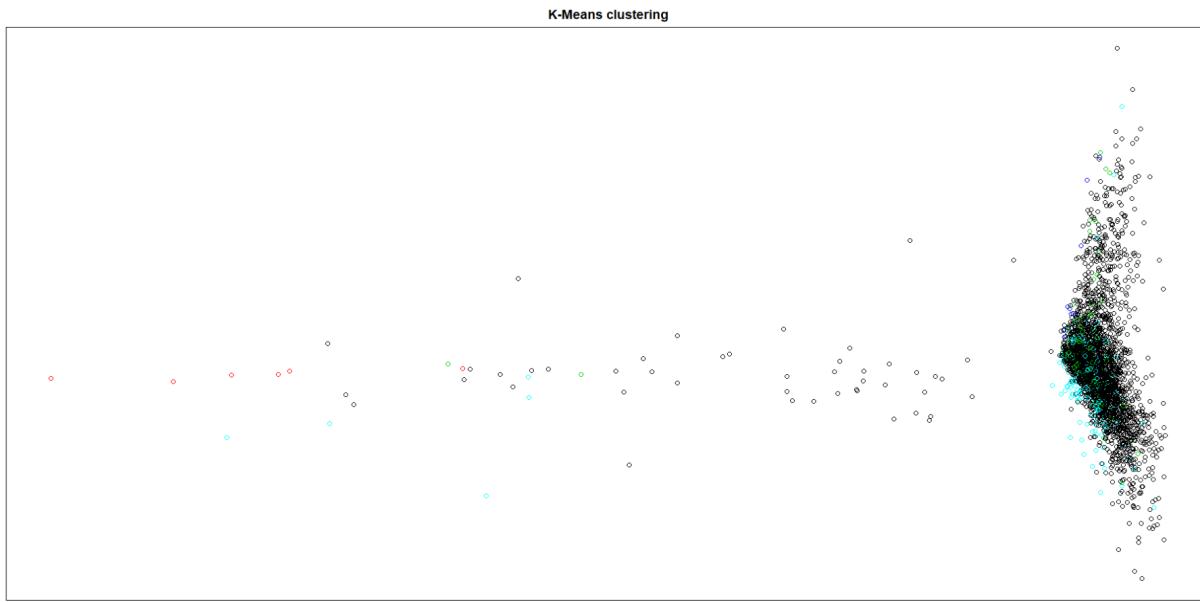


Figura 27: Agrupamiento particional por k-means.

Estuvimos debatiendo el significado de lo obtenido con esta técnica. Llegamos a la conclusión de que los métodos de clustering particional no son muy adecuados para el texto de nuestro dataset. Vemos que se genera una nube de puntos muy agrupada en la parte derecha y un porcentaje muy bajo de los items comienzan a esparcirse por la izquierda. Estás técnicas de agrupamiento particional necesitan que las nubes de puntos se encuentren disjuntas entre sí para poder situar los centroides correctamente. Casi todo el conjunto pertenece al grupo de color negro, se puede ver también como ha sacado un centroide de color azul cian en la parte izquierda inferior de la nube de puntos grande.

En esta ocasión no hemos realizado validación por silueta de los grupos obtenidos, a simple vista se puede ver que los grupos no son muy buenos y que no están claramente diferenciados.

Entonces, decidimos probar con un agrupamiento no particional, como es el jerárquico.

```
#Segunda técnica
clustering.hierarchical()
```

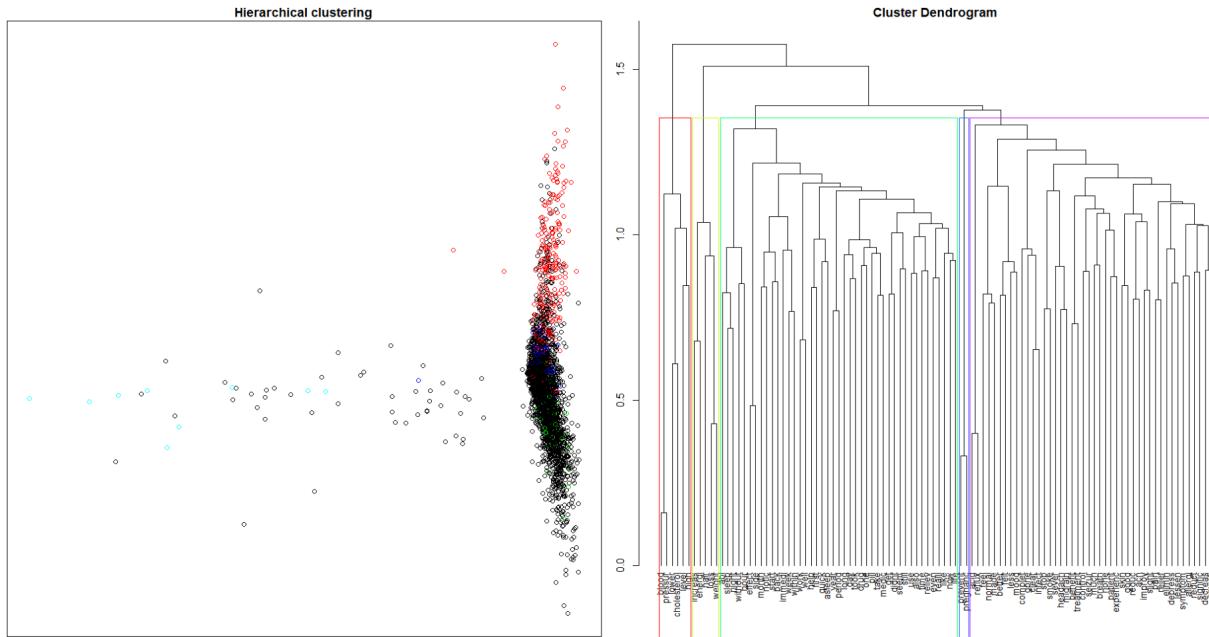


Figura 28: Agrupamiento jerárquico y muestra de anidamiento con los 100 términos más frecuentes.

En este caso, vemos que teniendo la misma nube de puntos, los resultados son más razonables. Suponemos que se debe a su criterio de anidación de términos, en los que a partir de la matriz de distancias va agrupando los términos cada vez en grupos más grandes. Vemos que hay una mejor cohesión en los grupos obtenidos, sobretodo en el grupo rojo y negro.

Como esta es la mejor técnica que sacamos, quisimos mostrar más información del proceso. Obtuvo los 100 términos más frecuentes como se mencionó anteriormente y obtuvimos un dendograma como los vistos en clase de teoría. Podemos ver que los grupos alcanzados tienen coherencia. Por ejemplo, en el primero agrupa sangre, presión, más bajo, alto, nivel...

Dependiendo del número de cluster que fijemos, especifica los grupos en niveles más altos o bajos de las llaves que vemos en la imagen. Por lo que podemos decir que siempre sigue la misma técnica de agrupamiento y lo que cambia es el momento en el que separa los cluster cuando especificamos el número de grupos que queremos tener, lo cual nos pareció curioso destacar.

```
# Tercera técnica
clustering.dbscan()
```

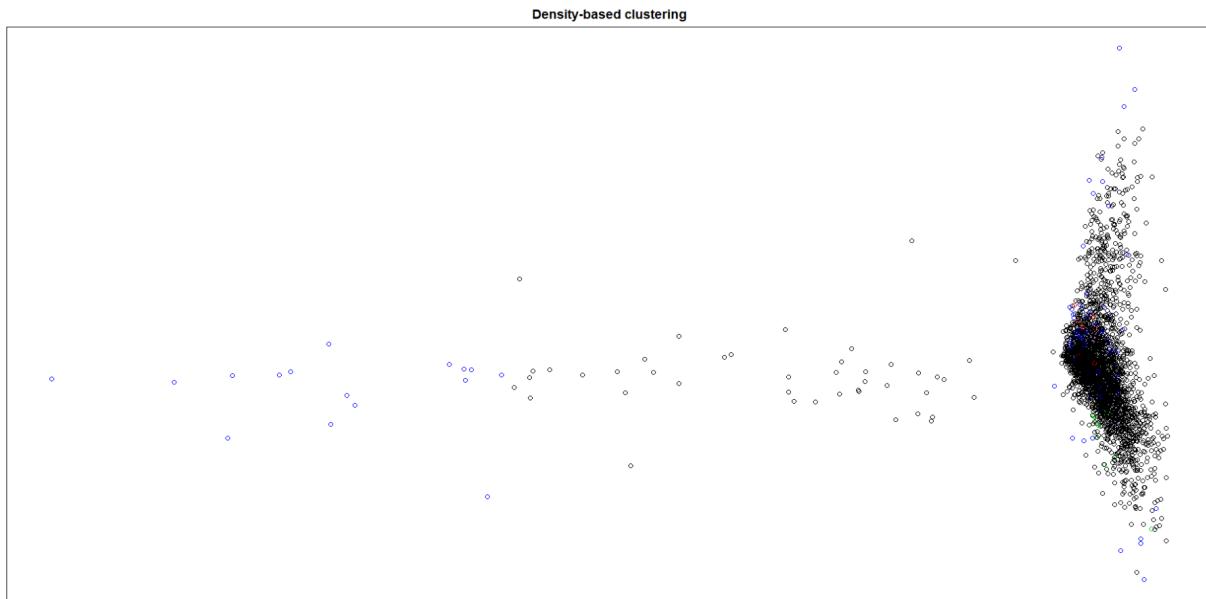


Figura 29: Agrupamiento particional por densidad.

En el agrupamiento por densidad tenemos el mismo problema que con k-medias. Los agrupamientos particionales no funcionan bien cuando los puntos en el espacio se encuentran tan unidos de forma uniforme y no disjunta.

Intentamos también realizar un word cloud en función de estos clústeres, aunque de momento no hemos tenido éxito en ese aspecto.

6.4. Density based clustering (DBSCAN)

Por último, vamos a aplicar una técnica más, la cuál se corresponde con una técnica de agrupamiento particional. Se correponde con la técnica *Density-based spatial clustering of applications* y consiste en

Recordemos que, tal y como hemos visto en la asignatura, puede darse el caso de tener zonas del espacio que sean muy densas,y otras con una densidad baja (dando lugar a grupos de baja densidad). Si tenemos en cuenta, los mismos parámetros para los dos tipos de zonas planteadas, al final estaríamos perdiendo mucha información, lo cuál no es deseable en una técnica de este tipo.

A continuación se muestra los resultados que hemos obtenido estableciendo $\text{esp}=0.15$ y $\text{MinPt}=5$. Como se aprecia en la siguiente gráfica, disponemos tanto de zonas de baja densidad como de alta densidad. Sin embargo, es muy notable la cantidad de puntos ruido que existen en nuestro conjunto, lo cual nos indica que disponemos de muchos puntos que no forman parte de ningún grupo. Este hecho nos lleva a pensar que este algoritmo no obtiene un resultado apropiado, en comparación a los métodos anteriores, y que **no se está ajustando bien a nuestro problema**.

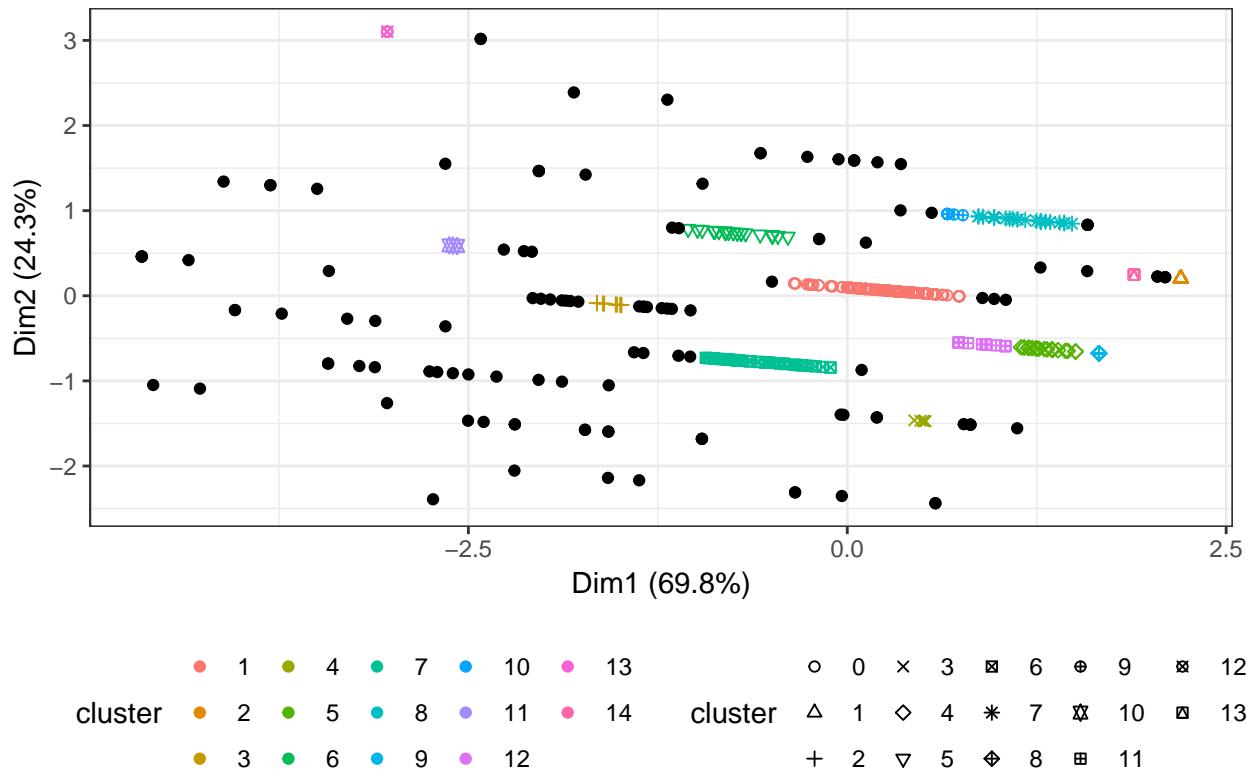
```
datos_ <- scale(data_train_procesado)
dbscan_cluster <- fpc:::dbscan(data = datos_, eps = 0.15, MinPts = 5)
# Visualización de los clusters
fviz_cluster(object = dbscan_cluster, data = datos_, stand = FALSE,
              geom = "point", ellipse = FALSE, show.clust.cent = FALSE,
```

```

pallete = "jco") +
theme_bw() +
theme(legend.position = "bottom")

```

Cluster plot

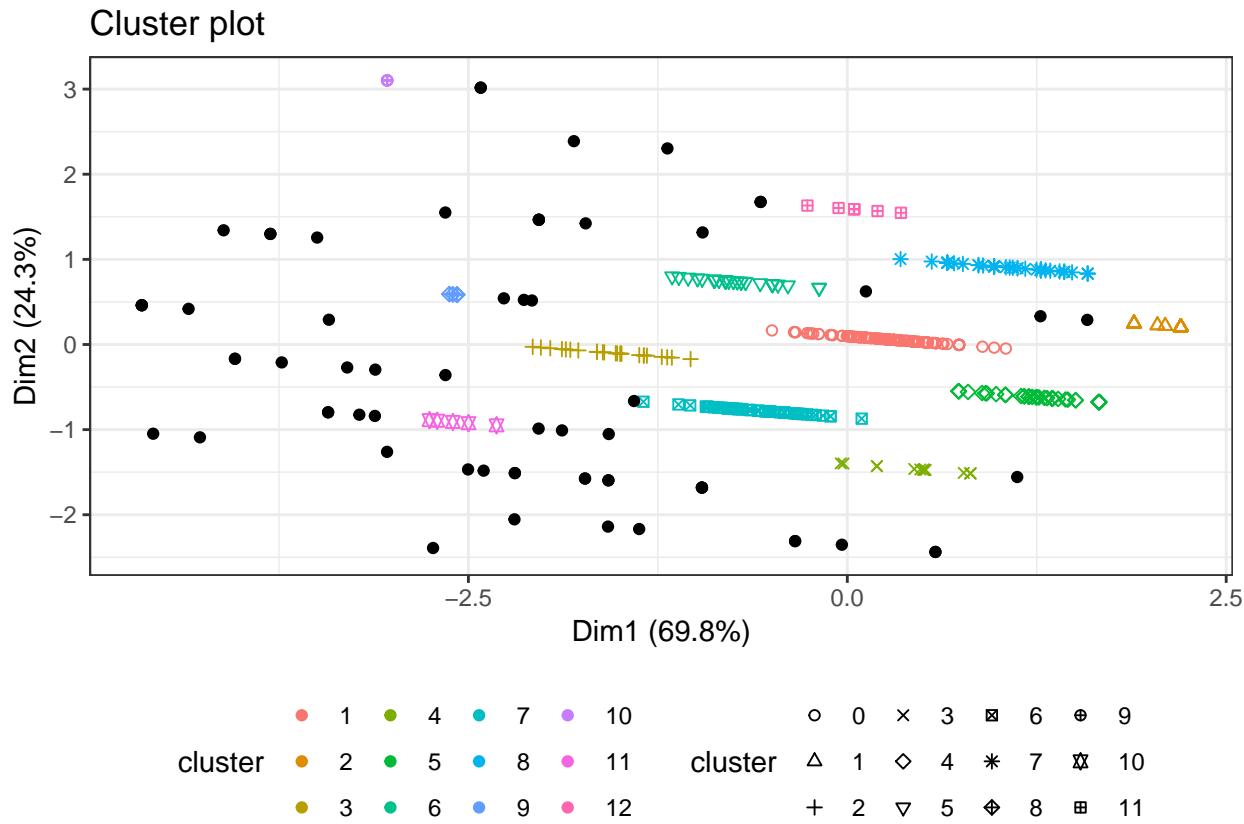


Podemos intenta ponerle solución, estableciendo radios de mayor amplitud, para así permitir que más puntos formen parte de los núcleos que se obtengan con la técnica. A continuación podemos ver el resultado obtenido. De nuevo, aunque podemos ver que abarcamos mayor cantidad de items, una gran cantidad de los datos siguen siendo considerados ruido. Además, se puede apreciar claramente como grandes zonas del espacio no son capaces de agruparse. **Por ello, pensamos que, por las características de nuestro problema, esta técnica no es apropiada.**

```

datos_ <- scale(data_train_procesado)
dbSCAN_cluster <- fpc::dbSCAN(data = datos_, eps = 0.4, MinPts = 5)
# Visualización de los clusters
fviz_cluster(object = dbSCAN_cluster, data = datos_, stand = FALSE,
             geom = "point", ellipse = FALSE, show.clust.cent = FALSE,
             pallete = "jco") +
theme_bw() +
theme(legend.position = "bottom")

```



6.5. KNN

En esta sección vamos a realizar técnicas de clasificación basada en instancias. Concretamente el de vecinos más cercanos (KNN). Esta técnica es puramente predictiva, por lo que no proporciona ninguna información acerca de la dependencia de variables. Dado un texto, busca en el conjunto de entrenamiento los items que tienen un mayor parentesco del que estamos tratando de deducir y copia su clase. La clase que vamos a intentar reducir es **ratingLabel** que es la columna preprocesada de **rating**. Nos hubiera gustado probar y comentar esta técnica tratando de clasificar otras clases, pero no tenemos tiempo suficiente como para ello.

Esta técnica es buena cuando el conjunto de items se encuentran disjuntos entre sí debido a sus clases. Aunque el clustering es una técnica no supervisada en la que de antemano no teníamos etiquetas concretas que clasificar, nos dimos cuenta de que no era una opción que nos aportara mucho debido a este hecho (la nube de puntos no se encontraba disjuntas en grupos claros). Por lo que de antemano pensamos que no va a devolvernos unas predicciones muy prometedoras.

Antes de comenzar, destacar que probamos con variables numéricas antes de trabajar con los textos. Como nuestro dataset no tiene variables continuas, probamos con variables discretas. Tal y como se dice en la teoría de esta asignatura, este algoritmo no trabaja bien con este tipo de valores numéricos debido a que el cálculo de la distancias hacia que hubiera una gran cantidad de distancias iguales e items superpuestos en un plano bidimensional. En definitiva, tuvimos problemas a la hora de procesar el algoritmo (concretamente “too many ties in KNN”, debido a lo comentado anteriormente). Así que nos pusimos a trabajar con cálculos de distancias en los comentarios, lo cual es más apropiado para esta técnica debido a que no tenemos valores continuos como tal en nuestro dataset.

Nos encontramos con un problema que tuvimos en la implementación de este algoritmo y el cual nos llevó mucho tiempo detectar y solucionar. Básicamente, nosotros estamos trabajando con dos dataset ya diferenciados desde el comienzo; el train y el test. Para esta técnica debemos de calcular la matriz de términos, si lo hacemos de los dos dataset por separado, tenemos el problema de que ambas matrices de términos no

tienen exactamente el mismo formato (las columnas pueden tener diferentes palabras). Entonces, teníamos errores de dimensiones al aplicar el algoritmo, ya que obteníamos un clasificador que no era apto para predecir el test.

La solución es calcular la matriz de términos de un dataset, y una vez realizados todos los cálculos y preparación de la matriz de términos que explicaremos más adelante, separamos esa misma matriz de términos (un 70 % para el train y un 30 % para el test). De esta forma nos aseguramos de que ambas matrices tienen exactamente las mismas columnas y los mismos términos y ya podemos trabajar correctamente con el algoritmo KNN.

Dicho esto, vamos a comenzar. Primero cargamos los datos.

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

El siguiente paso consiste en obtener el Corpus como hemos hecho en técnicas anteriores. Primero vamos a hacerlo sobre los comentarios acerca de los beneficios de los medicamentos. Acto seguido, obtenemos la matriz de términos correspondiente con sus vectores de pesos.

Para que el algoritmo funcione, es importante que esa matriz sea de tipo *matrix*. Una vez hecho esto, podemos separar la matriz en el conjunto de entrenamiento y prueba. Obtenemos el clasificador y predecimos solo el error del test porque la función obtiene el clasificador y la predicción en la misma llamada, por lo que para calcular el error de entrenamiento tendríamos que recalcular el mismo clasificador y nos parecía ineficiente.

Como ya hemos mencionado anteriormente, KNN selecciona los **k** items del conjunto de entrenamiento más parecidos al que estamos deduciendo (calculando este parentesco con distancia euclídea). Acto seguido, el algoritmo deduce la clase del ítem siendo la clase más frecuente de entre sus vecinos.

Por consiguiente, el valor de **k** es muy importante a la hora de realizar el clasificador. Un valor muy alto puede producir sobreaprendizaje y un valor muy bajo puede producir un aumento significativo en la tasa de error, por lo que probaremos con distintos valores.

```
# Para train KNN
corpus = tm::Corpus(tm::VectorSource(datos_preprocesados$benefits_preprocesado))
tdm <- tm::DocumentTermMatrix(corpus)

# Transform dtm to matrix and then back into a data frame for modeling
mat.df <- as.data.frame(data.matrix(tdm), stringsAsFactors = FALSE)

# Column bind category (known classification)
mat.df <- cbind(mat.df, datos_preprocesados$ratingLabel, row.names = NULL)

colnames(mat.df)[ncol(mat.df)] <- "ratingLabel"

# Classifier variable based on the personality typ
cl <- mat.df[, "ratingLabel"]

# Create model data and remove "type"
modeldata <- mat.df[, !colnames(mat.df) %in% "ratingLabel"]

train <- sample(nrow(mat.df), ceiling(nrow(mat.df) * .70))
test <- (1:nrow(mat.df))[-train]
```

Para **k=1**:

```
knn.pred <- knn(modeldata[train,], modeldata[test,], cl[train], k=1)

conf.mat <- table("Predictions" = knn.pred, Actual = cl[test])
conf.mat
```

```
(accuracy <- sum(diag(conf.mat))/length(test) * 100)
```

		Actual
		0 1
Predictions	0	137 277
	1	88 429
		[1] 60.79484

Figura 30: Matriz de confusión y tasa de precisión en modelo KNN con beneficios (k=1).

Para K=1 vemos que obtenemos de por sí una tasa de precisión razonable a sabiendas de que no iba a funcionar muy bien. Es decir, clasifica bien el *ratingLabel* en más de la mitad de las ocasiones.

Como podemos observar en la matriz de confusión, tiene un error alrededor del 40 % para el conjunto del test. Tal y como nos temíamos, la técnica no es muy apropiada para nuestro problema debido a la disposición del los textos. Sin embargo, vamos a probar con valores distintos de k para intentar mejorar hasta cierto punto el desempeño del algoritmo.

Para **k=3**:

```
knn.pred <- knn(modeldata[train,], modeldata[test,], cl[train], k=3)

conf.mat <- table("Predictions" = knn.pred, Actual = cl[test])
conf.mat

(accuracy <- sum(diag(conf.mat))/length(test) * 100)
```

		Actual
		0 1
Predictions	0	138 281
	1	87 425
		[1] 60.47261

Figura 31: Matriz de confusión y tasa de precisión en modelo KNN con beneficios (k=3).

Para k=3 el cambio no es significativo, incluso empeora un poco. Probamos con **k=5**:

```
knn.pred <- knn(modeldata[train,], modeldata[test,], cl[train], k=5)

conf.mat <- table("Predictions" = knn.pred, Actual = cl[test])
conf.mat

(accuracy <- sum(diag(conf.mat))/length(test) * 100)
```

		Actual
		0 1
Predictions	0	139 278
	1	86 428
		[1] 60.90226

Figura 32: Matriz de confusión y tasa de precisión en modelo KNN con beneficios (k=5).

Para k=5 tenemos el mejor clasificador hasta el momento, aunque vemos que se mantiene bastante estable, sin cambios realmente significativos.

Para **k=10**:

```

knn.pred <- knn(modeldata[train,], modeldata[test,], cl[train], k=10)

conf.mat <- table("Predictions" = knn.pred, Actual = cl[test])
conf.mat

(accuracy <- sum(diag(conf.mat))/length(test) * 100)

```

		Actual
		0 1
Predictions	0	162 363
	1	63 343

[1] 54.24275

Figura 33: Matriz de confusión y tasa de precisión en modelo KNN con beneficios (k=10).

En este punto, el clasificador tiene un bajón de un 6 % en la precisión. Vemos que siempre tenemos más falsos positivos que negativos. Es decir, el clasificador es positivo y suele dar más ratings altos que bajos cuando se equivoca.

Para **k=15**:

```

knn.pred <- knn(modeldata[train,], modeldata[test,], cl[train], k=15)

conf.mat <- table("Predictions" = knn.pred, Actual = cl[test])
conf.mat

(accuracy <- sum(diag(conf.mat))/length(test) * 100)

```

		Actual
		0 1
Predictions	0	180 441
	1	45 265

[1] 47.79807

Figura 34: Matriz de confusión y tasa de precisión en modelo KNN con beneficios (k=15).

Para **k=30**:

```

knn.pred <- knn(modeldata[train,], modeldata[test,], cl[train], k=30)

conf.mat <- table("Predictions" = knn.pred, Actual = cl[test])
conf.mat

(accuracy <- sum(diag(conf.mat))/length(test) * 100)

```

		Actual
		0 1
Predictions	0	196 596
	1	11 128

[1] 34.80129

Figura 35: Matriz de confusión y tasa de precisión en modelo KNN con beneficios (k=30).

Para **k=50**:

```

knn.pred <- knn(modedata[train,], modedata[test,], cl[train], k=50)

conf.mat <- table("Predictions" = knn.pred, Actual = cl[test])
conf.mat

(accuracy <- sum(diag(conf.mat))/length(test) * 100)

```

		Actual	
		0	1
Predictions	0	199	650
	1	8	74
	[1]	29.32331	

Figura 36: Matriz de confusión y tasa de precisión en modelo KNN con beneficios ($k=50$).

Como podemos observar, el modelo mantiene una precisión estable. Sin embargo a partir del valor 5 para **k** cuanto más vecinos, peor es el modelo. Podemos verlo de una forma más sencilla en la siguiente gráfica.

```

valores_k <- c(1,3,5,10,15,30,50)
precision <- c(60.79484,60.47261,60.90226,54.24275,47.79807,34.80129,29.32331)

plot(x=valores_k,y=precision, xlab = "Valores de K", ylab="Precisión(%)",
      ylim = c(0,100), type="o", col="green")
title("Precisión del modelo KNN en función del número de vecinos(K)")

```

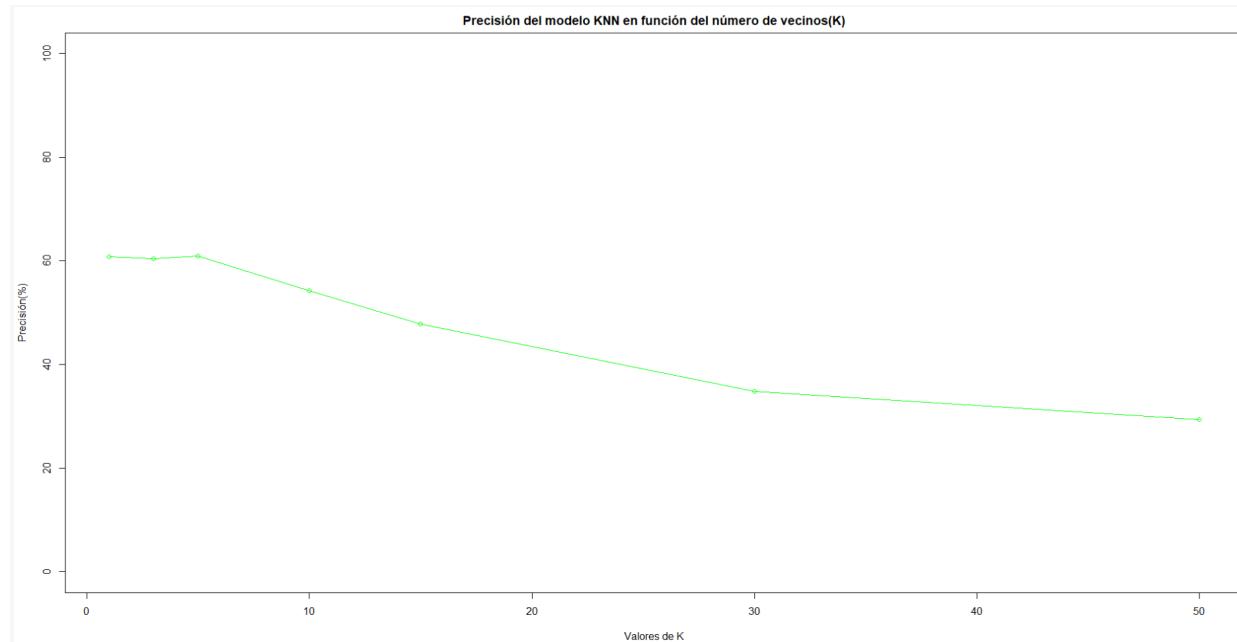


Figura 37: Gráfico del progreso del modelo KNN en función del valor de k .

Hacemos lo mismo para los comentarios de efectos secundarios que tenemos disponibles en nuestro dataset.

```

# Para train KNN
corpus = tm::Corpus(tm::VectorSource(datos_preprocesados$effects_preprocesado))
tdm <- tm::DocumentTermMatrix(corpus)

```

```

# Transform dtm to matrix and then back into a data frame for modeling
mat.df <- as.data.frame(data.matrix(tdm), stringsAsFactors = FALSE)

# Column bind category (known classification)
mat.df <- cbind(mat.df, datos_preprocesados$ratingLabel, row.names = NULL)

colnames(mat.df)[ncol(mat.df)] <- "ratingLabel"

# Classifier variable based on the personality typ
cl <- mat.df[, "ratingLabel"]

# Create model data and remove "type"
modeldata <- mat.df[, !colnames(mat.df) %in% "ratingLabel"]

train <- sample(nrow(mat.df), ceiling(nrow(mat.df) * .70))
test <- (1:nrow(mat.df))[-train]

```

Para k=1:

```

knn.pred <- knn(modeldata[train,], modeldata[test,], cl[train], k=1)

conf.mat <- table("Predictions" = knn.pred, Actual = cl[test])
conf.mat

(accuracy <- sum(diag(conf.mat))/length(test) * 100)

```

		Actual	
		0	1
Predictions	0	38	54
	1	158	681
	[1]	77.22879	

Figura 38: Matriz de confusión y tasa de precisión en modelo KNN con efectos secundarios (k=1).

Para k=3:

```

knn.pred <- knn(modeldata[train,], modeldata[test,], cl[train], k=3)

conf.mat <- table("Predictions" = knn.pred, Actual = cl[test])
conf.mat

(accuracy <- sum(diag(conf.mat))/length(test) * 100)

```

		Actual	
		0	1
Predictions	0	18	17
	1	178	718
	[1]	79.05478	

Figura 39: Matriz de confusión y tasa de precisión en modelo KNN con efectos secundarios (k=3).

Para k=5:

```

knn.pred <- knn(modeldata[train,], modeldata[test,], cl[train], k=5)

```

```

conf.mat <- table("Predictions" = knn.pred, Actual = cl[test])
conf.mat

(accuracy <- sum(diag(conf.mat))/length(test) * 100)

```

Parece que este vuelve a ser el mejor valor para k en la construcción del modelo.

		Actual
Predictions		
	0	1
0	11	6
1	185	729
[1]		79.48443

Figura 40: Matriz de confusión y tasa de precisión en modelo KNN con efectos secundarios (k=5).

Para k=15:

```

knn.pred <- knn(modeldata[train,], modeldata[test,], cl[train], k=15)

conf.mat <- table("Predictions" = knn.pred, Actual = cl[test])
conf.mat

(accuracy <- sum(diag(conf.mat))/length(test) * 100)

```

		Actual
Predictions		
	0	1
0	4	0
1	192	735
[1]		79.37701

Figura 41: Matriz de confusión y tasa de precisión en modelo KNN con efectos secundarios (k=15).

Es curioso que en este último caso solo tiene falsos positivos, no negativos.

Vemos que ocurre algo similar que con los comentarios referentes a los beneficios, solo que con una tasa de precisión más alta (solo se equivocaría 1 de cada 5 predicciones aproximadamente). Por algún motivo, es capaz de predecir mejor este tipo de comentarios sobre efectos secundarios. Suponemos que porque, entre otras cosas, la nube de puntos es más apropiada en esta ocasión para esta técnica.

Llegamos a la conclusión en ambos tipo de comentarios que los valores apropiados para k deben ser bajos (valor de 5), por los datos mostrados anteriormente.

6.6. Árboles de decisión

El árbol de decisión es una técnica clásica de clasificación en la que se realizan particiones binarias de los datos de forma recursiva. El resultado se puede representar con un árbol.

Para la construcción del árbol también se puede utilizar la función **rpart**. El principal parámetro de esta función es la complejidad, **cp**. Éste permite simplificar los modelos ajustados mediante la poda de las divisiones que no merecen la pena. Otros parámetros importantes son **minsplit**, número mínimo de observaciones que debe haber en un nodo para que se intente una partición, y **minbucket**, número mínimo de observaciones de un nodo terminal. Por defecto **minsplit= 20**, **minbucket=round(minsplit/3)** y **cp= 0.01**.

A la hora de determinar los parámetros, el procedimiento habitual para determinar los parámetros del modelo consiste en dejar que el árbol crezca hasta que cada nodo terminal tenga menos de un número mínimo de

observaciones, ya que en algunas divisiones puede que apenas mejore y en las siguientes sí lo haga. Para determinar cp se considera como 0, para que el árbol crezca lo máximo posible.

A continuación se va a aplicar dicha técnica para realizar las siguientes predicciones:

6.6.1. Predicción de effectivenessNumber en función del rating

En esta sección se va a predecir la efectividad (de 1 a 5, siendo 1 menos efectiva y 5 más efectiva) en función del rating. Para ello se va a seguir con el procedimiento que se ha descrito en la introducción.

A continuación se va a mostrar el árbol de decisión que se ha generado utilizando **rpart**.

```
library("rattle")
library("rpart")
library("rpart.plot")

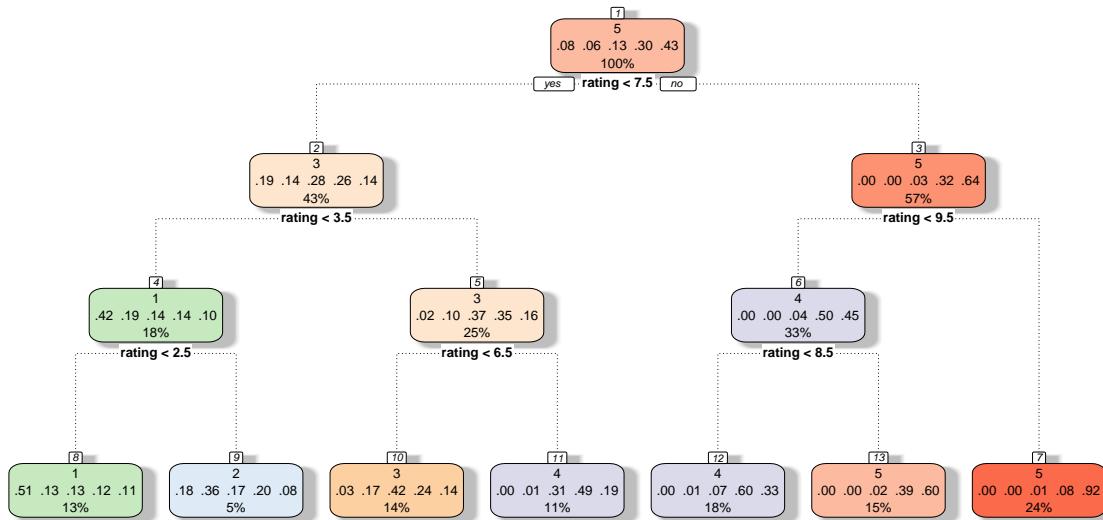
seed <- 42

set.seed(seed)

rpart <- rpart(effectivenessNumber ~ rating, data=datos_train,
                method="class",
                parms=list(split="information"),
                control=rpart.control(minsplit=30,
                                      minbucket=10,
                                      cp=0.00,
                                      usesurrogate=0,
                                      maxsurrogate=0)
               )

fancyRpartPlot(rpart, main="Decision Tree effectivenessNumber $ ratingLabel")
```

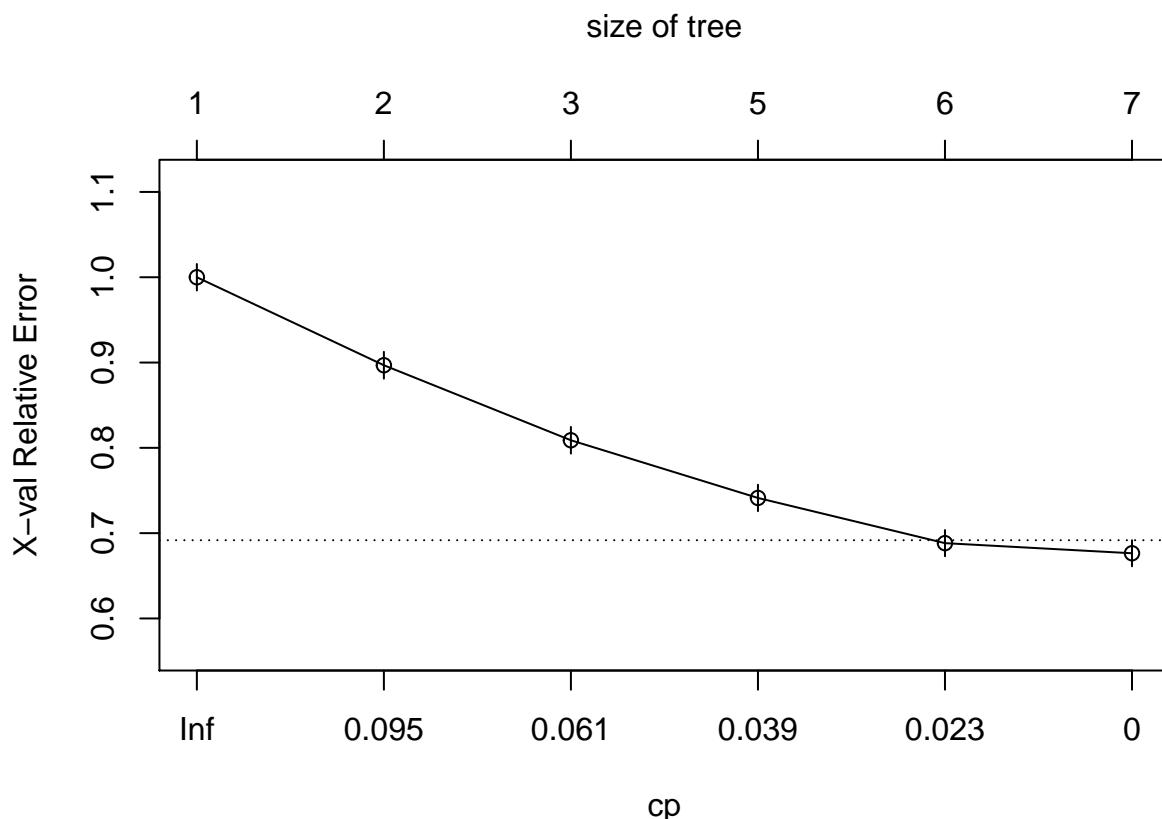
Decision Tree effectivenessNumber \$ ratingLabel



Rattle 2019–ene–30 19:06:31 gema

Como el árbol que se obtiene es muy grande, se podaría considerando una función de coste basada en la complejidad. Para esto, se calcula el error de la validación cruzada y se elige el menor.

```
plotcp(rpart)
```



Mostramos la tabla de resultados

```
printcp(rpart)
```

```
##
## Classification tree:
## rpart(formula = effectivenessNumber ~ rating, data = datos_train,
##       method = "class", parms = list(split = "information"), control = rpart.control(minsplit = 30,
##                                     minbucket = 10, cp = 0, usesurrogate = 0, maxsurrogate = 0))
##
## Variables actually used in tree construction:
## [1] rating
##
## Root node error: 1774/3104 = 0.57152
##
## n= 3104
##
##          CP nsplit rel error  xerror      xstd
## 1 0.103157     0    1.00000 1.00000 0.015541
## 2 0.087937     1    0.89684 0.89684 0.015698
## 3 0.042841     2    0.80891 0.80891 0.015658
## 4 0.034949     4    0.72322 0.74126 0.015519
## 5 0.014656     5    0.68828 0.68828 0.015342
## 6 0.000000     6    0.67362 0.67644 0.015294
```

Tal y como se puede ver en la tabla anterior, el mínimo error se alcanza en el nodo 6.

```
6 0.000000      6  0.67362 0.67644 0.015294
```

Típicamente se considera que hasta la línea discontinua de la figura anterior no hay diferencias significativas.

Esta línea es la suma del mínimo error y la desviación típica, es decir:

$$0.67644 + 0.015294 = 0.691734$$

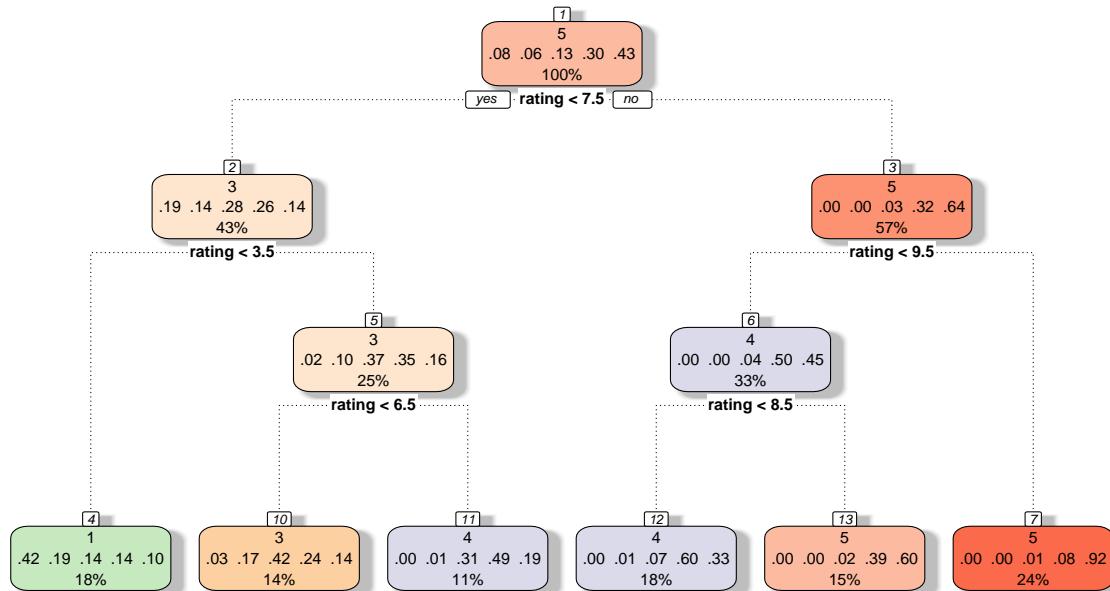
Observamos en el gráfico anterior que dicho error tiene un cp de 0.023, por lo que se va a volver a generar el árbol con rpart utilizando ahora como valor de cp 0,023.

```
set.seed(seed)
rpart <- rpart(effectivenessNumber ~ rating, data=datos_train,
               method="class",
               parms=list(split="information"),
               control=rpart.control(minsplit=30,
                                     minbucket=10,
                                     cp=0.023,
                                     usesurrogate=0,
                                     maxsurrogate=0)
)
# Vista textual del modelo árbol de decisión:
print(rpart)

## n= 3104
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 3104 1774 5 (0.08 0.06 0.13 0.3 0.43)
##      2) rating< 7.5 1324  959 3 (0.19 0.14 0.28 0.26 0.14)
##         4) rating< 3.5 553   318 1 (0.42 0.19 0.14 0.14 0.099) *
##         5) rating>=3.5 771   485 3 (0.016 0.096 0.37 0.35 0.16)
##            10) rating< 6.5 422   245 3 (0.028 0.17 0.42 0.24 0.14) *
##            11) rating>=6.5 349   178 4 (0 0.0086 0.31 0.49 0.19) *
##            3) rating>=7.5 1780   632 5 (0 0.0028 0.028 0.32 0.64)
##               6) rating< 9.5 1038   518 4 (0 0.0039 0.044 0.5 0.45)
##                  12) rating< 8.5 558   224 4 (0 0.0072 0.068 0.6 0.33) *
##                  13) rating>=8.5 480   194 5 (0 0 0.017 0.39 0.6) *
##                  7) rating>=9.5 742    62 5 (0 0.0013 0.0054 0.077 0.92) *

fancyRpartPlot(rpart, main="Árbol de decisión effectivenessNumber $ rating")
```

Árbol de decisión effectivenessNumber \$ rating



Rattle 2019-ene-30 19:06:32 gema

A continuación se lleva a cabo la predicción del conjunto de prueba y se calcula la matriz de confusión.

Para cada uno de los datos del conjunto de prueba se sigue el árbol según el valor de sus variables hasta llegar a los nodos terminales, allí, se clasifica con la categoría más probable de ese nodo terminal, como se vio en el árbol anterior.

La denominada matriz de confusión permite la visualización de los resultados del clasificador. Las filas muestran los valores reales y las columnas las predicciones del modelo. A partir de esta tabla se calcula la precisión del modelo.

```

pred <- predict(rpart, newdata=datos_test, type="class")

# Matriz de confusión
tabla <- table(datos_test$effectivenessNumber, pred, useNA="ifany",
dnn=c("Real", "Predicho" ))

tabla

##      Predicho
## Real   1    2    3    4    5
##   1    74    0    5    0    2
##   2    46    0   28    2    0
##   3    34    0   65   56    2
##   4    28    0   41  170   70
##   5    14    0   30   82  285
  
```

Esta tabla es la denominada matriz de confusión, que permite la visualización de los resultados del clasificador. Las filas muestran los valores reales y las columnas las predicciones del modelo. A partir de esta tabla se calcula la precisión del modelo.

```
sum(diag(tabla))/nrow(datos_test)
```

```
## [1] 0.5744681
```

La precisión de este modelo es del **57,44 %** aproximadamente.

El poder de predicción de los árboles de decisión no suele ser muy bueno, pero el algoritmo es sencillo y los modelos resultantes tienen una fácil interpretación.

6.6.2. Predicción de sideEffectsNumber en función del rating

Siguiendo el mismo proceso que se ha seguido para la obtención del árbol generado anteriormente, se va a predecir el valor de efectos secundarios en función del rating del medicamento.

```
seed <- 42

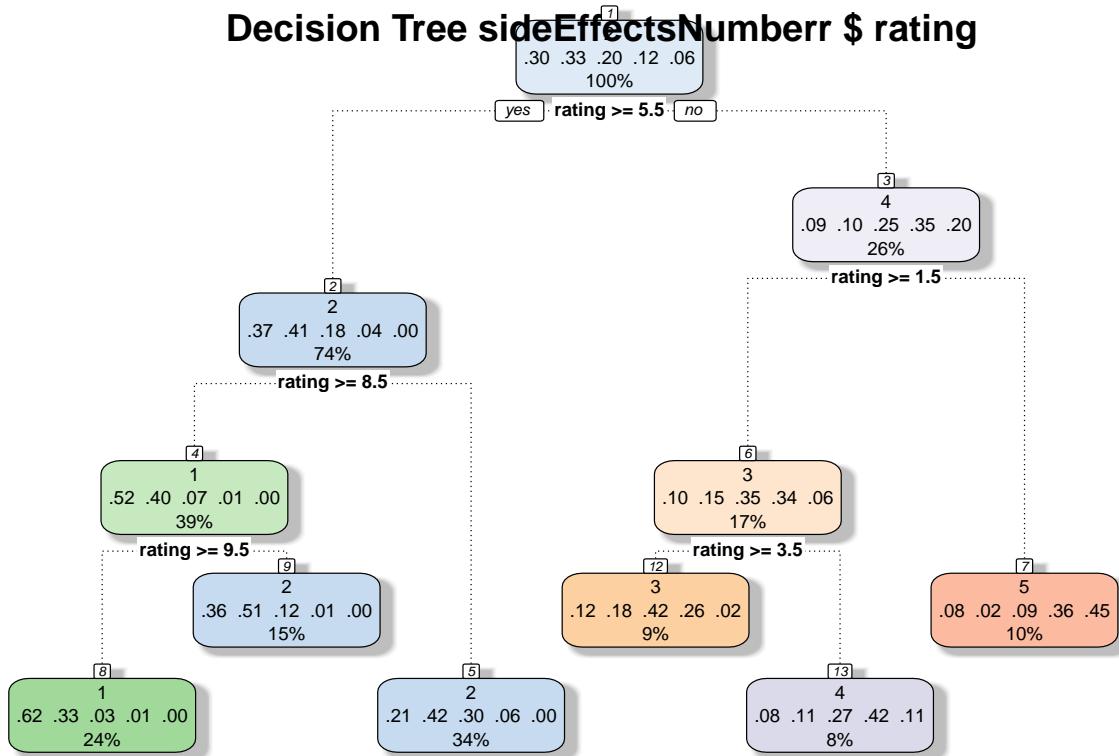
set.seed(seed)

rpart <- rpart(sideEffectsNumber ~ rating, data=datos_train,
                method="class",
                parms=list(split="information"),
                control=rpart.control(minsplit=30,
                                      minbucket=10,
                                      cp=0.01,
                                      usesurrogate=0,
                                      maxsurrogate=0)
               )

print(rpart)

## n= 3104
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 3104 2085 2 (0.3 0.33 0.2 0.12 0.056)
##      2) rating>=5.5 2286 1349 2 (0.37 0.41 0.18 0.036 0.0035)
##         4) rating>=8.5 1222 591 1 (0.52 0.4 0.067 0.012 0.0025)
##            8) rating>=9.5 742 282 1 (0.62 0.33 0.03 0.013 0.004) *
##            9) rating< 9.5 480 236 2 (0.36 0.51 0.12 0.01 0) *
##            5) rating< 8.5 1064 618 2 (0.21 0.42 0.3 0.064 0.0047) *
##            3) rating< 5.5 818 533 4 (0.093 0.1 0.25 0.35 0.2)
##               6) rating>=1.5 513 333 3 (0.1 0.15 0.35 0.34 0.06)
##                  12) rating>=3.5 265 153 3 (0.12 0.18 0.42 0.26 0.015) *
##                     13) rating< 3.5 248 144 4 (0.085 0.11 0.27 0.42 0.11) *
##                     7) rating< 1.5 305 169 5 (0.075 0.023 0.092 0.36 0.45) *
```

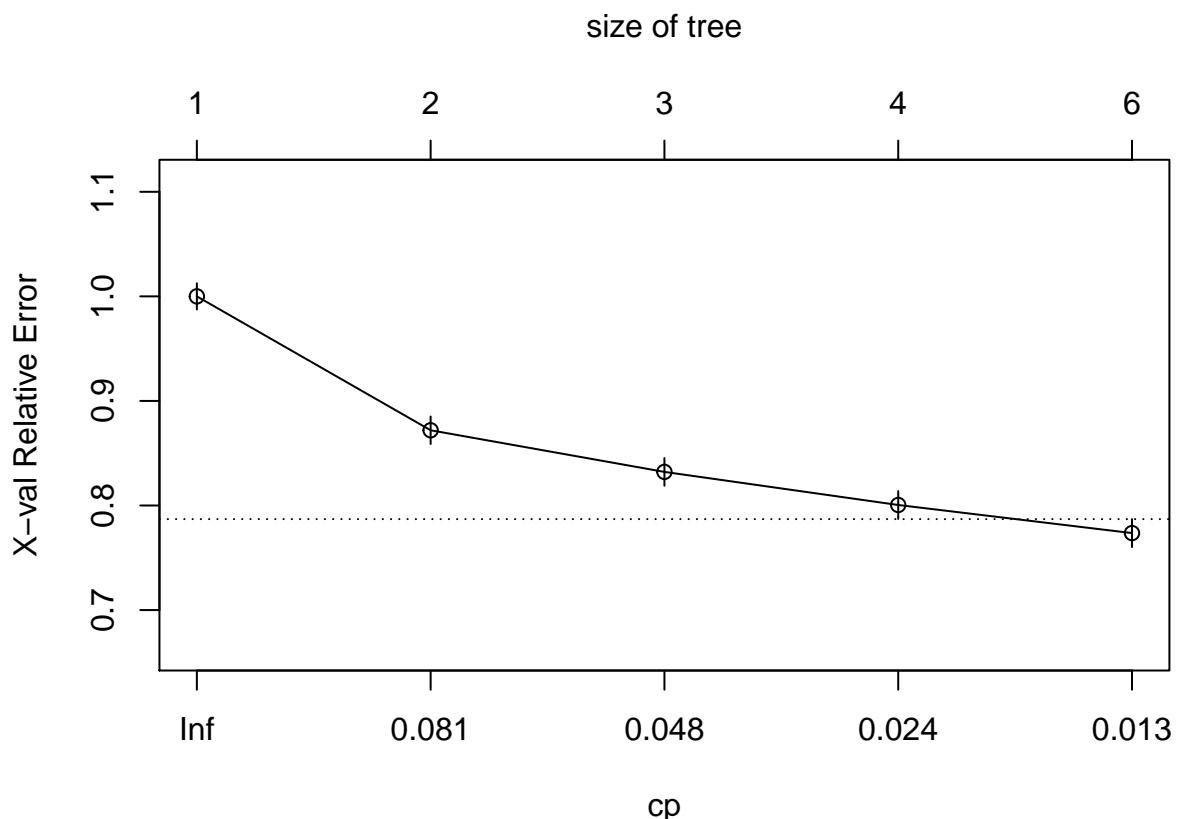
```
fancyRpartPlot(rpart, main="Decision Tree sideEffectsNumber $ rating")
```



Rattle 2019–ene–30 19:06:32 gema

Mostramos la gráfica del error de la validación cruzada.

```
plotcp(rpart)
```



Generamos la predicción

```
pred <- predict(rpart, newdata=datos_test, type="class")

# Matriz de confusión
tabla <- table(datos_test$sideEffectsNumber, pred, useNA="ifany",
dnn=c("Real", "Predicho" ))

tabla

##      Predicho
## Real    1    2    3    4    5
##   1 141 109   7   7   4
##   2  74 219  24  10   2
##   3   9 143  48  26  10
##   4   2  24  30  31  35
##   5   0   6   2   8  63
```

Obtenemos el resultado

```
sum(diag(tabla))/nrow(datos_test)
```

```
## [1] 0.4854932
```

La precisión de este modelo es del **50,58 %** aproximadamente.

6.6.3. Predicción de effectivenessNumber en función del weightedRating

Siguiendo el mismo proceso que se ha seguido para la obtención del árbol generado anteriormente, se va a predecir el valor de la efectividad del medicamento en función del rating ponderado que se ha generado

durante el procesamiento (70 % efectos secundarios, 30 % efectividad).

```
seed <- 42

set.seed(seed)

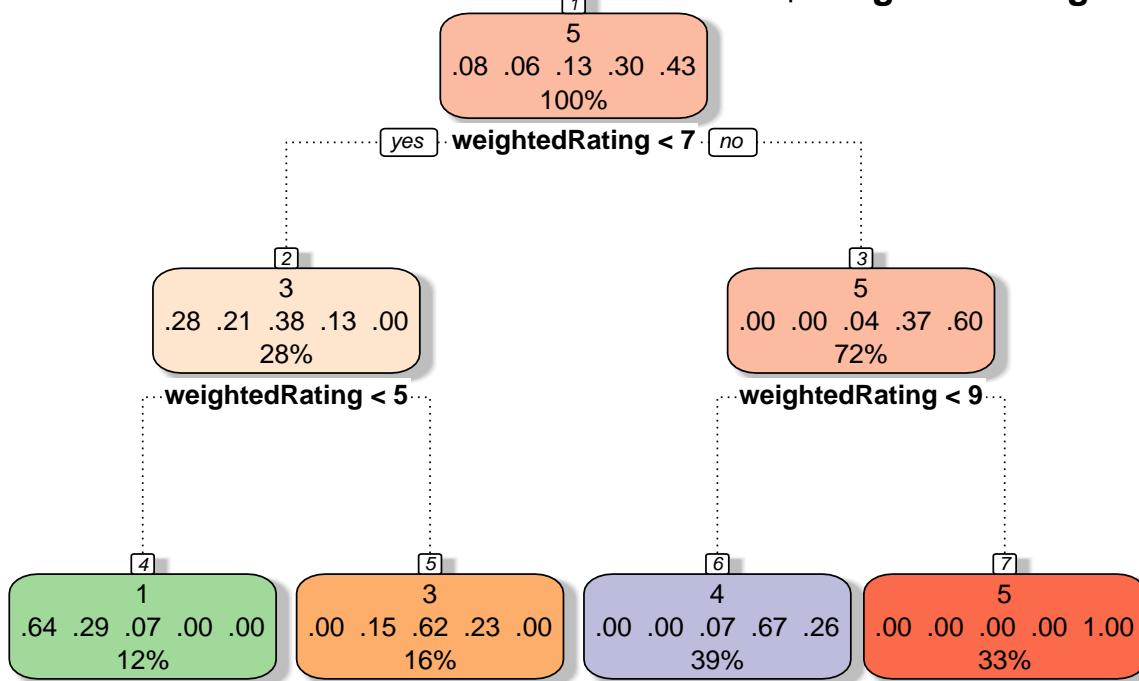
rpart <- rpart(effectivenessNumber ~ weightedRating, data=datos_train,
                method="class",
                parms=list(split="information"),
                control=rpart.control(minsplit=30,
                                      minbucket=10,
                                      cp=0.03,
                                      usesurrogate=0,
                                      maxsurrogate=0)
                )

print(rpart)

## n= 3104
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 3104 1774 5 (0.08 0.06 0.13 0.3 0.43)
##    2) weightedRating< 7 881  547 3 (0.28 0.21 0.38 0.13 0)
##      4) weightedRating< 5 387   140 1 (0.64 0.29 0.067 0 0) *
##      5) weightedRating>=5 494   186 3 (0 0.15 0.62 0.23 0) *
##    3) weightedRating>=7 2223  893 5 (0 0 0.036 0.37 0.6)
##      6) weightedRating< 9 1212   400 4 (0 0 0.067 0.67 0.26) *
##      7) weightedRating>=9 1011     0 5 (0 0 0 0 1) *

fancyRpartPlot(rpart, main="Decision Tree effectivenessNumber $ weightedRating")
```

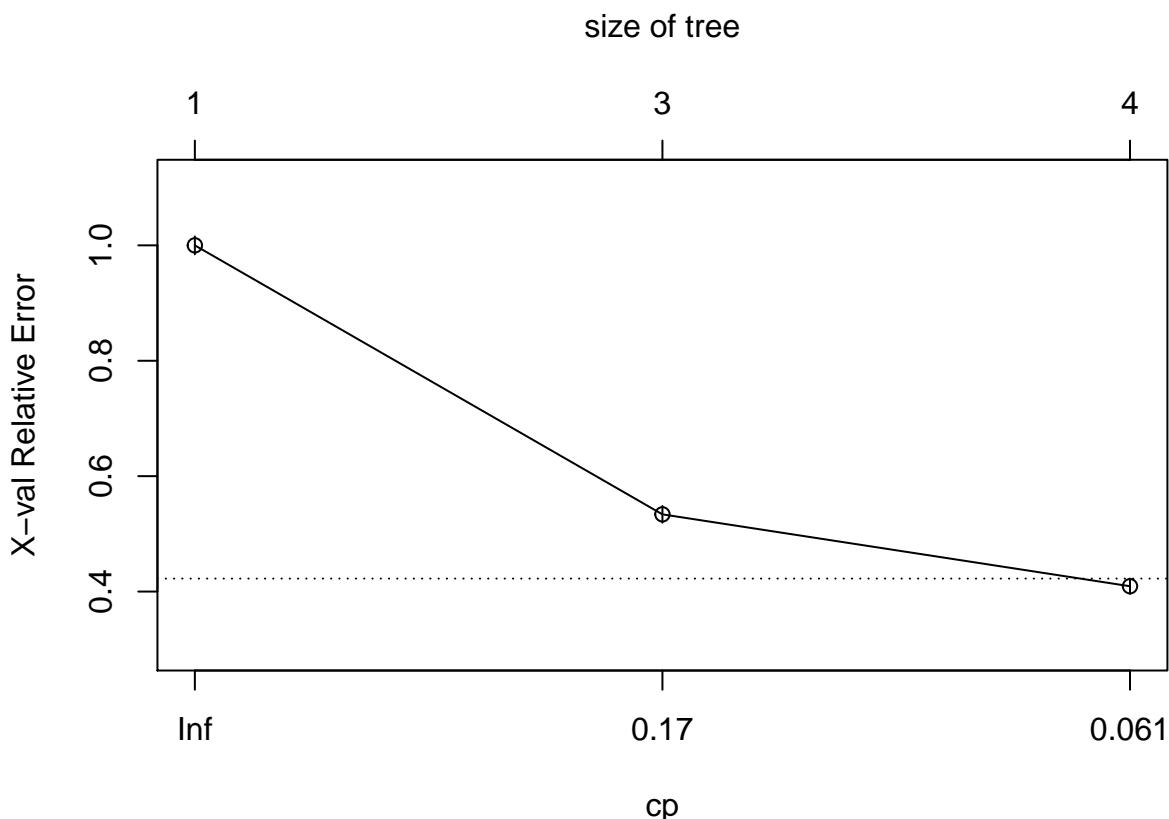
Decision Tree effectivenessNumber \$ weightedRating



Rattle 2019–ene–30 19:06:33 gema

Mostramos la gráfica del error de la validación cruzada.

```
plotcp(rpart)
```



Generamos la predicción

```

pred <- predict(rpart, newdata=datos_test, type="class")

# Matriz de confusión
tabla <- table(datos_test$effectivenessNumber, pred, useNA="ifany",
dnn=c("Real", "Predicho" ))

tabla

##      Predicho
## Real    1    2    3    4    5
##   1    63    0    9    9    0
##   2    25    0   43    8    0
##   3    42    0   53   62    0
##   4    14    0  108  123   64
##   5    12    0   25  210  164

```

Obtenemos el resultado

```
sum(diag(tabla))/nrow(datos_test)
```

```
## [1] 0.3897485
```

La precisión de este modelo es del **53,57 %** aproximadamente.

6.6.4. Conclusión

Haciendo un breve resumen de lo obtenido, tenemos lo siguiente:

- Porcentaje de acierto en la predicción de la efectividad en función del rating: **57,44 %**
- Porcentaje de acierto en la predicción de los efectos secundarios en función del rating: **50,58 %**
- Porcentaje de acierto en la predicción de la efectividad en función del rating ponderado: **53,57 %**
- Porcentaje de acierto en la predicción de los efectos secundarios en función del rating ponderado: **73,21 %**

Como se puede observar, el porcentaje de acierto de la efectividad teniendo en cuenta el rating (dados por los usuarios) y el rating ponderado es muy similar, sin embargo la predicción para los efectos secundarios difiere en un 23% a favor del rating ponderado. Esto es debido a que en el rating ponderado se ha tenido más en cuenta el impacto de los efectos secundarios y esto hace que la predicción de dichos efectos secundarios sea más precisa.

6.6.5. Predicción del rating en función del comentario de beneficios

En esta sección se va a realizar una predicción de la efectividad en base a los comentarios aportados por los usuarios (benefitsReview).

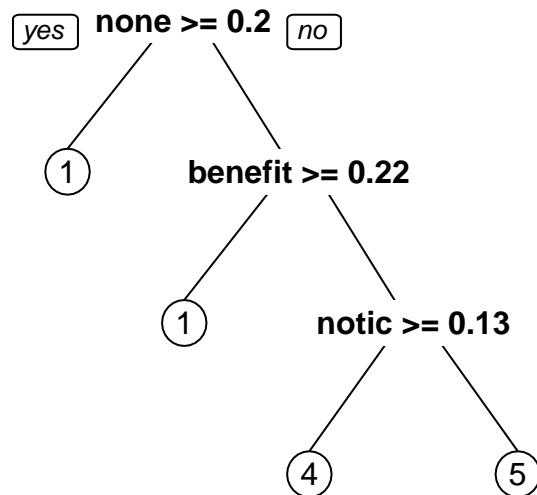
```
library(dplyr)
library(tm)
library(SnowballC)

datos_train = read.csv("datos/datos_train_preprocesado.csv", stringsAsFactors = F)
datos_test = read.csv("datos/datos_test_preprocesado.csv", stringsAsFactors = F)
datos_total <- bind_rows(datos_train, datos_test)
corpus = Corpus(VectorSource(datos_total$benefits_preprocesado))

# Creamos matriz de términos con las palabras de los comentarios. Entonces cada fila, va a estar formada por
# las palabras de un comentario.
tdm <- tm::DocumentTermMatrix(corpus)
tdm.tfidf <- tm::weightTfIdf(tdm)
reviews = as.data.frame(cbind(datos_total$effectivenessNumber, as.matrix(tdm.tfidf)))
preparado_train <- reviews[1:1500,]
preparado_test <- reviews[-(1:1000),]
```

A continuación se va a generar el modelo utilizando como variable a predecir el effectivenessNumber con el resto de palabras que se ha almacenado en preparado_train.

```
reviews.tree = rpart(V1 ~ ., method = "class", data = preparado_train );
prp(reviews.tree)
```



En realidad, es lógico que si utilizamos un conjunto de datos mayor, sobretodo en el caso de los textos, que

haga predicciones muy muy restringidas, ya que al haber tanta cantidad de palabras tiene que generalizar un montón y al final, esa generalización se reduce a utilizar pocas palabras para clasificar en las distintas etiquetas. Por ello, hemos ido probando distintos tamaños hasta dar con uno que visualmente nos compense. Pero obviamente, la importancia no está en que visualmente nos parezca un árbol representativo y que de buenos resultados, sino en que a la hora de predecir el conjunto de test, nos de resultado realmente representativos.

Por ello vamos a hacer predict con el árbol que hemos generado anteriormente. Simplemente le pasamos a la función predict de la librería `rattle` el árbol que hemos entrenado.

```
library(rattle)
pred <- predict(reviews.tree, newdata=preparado_test, type="class")
table(pred)
```

```
## pred
##    1    2    3    4    5
## 224    0    0   112 2802

table(preparado_test$V1)

##
##    1    2    3    4    5
## 244  205  438  933 1318

# Matriz de confusión
tabla <- table(preparado_test$V1, pred, useNA="ifany",
dnn=c("Real", "Predicho" ))
tabla
```

```
##      Predicho
## Real    1    2    3    4    5
##   1    116    0    0    3 125
##   2     32    0    0   15 158
##   3     20    0    0   21 397
##   4     27    0    0   45 861
##   5     29    0    0   28 1261
```

Obtenemos el resultado

```
sum(diag(tabla))/nrow(datos_test)
```

```
## [1] 1.375242
```

Con el fin de mejorar esta predicción, se puede seguir la idea del bagging de combinar muchos métodos sencillos, como se verá en el método de bosques aleatorios.

7. Árboles de decisión y Clasificación

7.1. Naive Bayes

En esta sección vamos a aplicar un clasificador basado en el teorema de Bayes, denominado *Naive Bayes*. Este es un modelo predictivo, por lo que tendremos la posibilidad de deducir las clases de los items del conjunto de test por medio de un clasificador que entrenemos con esta técnica.

Hemos decidido realizar esta técnica para barajar la posible componente aleatoria que puedan tener las clases de los medicamentos. Tal y como hemos visto en clase, esta técnica es útil cuando tenemos situaciones en las que, con el mismo contenido en las variables, tenemos asociado una etiqueta diferente, y este hecho, es muy propicio a darse en nuestro conjunto de datos (para las mismas características, un usuario puede asignar un nivel 4 de efectos secundarios, mientras otro un nivel 5, puesto que hay un gran factor de subjetividad). Además, nos viene perfecto ya que esta técnica funciona con variables discretas para poder crear el clasificador.

Vamos a realizar esta técnica directamente con el texto, porque nos parece más interesante en nuestro problema. Sin embargo, no debería ser muy complicado adaptarlo a variables numéricas (incluso más sencillo del que hemos realizado), ya que nuestro dataset solamente tiene variables discretas, por tanto, preferimos centrarnos directamente en el texto.

```
datos_train <- read.table("datos/datos_train_preprocesado.csv", sep=",",
                           comment.char="", quote = "\"", header=TRUE)
datos_test <- read.table("datos/datos_test_preprocesado.csv", sep=",",
                         comment.char="", quote = "\"", header=TRUE)
# Establecemos la semilla
set.seed(3)
```

El algoritmo realizado es el siguiente:

```
algoritmo.naiveBayes <- function(texto_train, texto_test, label_train, label_test){

  # Calculamos los corpus de los comentarios pasados
  corpus_train = tm::VCorpus(tm::VectorSource(texto_train))
  corpus_test = tm::VCorpus(tm::VectorSource(texto_test))

  # Obtenemos la matriz de términos de esos comentarios
  dtm_train <- DocumentTermMatrix(corpus_train)
  dtm_test <- DocumentTermMatrix(corpus_test)

  # Obtenemos los términos más frecuentes
  # (Aquellos en los que se repiten en más de 5 textos)
  freq.words <- findFreqTerms(dtm_train, 5)

  # Recalculamos la matriz de términos en función de este concepto (Sparsity)
  dtm_freq_train <- DocumentTermMatrix(corpus_train, control=list(dictionary = freq.words))
  dtm_freq_test <- DocumentTermMatrix(corpus_test, control=list(dictionary = freq.words))

  # Función para convertir el peso de los términos en valores binarios:
  # yes=presente, no=ausente
  convert_counts <- function(x) {
    x <- ifelse(x > 0, "Yes", "No")
  }

  # Obtenemos matriz de términos con valores binarios en lugar de pesos (continuos)
  trainNB <- apply(dtm_freq_train, MARGIN = 2, convert_counts)
```

```

testNB <- apply(dtm_freq_test, MARGIN = 2, convert_counts)

# Entrenamos el clasificador con el conjunto de entrenamiento
classifier <- naiveBayes(trainNB, as.factor(label_train), laplace = 1)

# Hacemos las predicciones sobre el conjunto de test y calculamos los errores que tienen
pred_test <- predict(classifier, newdata=testNB)
pred_train <- predict(classifier, newdata=trainNB)
Etest <- mean(pred_test!=label_test)
Etrain <- mean(pred_train!=label_train)

cat("-----\n")
cat("**MATRIZ DE CONFUSIÓN TEST**\n")

print(table(pred=pred_test,real=label_test))

cat(paste("Error de Test: ",Etest*100, "%\n\n"))

cat("-----\n")
cat("**MATRIZ DE CONFUSIÓN TRAIN**\n")

print(table(pred=pred_train,real=label_train))

cat(paste("Error de Train: ",Etrain*100, "%\n"))
cat("-----")
}

```

Lo primero que hacemos es calcular los corpus correspondientes, tanto para el conjunto de entrenamiento como para el de prueba. Acto seguido nos generamos nuestras matrices de términos del mismo modo que en otras técnicas.

En un intento de hacer esta matriz más pequeña para que fuese algo más eficiente, tratamos de encontrar aquellos términos que se encuentran en menos de 5 documentos (comentarios) en la totalidad de nuestro dataset de entrenamiento. Entonces, recalcularon la matriz de términos despreciando estos términos. En otras palabras, aplicamos la técnica de Sparsity. Es cierto que esto es una parte que podría ir perfectamente en el procesamiento, tal y como se ha hablado en el apartado correspondiente. No obstante, no vimos conveniente realizarla en el resto de técnicas salvo en esta. Por lo que lo hacemos de forma específica y exclusiva para este problema.

Como ya hemos mencionado antes, *Naive Bayes* es un clasificador que funciona bien con variables discretas. Por ello, es importante discretizar nuestras matrices de términos para este problema. Tenemos por cada documento un vector de pesos que varía dependiendo de las frecuencias de cada término en cada documento. Entonces, tal y como se menciona en la teoría de esta asignatura, decidimos discretizar estos valores bajo el criterio de si los términos se encuentran presentes (valor “yes”) o ausentes (valor “no”) en cada documento.

En otras palabras, transformamos nuestras matrices de términos en matrices binarias basándonos en este concepto de presente o ausente. Con esto ya tendríamos los comentarios de nuestro dataset preparados para poder crear el clasificador basado en el teorema de Bayes. Mencionar que las etiquetas pasadas deben de ser de tipo “factor”, de lo contrario no funcionará (se realiza la conversión dentro de la función por si no estuviera en este tipo). Este fue un error que nos retrasó mucho en el desarrollo.

Después, simplemente tendríamos que realizar las predicciones correspondientes, obtener las matrices de confusión y el error derivado de las mismas.

Ha sido realizado en una función de tal forma que podamos pasarle cualquier texto y cualquier etiqueta de nuestro dataset que sea apta para clasificar. Ahora vamos a ver los resultados que nos proporciona esta

técnica para los comentarios de beneficios y efectos secundarios.

Vamos a deducir el **el nivel de efectividad en función de los comentarios sobre los beneficios.**

```
algoritmo.naiveBayes(datos_train$benefits_preprocesado,datos_test$benefits_preprocesado,
                      datos_train$effectivenessNumber,datos_test$effectivenessNumber)
```

```
-----
**MATRIZ DE CONFUSIÓN TEST**
real
pred   1    2    3    4    5
  1  44  13   7  12   6
  2   3   3   4   7   1
  3   6   8  21  29  27
  4  14  32  80 150 148
  5  14  20  45 111 229
Error de Test:  56.7698259187621 %

-----
**MATRIZ DE CONFUSIÓN TRAIN**
real
pred   1    2    3    4    5
  1 196  20  13  26  34
  2   1  55   1   7   9
  3   3  23 198  41  55
  4  25  60 131 665 359
  5  22  28  72 187 873
Error de Train: 35.985824742268 %
```

Figura 42: Matriz de confusión y error para la predicción del nivel de efectividad de los medicamentos a partir de comentarios sobre beneficios.

Como podemos observar en los resultados obtenidos, tenemos un error de 35.9858 % para el conjunto de entrenamiento y 56.7698 % para el conjunto de test. Los resultados pueden parecer a priori nefastos, ya que podemos decir que se equivoca en un poco más de la mitad de las ocasiones. No obstante, debemos decir en defensa de esta técnica que debe poner una nota de efectividad (del 1 al 5) del medicamento en función del comentario que haya realizado esa persona sobre el mismo. En otras palabras, de 5 posibles valores de nota que puede poner debe decidir una.

Consideramos que es más interesante fijarse en la matriz de confusión que en el error que obtiene. La diagonal principal corresponde con los aciertos obtenidos por el clasificador. Lo interesante es que hay también valores altos cerca de esta diagonal. ¿Qué significa esto?. Quiere decir que el clasificador estima bastante bien las notas incluso cuando se equivoca. Cuando la nota real del medicamento es un 5, el clasificador suele equivocarse (cuando se equivoca) con una mayor frecuencia poniéndole un 4, no con un 1, por ejemplo, el cual es un error más pronunciado.

Otro ejemplo, solo se ha equivocado una vez poniendo un 2, cuando el nivel de efectividad del medicamento era un 5 en realidad. También hay que tener en cuenta que deducir el nivel de efectividad exacto del medicamento en función del comentario de una persona puede ser una tarea difícil incluso para una persona, ya que podemos deducir que tiene una efectividad alta o baja en función del comentario, pero saber el valor exacto de efectividad es otra historia.

Ahora, vamos a deducir el **nivel de efectividad en función de los comentarios sobre los efectos secundarios.**

```
algoritmo.naiveBayes(datos_train$effects_preprocesado,datos_test$effects_preprocesado,
                      datos_train$effectivenessNumber,datos_test$effectivenessNumber)
```

Vemos que las predicciones son parecidas, aunque con un poco más de error tanto en el test como el train. Parece ser que los efectos secundarios que refleja la gente en los comentarios son menos útiles para deducir el nivel de efectividad del medicamento. En realidad le vemos mucho sentido a este hecho, porque que un

```
-----  
**MATRIZ DE CONFUSIÓN TEST**  
    real  
pred   1   2   3   4   5  
  1  25  14  16  26  21  
  2   3   5   7  10  13  
  3   4   9  17  24  20  
  4  18  17  31  58  62  
  5  31  31  86 191 295  
Error de Test: 61.3152804642166 %  
  
-----  
**MATRIZ DE CONFUSIÓN TRAIN**  
    real  
pred   1   2   3   4   5  
  1 103   2  20  54  61  
  2   1  58   6  21  37  
  3  13  11 145  27  55  
  4  25  33  48 360  91  
  5 105  82 196 464 1086  
Error de Train: 43.5567010309278 %  
-----
```

Figura 43: Matriz de confusión y error para la predicción del nivel de efectividad de los medicamentos a partir de comentarios sobre efectos secundarios.

medicamento tenga unos efectos secundarios leves o graves no quiere decir en principio que sea efectivo o no. Por lo que podemos llegar a la conclusión de que este tipo de comentarios aporta un conocimiento menos útil al clasificador para poder deducir este tipo de etiqueta en concreto.

Aun así, vemos que no suele tener muchos errores “graves” ya que hay valores más altos cerca de la diagonal principal de la matriz de confusión para el conjunto de test, de la misma forma que en el caso anterior.

Ahora, vamos a deducir el ratingLabel en función de los comentarios sobre los beneficios.

```
algoritmo.naiveBayes(datos_train$benefits_preprocesado,datos_test$benefits_preprocesado,
                      datos_train$ratingLabel,datos_test$ratingLabel)
```

```
-----
**MATRIZ DE CONFUSIÓN TEST**
real
pred   0   1
      0 163 215
      1  77 579
Error de Test: 28.2398452611219 %

-----
**MATRIZ DE CONFUSIÓN TRAIN**
real
pred   0   1
      0 531 587
      1 129 1857
Error de Train: 23.0670103092784 %
-----
```

Figura 44: Matriz de confusión y error para la predicción del ratingLabel de los medicamentos a partir de comentarios sobre beneficios.

Vemos que los errores son más bajos que en los dos casos anteriores, esto es algo que consideramos normal, ya que ahora el clasificador solo tiene que decir si el rating que pone esa persona está por encima (1) o por debajo (0) del 5. Es como decir si un medicamento es *bueno* o *malo* para la persona que lo prueba en función del comentario que escribe.

Otro dato interesante es que el clasificador comete muchos más falsos negativos que positivos. En otras palabras, tiende a pensar que las personas establecen ratings más bajos de lo que realmente son, en general.

Vamos a analizar el ratingLabel en función de los comentarios sobre efectos secundarios.

```
algoritmo.naiveBayes(datos_train$effects_preprocesado,datos_test$effects_preprocesado,
                      datos_train$ratingLabel,datos_test$ratingLabel)
```

```
-----
**MATRIZ DE CONFUSIÓN TEST**
real
pred   0   1
      0 116 108
      1 124 686
Error de Test: 22.4371373307544 %

-----
**MATRIZ DE CONFUSIÓN TRAIN**
real
pred   0   1
      0 346 283
      1 314 2161
Error de Train: 19.2332474226804 %
-----
```

Figura 45: Matriz de confusión y error para la predicción del ratingLabel de los medicamentos a partir de comentarios sobre efectos secundarios.

Vemos que los errores son más bajos aún, con un 22.4371 % en el error de test. Para el ratingLabel ocurre al contrario, generamos un mejor modelo basándonos en los comentarios sobre efectos secundarios que en los beneficios. Suponemos que el rating lo establece las personas, no solo en función de su efectividad, sino de las sensaciones que el medicamento transmite en su bienestar general. Por lo que no nos parece un resultado descabellado.

7.2. SVM

En esta sección vamos a hablar sobre la técnica de Support Vector Machine (SVM). Principalmente presenta el inconveniente de que es poco escalable y costosa (computacionalmente y en tiempo), por lo que no vamos a hacer un análisis muy detallado del mismo.

Inicialmente, tratamos de utilizarlo con valores numéricos de nuestro dataset. El problema es que solo tenemos valores discretos y categóricos, por lo que esta técnica no funcionaba de forma correcta a la hora de calcular distancias. Tratar de hacer continuo un valor discreto es una pérdida de tiempo, ya que no podemos basarnos en nada para decidir el valor exacto que alcanzaría un valor discreto en un espacio continuo.

Tras darnos cuenta de esto, nos pasamos directamente al tratamiento del texto. Los pasos que llevamos a cabo fueron los siguientes:

```
# Creamos la matriz de términos
dtm_train <- create_matrix(datos_train$benefits_preprocesado)

# Creamos un contenedor a partir de la matriz de términos
contenedor <- create_container(dtm_train,datos_train$ratingLabel,
                                trainSize=1:length(dtm_train),virgin=FALSE)

# Entrenamos el modelo SVM con los parámetros que queramos
modelo_svm <- train_model(contenedor,"SVM",kernel="radial")

# Creamos la matriz de términos para el conjunto de test
dtm_test <- create_matrix(as.list(datos_test$benefits_preprocesado),
                           originalMatrix = dtm_train)

#Obtenemos el contenedor correspondiente a la matriz de términos del conjunto de Test
contenedor_test <- create_container(dtm_test,labels=as.factor(datos_test$ratingLabel),
                                      testSize=1:length(datos_test$ratingLabel),virgin=FALSE)

resultados_svm <- classify_model(contenedor_test,modelo_svm)
resultados_svm
```

Destacar de este proceso que hacemos uso de la matriz de términos como viene siendo común a la hora de manipular texto. Utilizamos el kernel de tipo radial y la totalidad del conjunto de entrenamiento para llevar a cabo la construcción del modelo.

Pero nos daba un error a la hora de ejecutar este código. Esto retrasó mucho el avance de la práctica ya que no conseguíamos darnos cuenta de que es lo que ocurría. Finalmente, parece ser una incompatibilidad de la librería en `create_matrix`. Por lo que tuvimos que cambiarla con la ayuda de `trace`. En el código hay un comentario que especifica el cambio a realizar en caso de tener este problema.

Posteriormente, tratamos de deducir el `ratingLabel` a partir de los comentarios sobre los beneficios de los medicamentos. Obteniendo finalmente los siguientes resultados:

Como podemos ver, el clasificador no funciona bien. Etiqueta todo a 0. Creemos que el error viene dado debido a la disposición de los términos (igual que vimos en el clustering). Al tener una nube de puntos tan densa y tan poco disjunta, esto hace que no haya una función aceptable para poder separarlas.

Por otro lado, SVM no funciona bien con texto a no ser que tengas las mismas palabras tanto en el test como en el train, cosa que no ocurre en nuestro caso. Esto hace que SVM no tenga todo lo necesario para generar una buena función, ya que nos impide predecir para futuros casos (puesto que tendrían una alta probabilidad de contener palabras no contempladas en el entrenamiento). Cosa que sí ocurre, por ejemplo, en este tutorial <https://www.svm-tutorial.com/2014/11/svm-classify-text-r/>.

En definitiva, decidimos no perder más tiempo con esta técnica, ya que parece no ser adecuada para nuestro

	SVM_LABEL <fctr>	SVM_PROB <dbl>
1	0	0.7585066
2	0	0.7585066
3	0	0.7585066
4	0	0.7585300
5	0	0.7585066
6	0	0.7585066
7	0	0.7585066
8	0	0.7585066
9	0	0.7585066
10	0	0.7585066
11	0	0.7585066
12	0	0.7585066
13	0	0.7585066
14	0	0.7585066
15	0	0.7585066
16	0	0.7585300
17	0	0.7585066
18	0	0.7585066
19	0	0.7585066
20	0	0.7585066
21	0	0.7585066
22	0	0.7585066
23	0	0.7585066
24	0	0.7585066
25	0	0.7585066
26	0	0.7585066
27	0	0.7585066
28	0	0.7585066
29	0	0.7585066
30	0	0.7585066
31	0	0.7585066
32	0	0.7585066
33	0	0.7585066
34	0	0.7585066
35	0	0.7585066
36	0	0.7585066
37	0	0.7585066
38	0	0.7585066
39	0	0.7585300
40	0	0.7585066
41	0	0.7585066
42	0	0.7585066
43	0	0.7585300
44	0	0.7585066
45	0	0.7585066
46	0	0.7585066
47	0	0.7585066
48	0	0.7585300

1-48 of 1,034 rows

Figura 46: Salida de SVM para el ratingLabel a partir de los comentarios de texto sobre los beneficios.

dataset y seguir avanzando en el resto que proponemos con esta práctica.

8. Regresión

En este apartado se va hacer uso de la técnica de **regresión**, con el objetivo de describir dependencias significativas entre las variables incluidas en la base de datos. La regresión es un modelo de predicción de variables continuas, en donde las variables independientes también son continuas o al menos numéricas. Dicho proceso viene caracterizado por aprender una función que aplica un conjunto de atributos X_1X_n en otro atributo Y . En nuestro caso, vamos a hacer uso de las columnas:

- **ratingLabel**: etiqueta de 0 ó 1, que contempla la opinión del paciente sobre el medicamento si es favorable (1) o no es favorable (0).
- **effectivenessNumber**: clasificación de la efectividad del medicamento según el paciente (5 posibles valores), en donde el 1 significa que es ineficaz y el 5 que es altamente eficaz.
- **sideEffectsInverse**: clasificación de los efectos secundarios del medicamento según el paciente (5 posibles valores), en donde el 1 significa que tiene efectos secundarios extremadamente graves y el 5 que es sin efectos secundarios.

effectivenessNumber	weightedRating	ratingLabel	sideEffectsInverse
5	10	0	4
5	8	0	2
5	10	1	5
2	6	0	4
2	4	0	2
1	2	0	2
5	10	1	4
4	8	1	5
5	10	1	5
1	2	0	1
4	8	1	3

Figura 47: Visualización de las columnas a las que aplica regresión

Como se observa en la gráfica 47, se va hacer uso de variables discretas. El objetivo de aplicar regresión, es obtener la curva ROC, la matriz de confusión y el error en el conjunto test, para obtener así la precisión de nuestro modelo y saber según el paciente qué medicamentos son favorables o no, tienen efectos secundarios o son efectivos.

Como se ha comentado, se va hacer uso de la matriz de confusión, la cual es una herramienta que permite la visualización del desempeño de un algoritmo, en donde cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real. Una de sus ventajas es que permite ver si la predicción falla o no.

	N (modelo)	S (modelo)
n (real)	Negativos Reales	Falsos Positivos
p (real)	Falsos Negativos	Positivos reales

Figura 48: Matriz de confusión para clasificador binario.

De esta forma, se observa como en la figura 48, la diagonal principal contiene la suma de todas las predicciones correctas (el modelo dice "S" y acierta, tiene efectos secundarios, o dice "N" y acierta también, no tiene

efectos secundarios). La otra diagonal refleja los errores del clasificador: los falsos positivos o “true positives” (dice que tiene efectos secundarios “S”, pero en realidad no tiene “n”), o los falsos negativos o “false negatives” (dice que no tiene efectos secundarios “N”, pero en realidad los tiene “p”). Además, otra de las medidas a usar, será la curva ROC, para caracterizar el comportamiento de la predicción.

Pero previo a la realización de las técnicas, se ha optado por eliminar los fármacos repetidos, para así obtener una predicción más acertada. Para dicha eliminación, se han cogido todos los medicamentos duplicados y eliminados los innecesarios. Así que si tenemos 5 filas de un mismo medicamento, se eliminan las 4 consecutivas al primero.

8.1. Regresión Lineal

La regresión lineal simple consiste en generar un modelo de regresión (ecuación de una recta) que permita explicar la relación lineal que existe entre dos variables. A la variable dependiente o respuesta se le identifica como Y y a la variable predictora o independiente como X . Con el comando `lm()` podemos ajustar el modelo. Algunos parámetros importantes de esta función son:

```
lm(formula, data, subset, weights)
```

- *fórmula*: definimos el modelo como: $Y \sim X$.
 - Regresión Múltiple: $y \sim X_1 + X_2 + X_n$.
 - Regresión Polinómica: $y \sim poly(x = X, degree = k)$
 - Interacción de variables: $y \sim X_1 \cdot X_2$. Si sólo queremos el término de interacción: $X1 : X2$
- *data* : especificamos el dataset a utilizar
- *subset* : si dividimos la muestra en training y testing, podemos indicar el subconjunto de entrenamiento con un vector que indique sus números de fila.

8.1.1. Regresión Lineal Simple

Se pretende predecir el valor del rating (etiqueta 0-1) en función de la puntuación de los medicamentos y, por otro lado para los efectos secundarios del mismo. Para ello se va a emplear la función `lm()`, la cual genera un modelo de regresión lineal por mínimos cuadrados en el que la variable respuesta es `ratingLabel` y el predictor `sideEffectsInverse`, y por otro lado, la variable respuesta será `ratingLabel` y el predictor `effectivenessNumber`.

El modelo de regresión lineal simple se describe de acuerdo a la siguiente ecuación:

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$

Siendo β_0 la ordenada en el origen, β_1 la pendiente y ϵ el error aleatorio. Este último representa la diferencia entre el valor ajustado por la recta y el valor real y recoge el efecto de todas aquellas variables que influyen en Y pero que no se incluyen en el modelo como predictores. Al error aleatorio también se le conoce como residuo.

A continuación, se realiza una función con la cual obtener los errores dentro y fuera de la muestra, y la matriz de confusión.

```
# Función que calcula los errores y E_test para regresión logística
errores_regresion_lineal <- function(m){
  probTr = predict(m, type="response")
  probTst = predict(m, data.frame(data_test_procesado), type="response")

  predTst = rep(0, length(probTst)) # predicciones por defecto 0
  predTst[probTst >= 0.5] = 1 # >= 0.5 clase 1

  predTr = rep(0, length(probTr)) # predicciones por defecto 0
```

```

predTr[predTr >= 0.5] = 1 # >= 0.5 clase 1 # Para el calculo del Etest

# Calculamos Etest y mostramos la matriz de confusión
print(table(pred=predTst, real=data_test_procesado$ratingLabel))

Etrain = mean(predTr != data_train_procesado$ratingLabel)
Etest = mean(predTst != data_test_procesado$ratingLabel)

# Devolvemos el error para el conjunto train y test
list(Etrain=Etrain*100, Etest=Etest*100)
}

```

Primero vamos a obtener un modelo en donde la variable respuesta es ratingLabel y el predictor sideEffectsInverse. Con el fin de obtener, que medicamentos favorables tienen efectos secundarios. Por lo que tendremos que hacer caso a la otra diagonal de la matriz de confusión.

```

# Creamos el modelo para ratingLabel ~ sideEffectsInverse
lm_effects = lm(data = data_train_procesado, formula = ratingLabel ~ sideEffectsInverse)
summary(lm_effects)

```

```

##
## Call:
## lm(formula = ratingLabel ~ sideEffectsInverse, data = data_train_procesado)
##
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -1.0164 -0.0164 -0.0164  0.1615  0.6953 
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)    
## (Intercept)  0.12683   0.05284   2.40   0.0167 *  
## sideEffectsInverse 0.17792   0.01311  13.57  <2e-16 *** 
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3328 on 500 degrees of freedom
## Multiple R-squared:  0.2691, Adjusted R-squared:  0.2676 
## F-statistic: 184.1 on 1 and 500 DF,  p-value: < 2.2e-16

# Evaluamos el modelo
errores_regresion_lineal(lm_effects)

```

```

##      real
## pred  0   1
##     0 47  11
##     1 27 228

## $Etrain
## [1] 12.3506
##
## $Etest
## [1] 12.14058

```

Para el modelo generado, tanto la ordenada en el origen como la pendiente son significativas (p-values < 0.05). El valor de R^2 indica que el modelo calculado explica el 26.91% de la variabilidad presente en la variable respuesta (ratingLabel) mediante la variable independiente (sideEffectsInverse). Como se observa

se obtiene que hay 11 falsos positivos, es decir, medicamentos que se dicen que tienen efectos secundarios pero no los tienen. Y existen 27 falsos negativos, es decir, medicamentos que se dicen que no tienen efectos secundarios pero que los tienen. Dichos resultados son los obtenidos según los comentarios de los pacientes. Como podemos apreciar, el error del test nos devuelve un valor de 12.14058, un error aceptable teniendo en cuenta los resultados obtenidos en la matriz de confusión. Pero posiblemente dicho error se pueda reducir aplicando sobre los datos otros modelos que veremos más adelante.

A continuación, vamos a crear el modelo para la variable respuesta ratingLabel y el predictor effectivenessNumber. Con el fin de obtener, que medicamentos favorables son efectivos. Por lo que tendremos que hacer caso a la otra diagonal de la matriz de confusión.

```
# Creamos el modelo para ratingLabel ~ effectivenessNumber
lm_effectiveness = lm(data = data_train_procesado, formula = ratingLabel ~ effectivenessNumber)
summary(lm_effectiveness)
```

```
##
## Call:
## lm(formula = ratingLabel ~ effectivenessNumber, data = data_train_procesado)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.01938 -0.01938 -0.01938  0.17667  0.76480
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)             0.03915   0.04733   0.827   0.409
## effectivenessNumber    0.19605   0.01145  17.129 <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.309 on 500 degrees of freedom
## Multiple R-squared:  0.3698, Adjusted R-squared:  0.3685
## F-statistic: 293.4 on 1 and 500 DF,  p-value: < 2.2e-16
```

```
# Evaluamos el modelo
errores_regresion_lineal(lm_effectiveness)
```

```
##      real
## pred  0   1
##     0 39   3
##     1 35 236
##
## $Etrain
## [1] 10.15936
##
## $Etest
## [1] 12.14058
```

Para el modelo generado, tanto la ordenada en el origen como la pendiente son significativas ($p\text{-values} < 0.05$). El valor de R^2 indica que el modelo calculado explica el 36.98 % de la variabilidad presente en la variable respuesta (ratingLabel) mediante la variable independiente (effectivenessNumber). Como se observa se obtiene que hay 3 falsos positivos, es decir, medicamentos que se dicen que son efectivos pero que no lo son. Y existen 35 falsos negativos, es decir, medicamentos que se dicen que no son efectivos pero los son. Como podemos apreciar, el error del test nos devuelve un valor de 12.14058, un error aceptable teniendo en cuenta los resultados obtenidos en la matriz de confusión. Pero posiblemente dicho error se pueda reducir aplicando sobre los datos otros modelos que veremos más adelante.

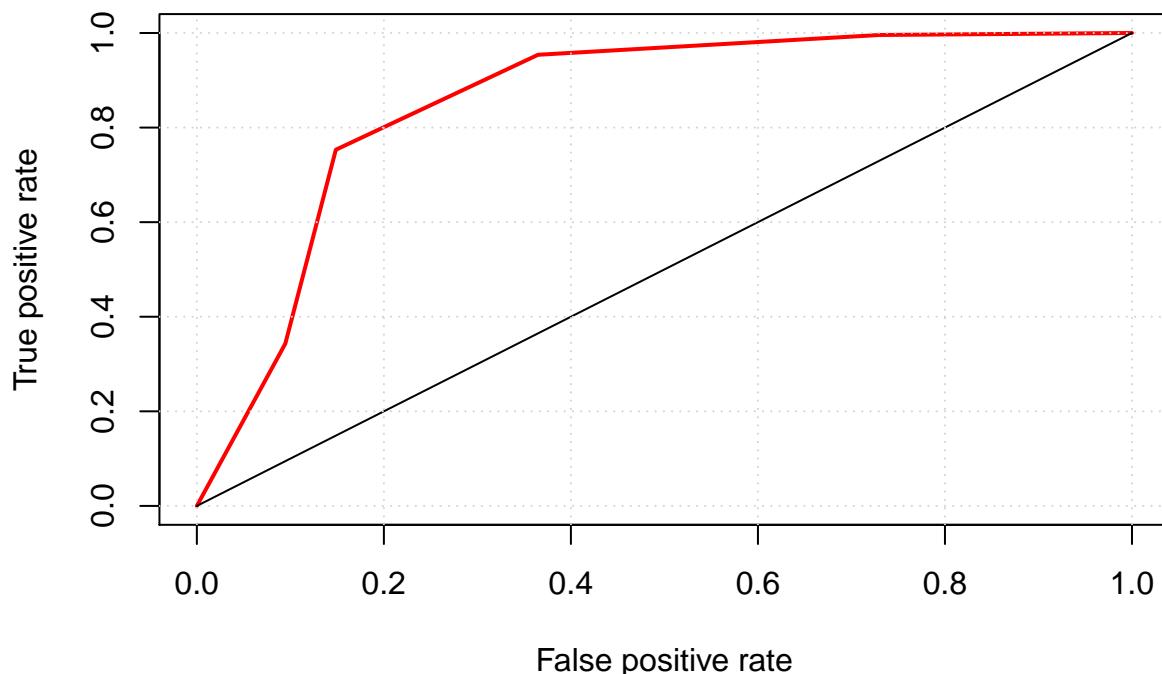
Una vez que hemos obtenidos las predicciones para nuestro modelo, vamos a obtener las probabilidades para

el mismo con el fin de obtener la curva ROC.

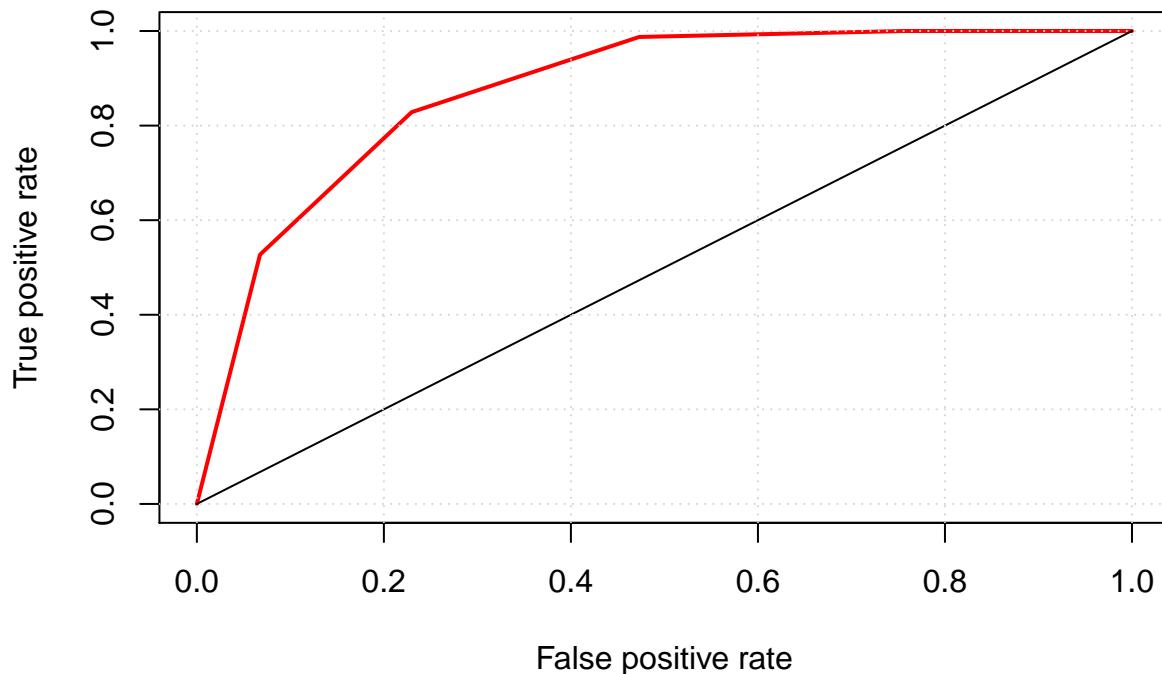
```
# Función que dibuja una curva ROC
plotROC <- function(modelo, etiq_real, adicionar=FALSE, color="red") {

  # Realizamos la predicción con la función de ROCR
  pred <- prediction(modelo, etiq_real)
  perf <- performance(pred, "tpr", "fpr")
  plot(perf, col=color, add=adicionar,
       main="Curva ROC - Regresión Lineal - Efectos secundarios", lwd = 2)
  segments(0, 0, 1, 1, col='black')
  grid()
}
```

Curva ROC – Regresión Lineal – Efectos secundarios



Curva ROC – Regresión Lineal – Efectos secundarios



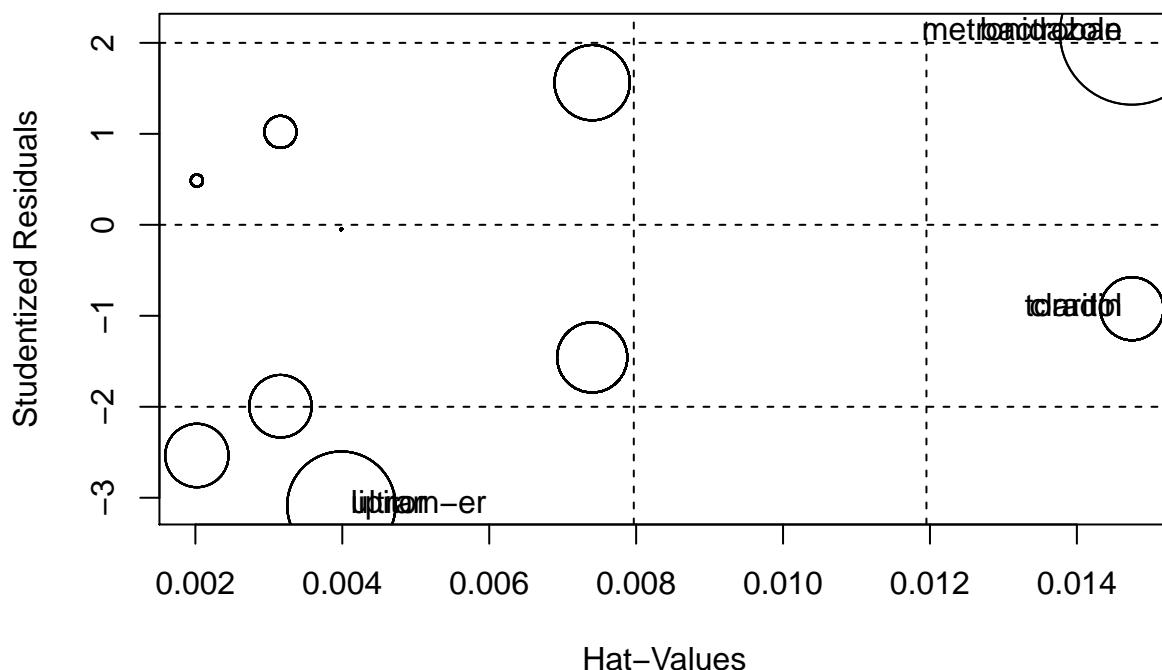
8.1.1.1. Casos atípicos

A continuación para el modelo lineal simple, vamos a obtener los casos atípicos, una forma rápida de identificarlos es usando la función `influencePlot()` del paquete `car`. Esta función produce un gráfico que señala a los casos atípicos influyentes.

Una de las fuentes de violaciones de los supuestos en el modelado lineal es la presencia de casos atípicos. Un caso atípico es aquel que, dados ciertos valores de $x \sim 1, x \sim 2, x \sim n$ tiene valores de y muy diferentes a los demás y por lo tanto producen un residuo muy alto. Estos casos atípicos pueden tener gran influencia sobre el modelo, ya que el criterio de mínimos cuadrados buscará minimizar el error y cambiará la pendiente para dar cuenta de estos casos.

Primero vamos a obtener los casos atípicos para el modelo de los efectos secundarios del medicamento.

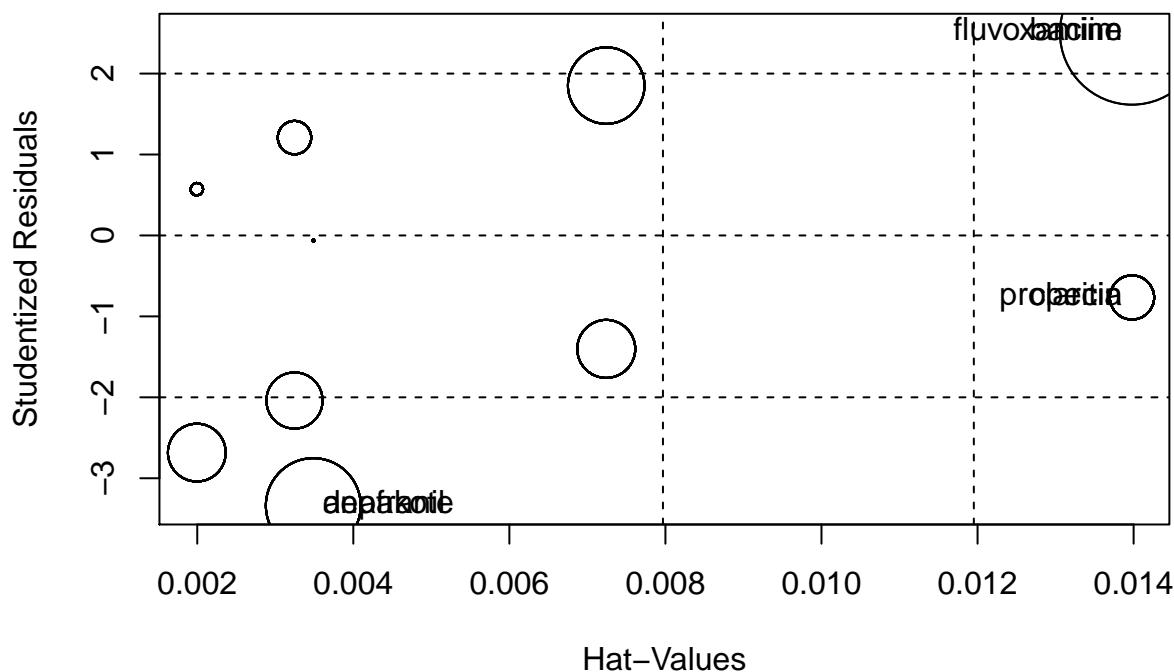
```
# Detección y visualización de observaciones influyentes para effects
influencePlot(lm_effects, xlab="Hat-Values", ylab="Studentized Residuals")
```



```
##           StudRes      Hat     CookD
## claritin -0.9223289 0.014750230 0.00636977
## lipitor   -3.0860011 0.003986799 0.01874047
## ultram-er -3.0860011 0.003986799 0.01874047
## toradol   -0.9223289 0.014750230 0.00636977
## metronidazole 2.1118521 0.014750230 0.03315542
## bactroban  2.1118521 0.014750230 0.03315542
```

Para dicho modelo, los medicamentos *claritin*, *lipitor*, *ultram-er*, *metronidazole*, *toradol* y *bactroban* aparecen como casos atípicos. A continuación, vamos a obtener los casos atípicos para el modelo de la efectividad del medicamento.

```
# Detección y visualización de observaciones influyentes para effectivenessNumber
influencePlot(lm_effectiveness, xlab="Hat-Values", ylab="Studentized Residuals")
```



```
##           StudRes      Hat     CookD
## propecia -0.7661251 0.013979059 0.004164089
## claritin -0.7661251 0.013979059 0.004164089
## depakote -3.3376924 0.003486567 0.019101083
## anafranil -3.3376924 0.003486567 0.019101083
## fluvoxamine 2.5054155 0.013979059 0.044031318
## baciim    2.5054155 0.013979059 0.044031318
```

Para dicho modelo, los medicamentos *propecia*, *claritin*, *depakote*, *anafranil* y *fluvoxamine* aparecen como casos atípicos.

8.1.1.2. Regresión Robusta

Una alternativa para controlar los casos atípicos es ajustar un modelo lineal robusto. El modelo lineal robusto utiliza criterios diferentes al de los mínimos cuadrados y pondera la influencia de los casos atípicos, por lo que producen coeficientes y errores estándar más confiables. El paquete MASS incluye la función `rlm()`, de sintaxis similar a `lm()` que implementa el método para el ajuste de los coeficientes y cálculo de los errores estándar.

Primero, calcularemos el modelo en donde la variable respuesta es `ratingLabel` y el predictor `sideEffectsInverse`. Con el fin de obtener, qué medicamentos favorables tienen efectos secundarios. Por lo que tendremos que hacer caso a la otra diagonal de la matriz de confusión.

```
# Creamos el modelo para ratingLabel ~ sideEffectsInverse
rlm_effects = MASS::rlm(ratingLabel ~ sideEffectsInverse, data = data_train_procesado)

## Warning in rlm.default(x, y, weights, method = method, wt.method =
## wt.method, : 'rlm' failed to converge in 20 steps
stargazer(lm_effects, rlm_effects, type = "text", model.numbers = FALSE,
           title="Comparación de modelo OLS y Robusto")

##
## Comparación de modelo OLS y Robusto
## =====
```

```

##                                     Dependent variable:
##                                     -----
##                                     ratingLabel
##                                     OLS          robust
##                                     linear
## -----
## sideEffectsInverse           0.178***   0.006***
##                               (0.013)    (0.0004)
## 
## Constant                    0.127**   0.971*** 
##                               (0.053)    (0.002)
## 
## -----
## Observations                 502        502
## R2                          0.269
## Adjusted R2                  0.268
## Residual Std. Error (df = 500) 0.333       0.007
## F Statistic                184.050*** (df = 1; 500)
## =====
## Note:                         *p<0.1; **p<0.05; ***p<0.01

# Creamos el modelo para ratingLabel ~ effectivenessNumber
rlm_effectiveness = MASS::rlm(ratingLabel ~ effectivenessNumber,
                                data = data_train_procesado)
stargazer(lm_effectiveness, rlm_effectiveness, type = "text", model.numbers = FALSE,
          title="Comparación de modelo OLS y Robusto")

## 
## Comparación de modelo OLS y Robusto
## =====
##                                     Dependent variable:
##                                     -----
##                                     ratingLabel
##                                     OLS          robust
##                                     linear
## -----
## effectivenessNumber            0.196***   0.213*** 
##                               (0.011)    (0.008)
## 
## Constant                     0.039       0.006
##                               (0.047)    (0.034)
## 
## -----
## Observations                 502        502
## R2                          0.370
## Adjusted R2                  0.369
## Residual Std. Error (df = 500) 0.309       0.210
## F Statistic                293.392*** (df = 1; 500)
## =====
## Note:                         *p<0.1; **p<0.05; ***p<0.01

```

El modelo robusto aumenta los coeficientes y reduce los errores estándar, aunque no en gran cuantía. En este caso interpretamos que los casos atípicos no son un problema grave.

8.1.2. Regresión Lineal Múltiple

Una vez que hemos la regresión lineal simple, vamos a intentar predecir el valor del ratingLabel en función de la puntuación de los medicamentos y de los efectos secundarios del mismo. Para ello se va a emplear la misma (`lm()`), la cual genera un modelo de regresión lineal por mínimos cuadrados en el que la variable respuesta es ratingLabel y los predictores son sideEffectsInverse y effectivenessNumber. Previamente a la realización del modelo, vamos a comprobar que las variables a usar pueden ser aplicadas juntas con la función `step()`, para así determinar la calidad del modelo.

```
# Creamos el modelo para ratingLabel ~ effectivenessNumber + sideEffectsInverse
lm_multiple <- lm(ratingLabel ~ sideEffectsInverse + effectivenessNumber,
                   data = data_train_procesado)

# https://rpubs.com/Cristina_Gil/Regresion_Lineal_Multiple
# Nos dice si existe alguna variable que estamos usando que no nos hace falta
step(lm_multiple, direction = "both", trace = 1)

## Start: AIC=-1296.57
## ratingLabel ~ sideEffectsInverse + effectivenessNumber
##
##          Df  Sum of Sq    RSS      AIC
## <none>            37.481 -1296.6
## - sideEffectsInverse  1    10.270 47.751 -1177.0
## - effectivenessNumber 1    17.903 55.384 -1102.6

##
## Call:
## lm(formula = ratingLabel ~ sideEffectsInverse + effectivenessNumber,
##      data = data_train_procesado)
##
## Coefficients:
##             (Intercept)  sideEffectsInverse  effectivenessNumber
##                 -0.3362           0.1311            0.1628

# Obtenemos el error d
errorres_regresion_lineal(lm_multiple)

##      real
## pred  0   1
##     0 50   5
##     1 24 234

## $Etrain
## [1] 8.366534
##
## $Etest
## [1] 9.265176
```

Como se aprecia en salida de la función `step(lm_multiple, direction = "both", trace = 1)`, las dos variables deben ser incluidas en el proceso de selección.

Por tanto, se observa que se obtiene que hay 5 falsos positivos, es decir, medicamentos que se dicen que son favorables pero que no lo son. Y existen 24 falsos negativos, es decir, medicamentos que se dicen que no son favorables pero los son. Se debe tener en cuenta que dichos resultados obtenidos han sido según los comentarios de los pacientes. Como podemos apreciar, el error del test nos devuelve un valor de 9.265176, un buen error si tenemos en cuenta, los obtenidos con la regresión simple.

8.2. Regresión Logística

A continuación, vamos a ajustar un modelo con regresión logística binaria sin regularización. De este modo vamos a comprobar, si se puede intentar encontrar un buen modelo lineal para nuestro problema. Además, discutiremos las necesidades de regularización, para ver si de esta forma conseguimos mejorarlo en cuanto a resultados.

Para la obtención de la regresión, vamos hacer uso del comando `glm()`.

8.2.1. Regresión Logística Simple

La regresión logística simple, es un método de regresión que permite estimar la probabilidad de una variable cualitativa binaria en función de una variable cuantitativa. Una de las principales aplicaciones de la regresión logística es la de clasificación binaria, en el que las observaciones se clasifican en un grupo u otro dependiendo del valor que tome la variable empleada como predictor. Es decir para predecir la probabilidad de pertenencia o no a una determinada clase (efectividad - no efectividad).

Para ello, vamos a utilizar la función `glm(..)` que en base al conjunto de muestras de entrenamiento suministrado, es capaz de calcular una probabilidad para cada dato. Si esta probabilidad sobrepasa el valor 0.5, entonces consideramos que dicha muestra pertenece a la clase 1, sin embargo, si su probabilidad es menor que 0.5, entonces consideramos que se encuentra en la clase con etiqueta 0. Una vez tengamos las etiquetas predichas a partir de las probabilidades que calcula el modelo, vamos a calcular el error dentro de la muestra y el error fuera de la muestra. Para ello, vamos a especificar como primer argumento, la variable respuesta denominada `ratingLabel` y a continuación, la variable en la que nos vamos a fijar para ajustar el modelo.

```
# Función que calcula los errores y E_test para regresión logística
errores_regresion_logistica <- function(m){
  probTr = predict(m, type="response")
  probTst = predict(m, data.frame(data_test_procesado), type="response")

  predTst = rep(0, length(probTst)) # predicciones por defecto 0
  predTst[probTst >= 0.5] = 1 # >= 0.5 clase 1

  predTr = rep(0, length(probTr)) # predicciones por defecto 0
  predTr[probTr >= 0.5] = 1 # >= 0.5 clase 1 # Para el calculo del Etest

  print(table(pred=predTst, real=data_test_procesado$ratingLabel)) # Calculamos Etest

  Etrain = mean(predTr != data_train_procesado$ratingLabel)
  Etest = mean(predTst != data_test_procesado$ratingLabel)

  list(Etrain=Etrain*100, Etest=Etest*100)
}
```

Primero vamos a obtener el modelo, que predice la variable `sideEffectsInverse` a partir del `ratingLabel`.

```
# Creamos el modelo para ratingLabel ~ sideEffectsInverse
gml_effects = glm(ratingLabel ~ sideEffectsInverse, family = binomial(logit),
                  data = data_train_procesado)
# Evaluamos el modelo
errores_regresion_logistica(gml_effects)

##      real
## pred   0   1
##     0  47  11
##     1  27 228
```

```
## $Etrain  
## [1] 12.3506  
##  
## $Etest  
## [1] 12.14058
```

Como se observa se obtiene que hay 11 falsos positivos, es decir, medicamentos que se dicen que tienen efectos secundarios pero no los tienen. Y existen 27 falsos negativos, es decir, medicamentos que se dicen que no tienen efectos secundarios pero que los tienen. Dichos resultados son los obtenidos según los comentarios de los pacientes. Como podemos apreciar, el error del test nos devuelve un valor de 12.14058, un error aceptable teniendo en cuenta los resultados obtenidos en la matriz de confusión. Pero se ha comprobado, que se obtiene un mejor error con la regresión lineal múltiple.

A continuación, vamos a crear el modelo para la variable de respuesta ratingLabel y el predictor effectivenessNumber. Con el fin de obtener, que medicamentos favorables son efectivos. Por lo que tendremos que hacer caso a la otra diagonal de la matriz de confusión.

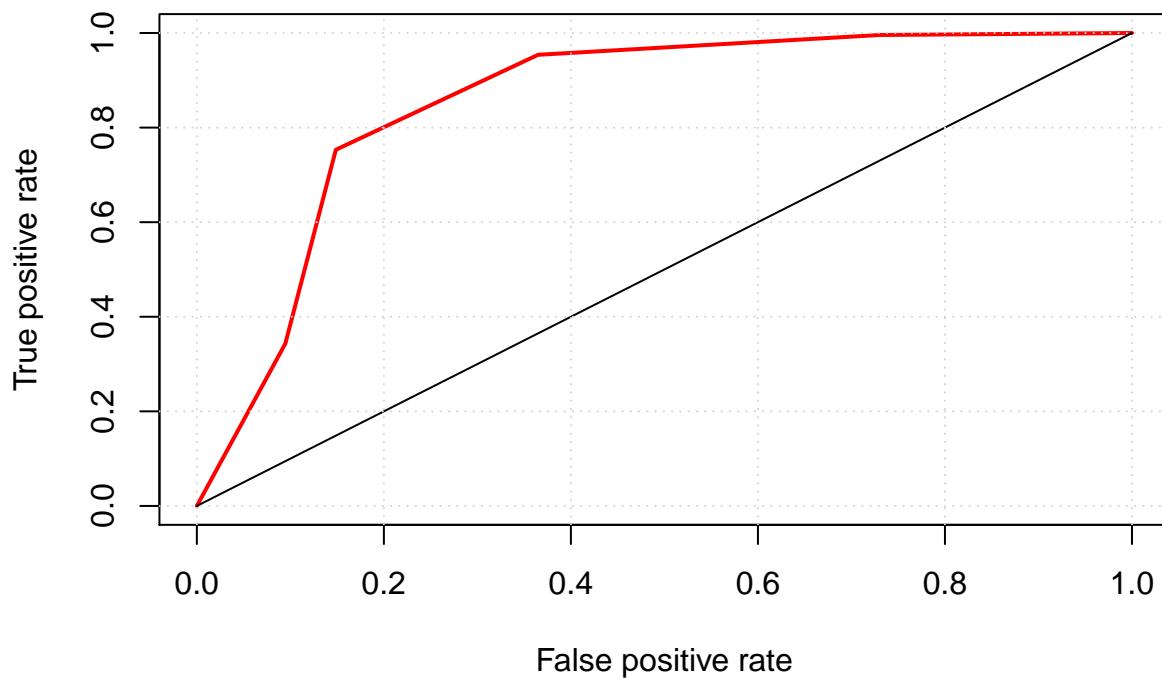
```
# Creamos el modelo para ratingLabel ~ effectivenessNumber  
gml_effectiveness = glm(ratingLabel ~ effectivenessNumber, family = binomial(logit),  
                           data = data_train_procesado)  
# Evaluamos el modelo  
errores_regresion_logistica(gml_effectiveness)
```

```
##      real  
## pred  0   1  
##     0 39   3  
##     1 35 236  
  
## $Etrain  
## [1] 10.15936  
##  
## $Etest  
## [1] 12.14058
```

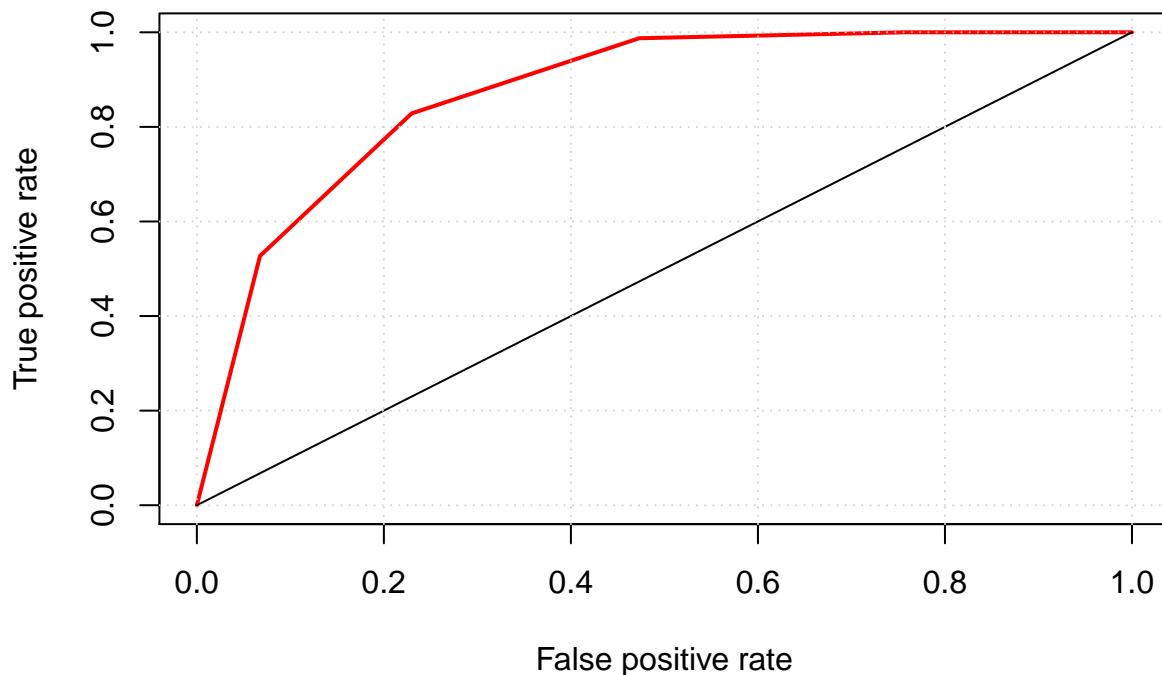
Como se observa se obtiene que hay 3 falsos positivos, es decir, medicamentos que se dicen que son efectivos pero que no lo son. Y existen 35 falsos negativos, es decir, medicamentos que se dicen que no son efectivos pero los son. Como podemos apreciar, el error del test nos devuelve un valor de 12.14058, un error aceptable teniendo en cuenta los resultados obtenidos en la matriz de confusión. Pero posiblemente dicho error se pueda reducir aplicando sobre los datos otros modelos que veremos más adelante.

Una vez que hemos obtenidos las predicciones para nuestro modelo, vamos a obtener la curva ROC.

Curva ROC – Regresión Lineal – Efectos secundarios



Curva ROC – Regresión Lineal – Efectos secundarios



8.2.2. Regresión Logística Multivariable

Una vez que hemos la regresión logística simple, vamos a intentar predecir el valor del ratingLabel en función de la puntuación de los medicamentos y de los efectos secundarios del mismo. Para ello se va a emplear la

misma (`glm()`).

```
# Creamos el modelo para ratingLabel ~ sideEffectsInverse + effectivenessNumber
modelo_glm_multiple <- glm(ratingLabel ~ sideEffectsInverse + effectivenessNumber, data = data_train_pr
# Evaluamos el modelo
errores_regresion_logistica(modelo_glm_multiple)

##      real
## pred   0   1
##     0 50   5
##     1 24 234

## $Etrain
## [1] 8.366534
##
## $Etest
## [1] 9.265176
```

8.2.3. Regularización en Regresión Logística

En este apartado vamos a ajustar un modelo a través de la regresión logística con regularización mediante la técnica Lasso Regression.

8.2.3.1. Ridge Regression

Esta técnica, en esencia, es la regresión lineal con Weight Decay que utilizamos en la Práctica 2, y cuya fórmula es: $(XTX + \lambda I) - 1 * X^T T$. Es el parámetro lambda, el que en función de su valor, marca el equilibrio entre los componentes de sesgo y varianza. Cuanto mayor sea su valor, mayor sesgo pero menor varianza. Es decir, cuanto mayor valor tenga lambda, mayor penalización y mayor reducción se les aplica a los valores de los coeficientes.

8.2.3.2. Lasso Regression

Esta técnica, además de realizar la misma tarea que la anterior, pero con una fórmula distinta para aplicar la penalización. Aún así, también utiliza el parámetro lambda para concretar el equilibrio entre sesgo y varianza. Además, también es capaz de realizar una selección de variables, anulando algunos coeficientes. Por lo que además de la reducción de los valores de estos coeficientes también reduce el número de estos necesarios para ajustar el modelo.

Por tanto, para aplicar la regularización, primero debemos averiguar el valor de alpha, que representa la técnica a utilizar. Si `alpha=0` entonces la regularización se aplica con la técnica Ridge Regression. Si por el contrario `alpha=1`, entonces la técnica a utilizar será Lasso Regression. Una vez sabemos qué técnica aplicar a través del valor de alpha, tendremos que concretar el valor de lambda, para que dicha técnica pueda ajustar el modelo con su respectiva penalización.

Para ello, vamos a usar la función `train(...)` que es capaz de probar distintos valores para alpha y lambda a través del ajuste de varios modelos, de modo que devuelve el mejor valor de lambda y el mejor valor de alpha. Para ello, como primer argumento debemos indicarle los valores de las etiquetas, que se corresponden con los valores de la variable explicativa. A continuación, vamos a darle la oportunidad de explorar con las columnas `sideEffectsInverse + effectivenessNumber`. Como segundo argumento le proporcionamos el conjunto de entrenamiento. El tercer argumento especifica el tipo de técnica a utilizar para explorar todos los modelos posibles. Con `glmnet()`, le concretamos que queremos que ajuste dichos modelos a través de la Lasso Regression y de Ridge Regression. Y por último, como cuarto argumento le especificamos la clase de funciones que debe utilizar para ajustar un modelo a través de regresión logística.

```

library(glmnet)
# Ajustamos un modelo a través de Regresión Logística multiclas
modelbest = train(form=as.factor(ratingLabel) ~ sideEffectsInverse + effectivenessNumber,
                  data=data_train_procesado, method="glmnet", family="binomial")

modelbest$bestTune$alpha

## [1] 0.1
modelbest$bestTune$lambda

## [1] 0.0004725087

```

Como se puede observar, la función nos dice que el mejor modelo que podemos ajustar para nuestro conjunto de entrenamiento, es el que tiene el valor de alpha 0.1. Con este valor tan cercano a 0 podemos intuir que la técnica que va a emplear es Ridge Regression, ya que esta se representa a través de alpha = 0.

En cuanto al valor de lambda, podemos comprobar que es bastante próximo a 0, y que por tanto, la penalización que va a aplicar Ridge Regression no es demasiado agresiva. Por lo que podemos intuir que el modelo ajustado tendrá un cierto grado de flexibilidad en referencia a las muestras mal clasificadas. También indica, que el modelo que va a ajustar, por el valor pequeño de lambda, se puede caracterizar por un bajo sesgo pero una alta varianza. Lo que puede desembocar en un modelo sobreajustado a los datos de entrenamiento, con una capacidad de generalización bastante escasa.

Para ajustar este modelo hemos utilizado la función `glmnet(..)`. Como primer argumento especificamos una matriz con las muestras de entrenamiento pero sin sus etiquetas, las cuales se le proporcionan a la función en el segundo argumento. Como tercer argumento le volvemos a indicar la familia de funciones que debe utilizar para que sea un modelo ajustado a partir de regresión logística. Los dos siguientes parámetros se corresponden con el mejor alpha y el mejor lambda que hemos obtenido en el proceso anterior.

```

# Conjunto Train

# Primero generamos una matriz con el conjunto de train separando las etiquetas de los datos
x = model.matrix(ratingLabel~sideEffectsInverse + effectivenessNumber,
                  data_train_procesado)[,-ncol(data_train_procesado)]
y = data_train_procesado$ratingLabel

# Conjunto Test
x.test = model.matrix(ratingLabel~sideEffectsInverse + effectivenessNumber,
                      data_test_procesado)[,-ncol(data_test_procesado)]
y.test = data_test_procesado$ratingLabel

# Reproducimos el modelo ajustado con el mejor lambda y el mejor alpha
ridge.mod = glmnet(x=x, y=y, family="binomial", alpha=modelbest$bestTune$alpha,
                    lambda=modelbest$bestTune$lambda, thresh=1e-12)

# Calculamos la predicción
predicciones.ridge = predict(ridge.mod, s=ridge.mod$lambda,
                             newx=x.test, type="response")

# Error de regresión con penalización Ridge
error.ridge = mean (( predicciones.ridge - y.test)^2)
cat("Error con la técnica Ridge Regression: ",error.ridge*100,"%\n")

## Error con la técnica Ridge Regression: 10.37528 %

```

Como se observa, se obtiene un error similar a los anteriores.

8.3. Regresión Polinomial

En algunos casos, la verdadera relación entre la variable respuesta y los predictores puede no ser lineal, por lo que podemos aplicar por ejemplo una regresión polinomial. Una forma simple de incorporar asociaciones no lineales en un modelo lineal es incluir versiones transformadas de los predictores, elevándolos a distintas potencias, evitando un exceso de grados para evitar el sobreajuste o overfitting.

La forma más sencilla de incorporar flexibilidad a un modelo lineal es introduciendo nuevos predictores obtenidos al elevar a distintas potencias el predictor original.

Partiendo del modelo lineal: $Y_i = \beta_0 + \beta_1 X_i + \epsilon$

Se obtiene un modelo polinómico de grado d a partir de la ecuación:

$$Y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \dots + \beta_d x_i^d + \epsilon_i$$

```
# Hacemos uso del poly para 2
lm_poly = lm(data = data_train_procesado,
              formula = ratingLabel ~ poly(sideEffectsInverse, 2) + effectivenessNumber)
summary(lm_poly)

##
## Call:
## lm(formula = ratingLabel ~ poly(sideEffectsInverse, 2) + effectivenessNumber,
##      data = data_train_procesado)
##
## Residuals:
##       Min     1Q   Median     3Q    Max 
## -1.06537 -0.06628  0.05724  0.09101  1.13669
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)    
## (Intercept)                 0.19607   0.04170   4.702 3.34e-06 ***
## poly(sideEffectsInverse, 2)1  3.37481   0.27214  12.401 < 2e-16 ***
## poly(sideEffectsInverse, 2)2 -1.82668   0.26304  -6.945 1.19e-11 ***
## effectivenessNumber          0.15638   0.01012  15.456 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.262 on 498 degrees of freedom
## Multiple R-squared:  0.549, Adjusted R-squared:  0.5463 
## F-statistic: 202.1 on 3 and 498 DF,  p-value: < 2.2e-16

errorres_regresion_lineal(lm_poly)

##      real
## pred 0 1
##   0 52 6
##   1 22 233
##
## $Etrain
## [1] 7.768924
##
## $Etest
## [1] 8.945687
```

Los p-values individuales de sideEffectsInverse, apuntan a que un polinomio de grado 2 es suficiente para modelar el ratingLabel en función de sideEffectsInverse.

```
# Calculo del polinomio de grado 2
modelo_poli2 <- lm(data = data_train_procesado, formula = ratingLabel ~ poly(sideEffectsInverse, 2))
summary(modelo_poli2)

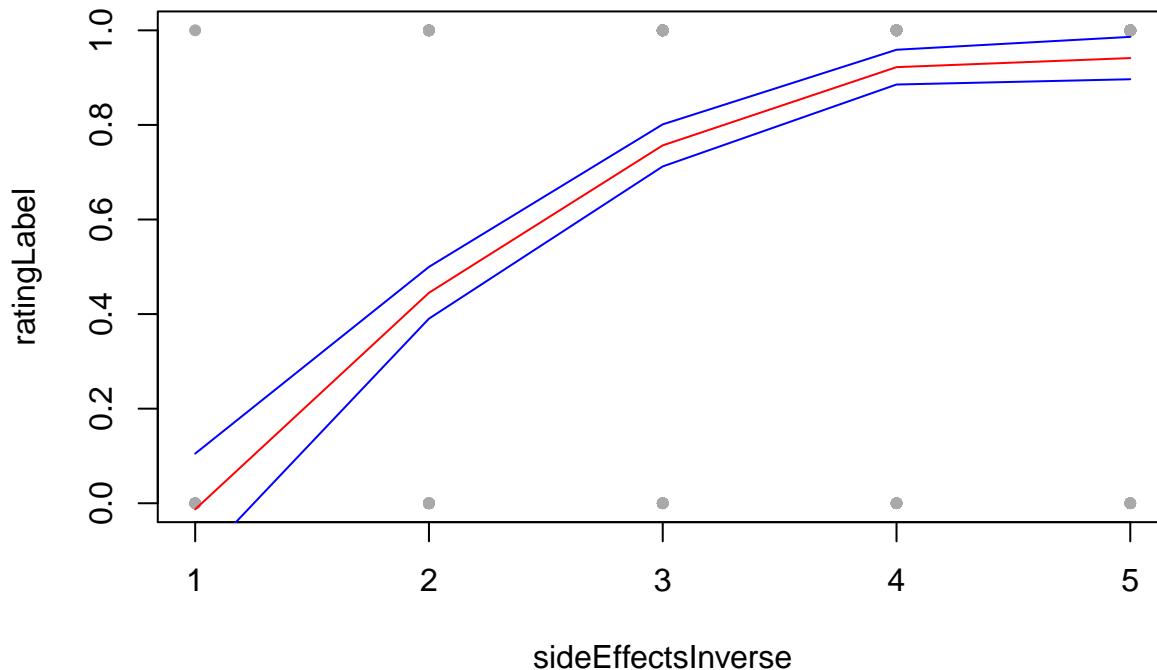
##
## Call:
## lm(formula = ratingLabel ~ poly(sideEffectsInverse, 2), data = data_train_procesado)
##
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -0.94138  0.05862  0.05862  0.07781  1.01293 
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)    
## (Intercept)             0.81474    0.01421  57.345 < 2e-16 ***
## poly(sideEffectsInverse, 2)1 4.51518    0.31833 14.184 < 2e-16 ***
## poly(sideEffectsInverse, 2)2 -2.19534    0.31833 -6.897 1.62e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3183 on 499 degrees of freedom
## Multiple R-squared:  0.3327, Adjusted R-squared:  0.33 
## F-statistic: 124.4 on 2 and 499 DF,  p-value: < 2.2e-16

# Interpolacion de puntos dentro del rango predictos
limites <- range(data_train_procesado$sideEffectsInverse)
nuevos_puntos <- seq(from = limites[1], to = limites[2], by = 1)
nuevos_puntos <- data.frame(sideEffectsInverse = nuevos_puntos)

# Prediccción de la variable respuesta y del error estándar
predicciones <- predict(modelo_poli2, newdata = nuevos_puntos, se.fit = TRUE,
                        level = 0.95)

# Cálculo del intervalo de confianza superior e inferior 95%
intervalo_conf <- data.frame(inferior = predicciones$fit -
                                1.96*predicciones$se.fit,
                                superior = predicciones$fit +
                                1.96*predicciones$se.fit)

attach(data_train_procesado)
plot(x = sideEffectsInverse, y = ratingLabel, pch = 20, col = "darkgrey")
title("Polinomio de grado 2: ratingLabel ~ sideEffectsInverse")
lines(x = nuevos_puntos$sideEffectsInverse, predicciones$fit, col = "red", pch = 20)
lines(x = nuevos_puntos$sideEffectsInverse, intervalo_conf$inferior, col = "blue", pch = 4)
lines(x = nuevos_puntos$sideEffectsInverse, intervalo_conf$superior, col = "blue", pch = 4)
```

Polinomio de grado 2: ratingLabel ~ sideEffectsInverse

En la gráfica, que acabamos de visualizar se puede observar como existe una relación, en donde a mayor valor de sideEffectsInverse, mayor ratingLabel, es decir, cuanto más favorable es un medicamento menos efectos secundarios tiene.

Por tanto, se ha observado como el método de regresión sobre las variables numéricas, obtiene unas predicciones de etiquetas muy buenas, obteniendo con la regresión logística multivariable el mínimo error fuera de la muestra. Ya que dicho método, permite estimar de manera muy aceptable la probabilidad de una variable cualitativa binaria en función de una variable cuantitativa.

9. Conclusiones

Después de realizar las técnicas que acabamos de comentar, hemos comprendido la importancia que tiene el preprocesamiento en nuestros datos, los cuales sirven como entrada para las técnicas implementadas en la Minería de Textos. Es por eso que a partir de una base de datos sin estructura, hemos ido transformando dichos datos en datos con una coherencia y sentido. Debido a que nuestro dataset de primeras disponía de mucho ruido, cosa que hubiera afectado negativamente a la precisión de los predictores. Es por eso que se llevó a cabo un debido filtrado, como se pudo ver en el apartado 2. Una vez que tuvimos los datos tratados adecuadamente, se procedió a la realización del análisis exploratorio de los datos. Dicho análisis, fue llevado a cabo, principalmente, para saber y comprender en mayor medida el conjunto con el que estamos trabajando y, en particular, para estudiar la efectividad y los efectos secundarios de los medicamentos.

Por otro lado, se llevó a cabo un análisis de sentimientos sobre los sentimientos de las personas, mediante la opinión que tienen sobre los distintos medicamentos que se encuentran en nuestro dataset. Dicho análisis nos llevó a la conclusión, de que se tiene una opinión más negativa que positiva sobre la aplicación de los medicamentos. Sin embargo, esta conclusión tiene coherencia, ya que las personas tiende a extraer los contras de los medicamentos. Y a no ser que el medicamento cumpla su función eficazmente, el paciente no tenderá a ser positivo respecto al mismo.

Con respecto a las técnicas, debemos destacar la componente subjetiva que tienen las reglas de asociación a la hora de su elección. Se observó, como los consecuentes más frecuentes fueron effect y side, lo que es lógico ya que estamos midiendo los efectos que tienen los medicamentos. Por otro lado, se aplicó el método de regresión sobre las variables numéricas, dicho método obtuvo unas predicciones de etiquetas muy buenas, obteniendo con la regresión logística multivariable el mínimo error fuera de la muestra. Ya que dicho método, permite estimar de manera muy aceptable la probabilidad de una variable cualitativa binaria en función de una variable cuantitativa.