



*ugr* | Universidad  
de **Granada**

MÁSTER EN INGENIERÍA INFORMÁTICA

— TRATAMIENTO INTELIGENTE DE DATOS —

## *Memoria de las prácticas*

---

Lidia Sánchez Mérida

Fernando Roldán Zafra

Granada, Enero de 2020

# Contents

<b>Descripción del dataset</b>	<b>2</b>
<b>Objetivos y estructura del documento</b>	<b>2</b>
<b>Análisis exploratorio de los datos del conjunto de entrenamiento original.</b>	<b>3</b>
<b>Preprocesamiento de los datos.</b>	<b>13</b>
Preprocesamiento de datos textuales. . . . .	16
Análisis de correlación. . . . .	20
Análisis de la influencia de las variables. . . . .	22
<b>Análisis textual</b>	<b>23</b>
Nube de palabras . . . . .	23
Análisis de sentimientos . . . . .	24
<b>Reglas de asociación</b>	<b>26</b>
<b>Técnicas de clasificación.</b>	<b>32</b>
Clasificación Bayesiana . . . . .	32
K-NN . . . . .	34
Árboles de decisión . . . . .	37
Random Forest . . . . .	41
<b>Regresión para predecir la efectividad del fármaco por su puntuación.</b>	<b>42</b>
Regresión lineal simple. . . . .	43
Regresión logística simple. . . . .	46
Regresión polinomial . . . . .	47
<b>Regresión para predecir la efectividad del fármaco en base a las reviews.</b>	<b>49</b>
Regresión lineal simple. . . . .	50
Regresión logística simple. . . . .	51
Regresión polinómica. . . . .	52
<b>Agrupamiento y clustering</b>	<b>54</b>
Agrupamiento de los farmacos en función al rating obtenido. . . . .	55
Adecuación de los datos y estudio previo. . . . .	55
Algoritmos particionales . . . . .	56
Algoritmos jerárquicos . . . . .	61
Extensiones de los algoritmos . . . . .	66

Agrupamiento de los medicamentos en función de las reviews de los usuarios. . . . .	69
Algoritmos Particionales . . . . .	72
Algoritmo jerárquico . . . . .	73
Extensiones de los algoritmos anteriores . . . . .	75
<b>Conclusiones finales.</b>	<b>79</b>
<b>Bibliografía.</b>	<b>81</b>

## Descripción del dataset

El dataset escogido para realizar las prácticas de esta asignatura es el ***UCI ML Drug Review dataset*** [1]. En él se puede encontrar una gran cantidad de información acerca de la efectividad de los fármacos recetados para las enfermedades diagnosticadas así como la opinión de los pacientes al respecto. Para ello se recopilan hasta siete campos, cuya naturaleza puede ser numérica o textual. Los campos más relevantes son *drugName*, en el que se encuentran los medicamentos recetados, *condition* que recopila las enfermedades, *review* en el que se almacenan las críticas de los pacientes con respecto a sus respectivas dolencias y tratamientos, así como *rating* que es un valor numérico entre 0 y 10 que el paciente le asigna a un tratamiento según su efectividad. Asimismo, también dispone de un campo numérico denominado *usefulCount* que representa el número de pacientes a los que les ha parecido útil la opinión de otro paciente. Otros campos menos relevantes son *uniqueID* con el que se representa el identificador único para cada una de las críticas de los pacientes así como *date*, que hace referencia a la fecha de publicación de las mismas.

Una cualidad que debemos destacar es que en la propia página de *Kaggle* de donde obtuvimos este dataset [1] ya están divididos los datos en dos conjuntos independientes: uno para entrenamiento y otro para test. La elección de este dataset se fundamenta en diversos motivos. El más importante es que cumple con los requisitos establecidos por el profesor para, posteriormente, aplicar los algoritmos vistos en esta asignatura. Sin embargo, también tuvimos en cuenta la organización de competiciones de Machine Learning en las que se proponía este dataset como principal protagonista y, además, aportan ciertas ideas para aplicar los tipos de algoritmos más populares, como veremos a continuación.

## Objetivos y estructura del documento

Tal y como hemos podido comprobar en la descripción anterior del dataset, el campo más relevante es el de las *reviews* de los pacientes. Es por ello por lo que la mayoría de las técnicas que aplicaremos a lo largo de este documento se enfocarán hacia el uso de este atributo. A continuación mostramos la estructura que va a seguir este documento:

- En primer lugar analizaremos estadísticamente el conjunto de entrenamiento original proporcionado por *Kaggle* para conocer cómo están dispuestos los datos, sus clases mayoritarias y minoritarias, así como los datos característicos más relevantes.
- A continuación reduciremos el tamaño de sendos conjuntos originales hasta que estén compuestos por 5.000 registros cada uno.
- Continuaremos realizando un preprocessamiento de las opiniones de los pacientes con el objetivo de quedarnos sólamente con los términos más relevantes.
- Realizaremos dos análisis generales para comprobar si disponemos de variables correlacionadas así como la influencia de cada una de ellas para trabajar solo con aquellos campos que nos aporten información útil.

Una vez dispongamos de los dos conjuntos de datos normalizados y de las *reviews* de los pacientes preprocesadas, aplicaremos las siguientes técnicas:

- **Nube de palabras.** Obtendremos los términos más frecuentemente utilizados en las críticas de los pacientes para conocer los temas que tratan en sus respectivas opiniones.
- **Análisis de sentimientos.** Con esta técnica pretendemos conocer si predominan las opiniones positivas, negativas así como cuántas de ellas se podrían considerar como neutras.
- **Reglas de asociación.** En esta sección intentaremos conocer las relaciones existentes entre los diferentes términos obtenidos para conocer en un mayor grado de profundidad de los temas que hablan los pacientes en sus críticas.

- **Clasificación.** En esta sección se aplicarán diversos algoritmos vistos en clase con el objetivo de conocer cuál es la técnica que mejor tasa de error proporciona partiendo del objetivo propuesto en la propia página del dataset [1]: **predecir la enfermedad del paciente en función de su review**. De este modo, obtendremos un clasificador capaz de predecir la dolencia que padece un paciente en función de la opinión que escriba.
- **Regresión.** En este apartado llevaremos a cabo una modificación de la idea que se propone para regresión en la página del dataset [1], es decir, intentaremos conocer cuál es la variable que mejor predice la **efectividad del medicamento**. Para ello haremos uso de las dos principalmente relacionadas con este campo: **la puntuación del fármaco y las reviews de los pacientes**. Para ello aplicaremos diversos tipos de regresión con tal de comparar la capacidad de predicción de cada clasificador en función de la variable predictora.
- **Agrupamiento.** Aplicaremos diversos algoritmos de agrupamiento para encontrar que algoritmos se adecuan mejor a nuestros datos de forma que podamos agrupar los datos de la mejor forma posible, extrayendo así información valiosa del dataset. Ademas de esto, en esa misma sección se utilizarán tecnicas de **text mining** junto con las de agrupamiento de forma que podamos estudiar las valoraciones sobre un medicamento y obtener así conclusiones sobre la estructura del dataset.
- **Bibliografía.** En este último apartado se adjuntan los enlaces utilizados para llevar a cabo la práctica.

## Análisis exploratorio de los datos del conjunto de entrenamiento original.

Tal y como hemos comentado anteriormente, este dataset que hemos seleccionado proporciona dos conjuntos, uno de entrenamiento y otro de prueba, en los que se recogen las opiniones de los pacientes, diagnosticados de una determinada enfermedad, acerca de los tratamientos que han probado. Existen dos tipos de opiniones, principalmente, una de ellas consiste en valorar el medicamento recetado para una enfermedad en concreto de 0 a 10. Por otro lado también se recopila, para cada opinión de un paciente, el número de personas que consideran que su crítica es útil.

Para comenzar con el estudio estadístico cargamos los datos de entrenamiento.

```
# Establecemos una semilla para que los resultados sean reproducibles.
set.seed(0)
# Cargamos la biblioteca que utilizaremos para graficar los datos estadísticos.
library(ggplot2)
# Cargamos los datos de entrenamiento y test
training_dataset <- read.csv(file="./drugsComTrain_raw.csv", header=TRUE, sep=",")
test_dataset <- read.csv(file="./drugsComTest_raw.csv", header=TRUE, sep=",")
```

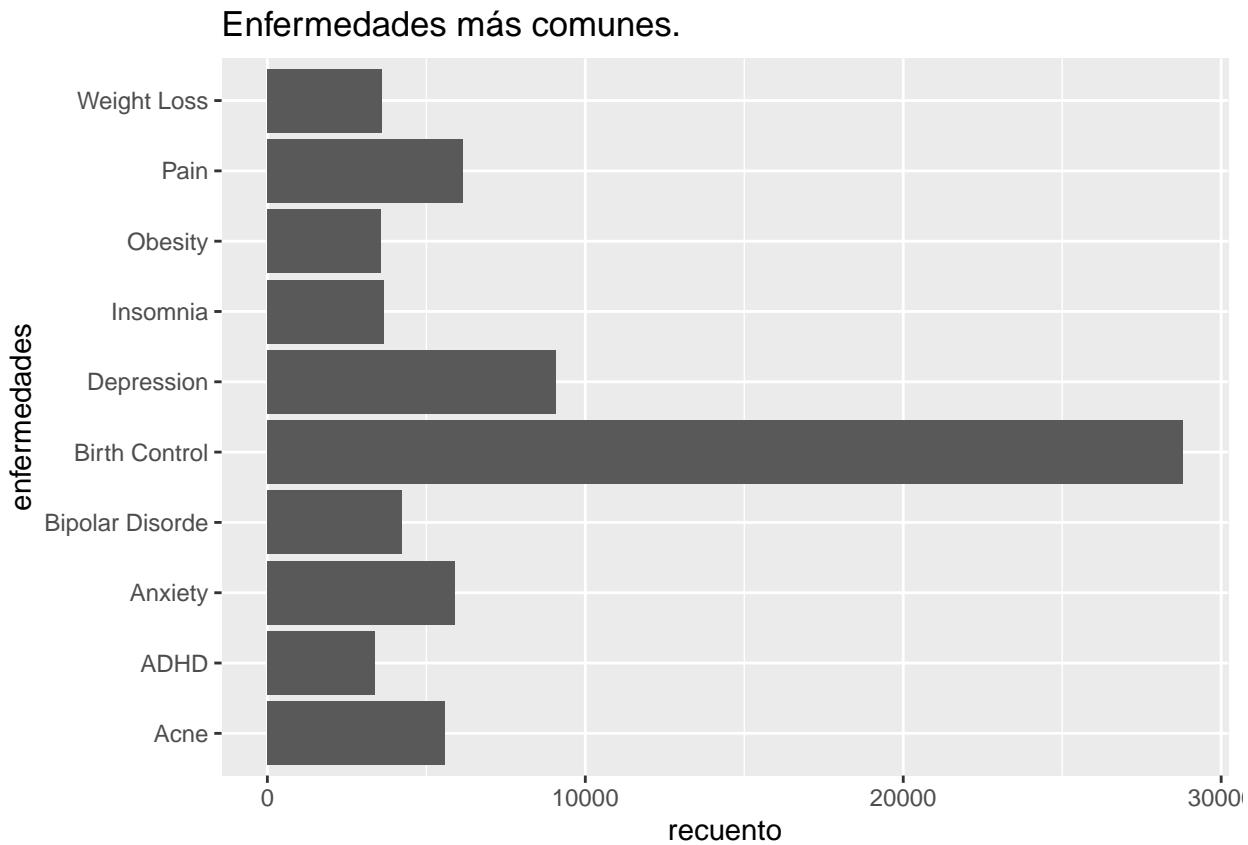
A continuación presentamos las estadísticas que hemos pensado son más relevantes para nuestro dataset en particular. En primer lugar averiguamos las diez enfermedades más comunes que se encuentran registradas en nuestro conjunto de datos con el objetivo de visualizar aquellas dolencias que se presentan más frecuente en un centro sanitario.

```
# 1) Enfermedades más comunes.
# Extraemos un resumen acerca de las enfermedades registradas.
recuento_enfermedades<-summary(training_dataset$condition)
# Convertimos los resultados anteriores en una matriz para poder trabajar con ellos.
enfermedades_matriz<-data.matrix(recuento_enfermedades)
# Obtenemos los nombres de las dolencias.
enfermedades<-names(recuento_enfermedades)
```

```

# También obtenemos el número de muestras de cada enfermedad.
recuento<-enfermedades_matriz[1:length(enfermedades_matriz)]
# Componemos un data frame con las enfermedades y el número de muestras para cada una.
dataframe<-data.frame(enfermedades=enfermedades[1:10], recuento=recuento[1:10])
# Las representamos en un gráfico de barras.
ggplot(data=dataframe, aes(x=enfermedades, y=recuento)) + geom_bar(stat="identity")+coord_flip() + ggtitle("Enfermedades más comunes")

```



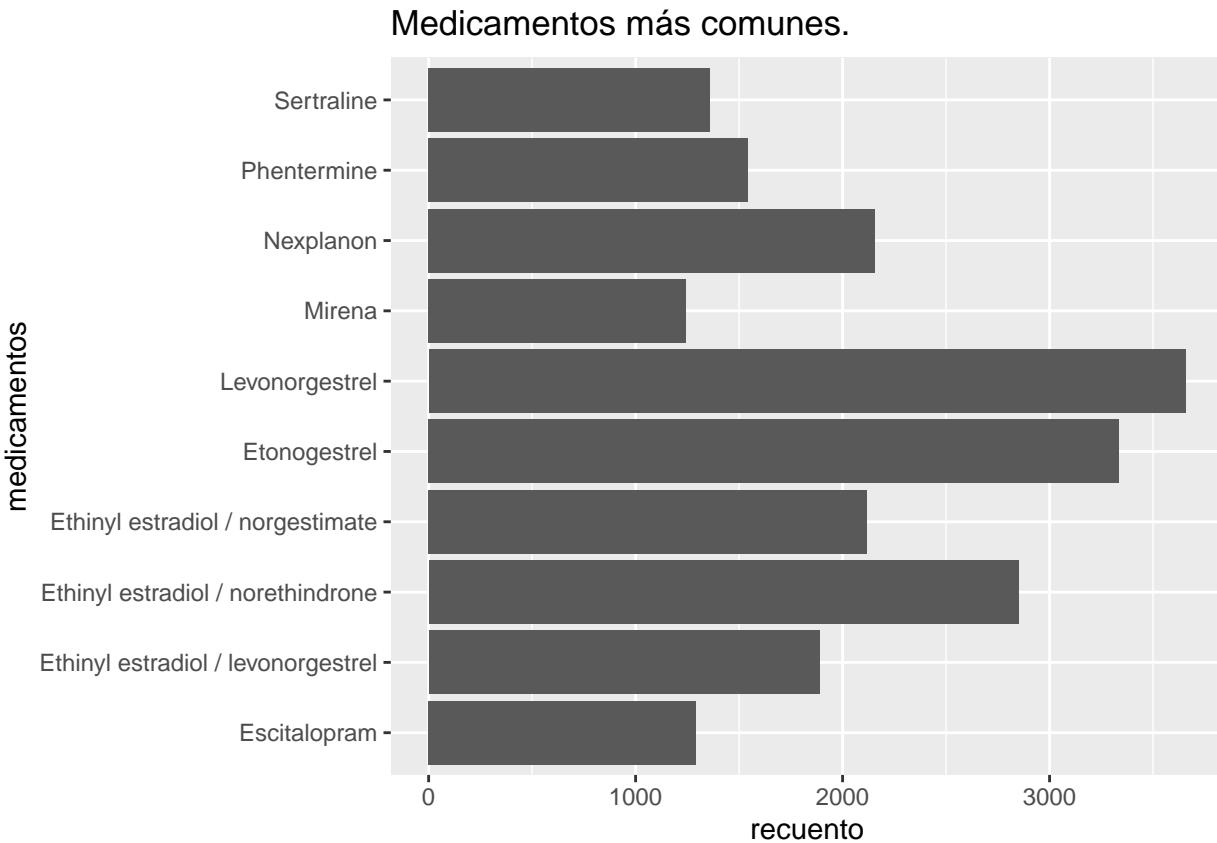
Tal y como se puede comprobar que la enfermedad *Birth Control* es considerablemente mayoritaria frente al número de muestras asociadas a las otras nueve enfermedades. Por ello, cuando reduzcamos nuestro conjunto de entrenamiento y test, deberemos tener en cuenta este aspecto de modo que todas las enfermedades escogidas tengan un número de muestras suficiente para ser representadas de forma equilibrada.

A continuación realizaremos el mismo experimento en relación a los medicamentos con el fin de conocer cuáles son los más recetados de forma general.

```

# 2) Medicamentos más comunes.
# Repetimos el procedimiento anterior para los medicamentos.
recuento_medicamentos<-summary(training_dataset$drugName)
medicamentos_matriz<-data.matrix(recuento_medicamentos)
medicamentos<-names(recuento_medicamentos)
recuento<-medicamentos_matriz[1:length(medicamentos_matriz)]
dataframe<-data.frame(medicamentos=medicamentos[1:10], recuento=recuento[1:10])
ggplot(data=dataframe, aes(x=medicamentos, y=recuento)) + geom_bar(stat="identity") + coord_flip() + ggtitle("Medicamentos más comunes")

```



Una vez hemos exploradas las enfermedades y medicamentos más comunes, los dos siguientes datos estadísticos consisten en asociar estos dos factores con el objetivo de conocer qué medicamentos más comunes se encuentran asociados a una misma enfermedad y cuáles de las enfermedades más comunes se encuentran asociadas para un mismo medicamento. En esta tercera estadística nos ocuparemos del primer caso.

```
# 3) Enfermedades comunes asociadas con un medicamento.
# Partimos del mismo calculo que en el apartado anterior donde obtenemos los medicamentos mas comunes
recuento_medicamentos<-summary(training_dataset$drugName)
matriz_medicamentos<-data.matrix(recuento_medicamentos)
medicamentos<-names(recuento_medicamentos)

# Creo un dataframe vacio donde meter los medicamentos mas comunes con sus enfermedades
dataframe_total<-data.frame()

# Bucle que itera sobre los 5 medicamentos más comunes
n_medicamento<-1
for(medicamento in medicamentos[1:5]){
  # Extraigo las filas del medicamento en cuestión
  df_medicamento<-training_dataset[training_dataset$drugName == medicamento, ]
  # Apunto el número de veces que se repite cada enfermedad
  recuento_enfermedades<-summary(df_medicamento$condition)
  # Transformo el dataframe a matriz
  enfermedades_matriz<-data.matrix(recuento_enfermedades)
  # Apunto el nombre de las enfermedades asociado al recuento de estas
  enfermedades<-names(recuento_enfermedades)
  # También obtenemos el número de muestras de cada enfermedad
```

```

recuento<-enfermedades_matriz[1:length(enfermedades_matriz)]
# Creo un dataframe computando por las 5 enfermedades más comunes con su número
dataframe<-data.frame(enfermedades=enfermedades[1:5], recuento=recuento[1:5])
# Creo el título del gráfico
text <-(paste0('Enfermedades más comunes para ', medicamento))
# Creo el gráfico para ese medicamento
ggplot(data=dataframe, aes(x=enfermedades, y=recuento)) + geom_bar(stat="identity")+coord_flip() + ggtitle(text)
ultimo<-paste("estadisticas_medicamento", n_medicamento, sep="")
ultimo<-paste(ultimo,".png",sep="")
ggsave(ultimo)
n_medicamento<-n_medicamento+1
}

```

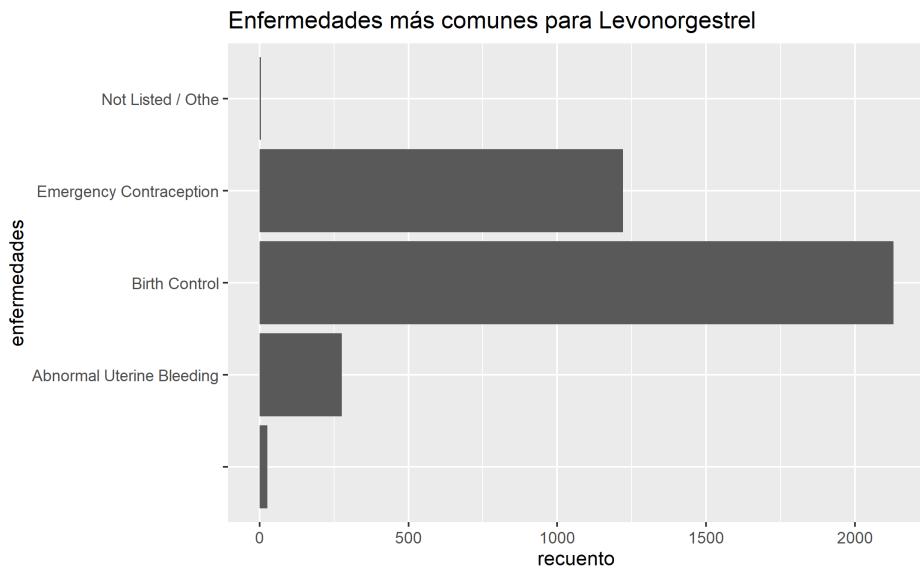


Figure 1: Estadísticas para el medicamento Levonorgestrel.

Como podemos comprobar, los cinco medicamentos más comunes son especialmente recetados para paliar la enfermedad denominada *Birth Control*. Del mismo modo podemos observar que el resto de dolencias a estos cinco tratamientos también están especializados en casos relacionados con la contracepción así como enfermedades asociadas con la menstruación y los órganos genitales femeninos. Esto nos indica que diversos tipos de dolencias, principalmente femeninas, pueden tratarse con los mismos medicamentos puesto que son indicados para un conjunto específico de dolencias de esta naturaleza en particular. Asimismo, podemos afirmar que si bien hemos comprobado que la enfermedad representada con un mayor número de muestras es *Birth Control*, tiene sentido que los cinco medicamentos más comunes estén dedicados a paliar esta dolencia, que también es sumamente común. Por tanto, podemos afirmar la relación existente entre los medicamentos más ampliamente recetados con las dolencias más altamente registradas.

Para el siguiente caso realizaremos el mismo cálculo pero en orden inverso, es decir, comprobaremos cuáles son los medicamentos más comunes para una misma enfermedad. De este modo podremos observar los distintos tratamientos que se pueden aplicar a una misma enfermedad además de comprobar su popularidad a la hora de recetar cada uno de ellos.

```

# 4) Medicamentos para cada enfermedad más común.
# Realizamos el mismo calculo que en el apartado anterior pero adaptandolo al caso contrario
recuento_enfermedades<-summary(training_dataset$condition)

```

```

matriz_enfermedades<-data.matrix(recuento_enfermedades)
enfermedades<-names(recuento_enfermedades)

dataframe_total<-data.frame()
n_enfermedad<-1
for(enfermedad in enfermedades[1:5]){
  df_enfermedades<-training_dataset[training_dataset$condition == enfermedad, ]
  recuento_medicamentos<-summary(df_enfermedades$drugName)
  medicamentos_matriz<-data.matrix(recuento_medicamentos)
  medicamentos<-names(recuento_medicamentos)
  recuento<-medicamentos_matriz[1:length(medicamentos_matriz)]
  dataframe<-data.frame(medicamentos=medicamentos[1:5], recuento=recuento[1:5])
  text <-(paste0('Medicamentos más comunes para ', enfermedad))
  ggplot(data=dataframe, aes(x=medicamentos, y=recuento)) + geom_bar(stat="identity")+coord_flip() +ggti
    titulo<-paste("estadisticas_enfermedad", n_enfermedad, sep="")
  titulo<-paste(titulo, ".png", sep="")
  ggsave(titulo)
}
n_enfermedad<-n_enfermedad+1
}

```

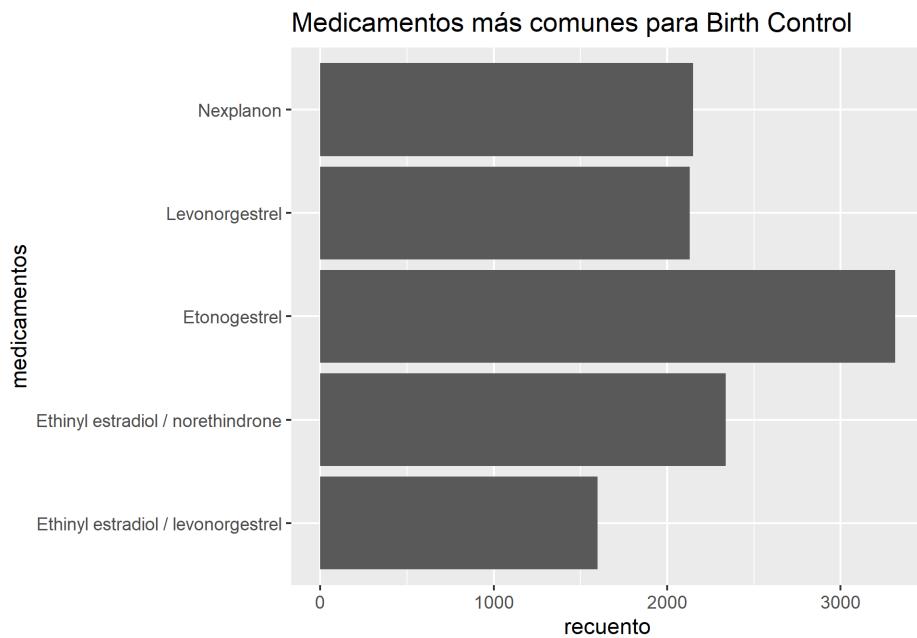


Figure 2: Estadísticas para la primera enfermedad.

En este caso, como se puede ver en las diferentes gráficas, los medicamentos se recetan de forma más distribuida, es decir para una misma enfermedad hay una mayor gama de tratamientos disponibles que se recetan con una frecuencia similar. Por lo tanto, no existen tratamientos recetados claramente mayoritarios, como sucedía en el caso anterior.

En el siguiente dato estadístico estudiaremos la valoración de los medicamentos para cada una de las diez enfermedades más comunes con el objetivo de conocer la eficacia, según los pacientes, de los tratamientos que en general se recetan para cada dolencia.

```

# 5) Eficacia general de los tratamientos recetados para las diez enfermedades más comunes.
# 5.1) Puntuación de los medicamentos asociados a "Birth Control".
# Extraemos un dataset solo con los registros asociados a la enfermedad en cuestión.
birth_control_dataset = subset(training_dataset, training_dataset$condition=="Birth Control")
# Calculamos variables estadísticas como la mediana, la media, el mínimo y máximo así como los cuartiles.
print("Estadísticas sobre los medicamentos para 'Birth Control'.")

## [1] "Estadísticas sobre los medicamentos para 'Birth Control'."

summary(birth_control_dataset$rating)

##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
##      1.00    3.00   7.00     6.09    9.00   10.00

# 5.2) Puntuación de los medicamentos asociados a "Depression".
depression_dataset = subset(training_dataset, training_dataset$condition=="Depression")
print("Estadísticas sobre los medicamentos para 'Depression'.")

## [1] "Estadísticas sobre los medicamentos para 'Depression'."

summary(depression_dataset$rating)

##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
##      1.000   5.000   8.000     7.099  10.000  10.000

# 5.3) Puntuación de los medicamentos asociados a "Pain".
pain_dataset = subset(training_dataset, training_dataset$condition=="Pain")
print("Estadísticas sobre los medicamentos para 'Pain'.")

## [1] "Estadísticas sobre los medicamentos para 'Pain'."

summary(pain_dataset$rating)

##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
##      1.000   7.000   9.000     7.632  10.000  10.000

# 5.4) Puntuación de los medicamentos asociados a "Anxiety".
anxiety_dataset = subset(training_dataset, training_dataset$condition=="Anxiety")
print("Estadísticas sobre los medicamentos para 'Anxiety'.")

## [1] "Estadísticas sobre los medicamentos para 'Anxiety'."

summary(anxiety_dataset$rating)

##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
##      1.000   7.000   9.000     7.691  10.000  10.000

```

```

# 5.5) Puntuación de los medicamentos asociados a "Acne".
acne_dataset = subset(training_dataset, training_dataset$condition=="Acne")
print("Estadísticas sobre los medicamentos para 'Acne'.")

## [1] "Estadísticas sobre los medicamentos para 'Acne'."

summary(acne_dataset$ratingng)

## Length Class Mode
##      0   NULL  NULL

# 5.6) Puntuación de los medicamentos asociados a "Bipolar Disorde".
bipolar_disorde_dataset = subset(training_dataset, training_dataset$condition=="Bipolar Disorde")
print("Estadísticas sobre los medicamentos para 'Bipolar Disorde'.")

## [1] "Estadísticas sobre los medicamentos para 'Bipolar Disorde'."

summary(bipolar_disorde_dataset$rating)

##    Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## 1.000  5.000  8.000  7.153 10.000 10.000

# 5.7) Puntuación de los medicamentos asociados a "Insomnia".
insomnia_dataset = subset(training_dataset, training_dataset$condition=="Insomnia")
print("Estadísticas sobre los medicamentos para 'Insomnia'.")

## [1] "Estadísticas sobre los medicamentos para 'Insomnia'."

summary(insomnia_dataset$rating)

##    Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## 1.000  3.000  8.000  6.724 10.000 10.000

# 5.8) Puntuación de los medicamentos asociados a "Weight Loss".
weight_loss_dataset = subset(training_dataset, training_dataset$condition=="Weight Loss")
print("Estadísticas sobre los medicamentos para 'Weight Loss'.")

## [1] "Estadísticas sobre los medicamentos para 'Weight Loss'."

summary(weight_loss_dataset$rating)

##    Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## 1.000  7.000  9.000  8.051 10.000 10.000

# 5.9) Puntuación de los medicamentos asociados a "Obesity".
obesity_dataset = subset(training_dataset, training_dataset$condition=="Obesity")
print("Estadísticas sobre los medicamentos para 'Obesity'.")

## [1] "Estadísticas sobre los medicamentos para 'Obesity'."
```

```

summary(obesity_dataset$rating)

##      Min. 1st Qu. Median     Mean 3rd Qu.    Max.
## 1.000   7.000  9.000  7.745 10.000 10.000

# 5.10) Puntuación de los medicamentos asociados a "ADHD".
adhd_dataset = subset(training_dataset, training_dataset$condition=="ADHD")
print("Estadísticas sobre los medicamentos para 'ADHD'.")
```

## [1] "Estadísticas sobre los medicamentos para 'ADHD'."

```
summary(adhd_dataset$rating)
```

```
##      Min. 1st Qu. Median     Mean 3rd Qu.    Max.
## 1.00   6.00   8.00   7.35   10.00 10.00
```

Como podemos comprobar en los resultados anteriores, si consideramos las medianas calculadas, es decir, la puntuación que se encuentra en la mitad del ranking, las enfermedades cuyos tratamientos, en general, son mejores valorados por los pacientes son *Pain*, *Anxiety*, *Acne*, *Weight Loss* y *Obesity*. Por el contrario, la que peor mediana tiene es *Birth Control*. Una de las ventajas de utilizar la mediana reside en la resistencia asociada a los valores de las puntuaciones, puesto que para calcularla lo único que se necesita es ordenar dichos valores y escoger el que se encuentra en la posición intermedia. Sin embargo, la media sí que se encuentra influida por los valores de las puntuaciones y, por lo tanto, se ve sesgada, principalmente, por los valores más bajos. Sin embargo, a través de esta variable estadística podemos confirmar las mismas conclusiones extraídas anteriormente, y además, nos permite agregar un mayor grado de concreción con respecto a los tratamientos mejor valorados. En este caso, podemos comprobar que la enfermedad cuyos tratamientos son mejor valorados es **Weight Loss, seguida de Obesity y Anxiety**.

A continuación procedemos cuáles son los medicamentos más efectivos, según la población, para estas tres enfermedades, con el objetivo de conocer las puntuaciones de los tratamientos que han provocado los buenos resultados observados anteriormente. Sin embargo, como el número de registros para cada una de las enfermedades es sumamente considerable, vamos a agregar una segunda condición. Esta trata de conseguir los medicamentos que a su vez son los más recetados. De este modo podremos conocer cuáles son los tratamientos más efectivos, según los pacientes afectados por las tres enfermedades anteriores, además de los más comúnmente recetados. Con todos estos criterios los resultados, para esta dolencia en concreto, se pueden visualizar a continuación.

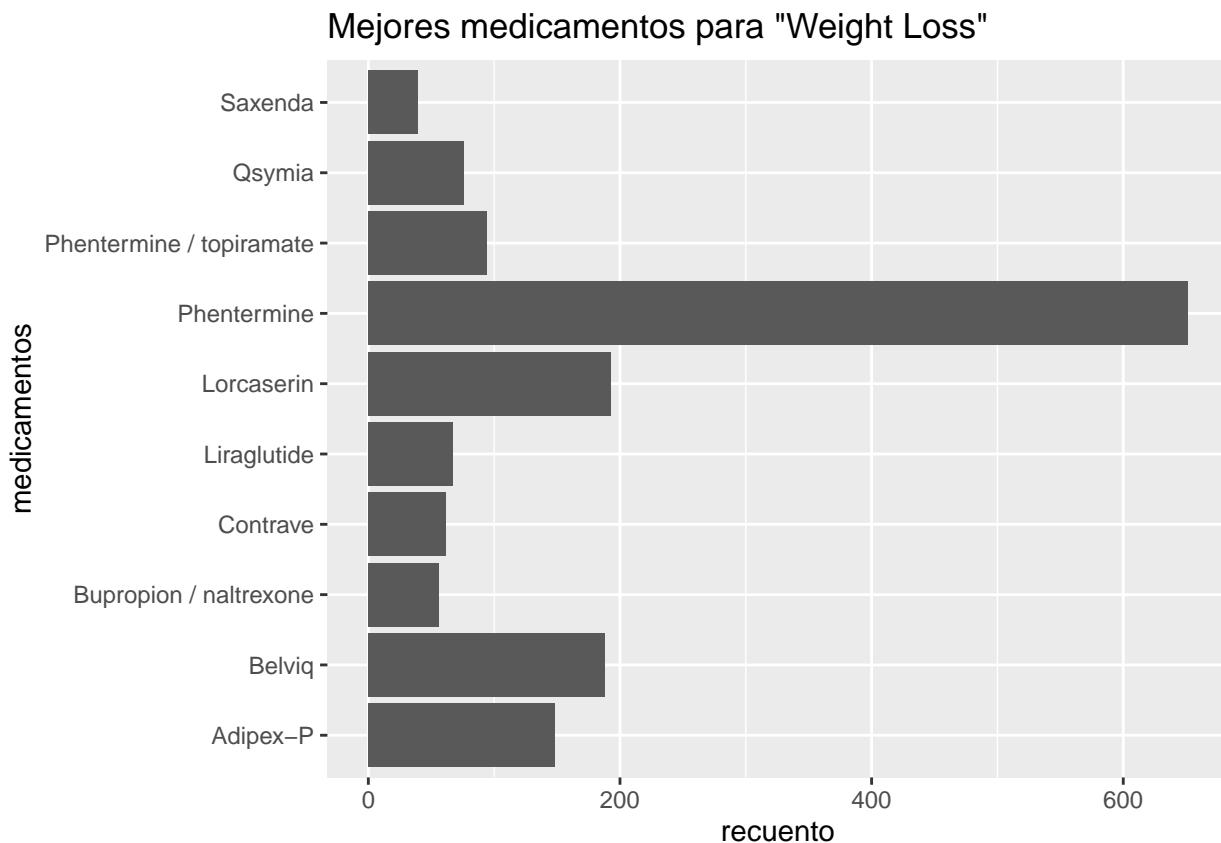
```

# 6) Medicamentos con mejores críticas y más veces recetados.
# 6.1) Para la primera enfermedad: 'Weight Loss'.
# Extraemos un dataset con las muestras asociadas solo a la enfermedad Weight Loss y con solamente los
# medicamentos que tengan una valoración 10/10.
birth_control_dataset = subset(training_dataset, training_dataset$condition=="Weight Loss" & training_
# Extraemos un resumen estadístico acerca de este pequeño dataset de Weight Loss.
recuento_medicamentos_weight_loss = summary(birth_control_dataset$drugName)
# Convertimos los datos a una matriz para poder trabajar con ellos.
medicamentos_matriz<-data.matrix(recuento_medicamentos_weight_loss)
# Obtenemos los nombres de los medicamentos para esta enfermedad en particular.
medicamentos<-names(recuento_medicamentos_weight_loss)
# Obtenemos el número de veces que han sido recetados cada uno de ellos.
recuento<-medicamentos_matriz[1:length(medicamentos_matriz)]
# Componemos un dataframe con estos últimos datos.
```

```

dataframe<-data.frame(medicamentos=medicamentos[1:10], recuento=recuento[1:10])
# Lo representamos mediante un gráfico de barras.
ggplot(data=dataframe,aes(x=medicamentos, y=recuento)) + geom_bar(stat="identity")+coord_flip() + ggtitle

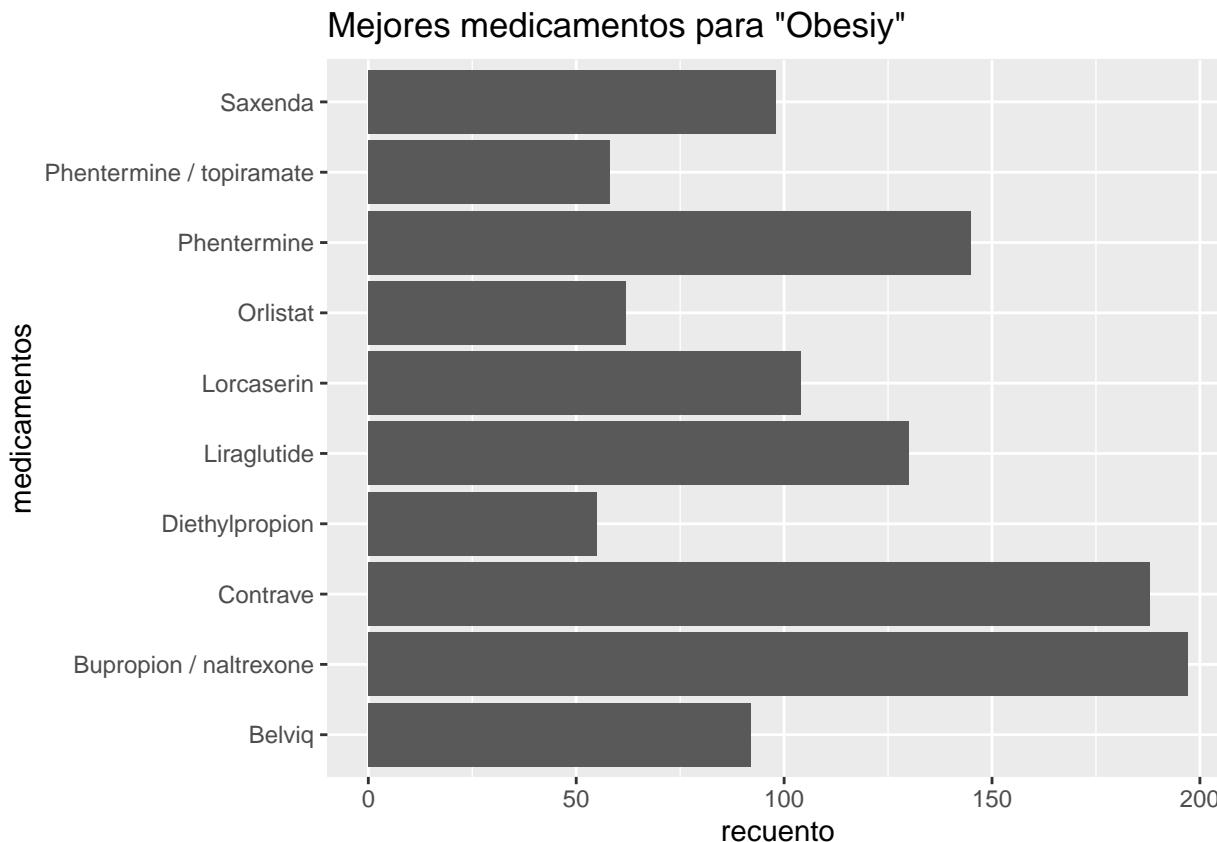
```



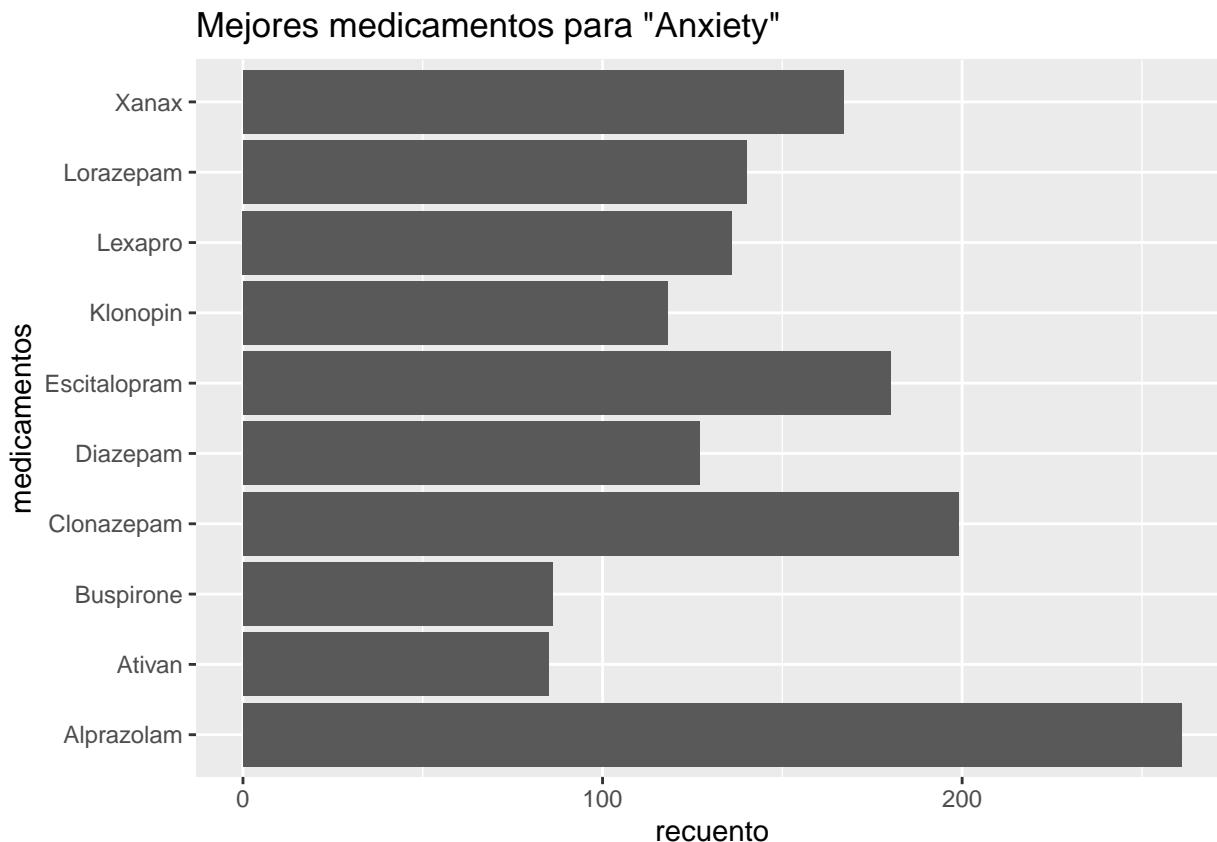
```

# 6.2) Para la segunda enfermedad: 'Obesity'
obesity_dataset = subset(training_dataset, training_dataset$condition=="Obesity" & training_dataset$rat
recuento_medicamentos_obesity = summary(obesity_dataset$drugName)
medicamentos_matriz<-data.matrix(recuento_medicamentos_obesity)
medicamentos<-names(recuento_medicamentos_obesity)
recuento<-medicamentos_matriz[1:length(medicamentos_matriz)]
dataframe<-data.frame(medicamentos=medicamentos[1:10], recuento=recuento[1:10])
ggplot(data=dataframe,aes(x=medicamentos, y=recuento)) + geom_bar(stat="identity") + coord_flip() + ggtitle

```



```
# 6.3) Para la tercera enfermedad: 'Anxiety'
anxiety_dataset = subset(training_dataset, training_dataset$condition=="Anxiety" & training_dataset$rate==1)
recuento_medicamentos_anxiety = summary(anxiety_dataset$drugName)
medicamentos_matriz<-data.matrix(recuento_medicamentos_anxiety)
medicamentos<-names(recuento_medicamentos_anxiety)
recuento<-medicamentos_matriz[1:length(medicamentos_matriz)]
dataframe<-data.frame(medicamentos=medicamentos[1:10], recuento=recuento[1:10])
ggplot(data=dataframe,aes(x=medicamentos, y=recuento)) + geom_bar(stat="identity") + coord_flip() + ggtitle("Mejores medicamentos para la 'Anxiety'")
```



En el primer gráfico resultante podemos comprobar que existe un medicamento con una puntuación máxima y que, a su vez, es claramente el más recetado denominado 'Phentermine/topiramate'. Por ello podemos determinar que en función de las críticas de los pacientes afectados por esta dolencia en concreto, este es el tratamiento mejor valorado y más comúnmente recetado.

No sucede lo mismo con el segundo gráfico en el que se muestran los diez medicamentos con una puntuación máxima y, a su vez, lo más recetados. En este caso existen dos tratamientos con un número bastante similar de recetas, por lo que podemos pensar que son los dos medicamentos más eficaces contra esta dolencia además de ser los dos más recetados.

Por último, en el tercer gráfico podemos observar que se plantea la misma dinámica que en el primero, es decir, de nuevo existe un medicamento contra, en este caso, la ansiedad que tiene la mejor puntuación posible (10 de 10) y que es el más recetado.

## Preprocesamiento de los datos.

Realizando el análisis exploratorio anterior, nos percatamos de que existen **reviews repetidas para una misma enfermedad**. Creemos que es debido a que para una misma dolencia se han recetado varios medicamentos y cuando el paciente ha aportado su opinión, esta se ha asignado a cada uno de los tratamientos recetados para la misma enfermedad los cuales ocupan un registro distinto. Por ello, actualmente disponemos de reviews iguales para varias filas dentro del conjunto de datos de entrenamiento y de prueba, asociadas a una misma enfermedad y a los diversos tratamientos recetados para paliarla. Con el objetivo de poder aplicar, posteriormente, los algoritmos vistos en esta asignatura y así obtener resultados representativos, vamos a aplicar un procesamiento equivalente a eliminar aquellos registros que dispongan de la misma crítica tanto en el dataset de entrenamiento como en el de prueba.

```

## Conjunto de entrenamiento
# Eliminamos las filas que tengan la misma crítica.
new_training_dataset<-training_dataset[!duplicated(training_dataset$review), ]
# Comprobamos que no existen reviews repetidas.
reviews_repetidas<-new_training_dataset[duplicated(new_training_dataset$review)]
ncol(reviews_repetidas)

## [1] 0

## Conjunto de prueba.
# Realizamos las mismas operaciones anteriores.
new_test_dataset<-test_dataset[!duplicated(test_dataset$review), ]
reviews_repetidas<-new_test_dataset[duplicated(new_test_dataset$review)]
ncol(reviews_repetidas)

## [1] 0

```

Una vez hemos eliminado aquellos registros que disponen de la misma crítica, comenzaremos con el submuestreo del dataset para **reducir su tamaño**. Y es que, tal y como se ha podido comprobar en los resultados estadísticos anteriores, el número de muestras del conjunto de entrenamiento es bastante considerable. Asimismo, también hemos podido comprobar que existen clases de medicamentos con un número de muestras extremadamente mayoritario, como es *Birth Control*, por lo que no todas cuentan con la misma representación dentro del dataset. Es por ello por lo que, a continuación, se procede a aplicar un tratamiento consistente en reducir el conjunto de entrenamiento a 5.000 muestras. Para ello reduciremos el **número de enfermedades a 10 y para cada una de ellas obtendremos 500 muestras**. De este modo equilibraremos las diversas clases de medicamentos existentes para, posteriormente, poder aplicar diversos algoritmos.

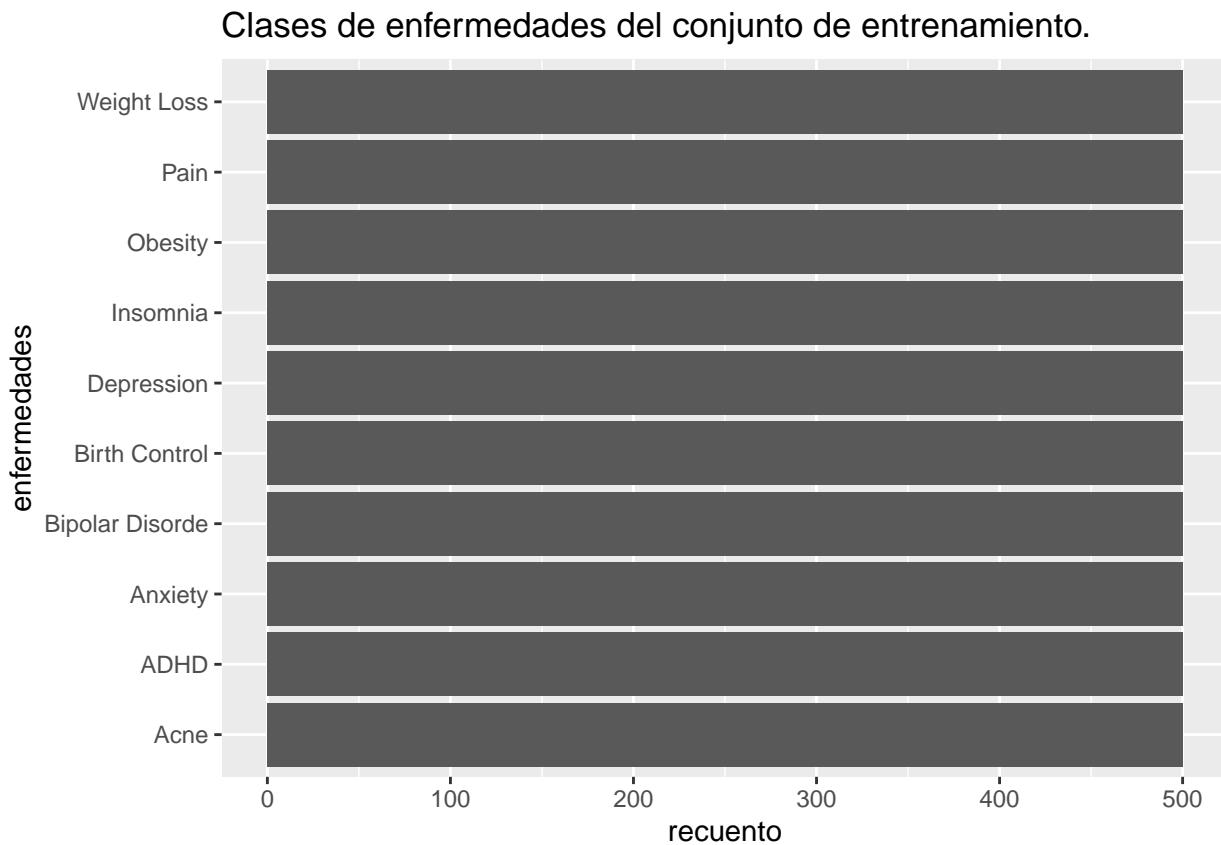
```

## CONJUNTO DE ENTRENAMIENTO
# Obtenemos un resumen acerca de las enfermedades registradas en el dataset.
recuento_etiquetas<-summary(new_training_dataset$condition)
# Recopilamos los nombres de las enfermedades.
etiquetas<-names(recuento_etiquetas)
# Formamos un dataset más pequeño con las 10 enfermedades con mayor número de ejemplos y les asociamos.
n_enfermedades_max = 10
n_muestras_max = 500
# Primera enfermedad
mini_training_dataset<-head(subset(new_training_dataset, new_training_dataset$condition==etiquetas[1]),
# Resto de enfermedades
for (i in c(2:n_enfermedades_max)) {
  mini_training_dataset<-rbind(mini_training_dataset, head(subset(new_training_dataset,
  new_training_dataset$condition==etiquetas[i]), n_muestras_max))
}
# Desorganizamos los datos
set.seed(42)
rows<-sample(nrow(mini_training_dataset))
mini_training_dataset<-mini_training_dataset[rows,]

# Repetimos el proceso que se ha aplicado anteriormente para la obtención de ciertas estadísticas para
recuento_enfermedades<-summary(mini_training_dataset$condition)
enfermedades_matriz<-data.matrix(recuento_enfermedades)
enfermedades<-names(recuento_enfermedades)
recuento<-enfermedades_matriz[1:length(enfermedades_matriz)]

```

```
dataframe<-data.frame(enfermedades=enfermedades[1:10], recuento=recuento[1:10])
ggplot(data=dataframe, aes(x=enfermedades, y=recuento)) + geom_bar(stat="identity") + coord_flip() + ggtitle("Clases de enfermedades del conjunto de entrenamiento.")
```



Como podemos comprobar en la gráfica anterior, en este dataset reducido ya no hay clases mayoritarias ni minoritarias, si no que todas disponen del mismo número de muestras representativas. Por lo tanto, hemos conseguido obtener un conjunto de datos de entrenamiento equilibrado con 5.000 muestras, es decir, 500 muestras para cada etiqueta que representa cada una de las diez enfermedades más comunes.

Del mismo modo procedemos con el **conjunto de prueba** de modo que también cuente con hasta 10 enfermedades diferentes con 500 muestras para cada una de ellas. Para ello repetiremos el mismo proceso anterior.

```
## CONJUNTO DE PRUEBA

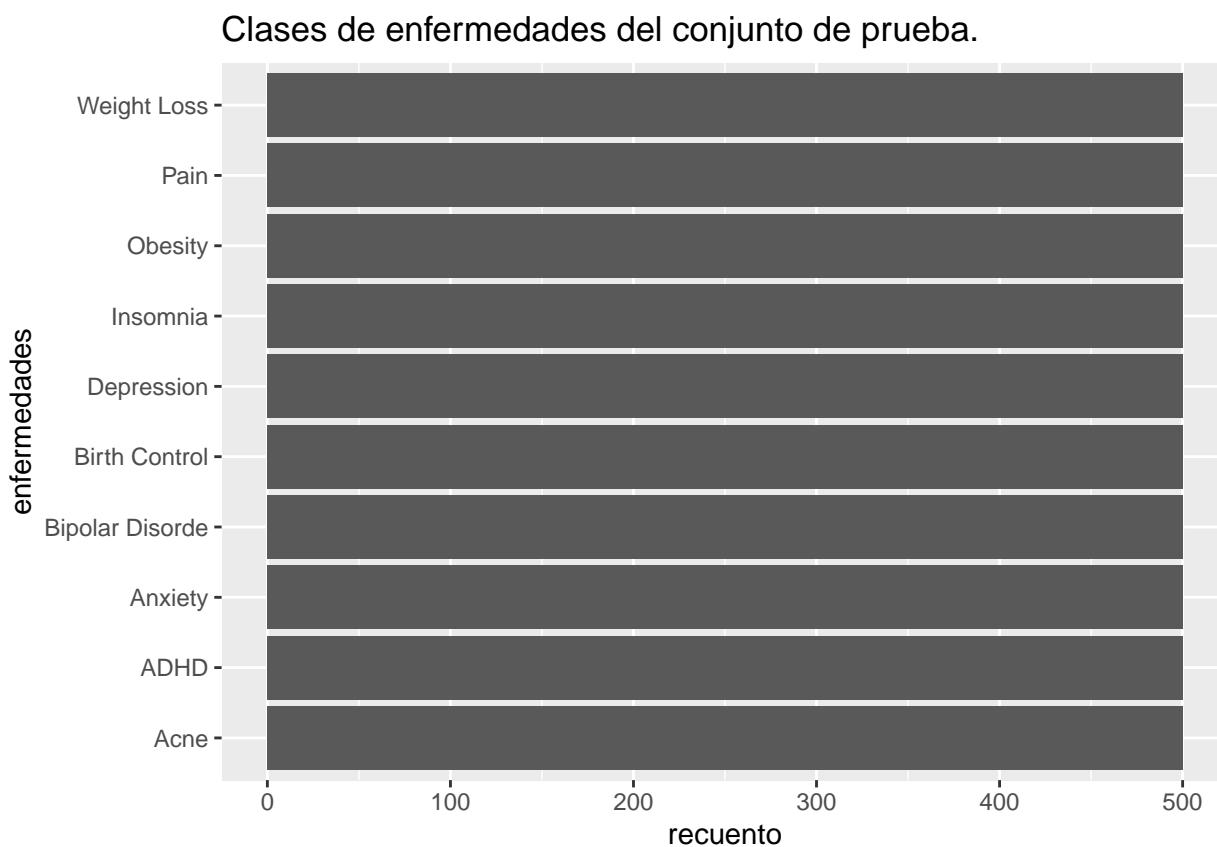
# Obtenemos un resumen acerca de las enfermedades registradas en el dataset.
recuento_etiquetas<-summary(new_test_dataset$condition)
# Recopilamos los nombres de las enfermedades.
etiquetas<-names(recuento_etiquetas)
# Formamos un dataset más pequeño con las 10 enfermedades con mayor número de ejemplos y les asociamos .
n_enfermedades_max = 10
n_muestras_max = 500
# Primera enfermedad
mini_test_dataset <- head(subset(new_test_dataset, new_test_dataset$condition==etiquetas[1]), n_muestras_max)
# Resto de enfermedades
for (i in c(2:n_enfermedades_max)) {
  mini_test_dataset <- rbind(mini_test_dataset, head(subset(new_test_dataset,
```

```

    new_test_dataset$condition==etiquetas[i]), n_muestras_max))
}
# Desorganizamos los datos
set.seed(42)
rows<-sample(nrow(mini_test_dataset))
mini_test_dataset<-mini_test_dataset[rows,]

# Repetimos el proceso que se ha aplicado anteriormente para la obtención de ciertas estadísticas para
recuento_enfermedades<-summary(mini_test_dataset$condition)
enfermedades_matriz<-data.matrix(recuento_enfermedades)
enfermedades<-names(recuento_enfermedades)
recuento<-enfermedades_matriz[1:length(enfermedades_matriz)]
dataframe<-data.frame(enfermedades=enfermedades[1:10], recuento=recuento[1:10])
ggplot(data=dataframe, aes(x=enfermedades, y=recuento)) + geom_bar(stat="identity") + coord_flip() + ggtitle("Clases de enfermedades del conjunto de prueba")

```



## Preprocesamiento de datos textuales.

En este subapartado procedemos a aplicar diversas técnicas para preparar los datos textuales, en especial las *reviews* de los pacientes, para que posteriormente podamos aplicar ciertos algoritmos con mayor facilidad. En primer lugar vamos a crear los vectores de documentos a partir de la columna anteriormente mencionada. Para ello convertiremos cada una de las críticas de los pacientes en un documento y las almacenaremos en un vector común denominado *corpus* [2].

```

# Cargamos el paquete que nos permitirá aplicar ciertas operaciones sobre textos.
library("tm")
# Datos de entrenamiento.
# Obtenemos en forma de vector la columna de las reviews.
train_reviews<-as.vector(mini_training_dataset$review)
# Lo convertimos en un documento y lo almacenamos en el corpus que posteriormente crearemos.
train_reviews_corpus<-VectorSource(train_reviews)
train_reviews_corpus<-VCorpus(train_reviews_corpus)
# Datos de test.
test_reviews<-as.vector(mini_test_dataset$review)
test_reviews_corpus<-VectorSource(test_reviews)
test_reviews_corpus<-VCorpus(test_reviews_corpus)

```

```
[1] "\"I have been on a lot of pain medicines since I was twelve years old, due to endometriosis. After thirteen surgeries, and abundant doctor visits with their many accompanying prescriptions, my doctor finally tried Norco 10/325 two twice daily for the pain, and it worked.\""
```

Figure 3: Contenido de una review.

Tal y como se puede observar, ahora cada una de las *reviews* de los pacientes se encuentra en un documento aparte, es decir, cada una de las opiniones de los pacientes ocupan una posición del vector de documentos. A continuación eliminaremos los **signos de puntuación** de las críticas porque, en principio, no nos aportan información relevante a la hora de aplicar técnicas relacionadas con la minería de textos. Asimismo, también hemos observado que ciertos signos de puntuación, como las comillas, no han sido procesados de forma adecuada y aparecen codificados como números. Obviamente este tipo de términos tampoco nos son útiles, por lo que procedemos, además, a **eliminar los números** de las críticas. Del mismo modo, con el objetivo de aplicar el mismo tratamiento a todas las palabras, procedemos también a **eliminar las mayúsculas**.

Por otro lado cabe destacar que para ciertas técnicas de análisis de textos, como podría ser encontrar los términos más comunes, existen algunas palabras, en todos los idiomas, cuyo número de apariciones es súmamente considerable pero no aportan ningún tipo de información relevante. Este tipo de términos se pueden clasificar en preposiciones, artículos, entre otros, y en este ámbito se suelen conocer como **stop words**. En nuestro caso, como todas las *reviews* se encuentran en inglés, especificaremos dicho idioma para eliminar este tipo de términos de las críticas de los pacientes. Tras este procedimiento, los términos eliminados dejan espacios en blanco que también vamos a eliminar en este primer preprocessamiento de los datos textuales. Para todo este proceso utilizaremos la librería *tm* especializada para el análisis y preprocessamiento de textos, y en particular la función *tm\_map* [3].

```

# Cargamos la librería que nos permitirá aplicar ciertas técnicas de preprocessamiento de textos.
library(tm)
# Datos de entrenamiento.
# Eliminamos los signos de puntuación con la función tm_map indicándoselo como segundo argumento.
train_reviews_corpus<-tm_map(train_reviews_corpus, content_transformer(removePunctuation))
# Eliminamos los números.
train_reviews_corpus<-tm_map(train_reviews_corpus, content_transformer(removeNumbers))
# Eliminamos las mayúsculas.
train_reviews_corpus<-tm_map(train_reviews_corpus, content_transformer(tolower))
# Eliminamos los stopwords en inglés.
train_reviews_corpus<-tm_map(train_reviews_corpus, content_transformer(removeWords), stopwords("english"))
# Eliminamos los espacios en blanco sobrantes tras el anterior procedimiento,
train_reviews_corpus<-tm_map(train_reviews_corpus, content_transformer(stripWhitespace))

# Datos de prueba.
test_reviews_corpus<-tm_map(test_reviews_corpus, content_transformer(removePunctuation))

```

```

test_reviews_corpus<-tm_map(test_reviews_corpus, content_transformer(removeNumbers))
test_reviews_corpus<-tm_map(test_reviews_corpus, content_transformer(tolower))
test_reviews_corpus<-tm_map(test_reviews_corpus, content_transformer(removeWords), stopwords("english"))
test_reviews_corpus<-tm_map(test_reviews_corpus, content_transformer(stripWhitespace))

```

```
[1] " alot pain medicines since twelve years old due endometriosis thirteen surgeries abundant doctor
visits many accompanying prescriptions doctor finally tried norco two twice daily pain worked "
```

Figure 4: Contenido preprocesado de una review.

Como podemos observar, tras este primer preprocesamiento de las *reviews* de los pacientes se presentan textos en los que únicamente aparecen términos que tienen un mayor grado de relevancia para el estudio de este dataset en particular. No obstante, a consecuencia de eliminar palabras, pueden existir documentos sin contenido y por lo tanto no disponen de ninguna utilidad. Este será el siguiente procesamiento que aplicaremos a continuación: **eliminar los documentos vacíos**. [3].

```

# Datos de entrenamiento.
# Borraremos los documentos que estén vacíos. Para ello los transformamos en una matriz de documentos.
train_reviews_matriz<-DocumentTermMatrix(train_reviews_corpus)
# Contamos el número de palabras de cada documento.
recuento_palabras<-apply(train_reviews_matriz, 1, sum)
# Comprobamos el número de documentos con 0 palabras.
cat("Documentos vacíos en el conjunto de entrenamiento:", recuento_palabras[recuento_palabras=0])

## Documentos vacíos en el conjunto de entrenamiento:

# Datos de prueba
test_reviews_matriz<-DocumentTermMatrix(test_reviews_corpus)
recuento_palabras<-apply(test_reviews_matriz, 1, sum)
cat("\nDocumentos vacíos en el conjunto de prueba", recuento_palabras[recuento_palabras=0])

##
## Documentos vacíos en el conjunto de prueba

# Estudio de la colección de documentos del conjunto de entrenamiento obtenida.
cat("\n\nEstudio de la colección de documentos del conjunto de entrenamiento tras el preprocesamiento.")

## 
## Estudio de la colección de documentos del conjunto de entrenamiento tras el preprocesamiento.

inspect(train_reviews_matriz)

## <<DocumentTermMatrix (documents: 5000, terms: 11826)>>
## Non-/sparse entries: 182380/58947620
## Sparsity : 100%
## Maximal term length: 65
## Weighting : term frequency (tf)
## Sample :
## Terms

```

```

## Docs day effects first ive now side started take taking years
## 148 0 0 0 0 1 0 1 2 0 0
## 2075 2 2 1 0 0 1 0 1 2 0
## 2191 1 1 0 0 0 1 2 0 2 0
## 2345 1 0 2 0 0 0 3 1 1 0
## 2979 0 1 0 0 0 2 0 0 0 2
## 3000 1 1 2 0 1 1 2 1 0 0
## 3680 1 0 0 0 0 0 0 0 1 2
## 3982 0 2 0 0 1 2 0 1 1 0
## 4723 0 1 0 0 1 1 1 0 0 0
## 854 0 0 0 0 1 1 0 0 0 0

```

Tal y como podemos comprobar, en nuestro caso, tras realizar el procedimiento por el cual eliminábamos palabras no relevantes, no existen documentos vacíos de contenido por lo que no podemos descartar ninguno. Asimismo, en función del estudio que se muestra a continuación acerca de la naturaleza de los documentos obtenidos del conjunto de entrenamiento, cabe destacar que se disponen de un total de 5.000 documentos y 11.825 términos. Sin embargo, en la lista de los términos más comunes podemos comprobar que algunos de ellos son variantes de otros, como por ejemplo son *take* y *taking*. Representa la misma información pero sin embargo se identifican como términos diferentes. Para eliminar este tipo de situaciones procedemos a aplicar una técnica conocida como **Lematización o Steaming**. Con ella se descartan aquellas palabras que tengan la misma raíz semántica y solo quedará un término de la misma familia [3].

```

# Datos de entrenamiento.
train_reviews_corpus<-tm_map(train_reviews_corpus, stemDocument)
train_reviews_matriz<-DocumentTermMatrix(train_reviews_corpus)
cat("Estudio de los documentos obtenidos tras el proceso de Lematización\n")

```

```
## Estudio de los documentos obtenidos tras el proceso de Lematización
```

```
inspect(train_reviews_matriz)
```

```

## <<DocumentTermMatrix (documents: 5000, terms: 8498)>>
## Non-/sparse entries: 176232/42313768
## Sparsity : 100%
## Maximal term length: 65
## Weighting : term frequency (tf)
## Sample :
## Terms
## Docs day effect feel month now start take week work year
## 148 0 0 0 2 1 1 2 1 0 0
## 2075 3 3 3 0 0 0 3 3 0 0
## 2191 1 1 3 0 0 3 2 1 0 0
## 2345 1 0 2 3 0 4 2 2 1 0
## 2979 1 4 0 0 0 0 0 0 1 2
## 3000 2 1 0 0 1 2 1 0 1 0
## 3680 2 0 0 2 0 0 1 1 0 2
## 3982 0 2 1 0 1 1 2 7 0 0
## 4723 1 1 0 1 1 1 0 0 0 0
## 854 0 1 0 5 0 1 0 0 0 0

```

```

# Datos de prueba.
test_reviews_corpus<-tm_map(test_reviews_corpus, stemDocument)
test_reviews_matriz<-DocumentTermMatrix(test_reviews_corpus)

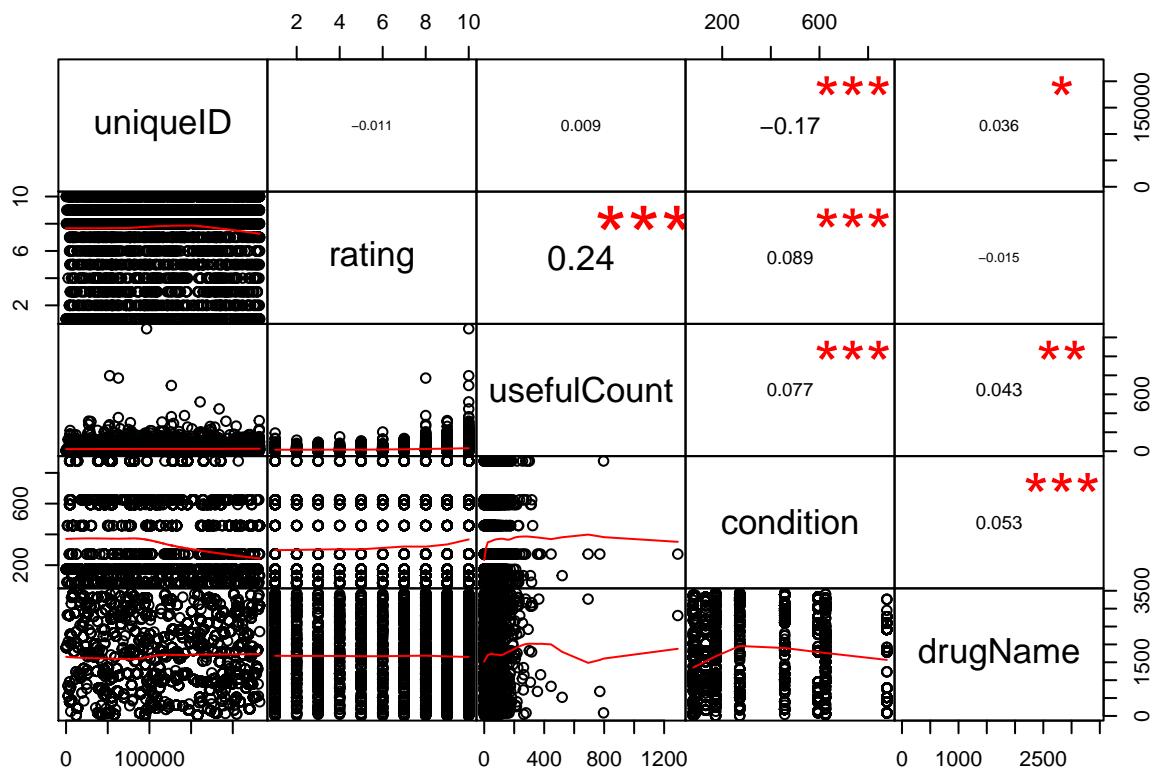
```

Tras la aplicación de esta última técnica podemos comprobar que se ha producido una reducción razonable de términos pasando a tener menos de 8.500 en total. Asimismo, podemos observar una mejora en relación a los términos más frecuentes puesto que todos ellos son diferentes y por lo tanto todos pueden aportar información relevante a la hora de aplicar los sucesivos algoritmos a lo largo de este documento.

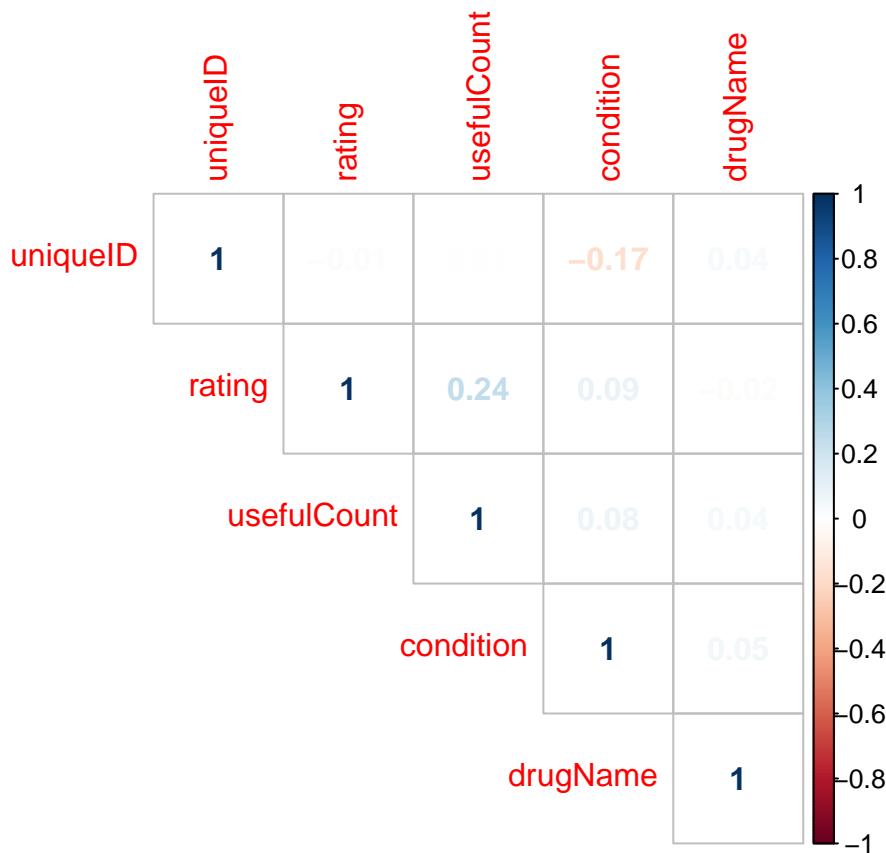
## Análisis de correlación.

En esta sección vamos a estudiar las relaciones que existen entre todas los campos de nuestro dataset para comprobar si algunos de ellos están correlacionados. Para ello vamos a convertir los atributos categóricos, como las enfermedades y los tratamientos, en valores numéricos sobre los que aplicar el estudio de la correlación tal y como se lleva a cabo en este tutorial [4]. Sin embargo, campos como la fecha en la que se publicó la crítica del paciente así como esta misma, los vamos a omitir puesto que no tiene sentido intentar convertirlos en variables numéricas.

```
# Copio el dataset original en otro que modificaremos para este apartado.
df_corr<-mini_training_dataset
# Eliminamos los campos categóricos puesto que vamos a convertir sus valores en valores numéricos.
df_corr$drugName <- NULL
df_corr$condition <- NULL
# También eliminamos los campos review y date puesto que no los vamos a utilizar en esta sección.
df_corr$review<-NULL
df_corr$date<-NULL
# Conversión de las variables categóricas, que son enfermedades y tratamientos, a variables numéricas.
df_corr["condition"] <- transform(as.numeric(mini_training_dataset$condition))
df_corr["drugName"] <- transform(as.numeric(mini_training_dataset$drugName))
# Calculo la matriz de correlación para las variables numéricas.
matriz_correlacion<-cor(df_corr) #, method="spearman")
# Calculamos los gráficos de dispersión y los coeficientes de correlación.
library(PerformanceAnalytics)
chart.Correlation(df_corr, histogram = F, pch = 19)
```



```
# Graficamos los coeficientes de correlación para todas las variables evaluadas.
library(corrplot)
corrplot(matriz_correlacion, method="number", type="upper")
```



En la primera imagen podemos comprobar los coeficientes de correlación entre cada par de variables existentes junto con la confianza asociada a esta correlación, representando la máxima con tres estrellas. Tal y como podemos comprobar, las variables que se encuentran más relacionadas son *rating* y *usefulCount* aunque su valor al ser de 0.24 no es considerablemente significante como para establecer una relación de correlación. Asimismo, si bien podemos pensar que los campos asociados a las enfermedades y a los tratamientos podrían mostrar una asociación, como podemos observar, esta no se hace patente a la hora de convertir los valores categóricos a numéricos puesto que de este modo el algoritmo no es capaz de relacionar ambos conceptos.

En la segunda imagen podemos visualizar los coeficientes de correlación entre cada par de variables clasificados y coloreados en función de cuán fuerte es su asociación. Tal y como podemos comprobar, la relación de correlación más fuerte, y por ende, la que dispone de un color más definido, es la que hemos comentado anteriormente: *rating-usefulCount*. El resto de asociaciones no cuentan con valores significativos y por lo tanto, apenas son visibles en la gráfica.

## Análisis de la influencia de las variables.

A continuación comprobaremos la importancia de cada uno de los campos que se incluyen en nuestro dataset. El objetivo es utilizar, solamente, aquellas columnas que sean relevantes para el estudio de este dataset en particular. Comenzamos eliminando directamente la columna *uniqueID* en ambos conjuntos de datos puesto que no aporta ningún tipo de información relevante ya que solo representa el identificador único para cada una de los registros del dataset. Del mismo modo también vamos a eliminar el campo en el que se recoge la fecha en la que se publicó la crítica puesto que tampoco nos aporta información útil para aplicar los sucesivos algoritmos.

```
## CONJUNTO DE ENTRENAMIENTO
mini_training_dataset = mini_training_dataset[-1]
```

```

mini_training_dataset = mini_training_dataset[-5]

## CONJUNTO DE PRUEBA
mini_test_dataset = mini_test_dataset[-1]
mini_test_dataset = mini_test_dataset[-5]

```

## Análisis textual

Tal y como hemos comentado al comienzo de este documento, una de las columnas más relevantes de nuestro dataset es la denominada *review*, en la que los pacientes expresan su opinión acerca del tratamiento recetado para su dolencia en particular. Es por ello por lo que, tras preprocessar estas críticas, procedemos a analizar el contenido de las mismas con el objetivo de conocer cuáles son los términos más repetidos así como el contento o descontento de los pacientes con sus respectivos medicamentos.

### Nube de palabras

Una vez disponemos de los datos preprocessados, tanto asociados al conjunto de entrenamiento como al de prueba, procedemos a representar gráficamente los términos obtenidos de los diversos documentos mediante una primera **nube de palabras**, en la cual se ordenarán las palabras de las críticas de los pacientes en función de su número de apariciones. De este modo podremos conocer los términos más utilizados en las *reviews*. Estos serán representados en un mayor tamaño y se irá reduciendo conforme disminuya el número de veces que aparece cada uno de los términos. Para dibujar la nube de palabras utilizaremos la librería *wordcloud2* [5] que es capaz de realizar gráficos más interesantes y vistosos que el paquete *wordcloud* convencional.



Figure 5: Nube de palabras.

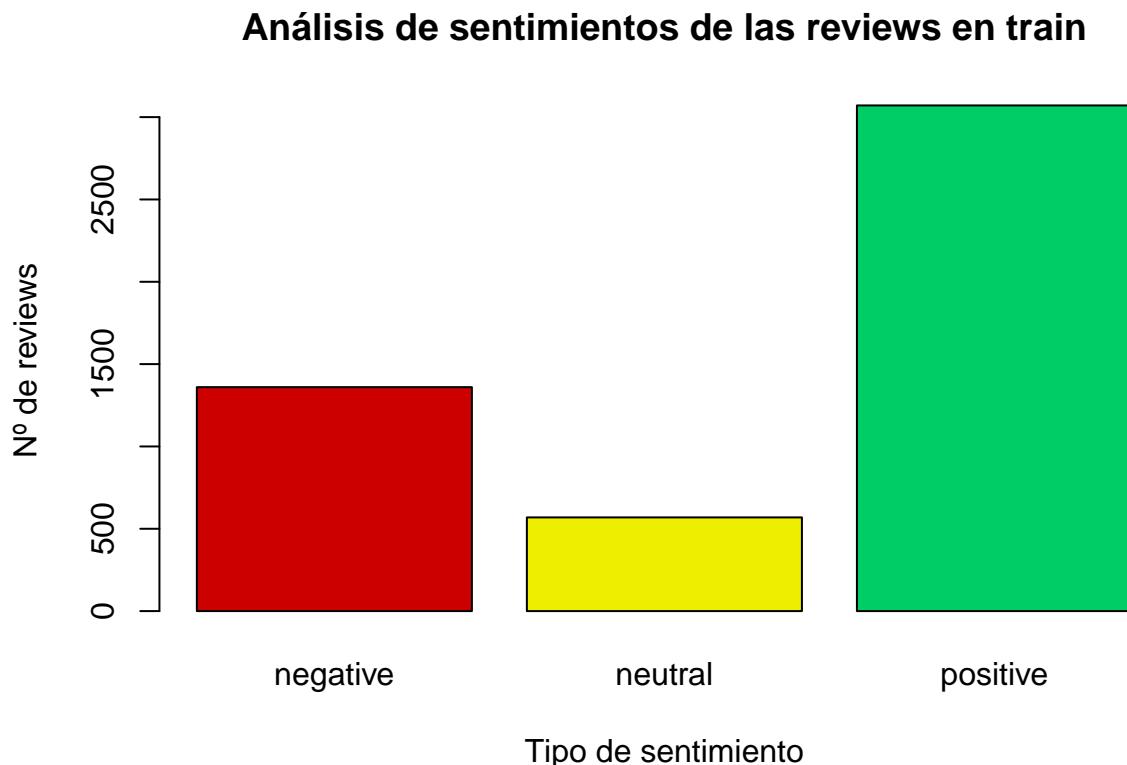
Tal y como podemos comprobar, la mayor parte de los términos más utilizados por los pacientes a la hora de expresar su opinión acerca de los fármacos recetados están directamente relacionados con el tiempo, como

*day, year*, entre otros. Posteriormente, podemos observar un segundo grupo de palabras relacionadas con los sentimientos como *feel* o *pain*. Incluso podemos afirmar que dos de las enfermedades que se encuentran en nuestro dataset aparecen esta nube de palabras: *pain* y *anxiety*. Para estudiar más a fondo el contexto en el que se encuentran estos términos a continuación procedemos a analizar los sentimientos que se encuentran tras cada una de las críticas de los pacientes.

## Análisis de sentimientos

En esta sección trataremos de conocer el grado de satisfacción general de los pacientes con respecto a los tratamientos que se les han recetado para su dolencia particular. Para ello, en primer lugar, procedemos a realizar una análisis general que nos informe acerca de la tendencia global de las críticas tal y como se realiza en este estudio [6]. Utilizaremos el corpus extraído tras el preprocesamiento de las críticas y le aplicaremos el método **analyzeSentiment** de la biblioteca *SentimentAnalysis* [7] con la cual obtendremos el tipo de sentimiento que se esconde tras cada una de las *reviews*: positivo, negativo o neutro. Para ello este tipo de biblioteca contiene una serie de diccionarios con los términos más relevantes para cada uno de los tres tipos de sentimientos. En nuestro caso particular utilizaremos el diccionario por defecto denominado *QDAP*.

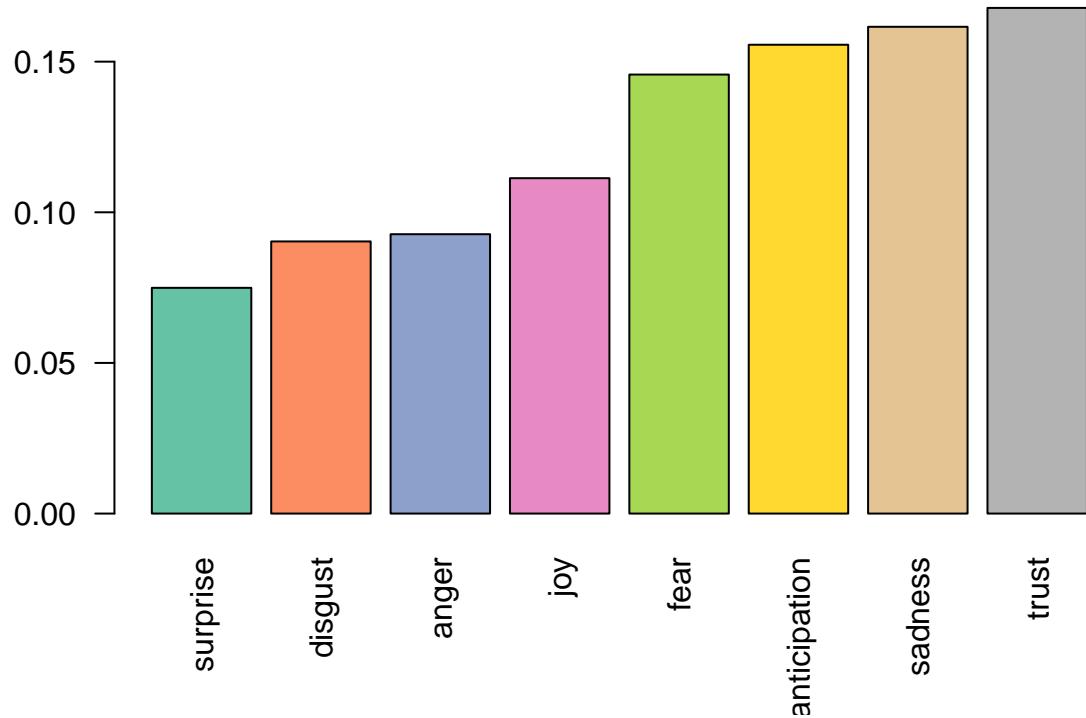
```
library(SentimentAnalysis)
# Obtenemos los sentimientos generales de las reviews
train_analisis_sentimientos<-convertToDirection(analyzeSentiment(train_reviews_corpus)$SentimentQDAP)
# Obtenemos su resumen estadístico
resumen_analisis = summary(train_analisis_sentimientos)
# Dibujamos los resultados obtenidos
barplot(resumen_analisis, main = "Análisis de sentimientos de las reviews en train",
        xlab = "Tipo de sentimiento", ylab = "Nº de reviews", col = c("red3", "yellow2", "springgreen3"))
```



Como podemos observar en ambos gráficos la gran mayoría de las críticas muestran un **sentimiento positivo**, por lo que podemos deducir que de forma general los pacientes están satisfechos con los tratamientos que han sido recetados para sus respectivas enfermedades. Asimismo, cabe destacar la considerable diferencia entre el número de críticas positivas y negativas. Por último, las críticas asociadas a un sentimiento neutro se corresponden con el grupo minotiratio puesto que es entendemos que es bastante complicado proporcionar una opinión sin demostrar nuestros sentimientos acerca de un fármaco.

A continuación procedemos a profundizar más acerca de los resultados obtenidos anteriormente con el objetivo de conocer cuáles son los sentimientos concretos más comunes que los pacientes demuestran a la hora de opinar sobre sus respectivos tratamientos. Para ello haremos uso de la librería *syuzhet*, y en concreto de la función *get\_nrc\_sentiment*, con la cual seremos capaces de medir el número de *reviews* que reflejan alguno de los ocho sentimientos posibles: enfado, expectación miedo, alegría, tristeza, confianza, sorpresa.

```
library(syuzhet)
library("tm")
library(ggplot2)
# Obtenemos el corpus de los comentarios de los pacientes ya preprocesados.
train_reviews_corpus2<-Corpus(VectorSource(train_reviews))
# Realizamos este nuevo análisis de sentimientos.
analisis_sentimientos2<-get_nrc_sentiment(train_reviews_corpus2$content)
# Representamos gráficamente los resultados ordenados de menor a mayor en función del porcentaje de crí
library(RColorBrewer)
colores<-brewer.pal(8, "Set2")
barplot(sort(colSums(prop.table(analisis_sentimientos2[, 1:8]))), las=2, col=colores)
```



Tal y como podemos comprobar el sentimiento que predomina en las críticas de los pacientes es la confianza, seguido de un segundo sentimiento negativo que es la tristeza y de un tercero que es la expectación. Estos

resultados pueden explicar el comportamiento de los seres humanos ante una enfermedad, y es que al ser diagnosticados es normal que sintamos tristeza por la situación actual y por la temporada que nos espera de tratamiento. Asimismo, también es razonable sentir confianza en los fármacos recetados por los especialistas en medicina para tratar nuestra dolencia y que nos hagamos ciertas expectativas acerca de la efectividad del mismo.

## Reglas de asociación

En esta técnica se pretende realizar un análisis acerca de uno de los campos más relevantes del dataset, a nuestro parecer: las *reviews* de los pacientes. Aplicando este procedimiento sobre nuestro conjunto de datos pretendemos extraer las relaciones existentes entre las palabras contenidas en estas críticas que redactan los enfermos asociadas a una determinada enfermedad y a uno o varios tratamientos. De este modo podremos conocer las relaciones existentes entre los términos que son más comunes en las *reviews*.

Inspirándonos en este estudio [8], el primer paso que procedemos a realizar consiste en separar los términos de las críticas de modo que obtengamos las palabras individuales y separadas del resto. Para ello haremos uso de la función *text\_tokens* aplicada al corpus creado y preprocesado anteriormente. A continuación podremos componer la **base de datos transaccional** en la que cada elemento se corresponde con una única palabra. Una vez dispongamos de los datos preparados podremos obtener las reglas de asociación mediante el paquete **aRules**. Para ello utilizaremos el algoritmo *apriori*[9], el cual extrae los conjuntos de palabras más frecuentes y los convierte en reglas de asociación. Como parámetros recibe la base de datos con las transacciones además de argumentos de configuración tales como el **soporte**, el cual indica el número de tuplas que soportan el proceso de inducción, y la **confianza**, la cual representa la calidad de cada una de las reglas. Además, para que sea aún más eficiente también estableceremos ciertos parámetros de rendimiento para que no cargue en memoria todas las reglas que obtenga. Asimismo también le indicamos que comience a analizar reglas con dos términos mínimo.

Según esta fuente [10], para ajustar el primer valor deberemos de conocer el número de transacciones contenidas en la base de datos así como las frecuencias máximas de los términos recogidos en las *reviews*.

```
# Importamos la librería para obtener los términos del corpus separados.
library(arules)
library(corpus)
# Separamos los términos de cada una de las críticas ya preprocesadas.
terminos_criticas<-text_tokens(train_reviews_corpus)
# Convertimos los términos a una base de datos transaccional
reviews_bd_transaccional<-as(terminos_criticas, "transactions")
# Obtenemos un resumen acerca de la bd creada.
cat("\n\nResumen de la base de datos transaccional.\n")

##
##
## Resumen de la base de datos transaccional.

summary(reviews_bd_transaccional)

## transactions as itemMatrix in sparse format with
## 5000 rows (elements/itemsets/transactions) and
## 8664 columns (items) and a density of 0.004202124
##
## most frequent items:
##    take      day     work   effect   start (Other)
```

```

##      2187     1823     1641     1616     1569   173200
##
## element (itemset/transaction) length distribution:
## sizes
##   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17  18
##  16  30  28  43  53  31  49  59  56  49  46  64  67  90  66  59  60  82
##  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##  69  76  99  92  80  81 114  94  80  77  82  86  92  62  87  90  90  89
##  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##  84  70  83  96  62  74  75  84  69  88  87  89  78 101  95 101 110  97
##  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  73
## 111  97 108 115 102  65  76  61  52  47  27  32  28  13  13  4   3   2
##  74  75  76  77  78  79  89  92  97 129
##   3   2   1   1   1   1   1   1   1   1
##
##      Min. 1st Qu. Median     Mean 3rd Qu.    Max.
##      1.00  22.00  37.00  36.41  52.00 129.00
##
## includes extended item information - examples:
##   labels
## 1      ^
## 2      \230
## 3      <
##
## includes extended transaction information - examples:
##   transactionID
## 1          1
## 2          2
## 3          3

cat("\n\nFrecuencia de las transacciones.\n")

##
##
## Frecuencia de las transacciones.

summary(itemFrequency(reviews_bd_transaccional))

```

```

##      Min. 1st Qu. Median     Mean 3rd Qu.    Max.
## 0.000200 0.000200 0.000400 0.004202 0.001200 0.437400

```

Como podemos observar existen 5.000 transacciones dentro de la base de datos obtenida. El término más repetido es *take* con hasta 2187 apariciones y su respectiva frecuencia es de 0.437, la máxima de entre todos los términos. Este valor nos indica el soporte máximo que podemos establecer. Sin embargo, para poder calcular este valor de una forma más precisa necesitamos establecer el número mínimo de apariciones que deseamos para cada uno de los términos y, posteriormente, lo dividimos entre el número total de transacciones. Comenzaremos estableciendo una frecuencia mínima de 500, por lo que el **soporte** =  $500/5000 = 0.1$ . Establecemos el valor por defecto para la confianza: 0.8.

```

# Importamos la librería para obtener las reglas de asociación y la base de datos transaccional
library(arules)
library(arulesViz)
# Obtenemos las reglas de asociación.
reglas_asoc<-apriori(reviews_bd_transaccional, parameter = list(support = 0.1, confidence = 0.8, target =

```

```

## Apriori
##
## Parameter specification:
##   confidence minval smax arem aval originalSupport maxtime support minlen
##           0.8      0.1     1 none FALSE                  TRUE       5     0.1      2
##   maxlen target ext
##       10  rules FALSE
##
## Algorithmic control:
##   filter tree heap memopt load sort verbose
##   0.1 TRUE TRUE  TRUE FALSE    2    TRUE
##
## Absolute minimum support count: 500
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[8664 item(s), 5000 transaction(s)] done [0.06s].
## sorting and recoding items ... [70 item(s)] done [0.00s].
## checking subsets of size 1 2 3 done [0.04s].
## writing ... [8 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].

```

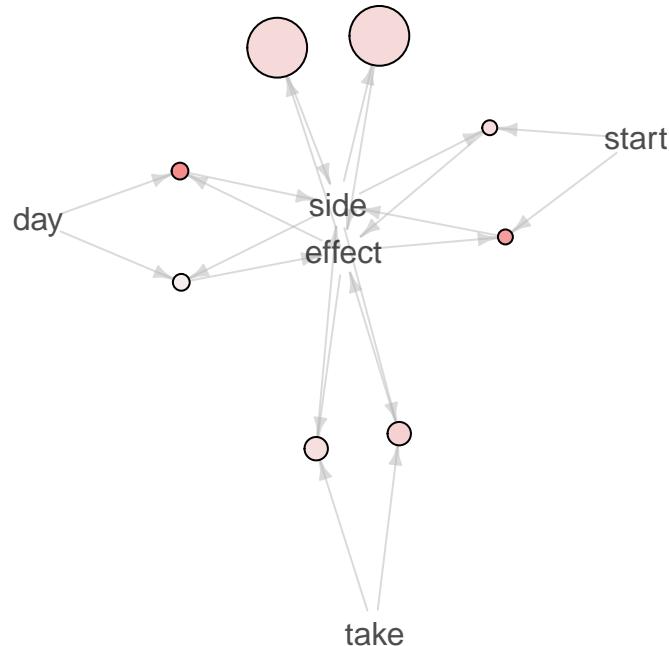
```

# Ordenamos las reglas obtenidas ordenadas por la confianza.
reglas_asoc<-sort(reglas_asoc, decreasing=TRUE, by="confidence")
# Dibujamos las reglas.
plot(reglas_asoc, method="graph")

```

## Graph for 8 rules

size: support (0.101 – 0.271)  
color: lift (2.875 – 2.981)



Como podemos observar hemos obtenido un total de ocho reglas de asociación, en las que los dos consecuentes más comunes son *side* y *effect*. Esto se debe a que, de forma general, los pacientes tienden a expresar en sus *reviews* los efectos que les producen sus tratamientos. Sin embargo, con un soporte tan alto apenas obtenemos más información, por lo que a continuación estableceremos una frecuencia mínima de 50 apariciones para cada término, obteniendo así un **soporte=0.01**.

```

library(arules)
library(arulesViz)
reglas_asoc<-apriori(reviews_bd_transaccional, parameter = list(support = 0.01, confidence = 0.8, target = "rules"))

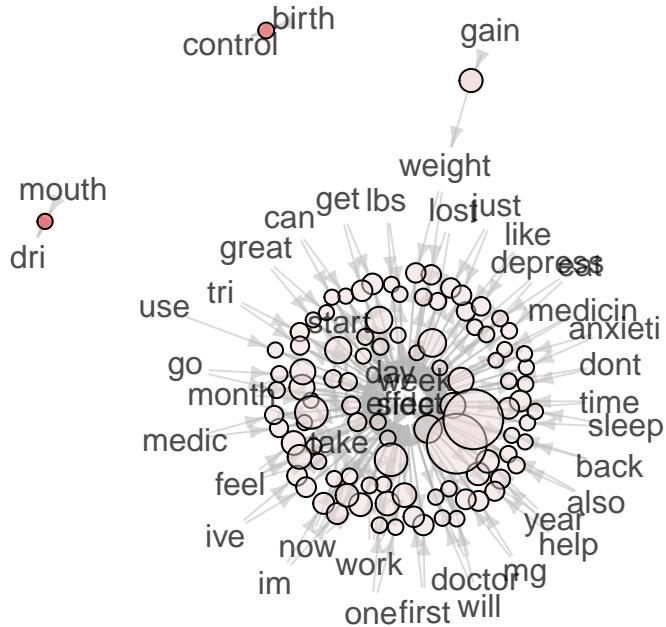
## Apriori
##
## Parameter specification:
##   confidence minval smax arem aval originalSupport maxtime support minlen
##             0.8     0.1    1 none FALSE           TRUE      5     0.01      2
##   maxlen target ext
##         10  rules FALSE
##
## Algorithmic control:
##   filter tree heap memopt load sort verbose
##   0.1 TRUE TRUE  TRUE FALSE    2    TRUE
##
## Absolute minimum support count: 50
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[8664 item(s), 5000 transaction(s)] done [0.08s].
## sorting and recoding items ... [646 item(s)] done [0.00s].
## checking subsets of size 1 2 3 4 5 6 done [0.19s].
## writing ... [4401 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
```

```

reglas_asoc<-sort(reglas_asoc, decreasing=TRUE, by="confidence")
plot(reglas_asoc, method="graph")
```

## Graph for 100 rules

size: support (0.043 – 0.271)  
color: lift (2.816 – 12.086)



Al disminuir el soporte hemos obtenido un mayor número de información puesto que las reglas obtenidas son menos estrictas a la hora de buscar asociaciones entre los términos de las *reviews* de los pacientes. En este caso podemos observar un mayor número de relaciones entre palabras, como *gain* y *weight*, los cuales están estrechamente relacionados puesto que una de las enfermedades que se encuentra en nuestro dataset es precisamente la pérdida de peso, por lo que es lógico que aparezcan estos dos términos. Del mismo modo ocurre con *birth* y *control*. Asimismo, se ha añadido un consecuente más al núcleo de las reglas de asociación: *take*. Este hecho se explica en relación a que el paciente además de expresar los efectos que tienen los fármacos sobre su dolencia, por lo general también suelen especificar las tomas que realizan de este medicamento durante el tratamiento impuesto por su médico. Es por ello por lo que estos tres términos se encuentran como los consecuentes más comunes de las reglas obtenidas. Relacionado con este aspecto se encuentran los términos asociados con espacios temporales, como *week*, *now*, *month*, entre otros, los cuales aparecen también cercanos al núcleo.

A continuación procedemos a experimentar con el valor de la **confianza**. Si bien este mide la calidad de las reglas obtenidas, podemos deducir que a menor valor mayor número de reglas de peor calidad. Para ello vamos a realizar un primer experimento con soporte=0.1. Nuestra teoría parte de que si disminuimos la calidad de las reglas que se puedan obtener, aparecerán un mayor número de ellas pese al soporte establecido.

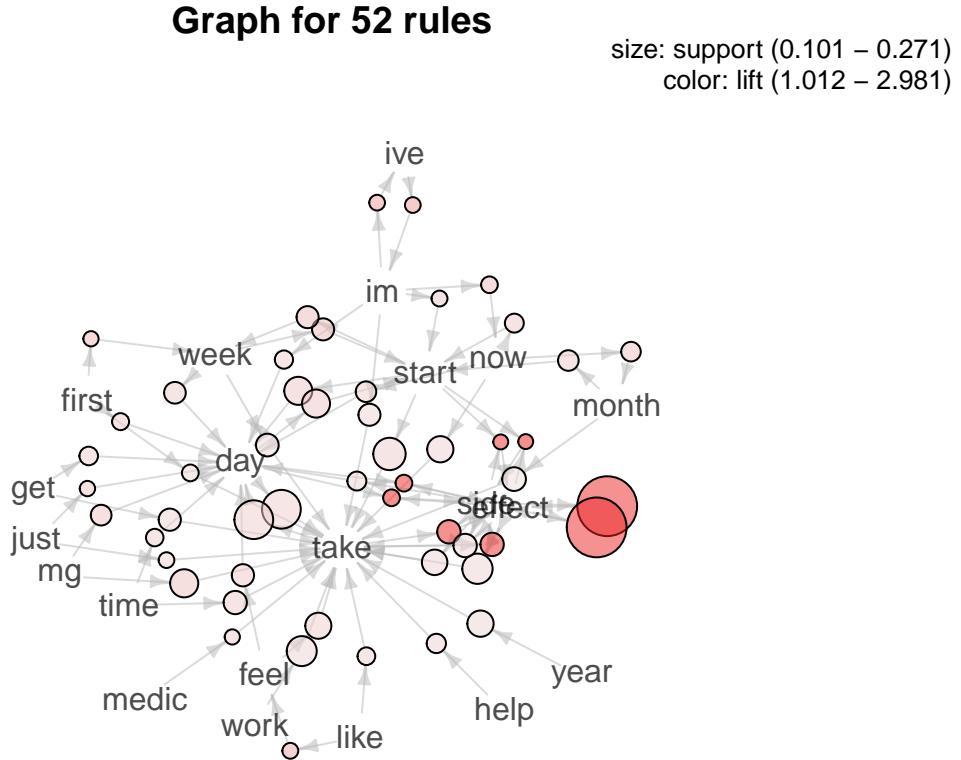
```
library(arules)
library(arulesViz)
reglas_asoc<-apriori(reviews_bd_transaccional, parameter = list(support = 0.1, confidence = 0.4, target = "rules"))
```

```
## Apriori
##
## Parameter specification:
##   confidence minval smax arem  aval originalSupport maxtime support minlen
```

```

##      0.4    0.1    1 none FALSE      TRUE      5    0.1    2
## maxlen target ext
##      10 rules FALSE
##
## Algorithmic control:
##   filter tree heap memopt load sort verbose
##   0.1 TRUE TRUE  TRUE FALSE    2    TRUE
##
## Absolute minimum support count: 500
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[8664 item(s), 5000 transaction(s)] done [0.06s].
## sorting and recoding items ... [70 item(s)] done [0.00s].
## checking subsets of size 1 2 3 done [0.04s].
## writing ... [52 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
```

`reglas_asoc<-sort(reglas_asoc, decreasing=TRUE, by="confidence")`  
`plot(reglas_asoc, method="graph")`



Tal y como podemos comprobar, en este caso se han podido obtener 52 reglas frente a las ocho que se obtuvieron con un valor de confianza del doble que el actual. No obstante, la calidad de estas reglas disminuye, es decir, las asociaciones de los términos pueden no ser del todo lógicas, como por ejemplo *ive* y *im*, que no aportan ningún tipo de información relevante. Es por ello, por lo que en consecuencia, podemos afirmar que un valor razonablemente alto de la confianza puede inducir a obtener relaciones entre términos que nos aporten información útil, mientras que un soporte demasiado elevado puede provocar cierta pérdida

de información al ser tan estrictos con el número de apariciones de cada término.

## Técnicas de clasificación.

La clasificación es un proceso consistente en aprender una determinada función con el objetivo, generalmente, de realizar predicciones futuras. De este modo se entrena un clasificador aportando un conjunto de datos de entrenamiento así como las etiquetas asociadas para que ajuste sus pesos y, posteriormente, sea capaz de predecir las etiquetas de un conjunto de prueba nunca visto anteriormente. En esta sección se entrenarán diferentes clasificadores a partir de la aplicación de las técnicas más populares. Nuestro objetivo principal se encuentra inspirado por el reto de Kaggle[1] en el cual se propone **predecir la enfermedad del paciente en función de su comentario**. Por ello las sucesivas técnicas que se detallan a continuación han sido aplicadas sobre el texto correspondiente a las *reviews* de los pacientes ya preprocessados.

### Clasificación Bayesiana

Es un método probabilístico que determina la clase más probable a la que pertenece un elemento aplicando, para ello, el teorema de Bayes. Es comúnmente utilizado cuando existe una componente aleatoria que provoca que dos elementos iguales estén categorizados en clases diferentes. Este aspecto encaja bastante bien con las críticas de los pacientes puesto que si bien sus términos pueden ser parecidos, en realidad sendos pacientes pueden sufrir dolencias diferentes.

Para aplicar esta técnica nos hemos basado en estos dos tutoriales [11, 12]. Como primer paso se deberá asignar a cada una de las enfermedades disponibles, tanto en el conjunto de entrenamiento como en el de test, una etiqueta numérica única, que sea capaz de identificarla de forma individual. Para ello utilizamos la función *factor* tal y como se muestra en este tutorial [13], la cual realizará este mismo proceso descrito obteniendo mediante *as.numeric* las etiquetas numéricas del 1 al 10 asociadas a cada una de las enfermedades de ambos dataset. Tras visualizar el resultado obtenido podemos afirmar que estas etiquetas numéricas se asignan de menor a mayor en función del orden alfabético, por lo que las dolencias que comienzan por *A* dispondrán de los primeros valores. A continuación obtenemos la matriz de documentos que contiene las *reviews* de los pacientes, como hemos realizado en anteriores técnicas, solo que en este caso solo nos vamos a quedar con aquellos con una frecuencia mínima de cinco apariciones. De este modo reducimos el número de palabras y consideraremos solo las más relevantes. Por último entrenamos el clasificador con el conjunto de entrenamiento y las etiquetas asociadas a las dolencias y mediremos su bondad a través de la tasa de error y de la matriz de confusión.

```
library(tm)
library(e1071)

# Asignamos a cada enfermedad diferente una etiqueta de 0 a 10 y la añadimos como una columna más a amb
trainNCondition<-as.numeric(factor(mini_training_dataset$condition))
testNCondition<-as.numeric(factor(mini_test_dataset$condition))
mini_training_dataset<-cbind(trainNCondition, mini_training_dataset)
mini_test_dataset<-cbind(testNCondition, mini_test_dataset)

# Obtenemos la matriz de los términos de las reviews de los conjuntos de entrenamiento y test.
train_reviews_matriz<-DocumentTermMatrix(train_reviews_corpus)
test_reviews_matriz<-DocumentTermMatrix(test_reviews_corpus)
# Nos quedamos con los términos más frecuentes (con un mínimo número de apariciones de 5)
terminos_frecuentes<-findFreqTerms(train_reviews_matriz, 5)
# Obtenemos de nuevo la matriz con los términos más frecuentes extraídos anteriormente tanto para train
train_reviews_frec_matriz<-DocumentTermMatrix(train_reviews_corpus, control=list(dictionary=terminos_frec
test_reviews_frec_matriz<-DocumentTermMatrix(test_reviews_corpus, control=list(dictionary=terminos_frec
# Función que transforma la frecuencia de los términos a una clase binaria: están o no están.
```

```

frecuencia_binaria<-function(x) { y<-ifelse(x > 0, "Yes", "No") }
# Obtenemos la matriz de los términos definiendo para cada uno si pertenece a la clase "No" (no está) o
train_reviews_matriz_bin<-apply(train_reviews_frec_matriz, 2, frecuencia_binaria)
test_reviews_matriz_bin<-apply(test_reviews_frec_matriz, 2, frecuencia_binaria)
# Entrenamos el modelo con Naive Bayes binario
clasificador_naive<-naiveBayes(train_reviews_matriz_bin, as.factor(mini_training_dataset$trainNCondition))
# Obtenemos las predicciones que hace el modelo sobre el conjunto de entrenamiento y de test.
train_pred<-predict(clasificador_naive, train_reviews_matriz_bin)
test_pred<-predict(clasificador_naive, test_reviews_matriz_bin)
# Calculamos sus errores
error_train<-mean(train_pred!=mini_training_dataset$trainNCondition)
error_test<-mean(test_pred!=mini_test_dataset$testNCondition)
# Mostramos los datos más relevantes.
cat("Orden de asignación de etiquetas 1-10\n")

## Orden de asignación de etiquetas 1-10

cat(levels(factor(mini_training_dataset$condition)))

## Acne ADHD Anxiety Bipolar Disorde Birth Control Depression Insomnia Obesity Pain Weight Loss

test_matriz_confusion_naive<-table(test_pred, mini_test_dataset$testNCondition)
cat("\n\nError en train: ", error_train*100,"% \nError en test: ",error_test*100,"%")

## 
## 
## Error en train: 15.18 %
## Error en test: 26.44 %

cat("\n\nMatriz de confusión en test\n")

## 
## 
## Matriz de confusión en test

print(test_matriz_confusion_naive)

## 
## test_pred   1   2   3   4   5   6   7   8   9   10
##      1 428   0   0   1   9   1   0   1   0   0
##      2   1 393   9   6   1  13   2   3   2   3
##      3   0   6 294  24   5  77  12   5   9   3
##      4   2  10  18 349   2  60   8   3   2   0
##      5  39   1   0   1 464   1   0   0   0   2
##      6   5  20  89  37   4 272   5  11  11   7
##      7   2  20  39  37   2  37 446   7  16  11
##      8   1   5   0   3   1   5   0 277   1 166
##      9  22  39  50  39  11  31  27  28 459  12
##     10   0   6   1   3   1   3   0 165   0 296

```

Tal y como podemos comprobar ambas tasas de error no son competitivas pero sí son razonablemente aceptables, teniendo en cuenta que el problema de clasificación planteado no es nada sencillo, puesto que el clasificador debe elegir una de las diez enfermedades posibles basándose solo en el contenido de las críticas preprocesadas de los pacientes. Estas, en la gran mayoría de ocasiones, no permiten al cien por cien conocer cuál es la dolencia a la que se enfrenta dicha persona puesto que suelen estar más enfocadas a la opinión suministrada en función de la efectividad del medicamento más que de la descripción de la propia enfermedad. Si analizamos la **matriz de confusión** podemos comprobar como en la mayoría de los casos no existe un número muy pronunciado de falsos positivos, exceptuando en la etiqueta 8, cuya enfermedad asociada *Insomnia* es bastante confundida con la relacionada a la etiqueta 10 que es *Weight Loss*, y viceversa. Esto nos lleva a pensar que los comentarios de los pacientes que padecen ambas enfermedades utilizan términos muy parecidos que confunden al clasificador.

## K-NN

Se trata de una de las técnicas predictivas que se encuentran dentro de la clasificación basada en instancias. Este algoritmo denominado **K-vecinos más cercanos** selecciona los  $k$  elementos más parecidos a un determinado ejemplo y les asigna la clase más frecuente. Si bien este método está pensado para trabajar con variables numéricas, vamos a aplicarlo de nuevo a la predicción de la enfermedad en base a las *reviews* de los pacientes, como con la técnica anterior.

Comenzamos esta sección probando la librería *kknn* vista en clase para entrenar un clasificador utilizando la técnica que nos ocupa en cuestión. El proceso que llevamos a cabo fue muy similar al anterior, exceptuando la extracción de los términos más comunes, es decir, directamente probamos a entrenar el clasificador con todos los términos de las *reviews* del conjunto de entrenamiento. Sin embargo, el rendimiento fue súmamente diferente puesto que obtuvimos una **tasa de error de un 99%**. Intentamos mejorar este resultado pero no encontramos ejemplos en internet en los que utilizasen esta librería en concreto aplicada a texto, si no que hacían uso de una denominada *class* [13]. Por tanto, decidimos probar suerte con la función *knn* almacenada en esta biblioteca inspirándonos, para ello, en este tutorial [14]. Al intentar entrenar el clasificador con el conjunto de entrenamiento y validar con el conjunto de test nos arrojaba un error acerca de las dimensiones diferentes que tenían las matrices de términos de sendos conjuntos, y es que si bien disponen del mismo número de filas, en cada conjunto de datos hay un número diferente de términos, por lo que las columnas no coincidían. Para solucionar este problema el tutorial mencionado anteriormente propone utilizar la técnica de **validación cruzada** para dividir el conjunto de entrenamiento en un conjunto para entrenar y otro para validar. De este modo nos aseguramos que en sendos conjuntos se encuentran todos los términos. De esta forma sí hemos sido capaces de entrenar un clasificador con esta técnica.

```
library(tm)
library(dplyr)
library(class)

# Obtenemos la matriz de términos del conjunto de entrenamiento.
train_reviews_matriz<-DocumentTermMatrix(train_reviews_corpus)
# Obtenemos los términos más frecuentes.
terminos_frecuentes_train<-findFreqTerms(train_reviews_matriz, 5)
# Obtenemos la nueva matriz de términos más frecuentes.
train_reviews_frec_matriz<-DocumentTermMatrix(train_reviews_corpus, control=list(dictionary=terminos_frecuentes_train))
# Convertimos a dataframe el conjunto de datos de entrenamiento para pasárselos al algoritmo.
train_df<-as.data.frame(data.matrix(train_reviews_frec_matriz), stringsAsFactors = FALSE)
# Añadimos la columna de clasificación de las enfermedades.
train_df<-cbind(train_df, trainNCondition)
# Obtenemos las etiquetas de las enfermedades para entrenar el clasificador.
etiquetas<-train_df[, "trainNCondition"]
# Obtenemos el modelo con el que vamos a entrenar el clasificador sin la columna de etiquetas, puesto que
modelo<-train_df[,-colnames(train_df) %in% "trainNCondition"]
# Validación cruzada para dividir el conjunto de train en un conjunto de entrenamiento y otro de validación.
```

```

train<-sample(nrow(train_df), ceiling(nrow(train_df) * .70))
test<-(1:nrow(train_df))[-train]
# Obtenemos las predicciones del conjunto de validación
knn.pred<-knn(modelo[train, ], modelo[test, ], etiquetas[train], k=1)
# Obtenemos la matriz de confusión y la tasa de error
m_conf<-table("Predicciones" = knn.pred, Real = etiquetas[test])
m_conf

##          Real
## Predicciones 1 2 3 4 5 6 7 8 9 10
##           1 59 6 4 2 20 2 4 3 3 3
##           2 1 19 4 3 5 4 3 1 1 0
##           3 20 33 56 27 15 27 29 9 16 14
##           4 8 16 12 35 12 22 22 6 8 14
##           5 18 1 0 1 46 2 0 2 2 4
##           6 9 29 19 25 16 40 11 7 4 4
##           7 10 22 19 23 6 14 78 9 11 13
##           8 9 13 8 12 7 13 1 60 2 70
##           9 14 15 14 9 13 8 16 12 106 12
##          10 10 5 2 3 12 4 1 24 1 31

test_error<-100-(sum(diag(m_conf))/length(test) * 100)
cat("\nError en el conjunto de prueba:", test_error, "%")

```

##  
## Error en el conjunto de prueba: 64.66667 %

Tal y como podemos comprobar, con  $k=1$ , es decir, considerando un único vecino más cercano la **tasa de error sobre el conjunto de prueba es mayor del 60%**. Si analizamos la matriz de confusión generada podemos observar que en la mayoría de los errores se concentran en torno a las enfermedades cuyas etiquetas asociadas son 2, 3, 6, 7 y 10. Esto significa que el clasificador no es capaz de distinguir correctamente entre estas dolencias en base a las *reviews* de los pacientes. En base a estos resultados podemos determinar que con esta configuración, el clasificador entrenado no dispone de una buena capacidad de generalización y por ello falla en más de la mitad de ocasiones. Investigando acerca del valor óptimo para  $k$ , es decir, encontrar el número de vecinos adecuado a considerar encontramos una idea en esta fuente [15]. Consiste en entrenar varios clasificadores mediante la técnica KNN utilizando validación cruzada y distintos valores para  $k$ . De este modo compara los resultados obtenidos en todos los casos y nos mostrará aquel valor de  $k$  para el cual se obtiene un mejor resultado. Como este procedimiento es computacionalmente muy costoso, solo lo hemos ejecutado una vez y tras el siguiente código se proporcionará la captura con los resultados obtenidos.

```

library(caret)
set.seed(400)
# Utiliza validación cruzada para probar diversos valores de k
ctrl<-trainControl(method="repeatedcv", repeats=3)
knnFit<-train(trainNCondition ~ ., data=train_df, method="knn", trControl=ctrl, preProcess=c("center", "scale"))
knnFit

```

Tal y como podemos observar, de entre todos los experimentos realizados el clasificador entrenado con  $k=5$  es el que mejor precisión ha obtenido. A continuación repetiremos el mismo proceso anterior solo que en lugar de  $k=1$  establecemos el nuevo valor óptimo descubierto para esta variable.

```

3500 samples
2552 predictors

Pre-processing: centered (2552), scaled (2552)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 3151, 3150, 3150, 3150, 3150, 3150, ...
Resampling results across tuning parameters:

k    RMSE      Rsquared     MAE
5    2.741332  0.10733977 2.252912
7    2.737911  0.09484553 2.288412
9    2.719579  0.10126438 2.290953

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was k = 9.

```

Figure 6: Validación cruzada para encontrar el k óptimo.

```

# Importamos la librería con la que aplicaremos el algoritmo KNN.
library(class)
# Obtenemos las predicciones del conjunto de validación
knn.pred<-knn(modelo[train, ], modelo[test, ], etiquetas[train], k=5)
# Obtenemos la matriz de confusión y la tasa de error
m_conf<-table("Predicciones" = knn.pred, Real = etiquetas[test])
m_conf

```

```

##          Real
## Predicciones 1 2 3 4 5 6 7 8 9 10
## 1            78 5 1 1 9 1 0 1 0 3
## 2            0 14 0 3 1 1 0 2 1 0
## 3            24 36 79 30 17 34 33 11 17 19
## 4            2 16 6 23 11 14 8 9 3 6
## 5            4 1 0 1 43 1 0 0 1 1
## 6            9 21 7 26 16 32 8 10 4 12
## 7            4 25 17 29 6 20 89 10 6 12
## 8            5 9 5 6 9 11 0 62 3 72
## 9            26 32 20 18 36 20 27 13 119 20
## 10           6 0 3 3 4 2 0 15 0 20

```

```

test_error<-100-(sum(diag(m_conf))/length(test) * 100)
cat("\nError en el conjunto de prueba:", test_error, "%")

```

```

##
## Error en el conjunto de prueba: 62.73333 %

```

Tal y como se puede comprobar el clasificador entrenado, considerando los cinco vecinos más cercanos, dispone de una **tasa de error sobre el conjunto de prueba también superior al 60%**, aunque menor que en el caso anterior. Si bien ha conseguido mejorar un poco con respecto al resultado anterior con  $k=1$ , podemos confirmar que esta técnica no nos aporta un clasificador capaz de predecir la enfermedad en función de las *reviews* de los pacientes, por lo que no es el mejor algoritmo para aplicarlo a nuestros datos. Si analizamos la matriz de confusión, podemos observar que la mayoría de errores se concentran a partir de la segunda etiqueta. Esto significa que el clasificador no es capaz de distinguir estas enfermedades de forma correcta en base a las *reviews* de los pacientes.

Posteriormente, para asegurarnos que realmente el valor de  $k$  óptimo es 5 hicimos una serie de pruebas variando el valor de este, que no vamos a adjuntar en este documento puesto que esta técnica es muy costosa

computacionalmente de aplicar. Sin embargo, pudimos confirmar que  $k=5$  es el mejor valor que podemos establecer para esta técnica puesto que ni con  $k=3$ , con el que obtuvimos un error de 64%, ni con  $k=10$  que proporcionaba un error 63.73%, obtuvimos una tasa de error menor.

## Árboles de decisión

Los árboles de decisión se utilizan para la clasificación debido a su facilidad tanto de interpretación de los resultados como de la extracción de las reglas que ayudan a entrenar un clasificador. Se trata de un algoritmo muy potente y especialmente dedicado a trabajar con dataset complejos y con un gran número de datos. Es por ello por lo que, como con técnicas anteriores, vamos a aplicarla a las *reviews*. El objetivo sigue siendo el mismo: intentar predecir las enfermedades en base a las críticas de los pacientes ya preprocessadas.

Para ello hemos investigado un poco acerca del proceso a seguir para entrenar un clasificador mediante árboles aplicado a texto y según hemos podido ver en esta fuente [16] el procedimiento es muy similar al de la técnica de Naive Bayes. En primer lugar obtenemos las matrices de términos así como la frecuencia de los mismos tanto para el conjunto de entrenamiento como para el de prueba. Estos valores serán los que se tendrá en cuenta a la hora de buscar indicios en las *reviews* que nos permitan predecir a qué enfermedad están haciendo alusión. Por último convertimos ambas matrices a dataframe con los que poder trabajar y les añadimos las respectivas columnas de clasificación de las enfermedades a sendos conjuntos de datos. Para entrenar el clasificador vamos a hacer uso de la librería *rpart* vista en clase y calcularemos tanto la matriz de confusión como las tasas de error tanto en entrenamiento como en validación.

```

library(tm)
library(rpart)

# Obtenemos la matriz de los términos de las reviews de los conjuntos de entrenamiento y test.
train_reviews_matriz<-DocumentTermMatrix(train_reviews_corpus)
test_reviews_matriz<-DocumentTermMatrix(test_reviews_corpus)
# Nos quedamos con los términos más frecuentes (con un mínimo número de apariciones de 5)
terminos_frecuentes<-findFreqTerms(train_reviews_matriz, 5)
# Obtenemos de nuevo la matriz con los términos más frecuentes extraídos anteriormente tanto para train
train_reviews_frec_matriz<-DocumentTermMatrix(train_reviews_corpus, control=list(dictionary=terminos_frecuentes))
test_reviews_frec_matriz<-DocumentTermMatrix(test_reviews_corpus, control=list(dictionary=terminos_frecuentes))
# Convertimos a dataframe el conjunto de datos de entrenamiento para pasárselos al algoritmo.
train_freq<-as.data.frame(data.matrix(train_reviews_frec_matriz), stringsAsFactors = FALSE)
test_freq<-as.data.frame(data.matrix(test_reviews_frec_matriz), stringsAsFactors = FALSE)
# Añadimos la columna de clasificación de enfermedades al conjunto de test para luego
# realizar la predicción y evaluar el clasificador.
train_freq<-cbind(train_freq, trainNCondition)
test_freq<-cbind(test_freq, testNCondition)
# Entrenamos el modelo árboles de decisión.
clasificador_arboles<-rpart(trainNCondition~., train_freq, method="class")
# Mostramos los resultados.
printcp(clasificador_arboles)

## 
## Classification tree:
## rpart(formula = trainNCondition ~ ., data = train_freq, method = "class")
## 
## Variables actually used in tree construction:
##   [1] acn      adderal adhd    anxieti bipolar lbs      pain     period
##   [9] pound    skin     sleep
## 
## Root node error: 4500/5000 = 0.9

```

```

## n= 5000
##          CP nsplit rel error xerror      xstd
## 1 0.081778      0 1.00000 1.02400 0.0042238
## 2 0.050000      2 0.83644 0.85533 0.0066148
## 3 0.039444      3 0.78644 0.79200 0.0071097
## 4 0.025444      7 0.62867 0.62778 0.0077901
## 5 0.013111      9 0.57778 0.58489 0.0078458
## 6 0.011778     10 0.56467 0.57067 0.0078538
## 7 0.010222     11 0.55289 0.56467 0.0078557
## 8 0.010000     12 0.54267 0.55933 0.0078566

# Calculamos el error y la matriz de confusión para entrenamiento.
train_predd<-predict(clasificador_arboles, train_freq, type="class")
train_confusion<-table(train_freq$trainNCondition, train_predd)
error_train<-sum(diag(train_confusion))/nrow(train_freq)
# Calculamos el error y la matriz de confusión para test.
test_predd<-predict(clasificador_arboles, test_freq, type="class")
test_confusion<-table(test_freq$testNCondition, test_predd)
test_confusion

##      test_predd
##      1   2   3   4   5   6   7   8   9   10
## 1  450   0   1   0   6  35   1   0   7   0
## 2   5 195  29   2   7 178  41   3  20  20
## 3   0   2 248   5   8 126  78   1  28   4
## 4   6   8  31 144   8 190  60  14  17  22
## 5 107   0  10   0 212 104   3   6  37  21
## 6   1   5 101  13  18 242  69   9  24  18
## 7   1   1   5   3  11 120 333   1  21   4
## 8   0   0   5   1   1 136  20 123  23 191
## 9   2   0   0   0  11  81   7   0 399   0
## 10  3   1   2   0   6 100  22 111  20 235

error_test<-sum(diag(test_confusion))/nrow(test_freq)
cat("\nError en entrenamiento:",error_train*100,"% \nError en test:",error_test*100,"%")

## 
## Error en entrenamiento: 51.16 %
## Error en test: 51.62 %

```

Tal y como podemos observar las tasas de error son bastante altas, por lo que el clasificador no dispone de una buena capacidad de generalización, y por tanto, no le permite diferenciar qué enfermedad está asociada a cada *review*. Si además observamos la matriz de confusión con respecto al conjunto de test, podemos visualizar que la mayoría de falsos positivos se encuentran a partir de la etiqueta 2, por lo que a partir de la segunda dolencia ya no es capaz de diferenciarlas claramente. Continuamos investigando cómo mejorar el rendimiento de este tipo de clasificador y encontramos varios tutoriales como este [17], en el que explican una serie de parámetros de control para poder ajustar el entrenamiento del clasificador [18]. El primero de ellos es *minsplit* con el cual le indicaremos el número mínimo de registros que debe tener un nodo para poder dividirse. Hemos probado con diversos valores y hemos comprobado que a mayor valor, peor tasa de error en sendos conjuntos. La conclusión a la que llegamos es que si establecemos un valor demasiado alto

esto no permitirá dividir el nodo en más caminos y por lo tanto podemos perder la solución óptima. Otro valor importante es *maxdepth*, con el que establecemos la profundidad máxima del árbol. Del mismo modo también hemos variado este valor y en función de los resultados así como del costo computacional, el mejor valor para nuestro dataset es que el árbol tenga profundidad 3.

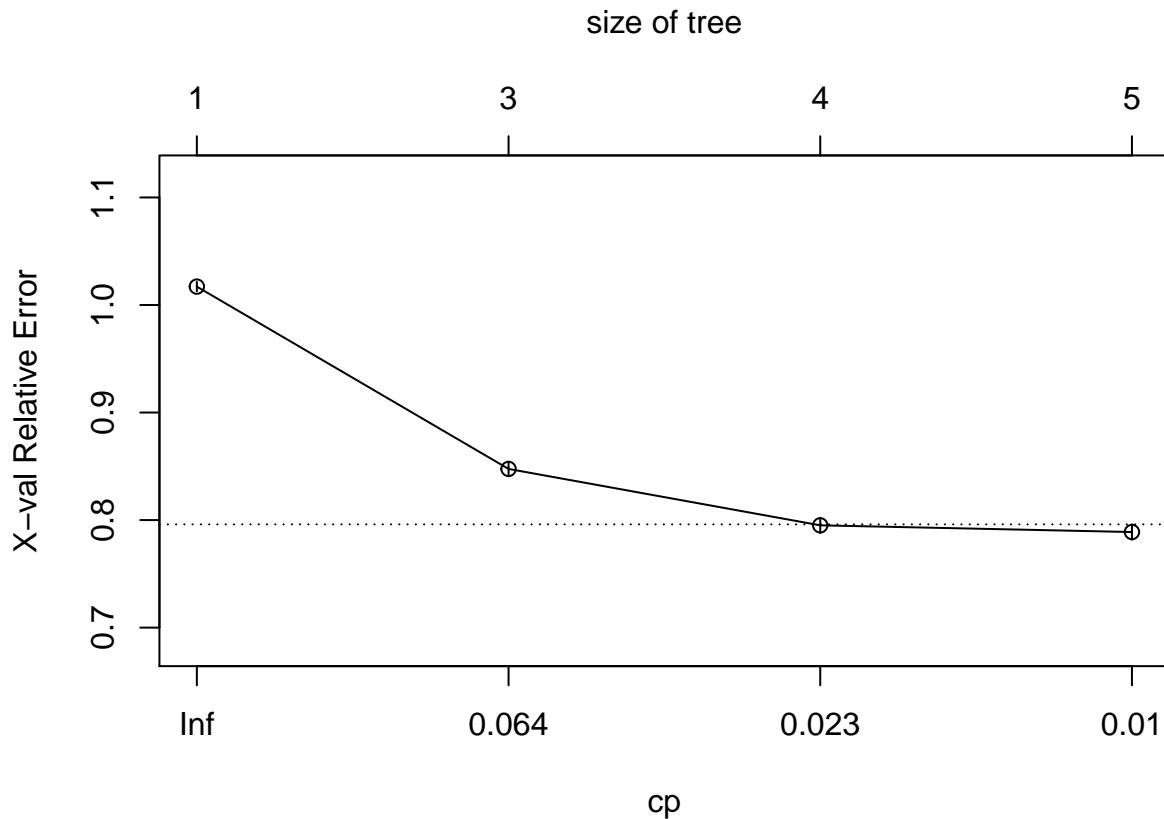
```
# Entrenamos el modelo árboles de decisión.
control<-rpart.control(minsplit=4, maxdepth=3)
clasificador_arboles<-rpart(trainNCondition~, train_freq, method="class", control=control)
# Calculamos el error y la matriz de confusión para entrenamiento.
train_predd<-predict(clasificador_arboles, train_freq, type="class")
train_confusion<-table(train_freq$trainNCondition, train_predd)
error_train<-sum(diag(train_confusion))/nrow(train_freq)
# Calculamos el error y la matriz de confusión para test.
test_predd<-predict(clasificador_arboles, test_freq, type="class")
test_confusion<-table(test_freq$testNCondition, test_predd)
test_confusion

##      test_predd
##      1   2   3   4   5   6   7   8   9   10
##  1 391 101  0   0   1   0   0   0   7   0
##  2   1 459  0   0   0   0   0   0  20  20
##  3   0 468  0   0   0   0   0   0  28   4
##  4   2 459  0   0   0   0   0   0  17  22
##  5 103 274  0   0  65  0   0   0  37  21
##  6   0 458  0   0   0   0   0   0  24  18
##  7   0 473  0   0   2   0   0   0  21   4
##  8   0 286  0   0   0   0   0   0  23 191
##  9   0  91  0   0  10  0   0   0 399   0
## 10   0 244  0   0   1   0   0   0  20 235

error_test<-sum(diag(test_confusion))/nrow(test_freq)
cat("\nError en entrenamiento:",error_train*100,"% \nError en test:",error_test*100,"%")

##
## Error en entrenamiento: 30.14 %
## Error en test: 30.98 %

plotcp(clasificador_arboles)
```



Tal y como podemos observar, ambas tasas de error se han reducido considerablemente hasta alcanzar niveles más normalizados. Asimismo, la matriz de confusión respecto al conjunto de prueba también ha mejorado considerablemente puesto que, tal y como se puede apreciar, el número de confusiones y enfermedades mal etiquetadas es mucho menor. Si observamos, a continuación, el gráfico que muestra la evolución del aprendizaje del árbol podemos observar que el valor óptimo para **cp=0.023**, es decir, si establecemos este valor conseguiremos que el rendimiento del clasificador sea óptimo así como su tasa de acierto puesto que no continuará explorando un camino que no mejore más allá de dicho valor. A continuación entrenamos un tercer clasificador fijando este parámetro.

```
# Entrenamos el modelo árboles de decisión.
control<-rpart.control(minsplit=4, maxdepth=3, cp=0.023)
clasificador_arboles<-rpart(trainNCondition~, train_freq, method="class", control=control)
# Calculamos el error y la matriz de confusión para entrenamiento.
train_predd<-predict(clasificador_arboles, train_freq, type="class")
train_confusion<-table(train_freq$trainNCondition, train_predd)
error_train<-sum(diag(train_confusion))/nrow(train_freq)
# Calculamos el error y la matriz de confusión para test.
test_predd<-predict(clasificador_arboles, test_freq, type="class")
test_confusion<-table(test_freq$testNCondition, test_predd)
test_confusion
```

```
##      test_predd
##      1   2   3   4   5   6   7   8   9   10
##  1 391 101  0  0  0  0  0  0  8  0
##  2 1459  0  0  0  0  0  0 20 20
##  3 468   0  0  0  0  0  0 28  4
```

```

##   4    2 459    0    0    0    0    0    0  17  22
##   5 103 274    0    0    0    0    0    0 102  21
##   6    0 458    0    0    0    0    0    0  24  18
##   7    0 473    0    0    0    0    0    0  23    4
##   8    0 286    0    0    0    0    0    0  23 191
##   9    0  91    0    0    0    0    0    0 409    0
##  10    0 244    0    0    0    0    0    0  21 235

error_test<-sum(diag(test_confusion))/nrow(test_freq)
cat("\nError en entrenamiento:",error_train*100,"% \nError en test:",error_test*100,"%")

##
## Error en entrenamiento: 29.22 %
## Error en test: 29.88 %

```

Tal y como podemos observar, se ha vuelto a reducir un poco ambos errores. Como conclusiones para terminar esta sección podemos afirmar que si bien hemos conseguido mejorar su rendimiento de forma considerable, esta técnica no proporciona buenos resultados en relación a nuestro objetivo principal, que era predecir enfermedades en función de las *reviews* de los pacientes.

## Random Forest

Esta técnica se presenta como una variante de otra denominada *Bagging* por la cual se introduce cierta aleatoriedad con el objetivo de mejorar la clasificación de los elementos dado un dataset. Para ello genera un gran número de árboles independientes entre sí y posteriormente combina sus resultados realizando un promedio entre todos ellos para etiquetar a cada uno de los elementos. A continuación, procedemos a aplicar este algoritmo como lo venimos haciendo con las anteriores técnicas de clasificación, pues nuestro propósito es averiguar cuál es la mejor técnica de clasificación que proporciona unos buenos resultados prediciendo la enfermedad a partir de la *review* del paciente.

Para entrenar un clasificador con esta técnica vamos a hacer uso de la librería *randomForest* vista en clase, en particular con la función de igual nombre **randomForest**[19]. Más concretamente, procedemos a utilizar la versión de dicha función en la cual se puede expresar la fórmula que definirá el modelo que deseamos entrenar. En un primer intento encontramos un error debido a que la matriz de términos del conjunto de entrenamiento y la del conjunto de prueba no disponen de los mismos términos, y por tanto, el clasificador no puede obtener las predicciones de términos que no ha visto previamente. Es por ello por lo que, en un segundo intento, aplicamos de nuevo la **validación cruzada** para dividir nuestro conjunto de entrenamiento original en un conjunto de *train* y otro de *test* para validar, posteriormente, el clasificador entrenado.

```

library(tm)
# Obtenemos la matriz de términos del conjunto de entrenamiento.
train_reviews_matriz<-DocumentTermMatrix(train_reviews_corpus)
# Obtenemos los términos más frecuentes.
terminos_frecuentes_train<-findFreqTerms(train_reviews_matriz, 5)
# Obtenemos la nueva matriz de términos más frecuentes.
train_reviews_frec_matriz<-DocumentTermMatrix(train_reviews_corpus, control=list(dictionary=terminos_frecuentes_train))
# Convertimos a dataframe el conjunto de datos de entrenamiento para pasárselos al algoritmo.
train_df<-as.data.frame(data.matrix(train_reviews_frec_matriz), stringsAsFactors = FALSE)
# Añadimos la columna de clasificación de las enfermedades.
train_df<-cbind(train_df, trainNCondition)
# Modificamos el nombre a las columnas para que el algoritmo pueda trabajar con los datos
colnames(train_df) <- paste(colnames(train_df), "_c", sep = "")

```

```

# Validación cruzada para dividir el conjunto de train en un conjunto de entrenamiento y otro de validación
train_vc<-train_df[1:3500, ]
test_vc<-train_df[3501:5000, ]
# Importamos la librería con la que aplicaremos la técnica Random Forest
library(randomForest)

## Warning: package 'randomForest' was built under R version 3.6.2

# Entrenamos el clasificador con un máximo de 100 árboles.
clasificador_rf<-randomForest(formula=train_vc$trainNCondition_c~, data=train_vc, ntree=100)
# Calculamos el error y la matriz de confusión para entrenamiento.
train_predd<-predict(clasificador_rf, train_vc, type="response")
train_confusion<-table(train_vc$trainNCondition_c, train_predd)
error_train<-sum(diag(train_confusion))/nrow(train_vc)
# Calculamos el error y la matriz de confusión para test.
test_predd<-predict(clasificador_rf, test_vc, type="response")
test_confusion<-table(test_vc$trainNCondition_c, test_predd)
error_test<-sum(diag(test_confusion))/nrow(test_vc)
clasificador_rf

## 
## Call:
##   randomForest(formula = train_vc$trainNCondition_c ~ ., data = train_vc,      ntree = 100)
##   Type of random forest: regression
##   Number of trees: 100
##   No. of variables tried at each split: 850
##
##   Mean of squared residuals: 2.682763
##   % Var explained: 67.18

cat("\nError en entrenamiento:",error_train*100,"% \nError en test:",error_test*100,"%")

##
## Error en entrenamiento: 0.6285714 %
## Error en test: 0.2 %

```

Como podemos comprobar el algoritmo ha aplicado regresión para poder entrenar el clasificador y así determinar si una determinada *review* hace referencia a una enfermedad u a otra. En base a las tasas de error obtenidas podemos determinar que el clasificador entrenado dispone de una muy buena capacidad de generalización puesto que ambos valores son mínimos, por lo que afirmamos que este modelo es capaz de diferenciar sin problema las diez enfermedades que se encuentran en nuestro dataset en función de la crítica que escribe el paciente.

Como conclusión para finalizar esta sección, podemos determinar que de todas las técnicas de clasificación probadas **Random Forest** es la que mejor resultados ha proporcionado entrenando un clasificador con una muy buena capacidad de predicción.

## Regresión para predecir la efectividad del fármaco por su puntuación.

El objetivo en esta primera sección consiste en intentar predecir la efectividad del medicamento en función de su puntuación, de 0 a 10, asignada por el paciente al que se le ha recetado. Para ello entrenaremos

diversos modelos de regresión para comprobar cuál es el que mejor resultados puede proporcionar. Con el fin de llevar a cabo esta idea, necesitamos añadir una nueva columna de clasificación binaria a nuestro dataset convirtiendo las puntuaciones de los medicamentos, que se encuentran en un rango entre 0 y 10, a dos valores binarios que representen la efectividad del tratamiento. El intervalo [0,4] corresponderá al valor 0 y por lo tanto representará que para un paciente en particular el medicamento no ha sido de ayuda, mientras que el intervalo [5-10] se corresponderá con el valor 1 que representará la eficacia del tratamiento.

## Regresión lineal simple.

En esta sección entrenaremos el modelo más sencillo posible aplicando, para ello, la **regresión lineal simple**. La variable dependiente será el campo ratingBinaryLabel, el cual contiene las etiquetas de clasificación de los fármacos explicadas anteriormente, mientras que la variable predictora será la categoría que contiene las puntuaciones originales de los tratamientos: *rating*.

```
# Recorremos las filas del conjunto de entrenamiento y comprobamos los valores en los intervalos definidos
for (i in 1:length(mini_training_dataset$rating)){
  if (mini_training_dataset$rating[i] < 5)
    mini_training_dataset$ratingLabel[i]<-0
  else
    mini_training_dataset$ratingLabel[i]<-1
}
# Realizamos el mismo procedimiento para el conjunto de test.
for (i in 1:length(mini_test_dataset$rating)){
  if (mini_test_dataset$rating[i] < 5)
    mini_test_dataset$ratingLabel[i]<-0
  else
    mini_test_dataset$ratingLabel[i]<-1
}
# Especificamos la variable a predecir, que en nuestro caso será el campo binario relacionado con la eficacia
modelo = mini_training_dataset$ratingLabel ~ mini_training_dataset$rating
# Entrenamos el modelo.
regresion_lineal = lm(modelo, mini_training_dataset)
# Comprobamos los resultados del modelo.
summary(regresion_lineal)

##
## Call:
## lm(formula = modelo, data = mini_training_dataset)
##
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -0.40112 -0.10550 -0.04893  0.12929  0.48148 
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)    
## (Intercept)             -0.0684667  0.0065808  -10.4   <2e-16 ***
## mini_training_dataset$rating  0.1173970  0.0008287   141.7   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1821 on 4998 degrees of freedom
## Multiple R-squared:  0.8006, Adjusted R-squared:  0.8006 
## F-statistic: 2.007e+04 on 1 and 4998 DF,  p-value: < 2.2e-16
```

Tal y como podemos comprobar en el resumen estadístico mostrado en primer lugar, el modelo predictivo es confiable en tanto en cuanto su p-value tanto en el origen como en la pendiente es menor que 0.05. Además, observando el valor del atributo *Multiple R*<sup>2</sup> podemos determinar que el modelo es capaz de predecir hasta un 80% de valores del campo *ratingLabel* mediante el atributo *rating*. Para ello aplica la siguiente ecuación, que se obtiene del *intercepto* y de la pendiente calculados previamente: *rating\_binario* = -0.0684667 + 0.1173970 \* *rating\_original*. Su interpretación reside en que por cada incremento en una unidad de la variable *rating* se corresponde a 0.117 del valor de *ratingLabel*.

Sin embargo, para medir la bondad del modelo de una forma más precisa y tradicional, procedemos a definir la siguiente función que es capaz de calcular las predicciones realizadas por el modelo entrenado previamente, tanto para el conjunto de entrenamiento como para el de test, para posteriormente calcular ambas **tasas de error**. Encapsulamos el código referente a este procedimiento en una función puesto que así podremos reutilizarlo más adelante en los sucesivos modelos de regresión que aplicaremos a nuestro dataset.

```
# Función que recibe el modelo de regresión entrenado para realizar las predicciones tanto del conjunto
evaluar_modelo_regresion<-function(modelo, datos_train, datos_test) {
  # Calculamos la probabilidad de cada muestra de pertenecer a una clase u a otra.
  train_probab<-predict(modelo, datos_train, type="response")
  test_probab<-predict(modelo, datos_test, type="response")
  # Inicializamos todas las etiquetas a 0.
  train_predicc<-rep(0, length(train_probab))
  test_predicc<-rep(0, length(test_probab))

  # En función de las probabilidades calculadas anteriormente clasificaremos el resto de muestras perten
  train_predicc[train_probab >= .5] = 1
  test_predicc[test_probab >= .5] = 1
  # Mostramos la matriz de confusión que nos aportará más información acerca de los falsos positivos.
  cat("\nMatriz de confusión.\n")
  print(table(pred=test_predicc, real=datos_test$ratingLabel))

  # Ahora calculamos la tasa de error en % para cada conjunto.
  train_error = mean(train_predicc != datos_train$ratingLabel)
  test_error = mean(test_predicc != datos_test$ratingLabel)
  cat("\nTasa de error en entrenamiento:",train_error*100,"%")
  cat("\nTasa de error en prueba:",test_error*100,"%")
}

evaluar_modelo_regresion(regresion_lineal, mini_training_dataset, mini_test_dataset)

## 
## Matriz de confusión.
##     real
## pred   0    1
##   0 222  831
##   1  836 3111
##
## Tasa de error en entrenamiento: 0 %
## Tasa de error en prueba: 33.34 %
```

Tal y como podemos comprobar, la **tasa de error sobre el conjunto de entrenamiento**, como era de esperar es bastante pequeña. Sin embargo, el hecho de que sea 0% no es favorable puesto que, por el contrario, **el error en el conjunto de prueba** es considerablemente alto, por lo tanto podemos reflexionar acerca de si el modelo se ha ajustado demasiado a los datos de entrenamiento y por ello su capacidad de generalización es menor. Asimismo, si analizamos la matriz de confusión, podemos comprobar que existe

una alta tasa de errores al clasificar muestras en la categoría 1 mientras que en realidad pertenecen a la clase cuya etiqueta es 0, y viceversa. Con el objetivo de evaluar la calidad del modelo entrenado procedemos a representar la **curva ROC** así como estimar el área existente debajo de la misma.

```
# Función que realiza las predicciones con la función del paquete asociado a la curva ROC para posteriormente calcular el área debajo de la curva
library("ROCR")

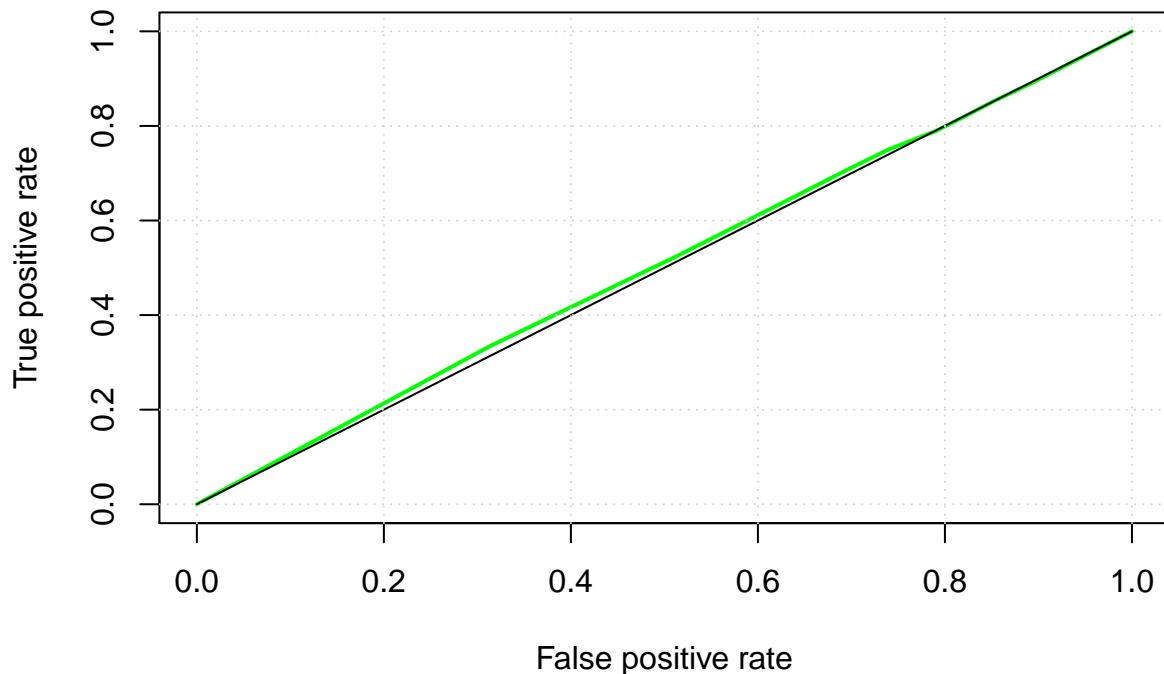
## Warning: package 'ROCR' was built under R version 3.6.2

## Warning: package 'gplots' was built under R version 3.6.2

curvaROC<-function(modelo, etiquetas) {
  # Obtenemos las predicciones.
  predicciones<-prediction(modelo, etiquetas)
  # Calculamos el rendimiento del modelo teniendo en cuenta los falsos positivos y los aciertos.
  curva<-performance(predicciones, "tpr", "fpr")
  # Dibujamos la curva.
  plot(curva, col="green", add=FALSE, main="Curva ROC. Regresión Lineal.", lwd = 2)
  segments(0, 0, 1, 1, col='black')
  grid()
  # Calculamos el área debajo de la curva
  curva.area = performance(predicciones, "auc")
  cat("\nEl área bajo la curva ROC es", curva.area@y.values[[1]]*100,"%\n")
}

# Curva ROC para el conjunto de prueba
test_probab = predict(regresion_lineal, mini_test_dataset, type=c("response"))
curvaROC(test_probab, mini_test_dataset$ratingLabel)
```

## Curva ROC. Regresión Lineal.



```
##  
## El área bajo la curva ROC es 50.90517 %
```

Como podemos comprobar la curva ROC es prácticamente lineal y por ende el área bajo ella es bastante escasa. Este aspecto nos indica que la capacidad de generalización del modelo entrenado es bastante escasa, con lo que explica la tasa de error tan elevada sobre el conjunto de prueba que hemos obtenido previamente.

## Regresión logística simple.

Aplicaremos esta técnica con el objetivo de entrenar un modelo predictivo capaz de averiguar, para cada muestra, la clase binaria a la que pertenece el tratamiento: 0 (no efectivo) o 1 (efectivo). Para ello se realizará un procedimiento similar al anterior solo que en este caso utilizaremos la función correspondiente a la **regresión logística** para llevar a cabo el entrenamiento del sistema.

```
# Entrenamos un modelo predictivo con regresión logística. Para ello especificamos como variable dependiente la respuesta binaria.  
regresion_logistica = glm(ratingLabel ~ rating, family=binomial(logit), data=mini_training_dataset)  
# Resumen acerca del modelo entrenado.  
summary(regresion_logistica)
```

```
##  
## Call:  
## glm(formula = ratingLabel ~ rating, family = binomial(logit),  
##       data = mini_training_dataset)  
##
```

```

## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -4.720e-05  2.100e-08  2.100e-08  2.100e-08  4.092e-05
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -186.68    9487.82  -0.02    0.984
## rating       41.52     2086.59   0.02    0.984
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 5.1475e+03 on 4999 degrees of freedom
## Residual deviance: 6.7698e-07 on 4998 degrees of freedom
## AIC: 4
##
## Number of Fisher Scoring iterations: 25

# Evaluación del modelo entrenado.
evaluar_modelo_regresion(regresion_logistica, mini_training_dataset, mini_test_dataset)
```

```

##
## Matriz de confusión.
##     real
## pred  0   1
##   0 1058  0
##   1    0 3942
##
## Tasa de error en entrenamiento: 0 %
## Tasa de error en prueba: 0 %
```

Tal y como podemos comprobar el modelo entrenado con la técnica actual no es confiable puesto que ambos campos tienen un p-value muy cercano a 1, por lo que podemos concluir que no puede ser predicha la etiqueta binaria mediante las puntuaciones originales de forma lineal.

## Regresión polinomial

Si bien los resultados de los modelos predictivos entrenados mediante técnicas lineales no han sido buenos, vamos a aplicar una técnica no lineal con el objetivo de comprobar si la variable binaria *ratingLabel* puede ser predicha mediante modelo polinómico. Para aplicar esta técnica es indispensable incluir algunos componentes no lineales, como pueden ser nuevos predictores obtenidos a partir de aplicar técnicas sobre los originales como elevarlos a una potencia. Comenzaremos utilizando un polinomio de grado 2, para lo que necesitaremos incluir la variable *rating*, con la que intentamos predecir la clase a la que pertenece cada tratamiento, elevada al cuadrado de la siguiente forma.

```

# Entrenamos el modelo con regresión polinomial estableciendo un polinomio de grado 2.
regresion_polinomica<-lm(formula=ratingLabel~poly(rating, 2), data=mini_training_dataset)
# Estudiamos el modelo entrenado.
summary(regresion_polinomica)
```

```

##
## Call:
## lm(formula = ratingLabel ~ poly(rating, 2), data = mini_training_dataset)
```

```

## 
## Residuals:
##      Min       1Q   Median      3Q     Max 
## -0.54128 -0.01262 -0.01190  0.01996  0.30233
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 0.789400  0.002032 388.39 <2e-16 ***
## poly(rating, 2)1 25.797041  0.143717 179.50 <2e-16 ***
## poly(rating, 2)2 -7.908205  0.143717 -55.03 <2e-16 ***
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 0.1437 on 4997 degrees of freedom
## Multiple R-squared:  0.8758, Adjusted R-squared:  0.8758 
## F-statistic: 1.762e+04 on 2 and 4997 DF,  p-value: < 2.2e-16

# Calculamos las tasas de error sobre entrenamiento y prueba.
evaluar_modelo_regresion(regresion_polinomica, mini_training_dataset, mini_test_dataset)

```

```

## 
## Matriz de confusión.
##      real
## pred    0    1
##   0  924    0
##   1  134 3942
## 
## Tasa de error en entrenamiento: 2.92 %
## Tasa de error en prueba: 2.68 %

```

Como podemos observar en el estudio del modelo entrenado los *p-values* de todos los componentes son bastante bajos por lo que podemos determinar que este nuevo modelo predictivo es confiable y es suficiente utilizar un polinomio de orden 2. Asimismo también podemos concluir que la variable *rating* está relacionada con la que intentamos predecir: *ratingLabel*. También podemos comprobar una mejoría sobre los resultados mostrados en la matriz de confusión puesto que el número de muestras erróneamente clasificadas ha descendido considerablemente, comparado con los anteriores modelos lineales. Este mismo hecho se demuestra al evaluar el modelo predictivo y obtener las tasas de error tanto en el conjunto de entrenamiento como en el de prueba, ya que como podemos observar, los errores entran dentro de un rango razonable.

A continuación procedemos a representar gráficamente la curva que describe el polinomio calculado para entrenar dicho modelo. Para ello, en primer lugar, debemos de obtener las predicciones para la etiqueta binaria mediante la interpolación de puntos dentro del rango del campo predictor. A mayor número de puntos, más precisa será la representación gráfica.

```

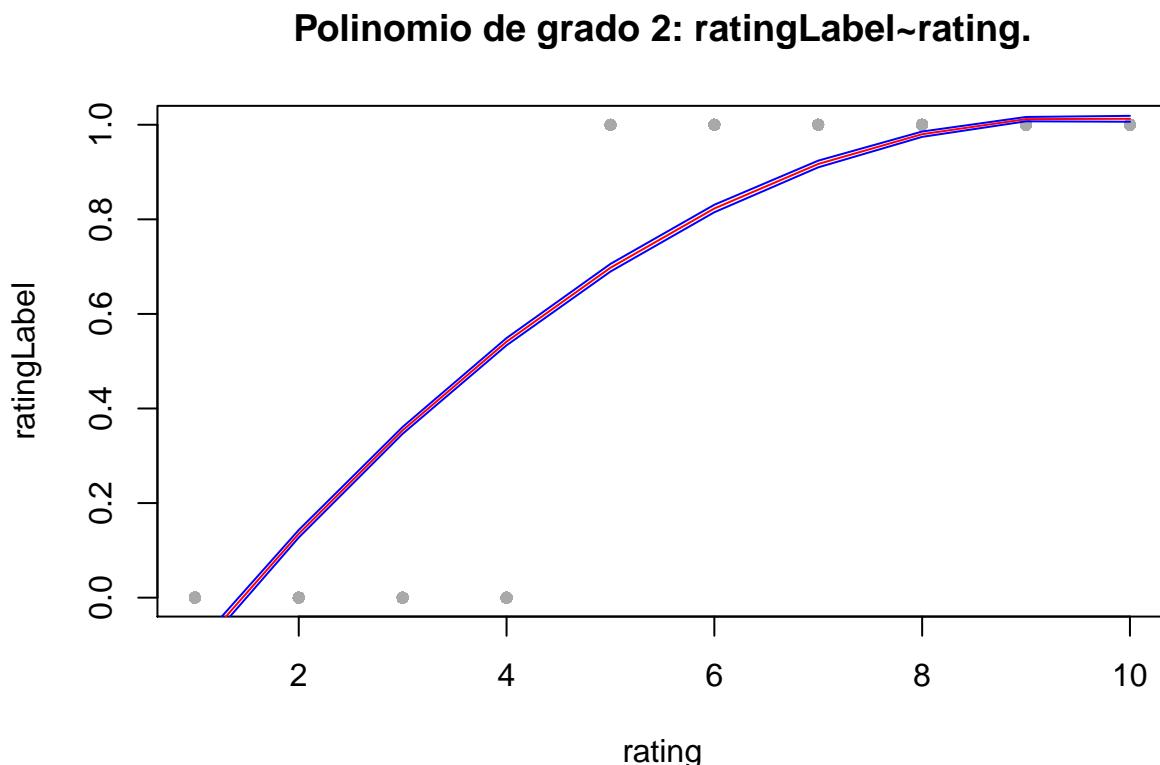
# Interpolación de puntos dentro del rango del campo rating
rango<-range(mini_training_dataset$rating)
puntos<-seq(from=rango[1], to=rango[2], by=1)
puntos<-data.frame(rating=puntos)
# Predicción de la etiqueta binaria y cálculo del error estándar del modelo. Para ello se establece un .
prediccciones<-predict(regresion_polinomica, newdata=puntos, se.fit=TRUE, level=0.95)
# Cálculo del intervalo de confianza superior e inferior al 95%.
intervalo_confianza<-data.frame(inferior=prediccciones$fit-1.96*predictions$se.fit, superior=prediccciones$fit+1.96*predictions$se.fit)
attach(mini_training_dataset)
plot(x=rating, y=ratingLabel, pch = 20, col = "darkgrey")

```

```

title("Polinomio de grado 2: ratingLabel~rating.")
lines(x = puntos$rating, predicciones$fit, col = "red", pch = 20)
lines(x = puntos$rating, intervalo_confianza$inferior, col = "blue", pch = 4)
lines(x = puntos$rating, intervalo_confianza$superior, col = "blue", pch = 4)

```



En esta gráfica queda claramente representada la relación existente entre la variable *rating* y la etiqueta binaria que deseamos predecir *ratingLabel*. Esta relación consiste en que a mayor puntuación, mayor es la efectividad del tratamiento y por lo tanto se encuentra dentro de la categoría de medicamentos efectivos.

En base a los estudios realizados con diversas variantes de la regresión, podemos determinar que la **regresión polinómica** ha sido el único con el que hemos obtenido un modelo competitivo, confiable y capaz de clasificar correctamente los tratamientos en efectivos o no efectivos en base a la puntuación, entre 0 y 10, asociada por cada uno de los pacientes afectados. Con ello podemos concluir que nuestro problema **no es lineal\***, y por lo tanto no se puede entrenar un modelo lineal para realizar una buena predicción.

## Regresión para predecir la efectividad del fármaco en base a las reviews.

En esta segunda sección dedicada a la regresión vamos a entrenar diversos modelos aplicando esta técnica del mismo modo que en el apartado anterior. El objetivo concreto es el mismo, es decir, vamos a entrenar modelos que sean capaces de averiguar si un tratamiento es efectivo o no en función de la *review* que el paciente escribió. Nuestro objetivo final es conocer si la puntuación original es una variable mejor predictora o si por el contrario las opiniones de los pacientes aportan una mayor cantidad de información que permita entrenar un modelo competente.

## Regresión lineal simple.

En este segundo apartado la variable dependiente será la misma, es decir, *ratingLabel* contendrá las etiquetas clasificadoras en relación a la efectividad de un medicamento. Sin embargo, en este caso serán los términos más frecuentes de las matrices los que jugarán el papel de variables independientes con las que intentaremos entrenar el clasificador para que, posteriormente, pueda predecir si un fármaco ha sido efectivo o no en función de la opinión del paciente.

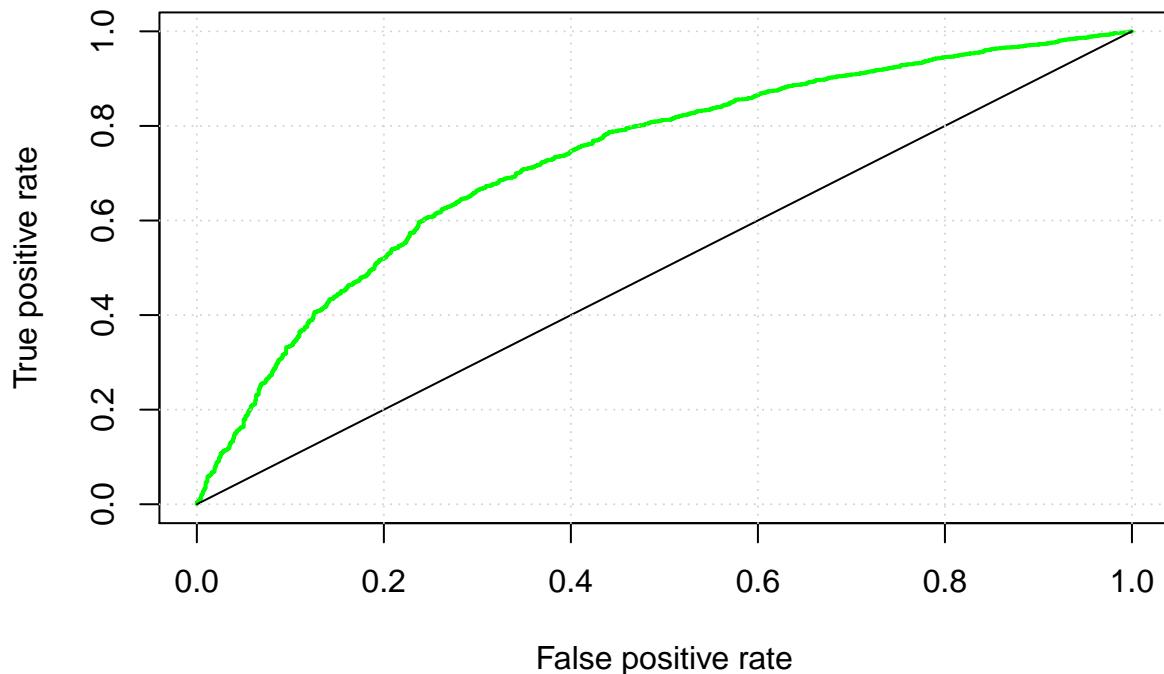
```
library(tm)
# Obtenemos la matriz de los términos de las reviews de los conjuntos de entrenamiento y test.
train_reviews_matriz<-DocumentTermMatrix(train_reviews_corpus)
test_reviews_matriz<-DocumentTermMatrix(test_reviews_corpus)
# Nos quedamos con los términos más frecuentes (con un mínimo número de apariciones de 5)
terminos_frecuentes<-findFreqTerms(train_reviews_matriz, 5)
# Obtenemos de nuevo la matriz con los términos más frecuentes extraídos anteriormente tanto para train
train_reviews_frec_matriz<-DocumentTermMatrix(train_reviews_corpus, control=list(dictionary=terminos_frecuentes))
test_reviews_frec_matriz<-DocumentTermMatrix(test_reviews_corpus, control=list(dictionary=terminos_frecuentes))
# Convertimos a dataframe el conjunto de datos de entrenamiento para pasárselos al algoritmo.
train_freq<-as.data.frame(data.matrix(train_reviews_frec_matriz), stringsAsFactors = FALSE)
test_freq<-as.data.frame(data.matrix(test_reviews_frec_matriz), stringsAsFactors = FALSE)
# Añadimos la columna de clasificación de la efectividad del medicamento.
train_freq<-cbind(train_freq, ratingLabel=mini_training_dataset$ratingLabel)
test_freq<-cbind(test_freq, ratingLabel=mini_test_dataset$ratingLabel)
# Entrenamos el modelo y a continuación lo evaluamos.
lm_reviews<-lm(train_freq$ratingLabel~., train_freq)
evaluar_modelo_Regresion(lm_reviews, train_freq, test_freq)

##
## Matriz de confusión.
##     real
## pred      0      1
##   0  476  645
##   1  582 3297
##
## Tasa de error en entrenamiento: 4.72 %
## Tasa de error en prueba: 24.54 %
```

Tal y como podemos observar el error en entrenamiento es bastante bajo mientras que el error en el conjunto de prueba es algo elevado. De nuevo podemos visualizar que el clasificador se ha adaptado demasiado a los datos de entrenamiento y eso ha provocado una disminución de su capacidad de predicción, como pasó en la primera sección. No obstante, en este caso, podemos notar que el sobreajuste no es tan acusado y que por tanto este modelo dispone de un poco más de capacidad de generalización utilizando las *reviews* en lugar de la puntuación original del fármaco. Si analizamos la matriz de confusión podemos observar que la mayoría de errores se concentran en etiquetar a los medicamentos como no eficaces (clase 0), es decir, el clasificador está siendo demasiado positivo y por tanto está categorizando fármacos como efectivos cuando no lo son. A continuación procedemos a visualizar la **curva ROC** que nos aportará más información acerca de la capacidad de generalización del modelo obtenido.

```
# Curva ROC para el conjunto de prueba
test_probab = predict(lm_reviews, test_freq, type=c("response"))
curvaROC(test_probab, test_freq$ratingLabel)
```

## Curva ROC. Regresión Lineal.



```
##  
## El área bajo la curva ROC es 73.01708 %
```

Si observamos la gráfica podemos comprobar, que en este tipo de regresión, el modelo entrenado mediante las opiniones de los pacientes dispone de una mayor capacidad de generalización que utilizando las puntuaciones de los fármacos, en el que recordemos que era prácticamente lineal. Asimismo, acompañando a esta teoría se encuentra el incremento del área bajo la curva.

## Regresión logística simple.

A continuación procedemos a entrenar un modelo predictivo no lineal mediante **regresión logística** especificando la misma dependencia entre la variable clasificatoria y los términos de las críticas con los que intentaremos averiguar la primera.

```
# Cargamos la librería que nos permitirá entrenar un modelo con esta técnica.  
library(MASS)  
# Entrenamos el modelo.  
glm_reviews = glm(train_freq$ratingLabel~, family=binomial(logit), data=train_freq)  
# Comprobamos la tasa de error sobre el conjunto de entrenamiento y de prueba.  
evaluar_modelo_regresion(glm_reviews, train_freq, test_freq)
```

```
##  
## Matriz de confusión.  
##      real
```

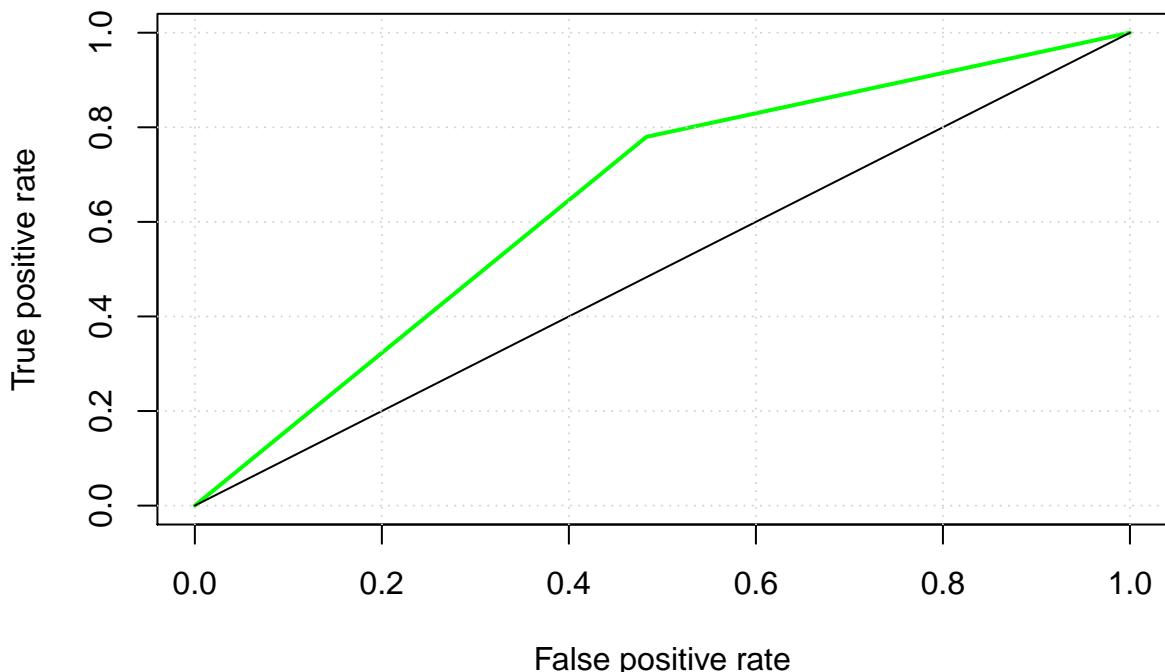
```

## pred      0      1
##      0  547  868
##      1  511 3074
##
## Tasa de error en entrenamiento: 3.7 %
## Tasa de error en prueba: 27.58 %

# Dibujamos su curva ROC
test_probab = predict(glm_reviews, test_freq, type=c("response"))
curvaROC(test_probab, test_freq$ratingLabel)

```

## Curva ROC. Regresión Lineal.



```

##
## El área bajo la curva ROC es 64.84102 %

```

En este caso, tal y como podemos apreciar, el modelo entrenado muestra unas tasas de error más elevadas que el entrenado con regresión lineal, por lo que podemos deducir que este clasificador dispone de una menor capacidad de generalización. Esto puede ser debido a que las dos clases existentes de la efectividad del fármaco pueden ser separables mediante una recta mejor que con otra función más compleja no lineal, y por lo tanto el clasificador entrenado con regresión lineal simple muestra mejores resultados.

## Regresión polinómica.

Por último vamos a entrenar un clasificador mediante **regresión polinómica** para intentar disminuir las tasas de errores obtenidas en técnicas anteriores con el modelo que nos ocupa en esta segunda sección. Para

ello, hemos comenzado repitiendo el grado del polinomio estableciendolo como en la primera sección, es decir, de grado 2, y los resultados obtenidos han sido en torno a un **2,9% de error en entrenamiento y un 31,66% en prueba**. Es por ello por lo que presentamos más resultados variando el grado del polinomio para comprobar si mejoran estos primeros resultados.

```
# Entrenamos el modelo con regresión polinomial estableciendo un polinomio de grado 2.  
lmpol_reviews_grado2<-lm(formula=train_freq$ratingLabel~poly(rating, 2), data=train_freq)  
# Calculamos las tasas de error sobre entrenamiento y prueba.  
cat("\nGrado del polinomio: 2")
```

```
##  
## Grado del polinomio: 2  
  
evaluar_modelo_regresion(lmpol_reviews_grado2, train_freq, test_freq)
```

```
##  
## Matriz de confusión.  
##      real  
## pred   0    1  
##     0 191 716  
##     1 867 3226  
##  
## Tasa de error en entrenamiento: 2.92 %  
## Tasa de error en prueba: 31.66 %
```

```
# Entrenamos el modelo con regresión polinomial estableciendo un polinomio de grado 3.  
lmpol_reviews_grado3<-lm(formula=train_freq$ratingLabel~poly(rating, 3), data=train_freq)  
cat("\n\nGrado del polinomio: 3")
```

```
##  
##  
## Grado del polinomio: 3  
  
evaluar_modelo_regresion(lmpol_reviews_grado3, train_freq, test_freq)
```

```
##  
## Matriz de confusión.  
##      real  
## pred   0    1  
##     0 222 831  
##     1 836 3111  
##  
## Tasa de error en entrenamiento: 0 %  
## Tasa de error en prueba: 33.34 %
```

```
# Entrenamos el modelo con regresión polinomial estableciendo un polinomio de grado 4.  
lmpol_reviews_grado4<-lm(formula=train_freq$ratingLabel~poly(rating, 4), data=train_freq)  
cat("\n\nGrado del polinomio: 4")
```

```
##  
##  
## Grado del polinomio: 4
```

```

evaluar_modelo_regresion(lmpol_reviews_grado4, train_freq, test_freq)

##
## Matriz de confusión.
##     real
## pred   0   1
##   0 222 831
##   1 836 3111
##
## Tasa de error en entrenamiento: 0 %
## Tasa de error en prueba: 33.34 %

# Entrenamos el modelo con regresión polinomial estableciendo un polinomio de grado 5.
lmpol_reviews_grado5<-lm(formula=train_freq$ratingLabel~poly(rating, 5), data=train_freq)
cat("\n\nGrado del polinomio: 5")

```

```

##
## Grado del polinomio: 5

evaluar_modelo_regresion(lmpol_reviews_grado5, train_freq, test_freq)

```

```

##
## Matriz de confusión.
##     real
## pred   0   1
##   0 222 831
##   1 836 3111
##
## Tasa de error en entrenamiento: 0 %
## Tasa de error en prueba: 33.34 %

```

Tal y como podemos observar, a partir de grado 3 los resultados empeoran por lo que aumentar el grado del polinomio así como su complejidad no es la solución para mejorar el modelo entrenado mediante esta técnica. Es por ello por lo que en base a los resultados obtenidos, podemos concluir que para **predecir la efectividad del medicamento en base a las reviews, el mejor modelo de regresión obtenido es el lineal**, puesto que es con el que hemos obtenido un clasificador que comete el menor número de fallos y dispone de una curva ROC cuya área es más de un 70%.

## Agrupamiento y clustering

En este último capítulo abordaremos el problema del agrupamiento de datos o clustering. Esta técnica consiste en una serie de métodos los cuales nos permiten agrupar ciertos datos en función de sus características. Dicho de otro modo, las técnicas de clustering nos permiten reconocer de un grupo, elementos parecidos entre si, de esta forma podremos crear subgrupos o clusters. Cada uno de estos clusters se encuentra formado por elementos parecidos entre si y diferentes de los elementos pertenecientes a otros grupos o clusters.

Las técnicas de clustering, al contrario de otras técnicas empleadas en este trabajo, pertenecen al conjunto de técnicas del “aprendizaje no supervisado” dentro del Machine Learning. Podríamos interpretar esto último como que son técnicas exploratorias, las cuales nos van a aportar información sobre los datos que ya tenemos pero no nos van a servir para, por ejemplo, predecir datos nuevos.

Una vez comprendido que son las técnicas de clustering, pasamos a ver como pueden ayudarnos en nuestro problema. En este caso y viendo los datos de los que disponemos hemos decidido estudiar dos posibles casos.

- **Agrupamiento de los fármacos en función al rating obtenido.** Parece obvio pensar, que una posible agrupación de los datos puede ser la que agrupa los mejores medicamentos (los que tienen mejor rating) y los peores medicamentos (peor rating). Para ello se utilizará tanto la variable "DrugName" como la variable "Rating"
- **Agrupamiento de los medicamentos en función a la review del paciente.** En este caso, vamos a comprobar si existe alguna relación entre las reviews de los pacientes y los medicamentos. Para ello utilizaremos dichas variables.

Cada uno de los dos casos anteriores se estudiará de forma independiente, aunque se seguirá un proceso similar. En ambos casos se implementarán 4 algoritmos muy utilizados hoy en día, 3 de ellos son algoritmos particionales y otro se trata de un algoritmo jerárquico. Cabe destacar la importancia que ha tenido la referencia [22] en el desarrollo de esta sección y es que nos ha aportado casi toda la información que necesitábamos

## Agrupamiento de los farmacos en función al rating obtenido.

### Adecuación de los datos y estudio previo.

Para comenzar con este problema, vamos a necesitar obtener los datos con los que trabajar. Estos datos serán las columnas "DrugName" y "Rating" de nuestro dataset. Sin embargo, hay que tener en cuenta que los algoritmos de clustering funcionan con variables numéricas, por lo tanto, tendremos que convertir la variable que almacena el nombre del fármaco a variable numérica. Además de esto y aprovechando que se va a realizar un estudio previo de los datos vamos a aplicar este último procedimiento al nombre de la enfermedad. De esta forma podremos ver que aspecto tienen los datos nuevamente. Hay que destacar que no podemos olvidarnos de replicarlo todo a nuestro conjunto de test. Las gráficas del modelo se han obtenido siguiendo las técnicas observadas en [20].

```
#Primero añadimos una semilla para obtener siempre los mismos resultados al ejecutar el algoritmo
set.seed(42)

#Realizamos una copia del dataset que modificaremos
clust_train_dataset <- mini_training_dataset

#Como se comentó anteriormente convertimos en numéricas las variables que utilizaremos y analizaremos
clust_train_dataset$condition_number <- as.numeric( factor(mini_training_dataset$condition) ) -1
clust_train_dataset$drugName_number <- as.numeric( factor(mini_training_dataset$drugName) ) -1

#Seleccionamos las columnas que vamos a analizar
clust_train_dataset = clust_train_dataset[c(5,9,8)]


library(mclust)
fit <- Mclust(clust_train_dataset, )
plot(fit) # plot results

#Recogemos los datos que vamos a utilizar en este caso.
rating_DrugName <- clust_train_dataset[,1:2]

#Debido a la diferencia de magnitud en las variables, escalamos los datos
```

```

rating_DrugName <- scale(rating_DrugName)
#Mostramos la forma que tendran los datos con los que trabajaremos
head(rating_DrugName)

```

```

##          rating drugName_number
## 4538    0.8663641   -0.05307259
## 3087    0.8663641    0.79023340
## 10761   0.5446066   -1.53283593
## 2157    0.8663641    0.61520763
## 6178    0.5446066   -1.58852595
## 1956    0.2228492   -1.00775861

```

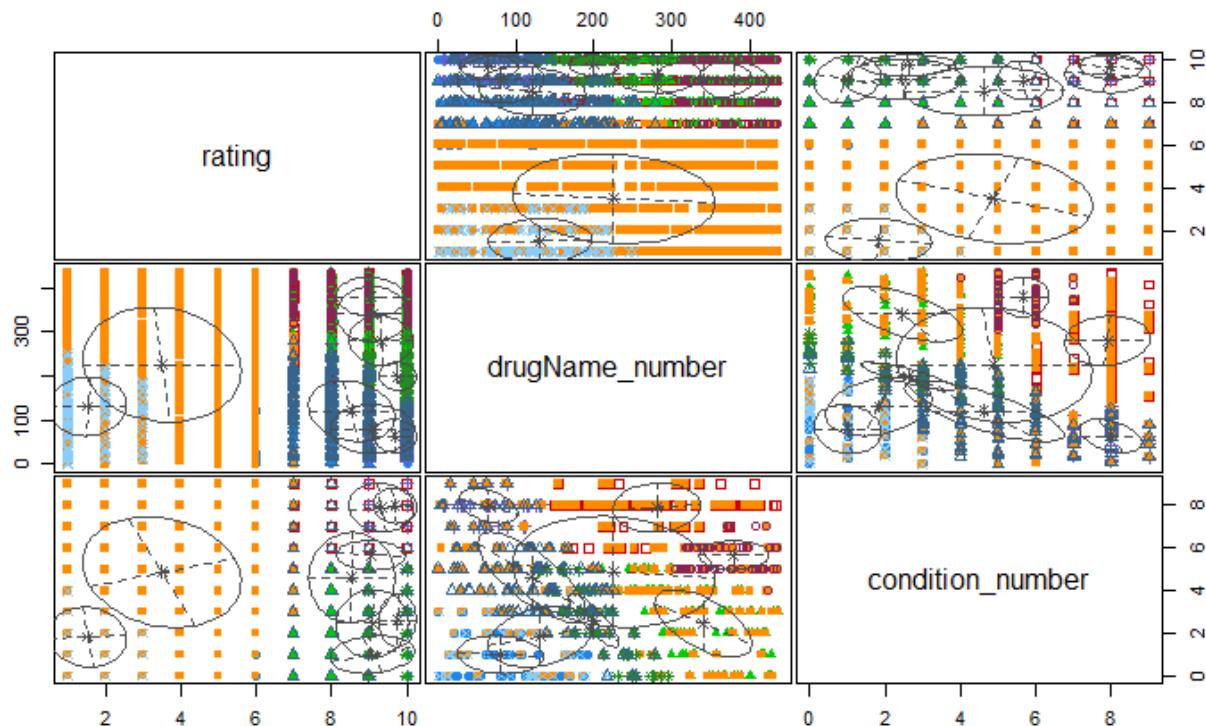


Figure 7: Análisis de clusters en datos

Como se puede ver en la imagen anterior, no se aprecia a simple vista ninguna separación obvia más allá de la separación que podemos ver debido al formato de los datos (como rating, por ejemplo). Sin embargo y aunque no se aprecien clusters a simple vista, probaremos con los algoritmos de clustering, a ver que pueden hacer.

### Algoritmos particionales

**K-medias** En este primer algoritmo, vamos a intentar agrupar los datos en un número definido de cluster de forma que la varianza interna dentro del cluster sea lo mejor posible. Primero, necesitaremos conocer el número óptimo de clusters, para ello utilizaremos la función fviz\_nblust del paquete “factoextra” con la opción Kmeans.

```

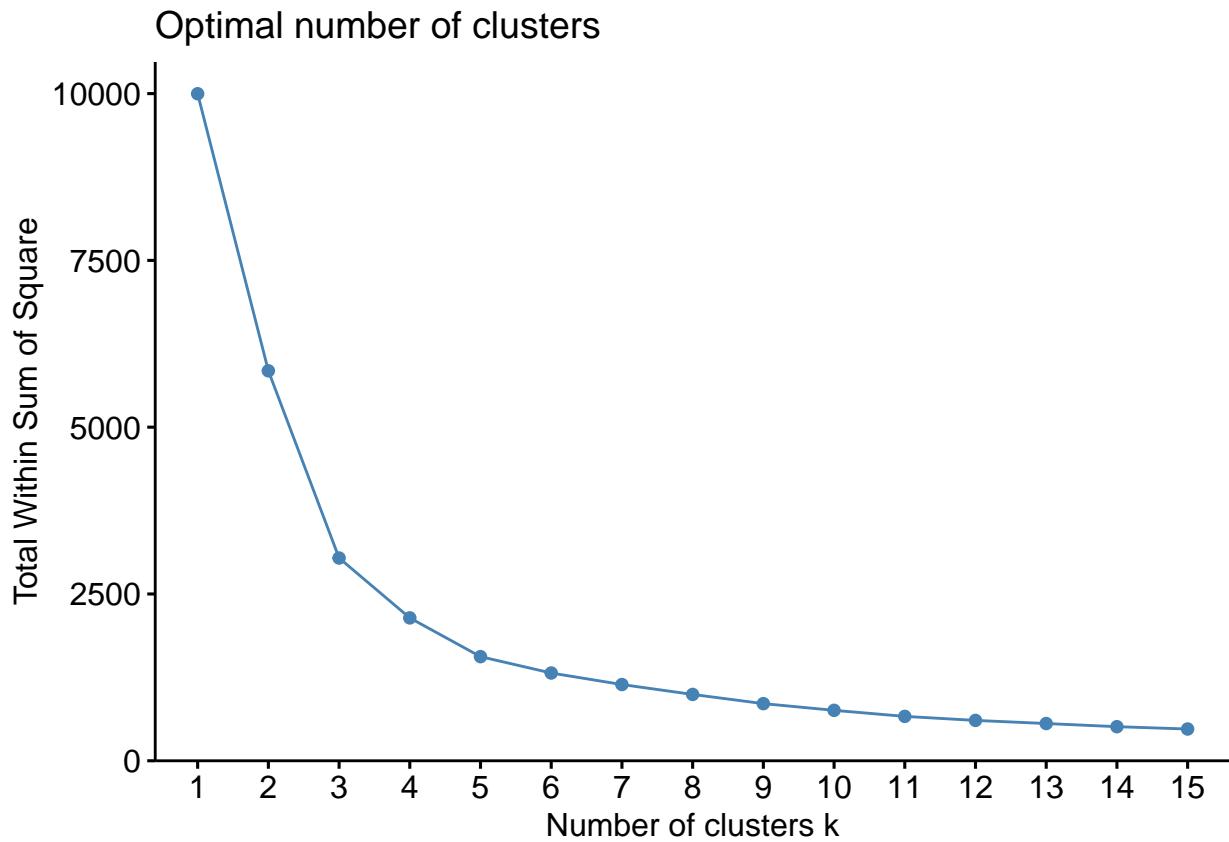
library(factoextra)

## Warning: package 'factoextra' was built under R version 3.6.2

## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa

fviz_nbclust(x = rating_DrugName, FUNcluster = kmeans, method = "wss", k.max = 15,
             diss = get_dist(rating_DrugName, method = "euclidean"), nstart = 50)

```



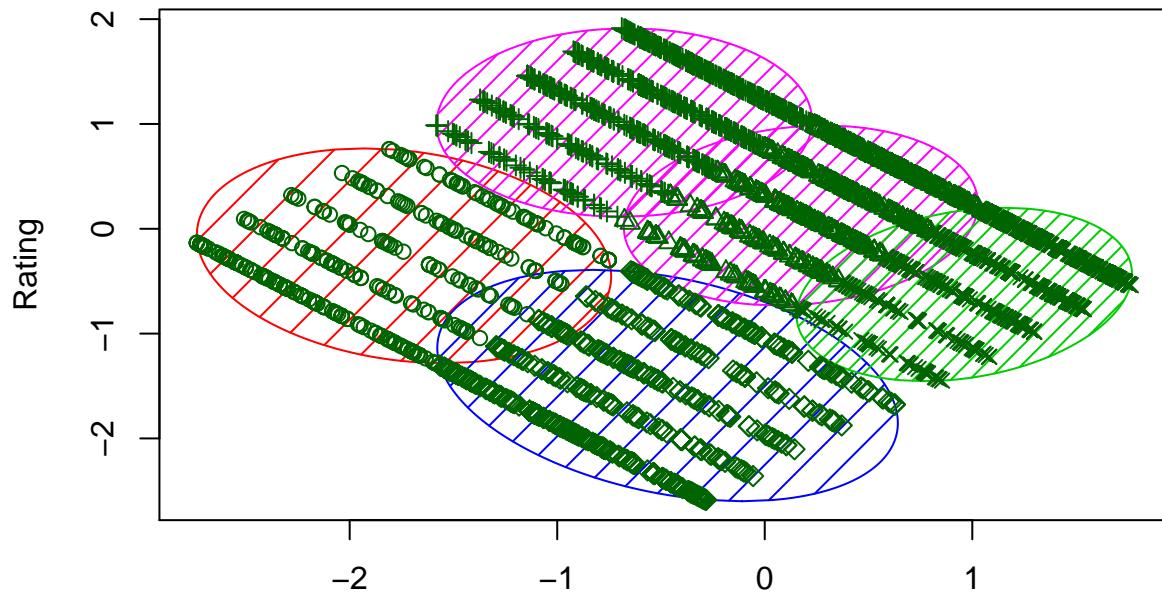
Como se puede ver en el gráfico anterior el número óptimo de clusters es 4 ya que en ese punto la varianza dentro del cluster deja de bajar de forma drástica. Sin embargo esto se separa un poco de nuestra suposición inicial en la que los medicamentos se agrupan en buenos y malos. Sin embargo, esto bien puede ser porque se agrupen en muy malos, malos, buenos y muy buenos. Por lo tanto no tenemos por qué darle mayor importancia. Así que ahora sí, vamos a intentar visualizar los clusters obtenidos por este método.

```

library(cluster)
set.seed(31)
#Ejecutamos el algoritmo k-means con 4 clusters
clusters <- kmeans(rating_DrugName, 5)
#Dibujamos los clusters
clusplot(rating_DrugName, clusters$cluster, color=TRUE, shade=TRUE, labels=0, lines=0, ylab = "Rating",

```

## CLUSPLOT( rating\_DrugName )



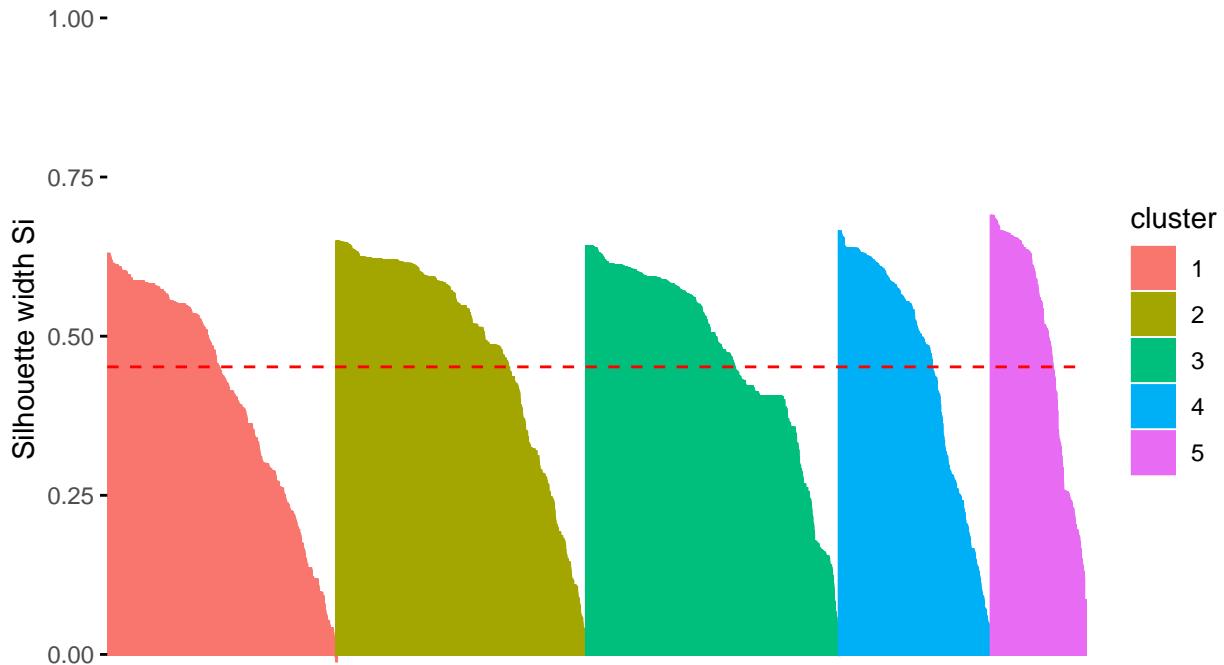
These two components explain 100 % of the point variability.

Como se puede ver, no se observa una separación clara entre los clusters que hemos obtenido con este método. Al igual que tampoco se puede apreciar una interpretación clara de estos datos y es que se puede ver perfectamente como con estos datos se agrupan los 10 ratings en 10 lineas. Esto no nos permite sacar ninguna conclusión en claro, esto se puede deber al carácter de nuestros datos y es que al tratarse la variable medicamento de una variable no numérica es difícil expresarla en forma numérica y que se aprecie una relación entre los datos. Sin embargo, comprobaremos como de bien se adaptan los clusters a nuestros datos, para ello utilizaremos el **índice de silueta**. Este índice consiste en la medición de las distancias entre los diferentes puntos de cada cluster con los otros puntos del mismo cluster y de fuera de él. Este índice se mueve en el rango [-1,1] y cualquier valor por encima de 0 nos dirá que ese dato esta en el cluster correcto, mientras que un valor inferior a 0 nos indicará justo lo contrario. El grafico se puede ver a continuación.

```
#Calculamos de nuevo los clusters y se los pasamos a la función fviz_silhouette del paquete "factoextra"
res.km <- eclust(seed = 12, rating_DrugName, "kmeans", k=5, graph = FALSE, labels(None))
fviz_silhouette(res.km)
```

```
##   cluster size ave.sil.width
## 1       1 1171      0.40
## 2       2 1275      0.48
## 3       3 1291      0.47
## 4       4  776      0.44
## 5       5  487      0.48
```

Clusters silhouette plot  
Average silhouette width: 0.45



Como se puede ver los clusters se adecúan bastante bien a nuestros datos. Sin embargo, esto no quiere decir que tengamos que creer ciegamente los resultados obtenidos y es que al no observarse ningún cluster bien definido no podemos confiar en estos resultados. Probaremos con otras técnicas de clustering para salir de dudas.

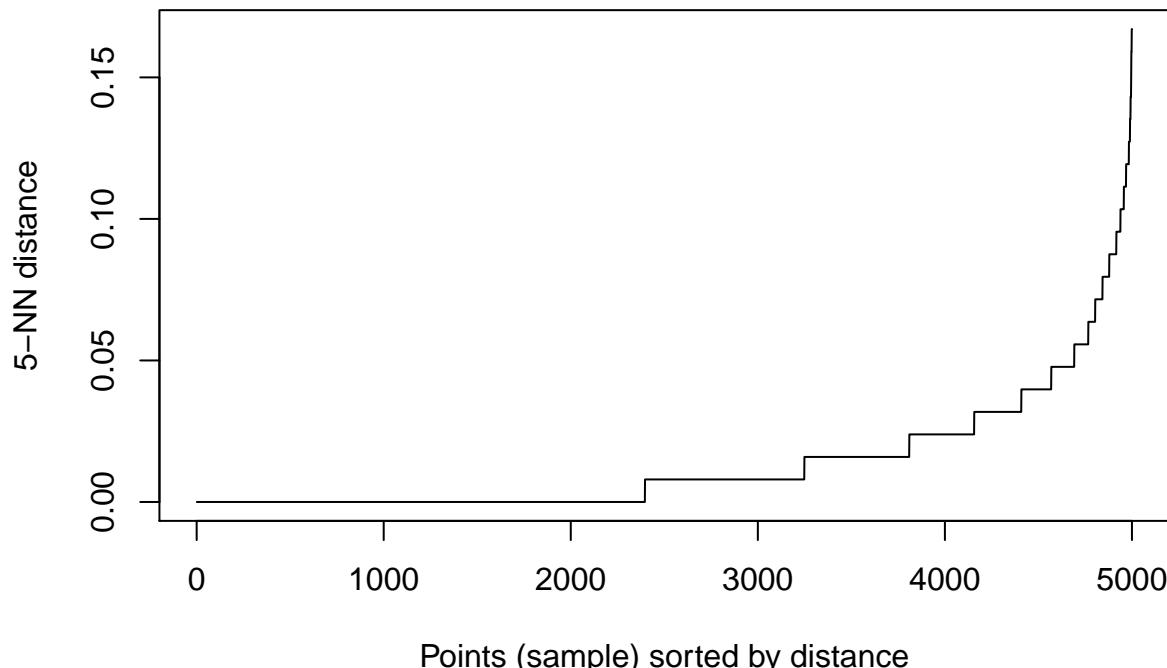
**Algoritmo DBSCAN** En este último algoritmo utilizaremos otro algoritmo particional, este se trata del algoritmo *dbscan* o *Density-based spatial clustering of applications with noise*. Este algoritmo al contrario que los anteriores, pretende agrupar los datos identificando regiones con alta densidad de observaciones separadas por regiones de baja densidad.

Al aplicar este algoritmo hay que tener en cuenta dos argumentos muy importantes:

- **minPts**: Cuanto mayor sea el tamaño de este argumento, mayor debe ser el valor mínimo de observaciones vecinas para formar un cluster.
- **epsilon**: Este argumento nos marcará que puntos se asignan a que cluster, es decir, nos marcará la distancia a la cual el algoritmo reconocerá que valor esta cerca de otro para que estos puedan formar un cluster.

Por lo tanto, uno de los argumentos más difíciles de ajustar será **epsilon**. Sin embargo, hay una técnica que nos permite acercarnos al valor ideal de dicho argumento. Se trata de estudiar las distancias promedio entre las  $k=\text{minPts}$  observaciones más próximas. El estudio mencionado, nos dibujará una curva en la cual el punto de inflexión suele ser el óptimo. Por lo tanto, pasamos a realizar dicho estudio.

```
#Realizamos un escalado de los valores del dataset para que sean un poco más manejables para el algoritmo
data <- scale(rating_DrugName)
#Representaremos la distancia promedio entre las k observaciones más próximas con minPts = 5.
dbSCAN::kNNdistplot(data, k = 5)
```

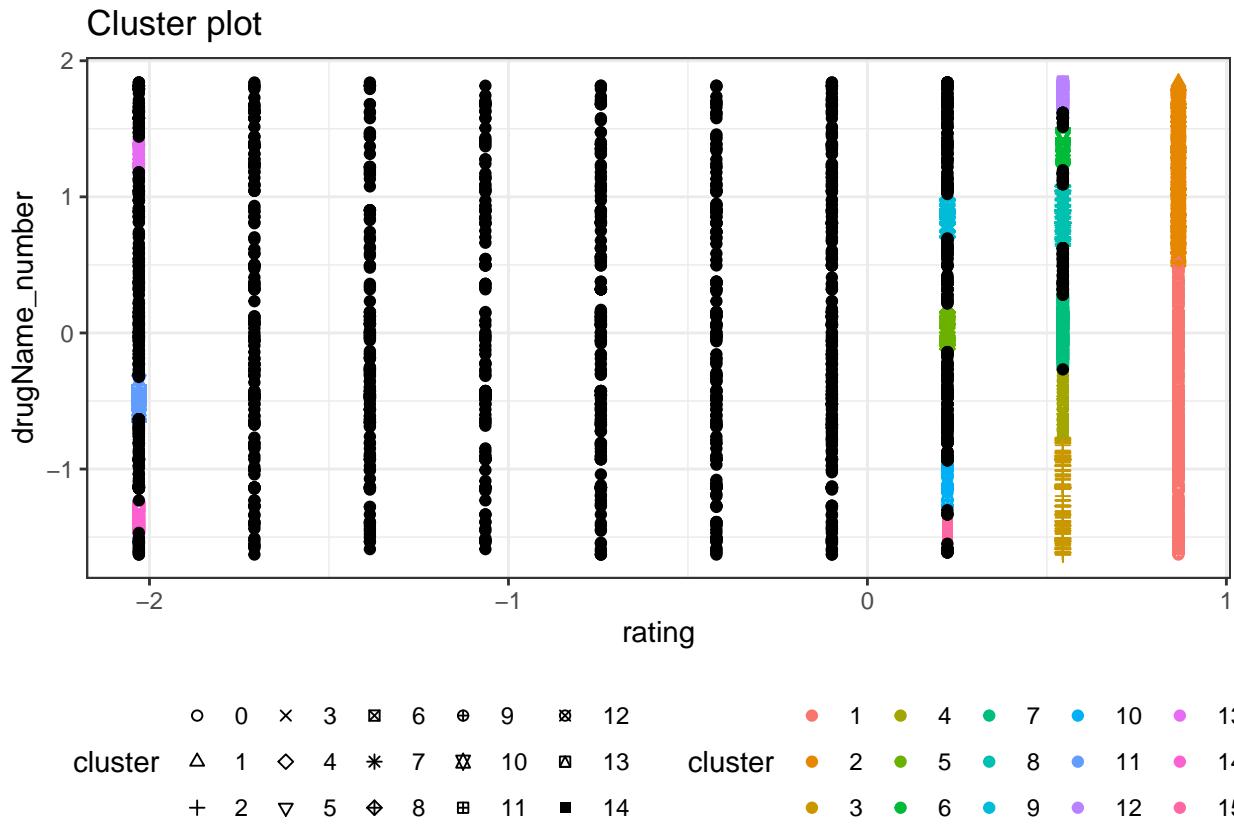


Como se ve en la gráfica, por las características de este problema, el punto de inflexión se encuentra entorno a 0.03. Por lo tanto, probaremos con  $\text{epsilon} = 0.03$  y  $\text{MinPts} = 5$  y representaremos la gráfica. Sin embargo, tras probar con dichos datos se obtienen demasiados clusters, cerca de 200. Por lo tanto, se modificarán dichos atributos intentando ser un poco más restrictivo. Para ello se aumentará el valor de  $\text{epsilon}$  y el valor de  $\text{minPts}$ .

Tras varias pruebas el único resultado obtenido que es algo interpretable es con los valores de  $\text{epsilon} = 0.1$  y  $\text{minPts} = 50$ . En este caso lo único que se puede obtener de esta técnica es que hay más densidad de medicamentos en las zonas de nota muy baja o muy alta. Por lo que la mayor parte de los clusters se encuentran en esa zona. Por todo esto, creemos que esta técnica no se adapta bien a nuestro problema ya que considera la mayor parte de los datos como ruido y no es capaz de realizar una agrupación de calidad de los medicamentos.

```
res.dbSCAN <- fpc::dbSCAN(data = data, eps = 0.1, MinPts = 50)

fviz_cluster(object = res.dbSCAN, data = data, stand = FALSE,
             geom = "point", ellipse = FALSE, repel = FALSE, show.clust.cent = FALSE,
             pallete = "jco") +
  theme_bw() +
  theme(legend.position = "bottom")
```



## Algoritmos jerárquicos

En este caso tratamos con un algoritmo no particional y a diferencia de los algoritmos particionales, con los algoritmos jerárquicos no es necesario definir un número de clusters al ejecutar el algoritmo.

Como hemos visto a lo largo de la asignatura, estos algoritmos se dividen en aglomerativos o divisivos. Debido a que las técnicas aglomerativas son más comunes, esta será la que utilizaremos. Aun así habrá que explicar en qué consiste este algoritmo. En las técnicas aglomerativas el agrupamiento se inicia en la base del árbol, donde cada observación forma un cluster individual. Los clusters se van combinando a medida que la estructura crece hasta converger en una única “rama” central. Por otra parte, los algoritmos divisivos toman el camino opuesto a los aglomerativos, se comienza con un único cluster que contiene todos los datos y se van realizando divisiones hasta que cada cluster es un dato.

Además de esto, a la hora de aplicar este tipo de algoritmos tendremos que tener en cuenta como se calcula la distancia entre clusters y es que esta distancia se usará para ir combinando los clusters más parecidos entre ellos hasta que quede uno solo. En este caso utilizaremos dos tipos de distancias diferentes:

- Complete o maximum: De entre todos los posibles pares de clusters se calcula la distancia y se selecciona el que tiene una distancia mayor.
- Average: De entre todos los posibles pares de clusters se calcula la distancia y se selecciona como distancia el valor promedio de todas estas medidas.

Hay multitud de formas de calcular el “linkage” o la distancia ademas de las anteriores, sin embargo por no caer en la redundancia en este trabajo solo emplearemos las dos anteriores. Una vez hayamos calculado los clusters tendremos que representar los resultados, esto se realizará mediante un dendograma. Esto nos

permitirá observar el árbol creado perfectamente. Pasamos por tanto a la aplicación del algoritmo y a su representación.

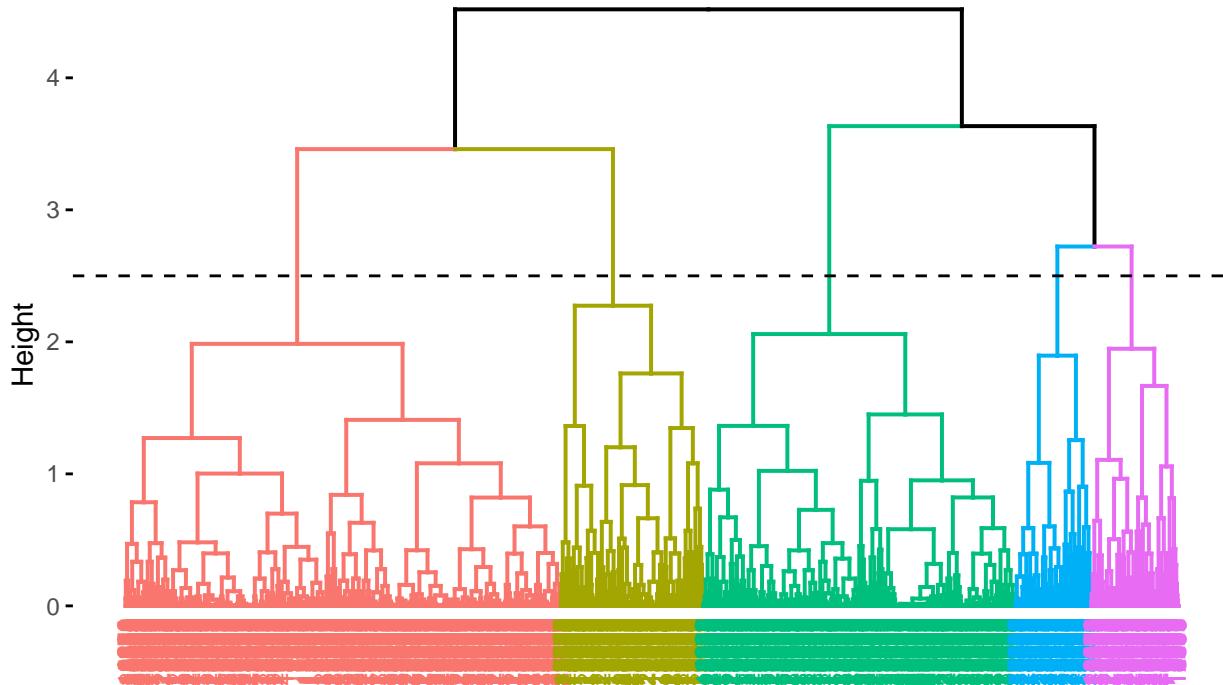
```
#Creamos el arbol del algoritmo o "cortamos las ramas" para generar el arbol

hc_euclidean_completo <- hclust(d = dist(x = rating_DrugName, method = "euclidean"),
                                    method = "complete")
hc_euclidean_media <- hclust(d = dist(x = rating_DrugName, method = "euclidean"),
                               method = "average")

#Representamos el arbol y utilizamos el numero óptimo de clusters que obtuvimos en el algoritmo anterior
fviz_dend(x = hc_euclidean_completo, k = 5, cex = 0.6) +
  geom_hline(yintercept = 2.5, linetype = "dashed") +
  labs(title = "Hierarchical clustering",
       subtitle = "Distancia euclídea, Lincage complete, K=5")
```

## Herarchical clustering

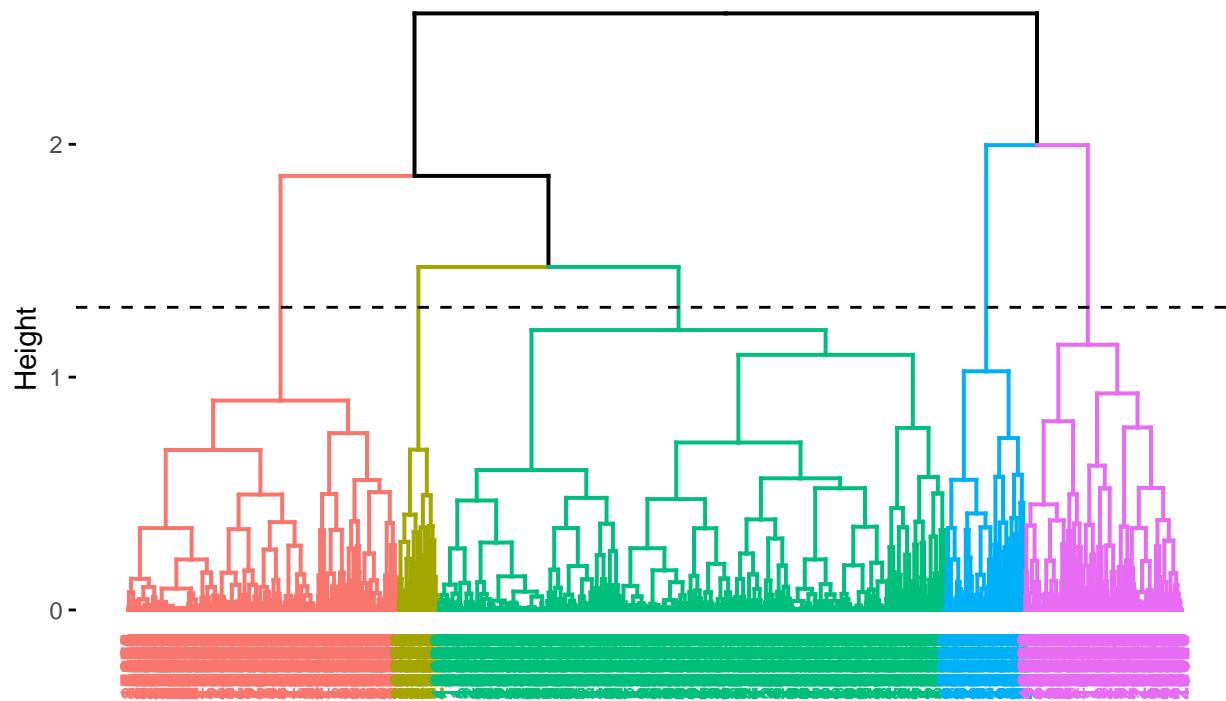
Distancia euclídea, Lincage complete, K=5



```
fviz_dend(x = hc_euclidean_media, k = 5, cex = 0.6) +
  geom_hline(yintercept = 1.3, linetype = "dashed") +
  labs(title = "Hierarchical clustering",
       subtitle = "Distancia euclídea, Lincage average, K=5")
```

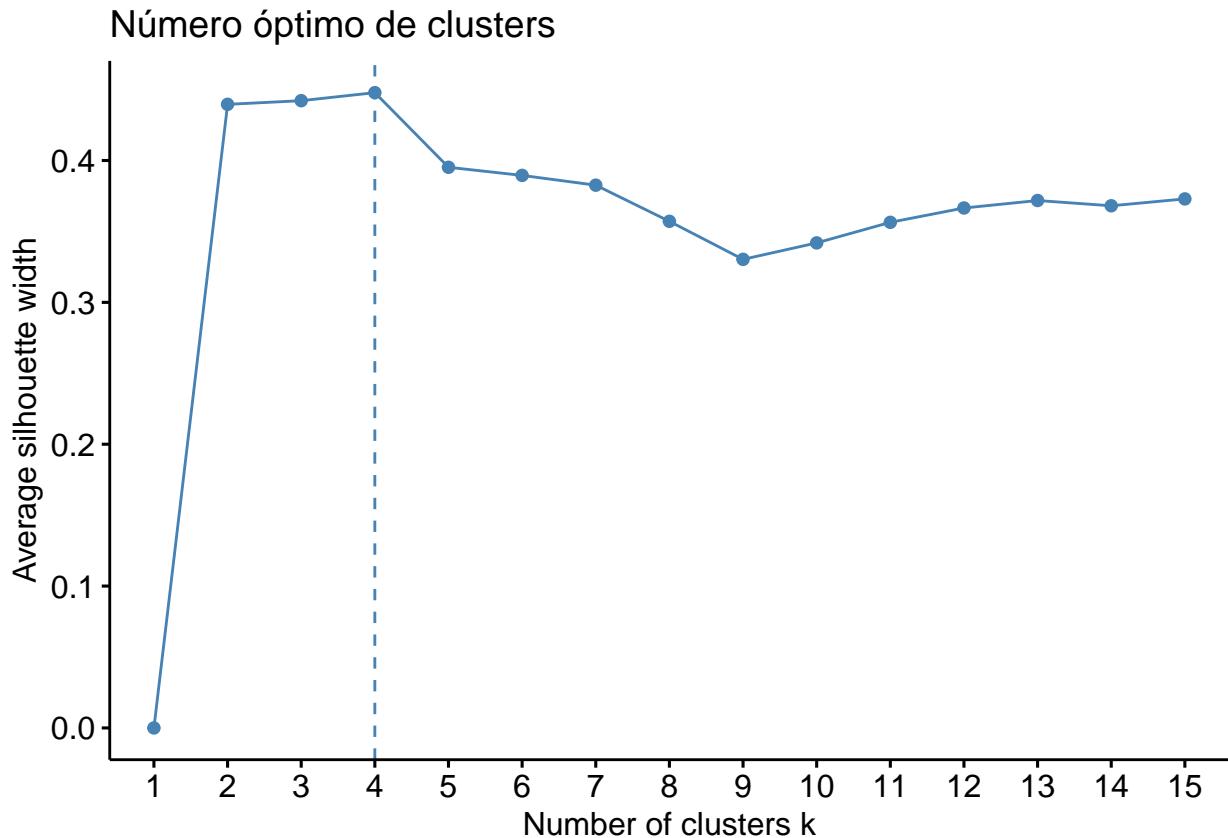
## Herarchical clustering

Distancia euclídea, Lincage average, K=5



Como se puede ver, no se aprecia mucha diferencia entre una medida de distancia y la otra, por lo tanto vamos a utilizar, siguiendo la temática mostrada en el algoritmo K-means, el índice de silueta asociado al número de clusters y a la media de dicho índice por cada número de clusters. De esta forma podremos confirmar si estábamos en lo correcto empleando 5 clusters.

```
fviz_nbclust(x = rating_DrugName, FUNcluster = hcut, method = "silhouette", k.max = 15,
             diss = dist(rating_DrugName, method = "euclidean") ) +labs(title = "Número óptimo de clust
```

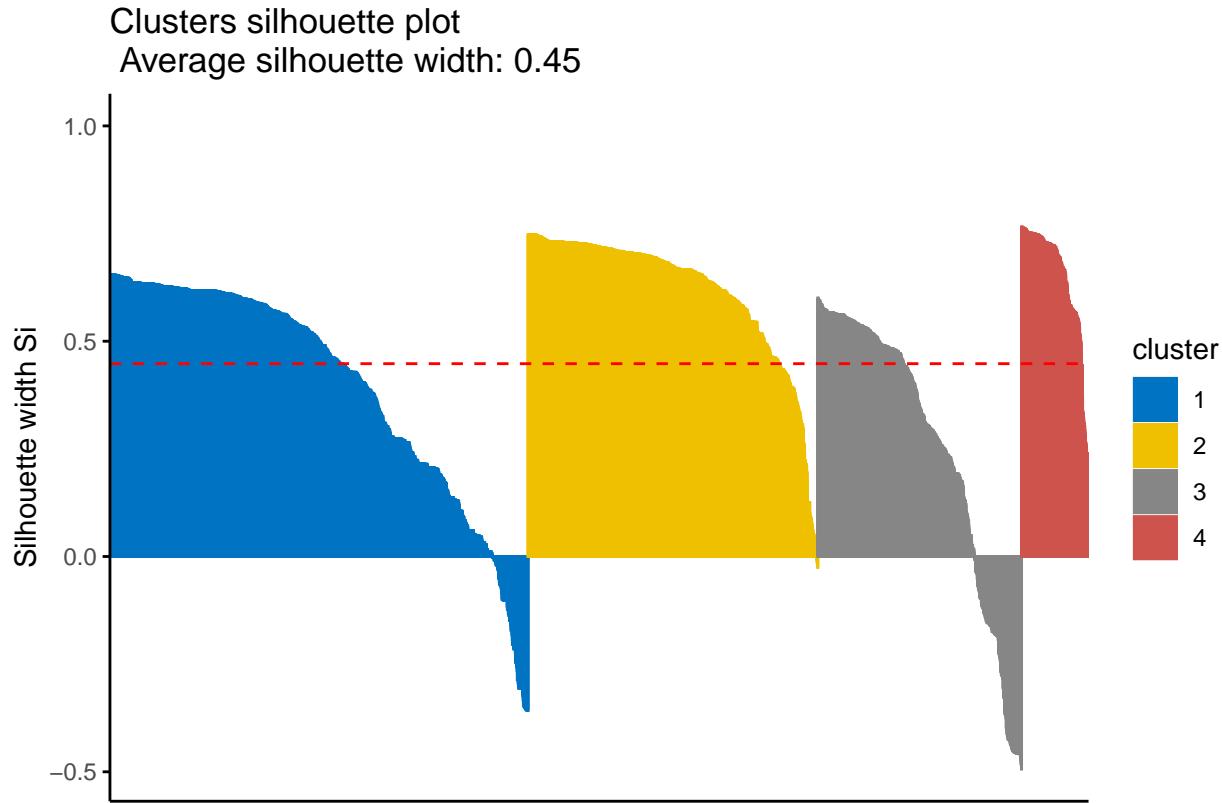


Como se puede ver en la grafica anterior. En este caso el número óptimo de clusters no es 5, si no 4 por lo que probaremos a dibujar el índice de silueta para ver como se adecúan estos clusters a nuestros datos. Dibujar de nuevo el dendograma no tendría mucho sentido ya que lo único que obtendríamos seria el mismo árbol solo que con un corte en el cluster en la parte superior. Por lo tanto para ahorrar espacio y redundancia no representaremos de nuevo dicho diagrama.

```
#Calculamos de nuevo el arbol
res.hcut <- eclust(x = rating_DrugName, FUNcluster = "hclust", k = 4, seed = 123,
                     hc_metric = "euclidean", graph = FALSE)
```

```
#Dibujamos el diagrama del índice de silueta
fviz_silhouette(sil.obj = res.hcut, print.summary = TRUE, palette = "jco",
                  ggtheme = theme_classic())
```

```
##   cluster size ave.sil.width
## 1        1 2136      0.40
## 2        2 1481      0.61
## 3        3 1041      0.25
## 4        4  342       0.65
```



Como podemos ver hay datos que no se asocian al cluster que deberían. Ademas de esto, el indice de silueta medio no es demasiado alto, por lo que podemos concluir que este metodo no es el mas idoneo para abordar nuestro problema. Aun así como se puede ver en [22] debemos validar los arboles. Una forma de hacerlo es empleando el coeficiente de correlación entre las distancias cophenetic del dendrograma (altura de los nodos) y la matriz de distancias original. Cuanto más cercano es el valor a 1, mejor refleja el dendrograma la verdadera similitud entre las observaciones. Valores superiores a 0.75 suelen considerarse como buenos.

```
mat_dist <- dist(x = rating_DrugName, method = "euclidean")
# Dendrogramas con linkage complete y average
hc_euclidean_complete <- hclust(d = mat_dist, method = "complete")
hc_euclidean_average <- hclust(d = mat_dist, method = "average")

cor(x = mat_dist, cophenetic(hc_euclidean_complete))

## [1] 0.6749662

cor(x = mat_dist, cophenetic(hc_euclidean_average))

## [1] 0.7803458
```

Como se puede ver, el método que emplea la media es ligeramente mejor que el del máximo. Sin embargo esto no significa necesariamente que agrupe muy bien nuestros datos y es que al ver la gráfica del cálculo del número ideal de clusters, se puede ver como no se observa demasiada diferencia entre utilizar 4 o más de dos clusters. Por ello, esto me hace pensar que tal vez nuestro problema no sea abordable mediante técnicas de clustering. Por otro lado, el hecho de que el índice de correlación entre las dos variables que estamos tratando (visto en el apartado de análisis de correlación) sea muy bajo no hace más que reforzar esta creencia. Sin embargo, comprobaremos el resultado de aplicar otros algoritmos más de clustering.

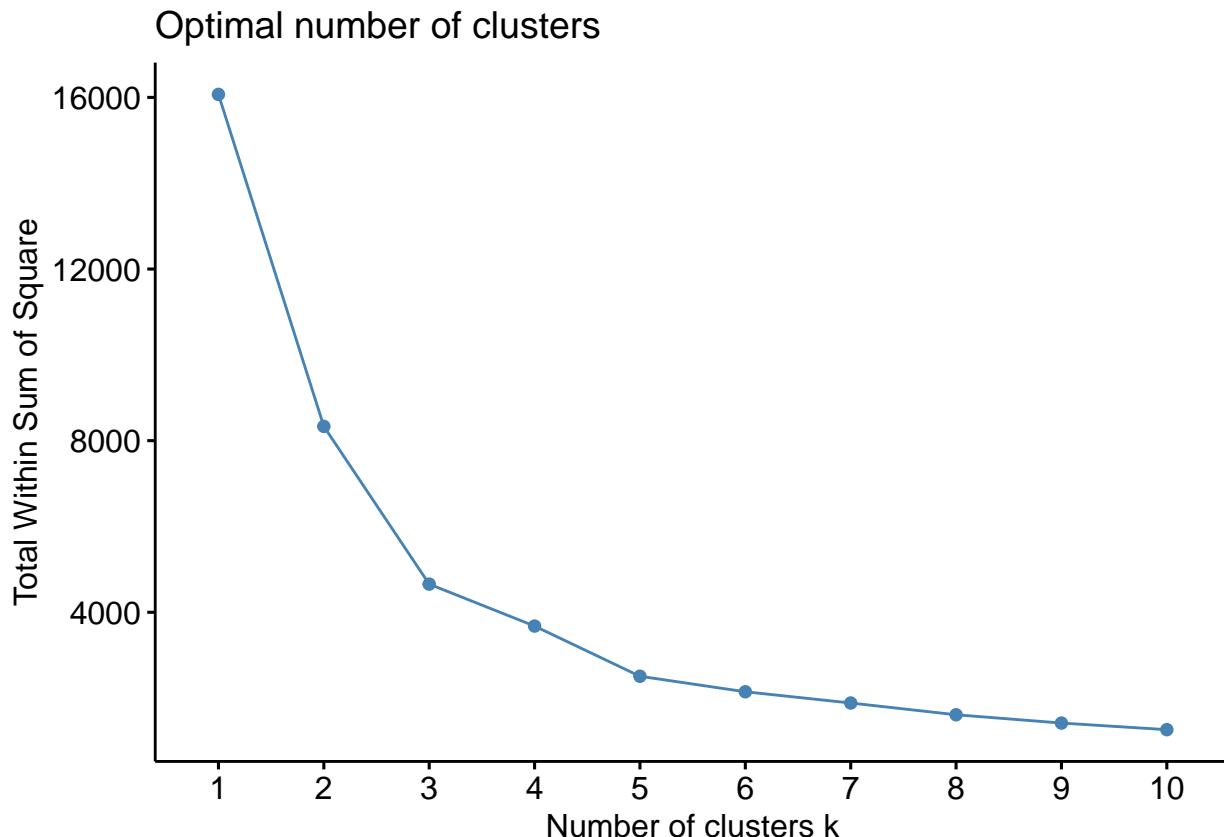
## Extensiones de los algoritmos

**K-medoides** En este apartado vamos a aplicar otra técnica similar a la técnica de k-medias, se trata de la técnica de k-medoides. Esta técnica difiere de la anterior en que cuando la anterior representa el cluster mediante su centroide, esta técnica lo representa mediante una observación o un dato en el conjunto, este punto será el medioide de cada cluster. Por otra parte, son muy similares en que ambas técnicas, dado el número de clusters k, agrupan los datos en k clusters.

Para aplicar esta técnica se suele utilizar el algoritmo PAM (Partitioning Around Medoids). En este algoritmo, a diferencia de en el caso anterior no se intenta minimizar la suma de las distancias al cuadrado con respecto al centroide si no que se intenta minimizar la suma de las diferencias de cada observación con respecto al medioide.

Una vez explicado todo esto pasamos a aplicar el algoritmo, en este caso seguiremos los mismos pasos que en el apartado anterior. Primero deduciremos el número óptimo de K y comprobamos este número mediante el índice de silueta. En este caso, para evitar redundancia de graficas dibujaremos los clusters obtenidos en el último momento. Además de esto, cabe destacar que se va a utilizar la distancia manhattan ya que como se ha podido ver en las referencias, es la distancia más utilizada en este algoritmo.

```
library(cluster)
library(factoextra)
#Dibujamos la curva que marcará el valor óptimo de clusters
fviz_nbclust(x =rating_DrugName, FUNcluster = pam, method = "wss", k.max = 10,
              diss = dist(rating_DrugName, method = "manhattan"))
```



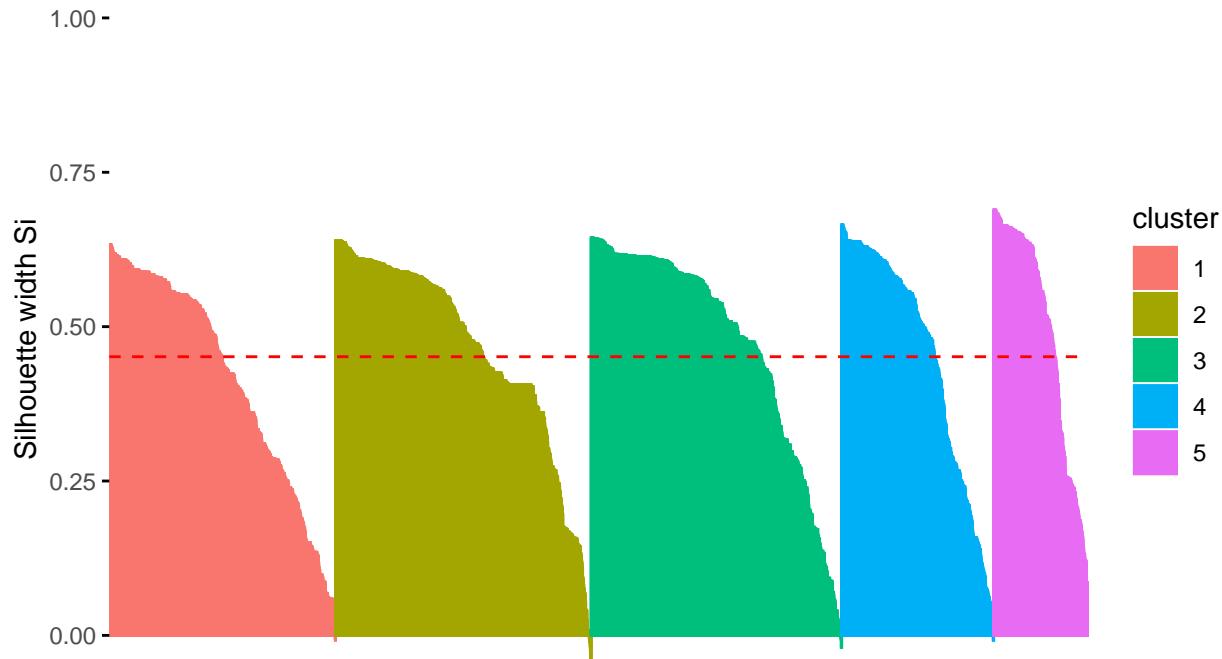
Como se puede ver el valor óptimo ronda los 5 clusters, ya que es donde la curva parece que se estabiliza. En este caso, podríamos decir que tiene sentido por de forma similar a lo comentado en el caso anterior.

Podríamos decir que es posible que la gente valore los medicamentos en 5 categorías principales. Aun así comprobaremos el índice para ver como se adecúan estos 5 clusters a nuestros datos.

```
#Calculamos el índice de silueta
res.pam <- eclust(seed = 12, rating_DrugName, "pam", k=5, graph = FALSE, labels(None))
#Lo dibujamos
fviz_silhouette(res.pam)
```

```
##   cluster size ave.sil.width
## 1       1 1155      0.41
## 2       2 1305      0.46
## 3       3 1280      0.47
## 4       4  776      0.45
## 5       5  484      0.48
```

Clusters silhouette plot  
Average silhouette width: 0.45

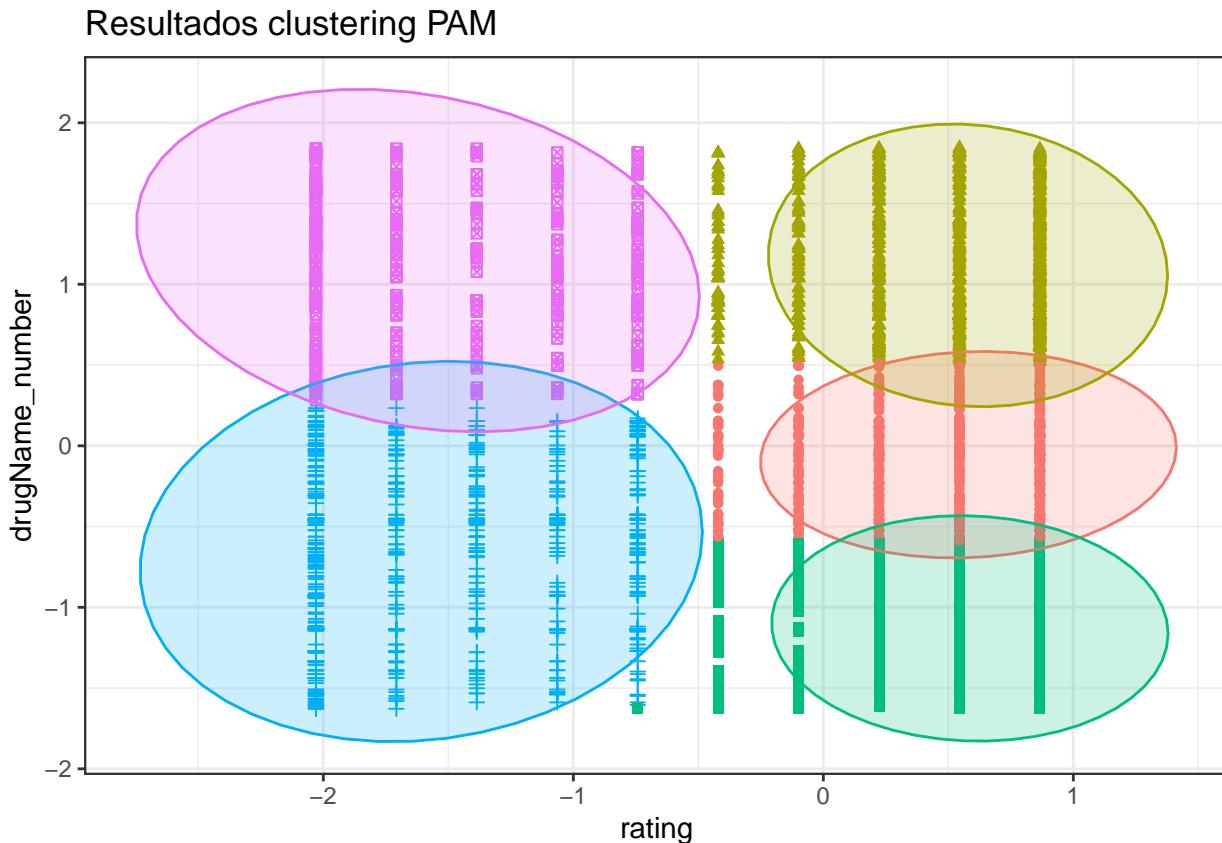


Como se puede ver en la imagen, los clusters se adecúan bastante bien a nuestros datos y es que si nos fijamos en el índice medio de silueta, es exactamente el mismo que con el algoritmo de k-medias de el caso anterior. En la siguiente imagen, pasamos por tanto, a representar los clusters. Como se puede ver en dicha imagen, los clusters que hemos obtenido son los mismos que hemos obtenido en el caso anterior. Sin embargo, parece que esta dividiendo los clusters en 2 grupos, por un lado los que tienen notas mayores que 5 y por otro las que tienen 5 o menos. Una vez hecho esto separa dichos grupos en grupos iguales. Esto no es muy interpretable y aunque pueda parecer que separa correctamente los mejores medicamentos de los peores esto no se cumple. Aun así este método al igual que el anterior es capaz de separar los datos en clusters y cubrirlos todos. Sin embargo, podemos decir que este algoritmo no es muy adecuado para nuestro problema. Hay que destacar la dificultad para interpretar los datos debido a la forma de nuestro dataset y es que el nombre del medicamento parece no estar relacionado con la nota de cada uno de estos.

```

fviz_cluster(object = res.pam, data = rating_DrugName, ellipse.type = "t",
             repel = FALSE, labelsize = 0) +
  theme_bw() +
  labs(title = "Resultados clustering PAM") +
  theme(legend.position = "none")

```



**Fuzzy Clustering** En todos los algoritmos anteriores, cada una de las técnicas se ha centrado en agrupar cada dato en un único cluster. En la técnica empleada en el fuzzy clustering o clustering difuso se hace todo lo contrario. Cada uno de los datos tendrá asociado un valor de pertenencia a todos los clusters. Esto se hace para intentar asemejar el razonamiento humano a las técnicas de clustering y es que como ejemplo podemos poner los colores. Un gris no es simplemente la combinación de negro y blanco, si no que también hay que tener en cuenta cuento negro y cuanto blanco tiene un gris ya que esto marcará tonos de grises diferentes. Esta técnica intentará, siguiendo el ejemplo de los colores, asignar un valor de pertenencia a cada dato con respecto a los diferentes clusters, negro y blanco en el ejemplo.

Una vez explicado de forma básica en que consiste el algoritmo pasamos a aplicarlo. De forma similar a los casos anteriores deberemos buscarnos un valor óptimo de k mediante el dibujo de la gráfica que asocia el índice medio de silueta y el número de clusters y una vez hecho esto representaremos los clusters. Debido a problemas de tiempo de cómputo no ha sido posible añadir en esta sección la gráfica del índice de silueta en función al número de clusters. Sin embargo, se utilizarán 2 clusters intentando agrupar los medicamentos en buenos o malos.

```

library(cluster)
library(factoextra)

```

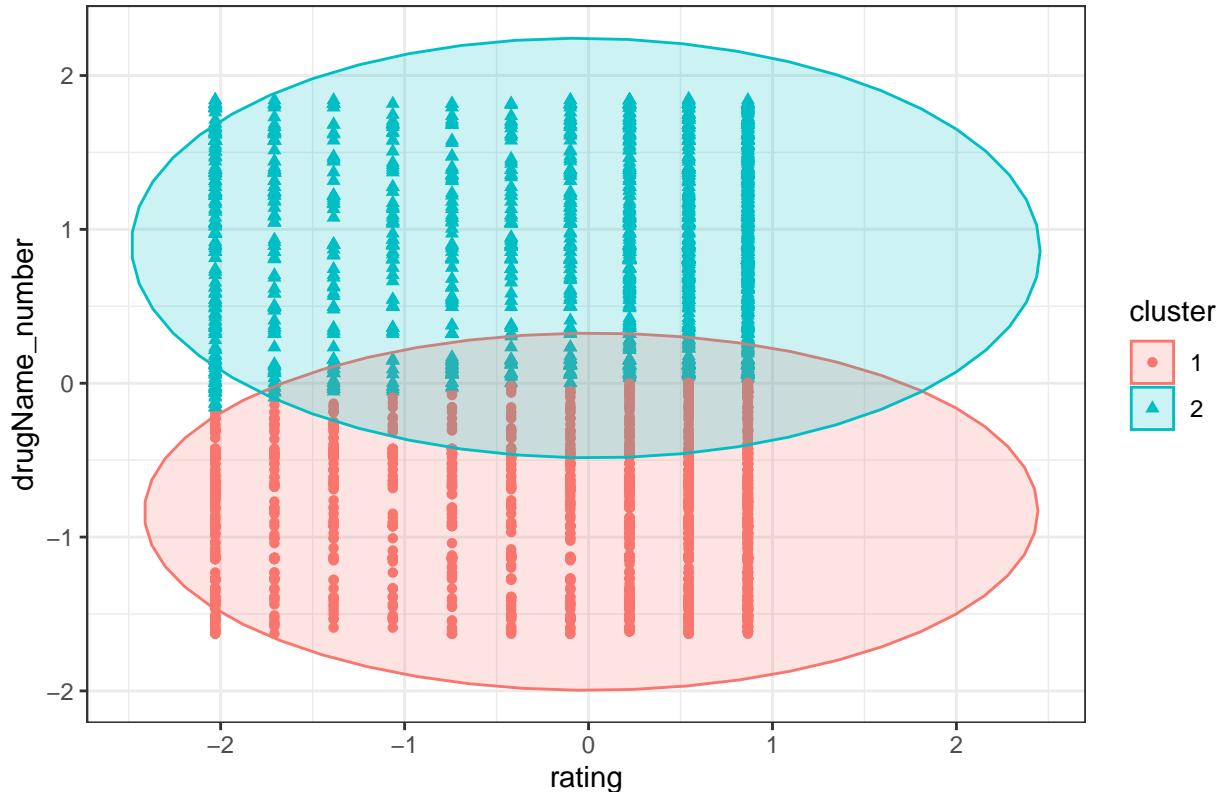
```

# fviz_nbclust(x = rating_DrugName, FUNcluster = fanny, method = "silhouette", k.max = 4,
#                diss = dist(rating_DrugName, method = "euclidean") ) +labs(title = "Número óptimo de clus-
res.fuzzy <- fanny(x = rating_DrugName, diss = FALSE, k = 2, metric = "euclidean",
                     stand = FALSE)

fviz_cluster(object = res.fuzzy, repel = FALSE, ellipse.type = "norm",
pallete = "jco", labelsize = 0) + theme_bw() + labs(title = "Fuzzy Cluster plot")

```

Fuzzy Cluster plot



Como se puede ver, en este caso el algoritmo esta dividiendo el conjunto de datos en dos mitades, dividiendolo por los medicamentos en lugar del rating. Esto nos lleva a pensar que este algoritmo no es especialmente útil para este caso. Ademas, se ha probado a utilizar un mayor numero de clusters pero en todos los casos el algoritmo con consigue converger en una solución. Por lo tanto esta ha sido la mejor solución obtenida.

### Agrupamiento de los medicamentos en función de las reviews de los usuarios.

En este apartado aplicaremos técnicas que nos permitirán analizar el texto de las reviews que ha obtenido cada tratamiento. Al igual que en el apartado anterior esta técnica pertenece al ámbito de aprendizaje no supervisado, por ello, se obtendrán agrupaciones que serás más o menos interpretables según nuestro criterio. Todas estas técnicas centradas en el análisis de textos se denominan “Text mining” o minería de textos. El text mining, por lo tanto, tiene como fin capturar conceptos y temas clave y descubrir relaciones y tendencias ocultas dentro de los textos y así poder obtener información valiosa sobre dichos textos o sobre sus autores.

Una vez definido que es el text mining hablaremos sobre como vamos a aplicar las técnicas de clustering a los textos. El principal problema se centra en que para aplicar estas técnicas necesitamos que los atributos sean numéricos. Para convertir todas las reviews en números utilizaremos la medida TF-IDF. Esta medida

consiste en que cada review será un vector de palabras y cada una de esas palabras tendrá asociado un peso de acuerdo a su frecuencia en la frase y a su importancia en el conjunto de todas las reviews. Pasamos por lo tanto a convertir las reviews en una matriz de pesos. Para calcular la matriz de distancias entre los distintos términos utilizaremos la medida de distancia de similitud del coseno. Esta medida se basa en calcular el coseno del angulo que forman dos vectores. Se utiliza esta distancia ya que en [23] se comenta que es especialmente útil para los algoritmos de clustering. Además de todo esto, aplicaremos los algoritmos a un conjunto de datos menor, ya que de esa forma podremos interpretar los resultados de forma más clara. Estos términos serán los 100 más comunes. De esta forma, al dibujar el dendograma será legible y tal vez nos permita obtener valiosa información de él. Por otra parte, también se creará otro conjunto con los 500 términos más comunes ya que para el algoritmo de fuzzy clustering no se ha conseguido ejecutar con la totalidad de los términos

```
#Partimos de la variable train_reviews_matriz calculada anteriormente.
#Calculamos el indice tfidf en función a ella
tdm.tfidf <- tm::weightTfIdf(train_reviews_matriz)

#Eliminamos las palabras que tienen una relevancia muy baja y lo convertimos a matriz
tdm.tfidf <- tm::removeSparseTerms(tdm.tfidf, 0.99)
tfidf.matrix <- as.matrix(tdm.tfidf)

# Utilizamos como medida de similitud la similitud del coseno (por tener un buen rendimiento con los algoritmos)
dist.matrix = proxy::dist(tfidf.matrix, method = "cosine")

## Registered S3 methods overwritten by 'proxy':
##   method           from
##   print.registry_field registry
##   print.registry_entry registry

#Mostramos la matriz de términos que hemos obtenido
tfidf.matrix[1:20, 1:20]

##      Terms
## Docs abl absolut accutan      acn      activ actual add adderal addict
##   1   0       0       0 0.00000000 0.0000000 0   0   0   0   0
##   2   0       0       0 0.00000000 0.0000000 0   0   0   0   0
##   3   0       0       0 0.07904579 0.0000000 0   0   0   0   0
##   4   0       0       0 0.00000000 0.0000000 0   0   0   0   0
##   5   0       0       0 0.00000000 0.0000000 0   0   0   0   0
##   6   0       0       0 0.00000000 0.0000000 0   0   0   0   0
##   7   0       0       0 0.00000000 0.0000000 0   0   0   0   0
##   8   0       0       0 0.00000000 0.0000000 0   0   0   0   0
##   9   0       0       0 0.00000000 0.0000000 0   0   0   0   0
##  10  0       0       0 0.00000000 0.0000000 0   0   0   0   0
##  11  0       0       0 0.00000000 0.0000000 0   0   0   0   0
##  12  0       0       0 0.00000000 0.0000000 0   0   0   0   0
##  13  0       0       0 0.00000000 0.0000000 0   0   0   0   0
##  14  0       0       0 0.00000000 0.0000000 0   0   0   0   0
##  15  0       0       0 0.00000000 0.0000000 0   0   0   0   0
##  16  0       0       0 0.00000000 0.0000000 0   0   0   0   0
##  17  0       0       0 0.13072958 0.1085357 0   0   0   0   0
##  18  0       0       0 0.00000000 0.0000000 0   0   0   0   0
##  19  0       0       0 0.00000000 0.0000000 0   0   0   0   0
##  20  0       0       0 0.00000000 0.0000000 0   0   0   0   0
```

```

##      Terms
## Docs addit adhd adipex adjust    affect age ago allow almost      along
## 1     0     0     0     0     0 0.0000000 0     0     0     0 0.00000000
## 2     0     0     0     0     0 0.0000000 0     0     0     0 0.00000000
## 3     0     0     0     0     0 0.0000000 0     0     0     0 0.00000000
## 4     0     0     0     0     0 0.0000000 0     0     0     0 0.00000000
## 5     0     0     0     0     0 0.0000000 0     0     0     0 0.00000000
## 6     0     0     0     0     0 0.0000000 0     0     0     0 0.00000000
## 7     0     0     0     0     0 0.0000000 0     0     0     0 0.00000000
## 8     0     0     0     0     0 0.0000000 0     0     0     0 0.00000000
## 9     0     0     0     0     0 0.0000000 0     0     0     0 0.00000000
## 10    0     0     0     0     0 0.0000000 0     0     0     0 0.00000000
## 11    0     0     0     0     0 0.0000000 0     0     0     0 0.00000000
## 12    0     0     0     0     0 0.0000000 0     0     0     0 0.00000000
## 13    0     0     0     0     0 0.0000000 0     0     0     0 0.00000000
## 14    0     0     0     0     0 0.0000000 0     0     0     0 0.00000000
## 15    0     0     0     0     0 0.0000000 0     0     0     0 0.00000000
## 16    0     0     0     0     0 0.0000000 0     0     0     0 0.00000000
## 17    0     0     0     0     0 0.0000000 0     0     0     0 0.00000000
## 18    0     0     0     0     0 0.5158429 0     0     0     0 0.00000000
## 19    0     0     0     0     0 0.0000000 0     0     0     0 0.08066643
## 20    0     0     0     0     0 0.0000000 0     0     0     0 0.00000000

##      Terms
## Docs already
## 1     0
## 2     0
## 3     0
## 4     0
## 5     0
## 6     0
## 7     0
## 8     0
## 9     0
## 10    0
## 11    0
## 12    0
## 13    0
## 14    0
## 15    0
## 16    0
## 17    0
## 18    0
## 19    0
## 20    0

```

```

#Obtenemos cuales son las frecuencias acumuladas de todos los terminos de la matriz.
#Creamos un vector vacio
vector_frecuencias <- c()
#Recorremos todos los terminos de la matriz
for(i in 1:length(tfidf.matrix[1])){
  #Almacenamos en el vector la frecuencia total de cada termino
  vector_frecuencias[i] <- sum(tfidf.matrix[,i])
}

```

```

#Ordeno y selecciono las 100 frecuencias mayores
frecuencias <- order(vector_frecuencias,decreasing = TRUE)[1:100]

#Creo una nueva matriz donde almacenar esos 100 resultados
tfidf.matrix_mini <- tfidf.matrix[,frecuencias]

#Almacenamos la matriz transpuesta, que será necesaria para algunos algoritmos.
tfidf.matrix_mini <- t(tfidf.matrix_mini)

#Calculo las distancias igual que en la matriz anterior
dist.matrix_mini = proxy::dist(tfidf.matrix_mini, method = "cosine")

#Para el algoritmo de fuzzy clustering se empleara una matriz de tamaño intermedio, que contendrá las 500 primeras frecuencias
frecuencias <- order(vector_frecuencias,decreasing = TRUE)[1:500]
tfidf.matrix_intermedia <- tfidf.matrix[,frecuencias]
tfidf.matrix_intermedia <- t(tfidf.matrix_intermedia)
dist.matrix_intermedia = proxy::dist(tfidf.matrix_intermedia, method = "cosine")

```

Como se puede ver tenemos una matriz de términos en la cual cada término tiene un vector el cual representa la relevancia de ese término en un documento. Una vez hecho esto podemos pasar a aplicar los algoritmos de agrupamiento. Los algoritmos que se aplicarán serán el algoritmo de k-medias, clustering jerárquico, dendograma y k-medoides. Además, hay que destacar que utilizaremos las mismas dos distancias que se utilizaron en el clustering jerárquico de la sección anterior para calcular ahora ese cluster. Para la elección del número de clusters se han dibujado primero los datos para ver la distribución de estos y se ha observado que se pueden apreciar 3 agrupaciones diferentes. Por lo tanto, se ha escogido como tamaño de cluster 3.

A continuación se va a separar la ejecución de cada algoritmo en bloques separados, primero el jerárquico y luego los particionales.

## Algoritmos Particionales

En este apartado ejecutaremos varios algoritmos e incluiremos sus resultados uno a uno. Al final se comentarán los resultados obtenidos.

```

#Definimos una variable que almacenará el número de clusters que se usarán.
truth.K = 3
#Calculamos los clusters
cluster_kmeans <- kmeans(tfidf.matrix, truth.K)
#Pintamos la nube de puntos
points <- cmdscale(t(dist.matrix), k = 2)
palette <- colorspace::diverge_hcl(truth.K) # Creating a color palette
plot(points, main = 'K-Means clustering', col = as.factor(cluster_kmeans$cluster),
     mai = c(0, 0, 0, 0), mar = c(0, 0, 0, 0),
     xaxt = 'n', yaxt = 'n', xlab = '', ylab = '')

```

**Algoritmo K-medias** En este caso podemos ver como el algoritmo no ha sido capaz de separar el conjunto de datos en diversos clusters. En lugar de eso el algoritmo ha incluido la gran mayoría de puntos en un solo cluster. Esto se puede deber a que estos datos se encuentren muy próximos los unos de los otros, de forma que los algoritmos particionales no pueden separar bien estos datos en diferentes grupos. De ser así, se obtendrán resultados parecidos en todos los algoritmos particionales.

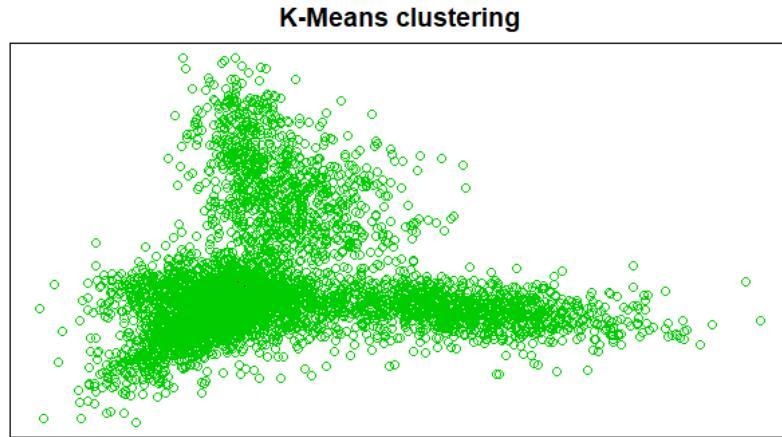


Figure 8: Resultado de algoritmo K-means en reviews.

```
##DBSCAN-----
# Calculamos el cluster
cluster_dbSCAN <- 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(truth.K)
plot(points, main = 'Density-based clustering',
     col = as.factor(cluster_dbSCAN$cluster),
     mai = c(0, 0, 0, 0), mar = c(0, 0, 0, 0),
     xaxt = 'n', yaxt = 'n', xlab = '', ylab = '')
```

**Algoritmo DBSCAN** Como era de esperar, este algoritmo particional tampoco ha sido capaz de encontrar ningun cluster. Como se comentó anteriormente, se cree que esto se debe a que los datos estan muy compactos y estos algoritmos no funcionan correctamente con datos no disjuntos.

### Algoritmo jerárquico

```
#Jerarquico-----
#Calculamos los clusters
#Distancia complete
jerarquico_complete <- hclust(dist.matrix, method = "complete")

#Distancia average
jerarquico_average <- hclust(dist.matrix, method = "average")

#Repetimos para el conjunto de 100 datos
#Calculamos los clusters
```

### Density-based clustering

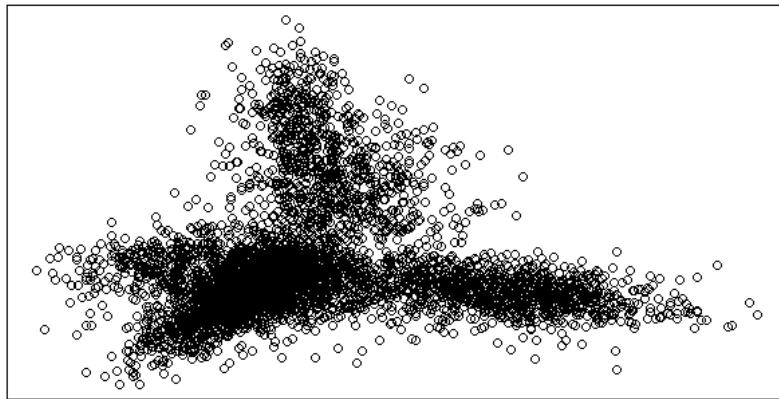


Figure 9: Resultado de algoritmo K-medioides en reviews.

```

#Distancia complete
jerarquico_complete_mini <- hclust(dist.matrix_mini, method = "complete")

#Distancia average
jerarquico_average_mini <- hclust(dist.matrix_mini, method = "average")

# Pintamos la nube de puntos
points <- cmdscale(dist.matrix, k = 2)
#palette <- colorspace::diverge_hcl(truth.K) # Creating a color palette
previous.par <- par(mfrow=c(1,2), mar = rep(1.5, 4))
#Nube de puntos
plot(points, main = 'Hierarchical clustering, distancia complete',
      col = as.factor(cutree(jerarquico_complete, k = truth.K)),
      mai = c(0, 0, 0, 0), mar = c(0, 0, 0, 0),
      xaxt = 'n', yaxt = 'n', xlab = '', ylab = '')

plot(points, main = 'Hierarchical clustering, distancia average',
      col = as.factor(cutree(jerarquico_average, k = truth.K)),
      mai = c(0, 0, 0, 0), mar = c(0, 0, 0, 0),
      xaxt = 'n', yaxt = 'n', xlab = '', ylab = '')

# Dibujamos el dendograma solo del conjunto mas pequeño ya que será el mas legible.

{plot(jerarquico_complete_mini, main = 'Dendrogram, linkage complete', cex=0.9, hang=-1)
rect.hclust(jerarquico_complete_mini, 3, border=rainbow(3))}

{plot(jerarquico_average_mini, main = 'Dendrogram, linkage average', cex=0.9, hang=-1)
rect.hclust(jerarquico_average_mini, 3, border=rainbow(3))}
```

En este caso se puede ver como mediante los algoritmos jerárquicos no se ha conseguido ningun resultado relevante y es que la gran mayoría de datos del conjunto no se han asignado a ningun dataset. Esto se puede deber a la forma que tienen nuestros datos. Sin embargo, mediante el análisis del dendograma

Hierarchical clustering, distancia completa

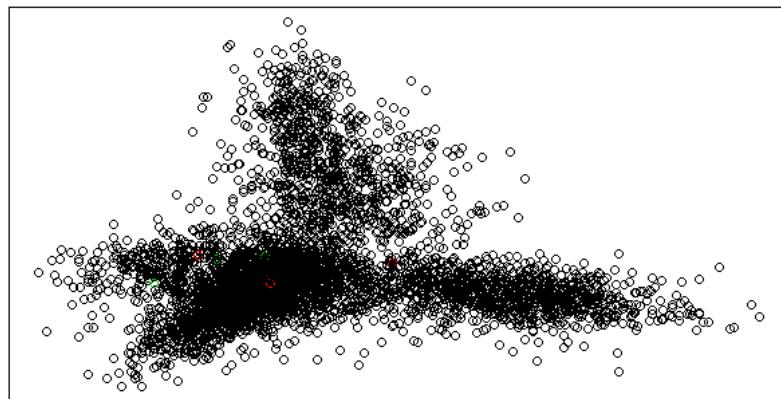


Figure 10: Resultado de algoritmo jerárquico con linkage completo o máximo.

se puede ver como hay algunas palabras que si que ha conseguido agrupar en clusters, en estos dendogramas se pueden ver los clusters obtenidos del análisis de las 100 palabras más relevantes. En el primer dendrograma se puede ver el obtenido mediante el linkage completo o máximo. Este es el linkage que nos ha proporcionado mejores resultados y es que los clusters aparecen más dispersos. Sin embargo, estos clusters son de difícil interpretación. En el primero parece hacer referencia a las reviews referentes a la enfermedad “acne”, esto se puede ver por términos como “face”, “acn” y “skin”. En el segundo cluster parecen agruparse términos referentes a “obesity” y a “birth Control”, llegamos a esta conclusión a través de términos como “eat”, “lbs”, “lost”, “pound”, “weight”, “period”, “control” y “pill”. Por último, en el tercer cluster se agrupan el resto de enfermedades. Esto se puede ver por las palabras “headache”, “insomnia”, “anxiety”, “depress” o “pain”.

Sin embargo, cabe destacar que estas agrupaciones y su interpretación no son más que meramente especulativas y subjetivas.

### Extensiones de los algoritmos anteriores

En este apartado se verán algoritmos los cuales son modificaciones o variaciones de los anteriores, en este caso se trata del algoritmo de los k-medoides y de fuzzy clustering.

```
#k-medoides-----
# Calculamos el cluster
library(factoextra)
library(cluster)
res.pam <- eclust(seed = 12, dist.matrix, "pam", k=truth.K, graph = FALSE, labels(None))
# Lo dibujamos
fviz_cluster(object = res.pam, data = dist.matrix, ellipse.type = "t",
             repel = FALSE, labelsize = 0) +
  theme_bw() +
  labs(title = "Resultados clustering PAM") +
  theme(legend.position = "none")
```

### Hierarchical clustering, distancia average

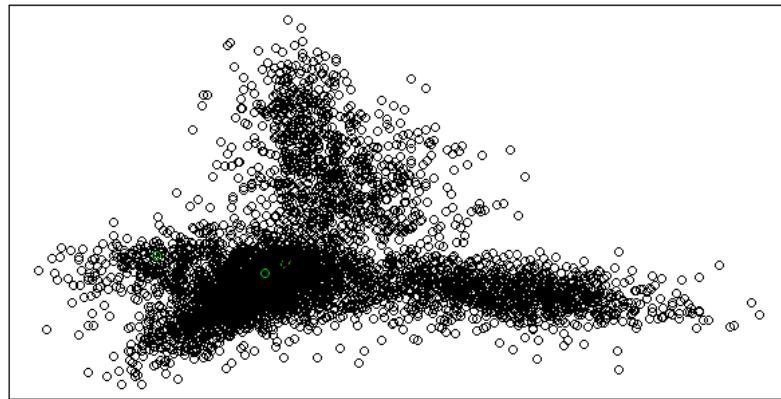


Figure 11: Resultado de algoritmo jerárquico con linkage medio.

**Algoritmo K-medoides** En este caso se puede ver como se obtiene un resultado parecido a los obtenidos en los algoritmos particionales. Por lo que no se puede extraer mucha información de este algoritmo.

```
library(cluster)
library(factoextra)
#Fuzzy Clustering-----
#Calculamos el cluster
set.seed(4)
res.fuzzy <- fanny(x = dist.matrix_intermedia, diss = FALSE, k = 3, metric = "euclidean",
                     stand = FALSE, memb.exp = 1.001)
#Lo dibujamos
fviz_cluster(object = res.fuzzy, repel = FALSE, ellipse.type = "norm",
              pallete = "jco", labelsize = 7) + theme_bw() + labs(title = "Fuzzy Cluster plot")
```

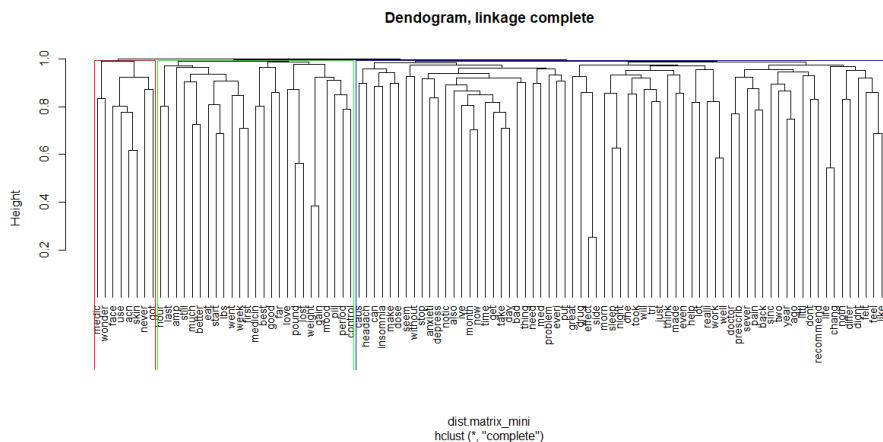


Figure 12: Dendogramma con linkage completo o máximo.

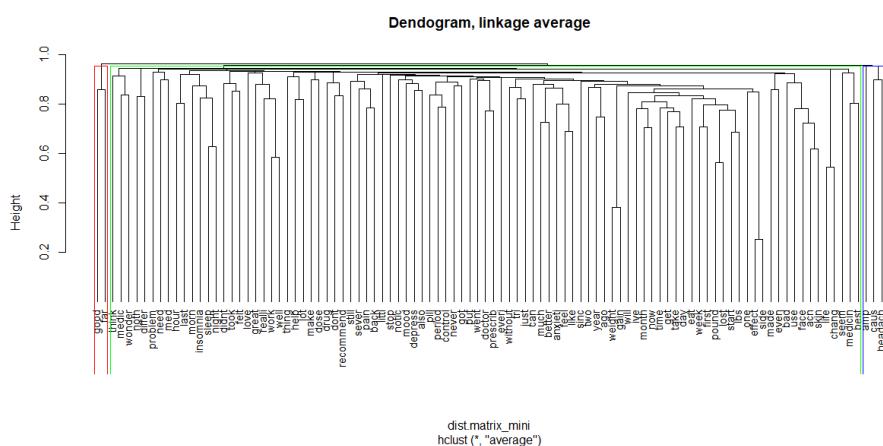
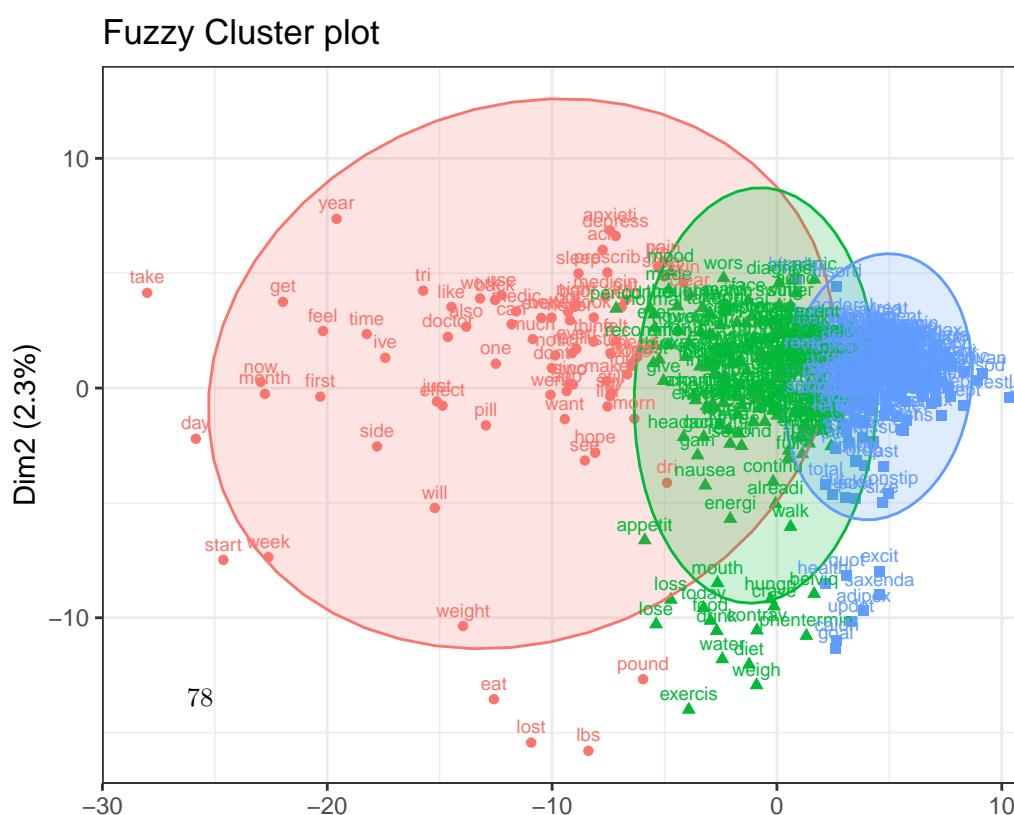


Figure 13: Dendogramma con linkage medio.



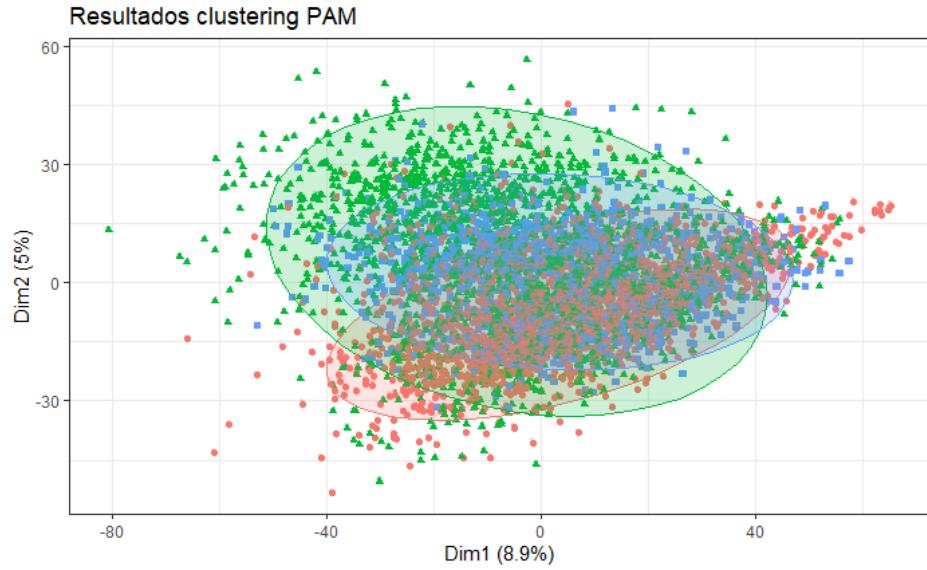


Figure 14: Resultado de algoritmo K-medoides en reviews.

En este caso se puede ver como este algoritmo si que ha sido capaz de separar los datos en 3 clusters. Veamos que terminos ha agrupado en cada cluster. Para ello vamos a mostrar 20 palabras de cada grupo para ver si se puede encontrar alguna relación entre ellas.

```
#Cluster 1
cat("Cluster 1:\n ")

## Cluster 1:
## 

(which(res.fuzzy$clustering == 1))[1:20]

##      work      pain      take      help      day      effect      sleep      month      week
##      1          2          3          4          5          6          7          8          9
##      feel      year      side      medic      start      now      medicin      anxieti      ive
##      10         11         13         14         15         16         17         18         19
##      use       get
##      20         21

#Cluster 2
cat("\nCluster 2:\n ")

## 
## Cluster 2:
## 

(which(res.fuzzy$clustering == 2))[1:20]

##      great      well      good      hour      drug      gain      made
##      12         33         38         43         48         51         54
```

```

##      period      didnt      far      last      control      lot      love
##      57          58          59          63          68          71          78
##      chang recommend mood      med      think problem
##      80          83          85          87          89          90

#Cluster 3
cat("\nCluster 3:\n ")

## 
## Cluster 3:
## 

(which(res.fuzzy$clustering == 3))[1:20]

##      caus   wonder     gave ambien    wake  asleep    amaz bipolar    fall
##      79       88      112     122      125      127      131      147      155
##      awesom dosag relief couldnt sure sometim adderal  xanax minut
##      158      163      165     172      173      188      189      196      200
##      thank product
##      203      207

```

Tras analizar los elementos de cada cluster se puede ver como la interpretación de estos es bastante difícil y subjetiva. Aun así vamos a intentar darle algún significado a cada uno de los clusters. En el primer cluster, las palabras “pain”, “feel”, “anxiety”, “help” o “sleep” nos sugieren que este grupo se puede haber formado entorno a términos que hacen referencia al estado en el que se encontraban los pacientes antes de tomar los medicamentos. Por otro lado, el segundo cluster nos sugiere que podrían tratarse de términos agrupados como una review al medicamento o a sus efectos beneficiosos. Esto podemos verlo en términos como “great”, “well”, “good”, “drug”, “love”, “mood” o “gain”. Por último, en el tercer cluster podemos ver como al contrario que en el caso anterior, este agrupa los términos por los efectos adversos del medicamento prescrito. Esto se puede entrever por los términos “caus”, “gave”, “asleep”, “wake”, “bipolar”, “couldnt” o por último “fall”.

Despues de ver todas estas graficas podemos ver como la mayoria de los algoritmos de clustering no se adecuan correctamente a estos datos. Y es que aunque parezca que hay una forma definida, a dichos algoritmos les cuesta reconocer los grupos que la componen al estar todos muy agrupados. Sin embargo, se puede ver que para el algoritmo de fuzzy clustering se obtiene el mejor resultado, esto puede deberse al carácter más permisivo de este algoritmo el cual asigna pesos de pertenencia a cada uno de los datos con respecto a los clusters obtenidos, de forma que cada dato pertenece a un cluster. Por otro lado se puede ver como dentro de los algoritmos jerárquicos, el linkage basado en la media no funciona muy bien, por lo tanto para este caso podremos decir que en este caso funciona mejor el linkage completo.

Además de todo esto, se puede ver como los clusters que se han podido interpretar, tanto en el dendograma del algoritmo jerárquico como en el algoritmo de clustering difuso, no tienen nada que ver unos con otros. Pudiendo tener uno más significado semántico que otro. Para nosotros, el clustering difuso ha sido el que ha obtenido unos mejores resultados, consiguiendo agrupar mejor que ningun otro los datos en grupos semánticamente relevantes. Sin embargo, cabe destacar por última vez que estos grupos son totalmente subjetivos, cosa que viene directamente relacionada con la subjetividad intrínseca al dataset.

## Conclusiones finales.

Como hemos podido comprobar a lo largo del documento, los tres campos más relevantes de ambos conjuntos de datos son aquellos que representan la enfermedad del paciente, el fármaco recetado así como su

propia opinión acerca del tratamiento. Es por ello por lo que la mayoría de los algoritmos aplicados se han concentrado en tales datos.

En primer lugar, destacamos que tras realizar el **análisis de sentimientos** acerca de las críticas de los enfermos nos pareció bastante sorprendente que la mayoría de ellas fueran **positivas**, puesto que no es lo habitual cuando se trata de datos asociados a enfermedades, por lo que podemos reflexionar que estos pacientes han sido correctamente atendidos y diagnosticados por especialistas competentes que han sabido, en la mayoría de ocasiones, paliar sus dolencias. En relación a las **reglas de asociación** obtenidas era de esperar que los términos con un mayor vínculo fuesen aquellos que definiesen el nombre de una enfermedad, como *Birth Control*, así como las expresiones comunes representativas de la toma de un medicamento tanto verbales como temporales.

En la sección de **clasificación** hemos entrenado el mejor clasificador obtenido en todo este documento a través de la técnica **Random Forest** obteniendo tasas de error aproximadas al 0.5%, por lo que podemos concluir que este es el modelo predictivo más competente y que mejor sabe predecir **las enfermedades en función de las opiniones de los pacientes** para nuestro conjuntos particulares de datos. Nuestra reflexión acerca del excelente comportamiento de este algoritmo para nuestro dataset reside en la capacidad de introducir cierta aleatoriedad en el entrenamiento, ya que esta cualidad le proporciona un alto grado de flexibilidad para no sobreajustarse a los datos de entrenamiento y lidiar con datos caracterizados por la incertidumbre. Y es que, efectivamente, cada crítica escrita por un paciente tiene un contenido distinto y es totalmente subjetiva y dependiente de la persona que la ha redactado. Es por ello por lo que pensamos que esta técnica ha proporcionado un clasificador muy competitivo para nuestro dataset y el modelo de predicción establecido basado en las *reviews*.

En relación a las técnicas de **regresión** hemos podido comprobar cómo ha variado la naturaleza del clasificador a la hora de predecir la efectividad del medicamento en función de **la puntuación o de las opiniones de los pacientes**. Gracias al análisis que hemos llevado a cabo en estas dos secciones hemos podido comprobar la importancia de escoger una buena variable predictora para poder entrenar un clasificador con una buena capacidad de generalización. Asimismo, también han quedado patentes las grandes diferencias entre la predicción numérica o con texto, pues pueden dar lugar a modelos predictivos totalmente distintos, como ha sido nuestro caso. En el primero el mejor clasificador ha sido entrenado con **regresión polinómica** de grado 2, mientras que con los términos de las *reviews* como variables independientes el mejor modelo predictivo lo hemos conseguido con **regresión lineal simple**.

Para acabar, hablaremos sobre las conclusiones a las que hemos llegado con respecto a las técnicas de **Clustering o Agrupación**. En este apartado no se consiguieron los resultados esperados y es que en la subjetividad del conjunto de datos que hemos utilizado ha marcado la difícil interpretación de los resultados y la baja adecuación de estos a nuestras expectativas. Sin embargo hemos podido ver y estudiar el funcionamiento de los algoritmos de clustering más utilizados hoy en día y es que estos resultados también se adecúan a lo que es el mundo real: No siempre todo está relacionado tal y como nosotros esperamos. Y es que, aunque para nosotros, dos variables pueden estar muy relacionadas es posible que en realidad no se encuentren correlacionadas. Esto último se ha podido ver en el análisis de rating y medicamentos que apriori podría parecer que se encuentran muy relacionadas pero sin embargo y como se vio en el estudio inicial del dataset, estas dos variables no se encuentran correlacionadas. Esto se puede deber a que lo que para una persona es bueno para otra no lo es tanto y más en los medicamentos cuando hay gente a la que una medicina le afecta mejor o peor que a otra.

En cuanto a la parte de **text mining** se puede ver que los resultados varían mucho en función de que algoritmo usemos, sin embargo, parece que el algoritmo de clustering difuso funciona mejor que el resto. Aun así la subjetividad del dataset, vuelve a jugar una mala pasada y es bastante difícil sacar alguna conclusión clara acerca del significado de los resultados.

## Bibliografía.

1. Jessica Li, Kaggle University Club Hackathon, *UCI ML Drug Review Dataset*, [https://www.kaggle.com/jessicali9530/kuc-hackathon-winter-2018#drugsComTrain\\_raw.csv](https://www.kaggle.com/jessicali9530/kuc-hackathon-winter-2018#drugsComTrain_raw.csv)
2. *Introduction to Corpus*, <https://cran.r-project.org/web/packages/corpus/vignettes/corpus.html>
3. Ingo Feinerer, 12/12/2019, *Introduction to the tm Package, Text Mining in R*, <https://cran.r-project.org/web/packages/tm/vignettes/tm.pdf>
4. Oscar Ramírez-Alán, *Correlación y Regresión Lineal*, 9/10/2017, <https://rpubs.com/osoramirez/316691>
5. *Wordcloud2 introduction*, 03/01/2018, <https://cran.r-project.org/web/packages/wordcloud2/vignettes/wordcloud.html>
6. Julia Silge, David Robinson, *Sentiment analysis with tidy data*, 24/11/2019, <https://www.tidytextmining.com/sentiment.html>
7. Stefan Feuerriegel, Nicolas Proellosch, *SentimentAnalysis Vignette*, 26/03/2019, <https://cran.r-project.org/web/packages/SentimentAnalysis/vignettes/SentimentAnalysis.html>
8. Joaquín Amat Rodrigo, *Reglas de asociación y algoritmo Apriori con R*, 2018, [https://rpubs.com/Joaquin\\_AR/397172](https://rpubs.com/Joaquin_AR/397172)
9. RDocumentation, *apriori*, <https://www.rdocumentation.org/packages/arules VERSIONS/1.6-4/topics/apriori>
10. Stackoverflow, *How can we find support and confident in apriori for rules?*, 2018, <https://stackoverflow.com/questions/43588163/how-can-we-find-support-and-confident-in-apriori-for-rules>
11. Aisha Javed, *A Deep Dive into Naïve Bayes for Text Classification*, 25/10/2018, <https://blog.datasciencedojo.com/unfolding-naive-bayes-from-scratch-part-1/>
12. Rohit Katti, *Naïve Bayes Classification for Sentiment Analysis of Movie Reviews*, 30/04/2016, <https://rpubs.com/cen0te/naivebayes-sentimentpolarity>
13. Manual del paquete *class*, <https://cran.r-project.org/web/packages/class/class.pdf>
14. RPubs by RStudio, *Text Mining MBTI with Knn*, [http://rpubs.com/JWB62/394875](https://rpubs.com/JWB62/394875)
15. Foro de Analytics Vidhya, *How to choose the value of K in knn algorithm*, 2015, <https://discuss.analyticsvidhya.com/t/how-to-choose-the-value-of-k-in-knn-algorithm/2606/5>
16. Stackoverflow *Decision Trees For Document Classification*, <https://stackoverflow.com/questions/3114734/decision-trees-for-document-classification>
17. Sibanjan Das, *Decision Trees and Pruning in R*, 2017, <https://dzone.com/articles/decision-trees-and-pruning-in-r>
18. Manual de *rpart.control*, <https://www.rdocumentation.org/packages/rpart VERSIONS/4.1-15/topics/rpart.control>
19. Manual de *randomForest*, <https://www.rdocumentation.org/packages/randomForest VERSIONS/4.6-14/topics/randomForest>
20. *Cluster Analysis*, <https://www.statmethods.net/advstats/cluster.html>
21. *Cluster Analysis* in R simplified and enhanced, <https://www.datanovia.com/en/blog/cluster-analysis-in-r-simplified-and-enhanced/>
22. Joaquín Amat Rodrigo, *Clustering y heatmaps: aprendizaje no supervisado*, Septiembre de 2017 [https://rpubs.com/Joaquin\\_AR/310338](https://rpubs.com/Joaquin_AR/310338)
23. *Text clustering* in R, <https://medium.com/@SAPCAI/text-clustering-with-r-an-introduction-for-data-scientists-c406e7454e76/>
24. Basic *Text Mining* in R, Philip Murphy, [https://rstudio-pubs-static.s3.amazonaws.com/265713\\_cbef910aee7642dc8b62996e38d2825d.html](https://rstudio-pubs-static.s3.amazonaws.com/265713_cbef910aee7642dc8b62996e38d2825d.html)
25. *K Means Clustering* in R, <https://www.r-bloggers.com/k-means-clustering-in-r/>
26. *Clustering* categorical data with R, <https://dabblingwithdata.wordpress.com/2016/10/10/clustering-categorical-data-with-r/>