



Universidade de Vigo

Trabajo Fin de Máster

Text Analytics para Procesado Semántico

María Calvo Torres

Máster en Técnicas Estadísticas

Curso 2016-2017

Propuesta de Trabajo Fin de Máster

| |
|---|
| Título en galego: Text Analytics para Procesado Semántico |
| Título en español: Text Analytics para Procesado Semántico |
| English title: Text Analytics for Semantic Processing |
| Modalidad: Modalidad B |
| Autor/a: María Calvo Torres, Universidade de Vigo |
| Director/a: Rubén Fernández Casal, Universidade da Coruña; |
| Tutor/a: Juan Ramón González Hernández, Oesía Networks S.L.; |
| Breve resumen del trabajo: <p>La Minería de Textos proporciona herramientas para obtener información y conocimiento de conjuntos de datos no estructurados. Durante los últimos años ha aumentado notablemente el interés en este tipo de técnicas y se han desarrollado numerosas herramientas.</p> <p>El trabajo propuesto consistiría inicialmente en una revisión de las técnicas más relevantes en este contexto y de las herramientas disponibles para su aplicación en la práctica (principalmente paquetes de R). El objetivo final sería la implementación de estos métodos para el análisis de conjuntos de datos reales.</p> <p>Entre las principales tareas que aparecen en un análisis de este tipo estarían:</p> <ul style="list-style-type: none">▪ El acceso a datos (en distintos formatos).▪ El preprocesamiento del texto y la generación del corpus lingüístico.▪ La construcción del espacio para el análisis semántico.▪ La obtención de distancias y medidas de similitud. <p>Todo esto apoyado por el conocimiento experto de Oesía en la materia.</p> |
| Recomendaciones: <p>Es recomendable que el alumno disponga de conocimientos básicos de programación, especialmente en R y/o Python.</p> |
| Otras observaciones: |

Don Rubén Fernández Casal, profesor contratado doctor de la Universidade da Coruña, y don Eduardo San Miguel Martín, jefe de proyecto en Oesía Networks informan que el Trabajo Fin de Máster titulado

Text Analytics para Procesado Semántico

fue realizado bajo su dirección por doña María Calvo Torres para el Máster en Técnicas Estadísticas. Estimando que el trabajo está terminado, dan su conformidad para su presentación y defensa ante un tribunal.

En Santiago, a 12 de Julio de 2017.

El director:



Don Rubén Fernández Casal

El jefe de proyecto en Oesía S.L.:



Don Eduardo San Miguel Martín

La autora:

Doña María Calvo Torres

Agradecimientos

Me gustaría agradecerles a Rubén Fernández Casal y a Eduardo San Miguel Martín el tiempo invertido en este proyecto y, especialmente, a Jordi Aceiton Cardona y a David Griñán Martínez, por tantos dolores de cabeza sufridos y tanto esfuerzo dedicado a que este clasificador diese sus frutos.

Muchísimas gracias también a Irma Riádigos Sánchez, por leerse cada una de las mil versiones del trabajo y, como no, a Daniel Rodríguez López, por todas esas horas sin dormir y el apoyo que me ha dado desde el primer momento.

A todos los que habéis puesto un poquito de vosotros en este trabajo, solamente puedo decir, GRACIAS.

Índice general

| | |
|---|-----------|
| Resumen | XI |
| 1. Introducción | 1 |
| 2. Manipulación de textos | 3 |
| 2.1. Obtención de los datos y creación del corpus | 3 |
| 2.2. Preprocesamiento | 4 |
| 2.3. Ejemplo | 6 |
| 3. Análisis exploratorio de textos | 9 |
| 3.1. Resumen numérico | 9 |
| 3.1.1. La matriz de documentos-términos | 9 |
| 3.1.2. Importancia de los términos | 10 |
| 3.2. Métodos gráficos: el diagrama de barras y la nube de palabras | 15 |
| 4. Métodos de inferencia | 17 |
| 4.1. Comparación de documentos y términos | 17 |
| 4.1.1. Distancia euclídea | 17 |
| 4.1.2. Distancia del coseno | 17 |
| 4.1.3. Correlación | 19 |
| 4.2. Técnicas de reducción de dimensión | 19 |
| 4.2.1. Frecuencias | 20 |
| 4.2.2. Sparsity | 21 |
| 4.2.3. Asignación de Dirichlet latente (LDA) | 22 |
| 4.3. Clasificación de un nuevo documento | 24 |
| 4.3.1. Particionamiento de los datos | 26 |
| 4.3.2. Métodos de clasificación basados en árboles | 26 |
| 4.3.3. Regresión logística | 37 |
| 4.3.4. Los k vecinos más próximos (k-nearest neighbour classification, kNN) | 38 |
| 4.3.5. SVM o máquina de soporte vectorial | 39 |
| 4.3.6. Resumen de los métodos para el ejemplo del clasificador. | 44 |
| 5. Aplicación en <i>Shiny</i> | 47 |
| Bibliografía | 51 |

Resumen

Resumen en español

Este trabajo trata de introducir las principales técnicas de la minería de textos y sus diversas aplicaciones. Para ello, en el primer capítulo, se comienza a explicar qué es la minería de textos y cuáles son sus principales usos actualmente. Seguidamente, se explican varias formas de cómo obtener los datos y luego se ilustra el preprocesado de los mismos con un ejemplo. En el tercer capítulo, se hace un análisis exploratorio de los datos y, a continuación, se explican los métodos más utilizados para la clasificación de documentos, se entrenan los modelos y se validan.

Por último, se muestra una aplicación *Shiny* del clasificador de textos que se ha construido siguiendo paso a paso las explicaciones del trabajo. Su finalidad es una mejor comprensión del tema con la ayuda de un método visual interactivo.

Para la realización de este trabajo, se ha recibido la ayuda de la empresa Oesía, experta en el ámbito de la minería de textos.

English abstract

This project tries to introduce the main text mining techniques and their different applications. For this, in the first chapter, it begins to explaining what text mining is and which are its main applications at the moment. Consecutively, several ways of how to obtain the data are explained and then data preprocessing is illustrated with an example. In the third chapter, an exploratory data analysis is done, and then the most used methods for document classification are explained, the models are trained and validated.

Finally, it shows a Shiny text classifier application that has been constructed following step by step explanations of the project. Its objective is a better understanding of the subject with the help of an interactive visual method.

For the realization of this project, the help of Oesía company, expert in the field of text mining, was received.

Resumo en galego

Este traballo trata de introduci-las principais técnicas da minería de textos e as súas diversas aplicacións. Para isto, no primeiro capítulo, comézase a explicar que é a minería de textos e cales son as súas principais usos actualmente. Deseguido, explicanse varios xeitos de como obte-los datos e, logo ilústrase o preprocesado dos mesmos cun exemplo. No terceiro capítulo, faise unha análise exploratoria dos datos e, a continuación, explicanse os métodos máis utilizados para a clasificación de documentos, adéstranse os modelos e válídanse.

Por último, amósase unha aplicación *Shiny* do clasificador de textos que se construíu seguindo paso a paso as explicacións do traballo. A súa finalidade é unha mellor comprensión do tema coa axuda dun método visual interactivo.

Para a realización deste traballo, recibíuse a axuda da empresa Oesía, experta no ámbito da minería de textos.

Capítulo 1

Introducción

En los últimos años ha proliferado el almacenamiento de enormes cantidades de datos, y aunque cada día aumenta el volumen de información disponible, en comparación, la capacidad de procesamiento de esta información permanece prácticamente constante. El análisis de textos, o *text analytics*, es un área nueva de investigación en el análisis de datos que trata de resolver esta sobrecarga de información al combinar técnicas de minería de datos, procesamiento del lenguaje natural, recuperación de información y gestión del conocimiento. Dicho término es sinónimo a minería de textos, que es con el que se trabajará en este documento, debido a que se utiliza con mayor frecuencia.

La minería de textos está conformada por un compendio de técnicas de diversos campos, entre ellos la lingüística, la estadística y el *machine learning*. Estas técnicas modelan y estructuran el texto de entrada, que puede ser cualquier agrupación de documentos basados en texto. Cabe destacar que estas colecciones suelen oscilar entre miles y millones de documentos y pueden ser estáticas o dinámicas, dependiendo de si el número inicial de documentos permanece invariable o si, por el contrario, se aplica a colecciones que se caracterizan por añadir documentos nuevos o actualizados con el paso del tiempo.

Es importante tener en cuenta que estas técnicas de minería de textos no se aplican directamente en las colecciones de documentos, sino que éstos necesitan una preparación. Estas operaciones previas reciben el nombre de preprocesado, el cual incluye una gran variedad de pasos, dependiendo del objetivo al que se pretenda llegar con los documentos.

El análisis de los datos obtenidos, después del preprocesado, extrae información de los datos en formato de texto de manera automatizada. Este es su objetivo principal, que normalmente se utiliza para la toma de decisiones en el ámbito empresarial.

De entre las múltiples aplicaciones de la minería de textos, destacan las siguientes:

- Extracción de información y enriquecimiento de contenidos: Aunque el trabajo con el contenido del texto todavía requiere un esfuerzo humano, las técnicas de minería de textos son capaces de gestionar de forma cada vez más eficaz grandes volúmenes de información. Con ello, se revela información semántica significativa, que puede ser utilizada como metadatos o para resumir el contenido disponible.
- Gestión del conocimiento: Se busca encontrar información importante lo más rápidamente posible con grandes volúmenes de datos.
- Inteligencia de negocios: Este proceso es utilizado comúnmente por las empresas para llevar a cabo la toma de decisiones, trabajando con miles de fuentes y analizando grandes volúmenes de datos para extraer de ellos sólo el contenido relevante.

- Minería de opiniones o análisis de sentimientos: Consiste en extraer y analizar opiniones, emociones y sentimientos que generan principalmente los usuarios de redes sociales en relación a diversas marcas y productos, ya que ayuda a revelar información sobre un tema específico apoyando a campos como la inteligencia de negocios y jugando un papel muy importante en la toma de decisiones. Las empresas lo utilizan principalmente para predecir las necesidades de los clientes y comprender la percepción de su marca.
- Atención al cliente: Se utiliza para proporcionar una respuesta rápida y automatizada al cliente, reduciendo la dependencia de los operadores, tanto de los centros de llamadas para resolver problemas, como en foros, correos o encuestas de opinión.
- Clasificación de documentos: La minería de textos puede implementarse para mejorar la eficacia de los métodos de filtrado estadístico, como por ejemplo el filtrado de correo no deseado. Ésto, facilita la recuperación y navegación de documentos.

A continuación, se procederá a explicar, pormenorizadamente, los pasos necesarios principales para obtener un buen resultado a partir de técnicas de minería de textos, utilizando para ejemplificar el proceso la clasificación de documentos.

El conjunto de datos con el que se va a trabajar es una colección de 19997 mensajes recopilados de 20 directorios, cada uno conteniendo los documentos de texto pertenecientes a un tema determinado. Los temas de los directorios son muy diversos, como por ejemplo, ateísmo, hockey o el espacio. Los datos pueden obtenerse en el siguiente enlace:

<http://www.cs.cmu.edu/afs/cs/project/theo-20/www/data/news20.html>

Para llevar a cabo esta tarea, se partirá de un preprocesado de los datos, utilizando un subconjunto de 7500 documentos con el fin de agilizar el tiempo de computación. Seguidamente se hará un análisis exploratorio de éstos para obtener información descriptiva de cada grupo y con ella, se procederá a elaborar un clasificador. Lo que se busca es que éste, al proporcionarle un nuevo documento, sea capaz de decir a qué carpeta de entre las 20 es más probable que pertenezca el texto. Este clasificador se mostrará a través de una aplicación *Shiny*, que es una herramienta para crear aplicaciones web interactivas programada en lenguaje R.

Todo el código de R utilizado en este trabajo, tanto el del clasificador, como el de la aplicación *Shiny*, se puede consultar en el siguiente enlace de *Dropbox*:

<https://www.dropbox.com/sh/5i3uccz6nvolyhy/AACDEqzBFZZIQDq7U100-NXva?dl=0>

En este capítulo introductorio se ha utilizado principalmente la referencia [21] de la bibliografía.

Capítulo 2

Manipulación de textos

2.1. Obtención de los datos y creación del corpus

En el campo de la minería de textos normalmente se comienza con un conjunto de documentos de entrada altamente heterogéneos, por lo tanto el primer paso será importarlos a un entorno informático. Para ello, se utilizará el programa R [20] a través, principalmente, del paquete `tm` [7], ya que es el estándar actual para el análisis estadístico de texto en R. Este paquete facilita el uso de formatos de texto heterogéneos y cuenta con soporte de base de datos integrado para minimizar las demandas de memoria.

Para poder trabajar con estos documentos, se necesita de una entidad conceptual similar a una base de datos que contenga y gestione documentos de texto de una manera genérica. Esta entidad es el denominado corpus, el cual es una colección de documentos de texto. Existen diversos tipos de corpus en `tm`. Los dos más importantes son el *Volatile Corpus* y el *Permanent Corpus*. Se utilizará uno u otro dependiendo de si se desea almacenar los datos en la memoria y, que cuando el objeto R sea destruido se elimine (*Volatile Corpus*), o de si se quiere que los documentos se almacenen físicamente fuera de R, por ejemplo en una base de datos, y que no se destruya en caso de eliminarse el objeto de R (*Permanent Corpus*).

Dado que los documentos de texto pueden estar presentes en diferentes formatos de archivo, el programa R tiene varios lectores de textos instalados para archivos `.DOC`, `.PDF`, `.XML`, etc. En el ejemplo del clasificador, ya se dispone de los datos en archivos `.txt`, por lo que únicamente se necesita generar un corpus con estos datos. A continuación, se ve un ejemplo de como se podría hacer.

```
filePath<- "news20/20_newsgroup/alt.atheism/49960"
texto <- suppressWarnings(readLines(filePath))
corpus <- VCorpus(VectorSource(texto))
```

Para representar documentos en un corpus hay diversas formas de tomar los datos. Los tipos más utilizados son los presentados a continuación:

- Caracteres: Las letras, los números, los caracteres especiales y los espacios individuales. La representación a nivel de carácter incluye el conjunto completo de todos los caracteres de un documento. Este tipo de representación es difícil de manejar para algunos tipos de técnicas de procesamiento de texto debido a que el espacio de características para un documento no está suficientemente optimizado. A pesar de eso, este espacio de características está considerado como el más completo de cualquier representación de un documento de texto real.

- **Palabras:** Seleccionar las palabras directamente de un documento es el nivel básico de riqueza semántica. Es posible que una representación a nivel de palabra de un documento incluya una característica para cada palabra dentro de ese documento. Esto puede llevar a algunas representaciones de nivel de palabra de colecciones de documentos de decenas o incluso cientos de miles de palabras únicas en su espacio de características. Para minimizar el número de palabras, la mayoría de las representaciones de documentos a nivel de palabra llevan a cabo algún tipo de optimización; por lo tanto, el espacio consiste en un subconjunto de características representativas filtradas.
- **Conceptos:** Esta representación tiene en cuenta la sinonimia y la polisemia. Las representaciones basadas en conceptos pueden ser procesadas para soportar jerarquías de conceptos muy sofisticadas, y posiblemente proporcionen las mejores representaciones. Aún así, tienen la gran desventaja de que su implementación es muy compleja y de que depende del dominio de múltiples conceptos.

Pueden existir enfoques híbridos de los tres tipos anteriores. Sin embargo, éstos necesitan una planificación, pruebas y optimización cuidadosas para evitar un crecimiento desmesurado en la dimensionalidad de las características de las representaciones de documentos.

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

2.2. Preprocesamiento

Dado que la representación del total de documentos puede tener una alta dimensión, una vez ya se ha construido el corpus el siguiente paso es el procesado de éste, es decir, la aplicación de métodos para limpiar y estructurar el texto de entrada e identificar un subconjunto simplificado de las características del documento que puedan representarlo en un análisis posterior. A continuación, se presenta una lista de las transformaciones más habituales, utilizando la función `tm_map` del paquete `tm`:

- Conversión de las mayúsculas en minúsculas:

```
tm_map(corpus, content_transformer(tolower))
```

Normalmente se convierte en minúsculas todas las letras para que los comienzos de oración no sean tratados de manera diferente por los algoritmos.

- Eliminación de *stopwords*:

```
tm_map(corpus, removeWords, stopwords("english"))
```

Las *stopwords*, o palabras *stop*, son las más utilizadas en la lengua con la que se está trabajando y, por lo tanto, no aportan información relevante en los textos. Los artículos, las preposiciones, las conjunciones, los verbos y adjetivos más comunes, los pronombres y los adverbios son palabras *stop*. En cada idioma varían, por lo que hay que especificar el idioma en el que están los textos.

Para la clasificación de documentos, la inclusión de estas palabras no debería ser muy útil, ya que se espera que tengan una distribución uniforme entre los diferentes documentos. La eliminación de palabras *stop* se utiliza principalmente para aumentar el rendimiento computacional, no tanto para mejorar la estimación.

- Eliminación de signos de puntuación:

```
tm_map(corpus, removePunctuation)
```

Se eliminan los símbolos: ! " # \$ % & ' () * + , - . / : ; < = > ? @ [\] ^ _ ` { | } ~

Esta función simplemente elimina la puntuación sin insertar espacios en blanco. Para insertar un espacio en blanco en su lugar se haría de la siguiente forma:

```
tm_map(corpus, content_transformer(gsub), pattern = '[-]+', replacement= ' ')
```

- Eliminación de números:

```
tm_map(corpus, removeNumbers)
```

- Eliminación de espacios en blanco innecesarios:

```
tm_map(corpus, stripWhitespace)
```

- Agrupación de sinónimos:

Con el fin de disminuir la dimensión del espacio a trabajar, se pueden identificar palabras distintas con el mismo significado y reemplazarlas por una sola palabra. Para ello se toman los sinónimos de dicha palabra. Es común llevar a cabo esta agrupación de sinónimos en R utilizando la librería `wordnet` [8].

WordNet (Fellbaum 1998) [10] es una gran base de datos léxica de más de 100.000 términos en inglés donde los sustantivos, verbos, adjetivos y adverbios se agrupan en conjuntos de sinónimos llamados *synsets*, cada uno de ellos expresando un concepto distinto.

```
Sys.setenv(WNHOME = "C:/Program Files (x86)/WordNet/2.1")
library(wordnet)
setDict("C:/Program Files (x86)/WordNet/2.1/dict")
synonyms('write', "VERB")
```

```
## [1] "compose"      "drop a line" "indite"      "pen"         "publish"
## [6] "spell"        "write"
```

- Simplificación:

Otro posible enfoque para disminuir las dimensión es la agrupación de términos de un mismo tema, por ejemplo, que cada vez que aparezca en el texto una marca de un coche, la palabra se sustituya por *carbrand*. El siguiente código muestra cómo se podrían cambiar los números por la etiqueta *labelnum*.

```
corpus <- tm_map(corpus, content_transformer(gsub), pattern = '[0-9]+',
replacement= 'labelnum')
```

Esta simplificación solamente es útil cuando existe un extenso número de palabras de un mismo tema en los textos.

- Lematización:

```
tm_map(corpus, stemDocument)
```

La lematización o stemming es la transformación en lemas de todos los términos que componen el corpus, entendiendo por lema la forma representante de todas las formas flexionadas de una misma palabra.

Es una técnica que reduce la complejidad sin pérdida de información severa. Uno de los algoritmos más conocidos es el propuesto por Martin F. Porter en [19] que elimina las terminaciones morfológicas e inflexiones comunes de las palabras. Esto es necesario, ya que así, palabras con el mismo lema se tienen en cuenta como si fuesen una única a la hora de contar la frecuencia de cada término; por ejemplo, las palabras *value*, *values* y *valuíng* comparten el lema, *valu*, por lo que para calcular la frecuencia de éste se suman las apariciones de estas tres palabras.

- Descartar las palabras debido a su frecuencia de aparición, por ejemplo, las palabras que salgan en todos los documentos o las que solo aparezcan en uno.

La función `tm_map` puede realizar varios preprocesados simultáneamente aunque se hayan explicado por separado.

En el corpus que se está tomando como ejemplo se llevó a cabo este mismo preprocesado, exceptuando la agrupación de sinónimos debido a que ésto supondría trabajar con enormes bases de datos, y la simplificación, ya que los textos no albergaban ningún tema claro con el que poder hacerlo. El último punto se explicará más a fondo en la sección 4.2.

A continuación se mostrará un trozo del primer texto de la carpeta *alt.atheism*, al que se le irán aplicando las distintas fases del preprocesado, para comprender mejor cómo es el texto que se obtiene finalmente.

2.3. Ejemplo

El texto de partida pertenece a la primera parte del documento 49960 de la carpeta *alt.atheism*:

FREEDOM FROM RELIGION FOUNDATION

Darwin fish bumper stickers and assorted other atheist paraphernalia are available from the Freedom From Religion Foundation in the US.

Write to: FFRF, P.O. Box 750, Madison, WI 53701.
Telephone: (608) 256-8900

Lo primero que se debe hacer es guardarlo en un corpus:

```
texto<-"FREEDOM FROM RELIGION FOUNDATION

Darwin fish bumper stickers and assorted other atheist paraphernalia are
available from the Freedom From Religion Foundation in the US.

Write to:  FFRF, P.O. Box 750, Madison, WI 53701.
Telephone: (608) 256-8900"

corpus <- VCorpus(VectorSource(texto))
```

Antes de empezar a trabajar con los datos se debe inspeccionar que se han cargado correctamente y que no ha habido ningún error, para ello se utiliza la función `inspect()`:

```
inspect(corpus)

## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 1
##
## [[1]]
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 246
```

El primer paso del preprocesado es la transformación en minúsculas:

```
corpus <- tm_map(corpus, content_transformer(tolower))

corpus[[1]]$content

## [1] "freedom from religion foundation\n\ndarwin fish bumper stickers and assorted
other atheist paraphernalia are\navailable from the freedom from religion foundation
in the us.\n\nwrite to: ffrf, p.o. box 750, madison, wi 53701.\nntelephone: (608)
256-8900"
```

El siguiente paso es la eliminación o sustitución de caracteres especiales y la supresión de las palabras *stop*. En el caso del ejemplo se utilizan las SMART, que es una extensa lista de palabras *stop* en inglés:

```
# Tratamiento para caracteres especiales:
corpus <- tm_map(corpus, content_transformer(gsub), pattern = '[-]+' ,
replacement= ' ')

corpus <- tm_map(corpus, removeWords, stopwords("SMART"))

corpus[[1]]$content
```

```
## [1] "freedom religion foundation\n\ndarwin fish bumper stickers assorted
atheist paraphernalia \n freedom religion foundation .\n\nwrite : ffrf,
.. box 750,
madison, wi 53701.\ntelephone: (608) 256 8900"
```

A continuación se elimina la puntuación, los números y una lista adicional de palabras *stop*:

```
# Eliminación puntuación:
corpus <- tm_map(corpus, removePunctuation)

# Eliminación números:
corpus <- tm_map(corpus, removeNumbers)

# Eliminación de palabras de la lista "mywords":
mywords <- readLines("./words/mywords.txt")
corpus <- tm_map(corpus, removeWords, mywords)

corpus[[1]]$content
```

```
## [1] "freedom religion foundation\n\ndarwin fish bumper stickers assorted
atheist paraphernalia \n freedom religion foundation \n\nwrite ffrf box
madison wi \ntelephone "
```

Seguidamente se suprimen los espacios en blanco innecesarios:

```
corpus <- tm_map(corpus, stripWhitespace)

corpus[[1]]$content
```

```
## [1] "freedom religion foundation darwin fish bumper stickers assorted atheist
paraphernalia freedom religion foundation write ffrf box madison wi telephone "
```

Finalmente, se lleva a cabo la lematización del texto y se obtiene el texto con el preprocesado completo:

```
corpus <- tm_map(corpus, stemDocument)

corpus[[1]]$content
```

```
## [1] "freedom religion foundat darwin fish bumper sticker assort atheist
paraphernalia freedom religion foundat write ffrf box madison wi telephon"
```

En este capítulo se ha trabajado principalmente con las referencias [6], [9], [16] de la bibliografía y, además, con el paquete `tm` [7] de R.

Capítulo 3

Análisis exploratorio de textos

3.1. Resumen numérico

3.1.1. La matriz de documentos-términos

A partir de que ya se ha depurado el corpus, un enfoque común de la minería de textos es la creación de la matriz de documentos-términos, donde las filas corresponden a los documentos y las columnas a los términos. Esto se debe a que muchas de las aplicaciones toman matrices de documentos-términos como entrada. En dicha matriz, las celdas contienen un recuento del número de apariciones de un término específico en un texto dado. Los términos con mayor número de ocurrencias en un texto son catalogados como importantes.

Para la creación de esta matriz en R se utiliza el comando `DocumentTermMatrix()`:

```
dtm <- DocumentTermMatrix(corpus)
```

Para generar la matriz de documentos-términos se utilizan dos nuevas funciones, `process_texts` y `X_y_dataset`. La primera transforma un vector de textos en una estructura tipo `VCorpus` y realiza el preprocesado visto en el capítulo anterior. La segunda, genera y guarda la matriz documentos-términos “X” a partir de `corpus` y el vector “y” de etiquetas numéricas para asignar grupo a cada documento de “X” a partir de las etiquetas (`labels`).

Debido a que la matriz de documentos-términos del clasificador de textos es muy grande y dispersa, para comprender mejor este capítulo se obtendrá la matriz de documentos-términos de un nuevo ejemplo.

```
textos <- c("The dog is man's best friend and dogs love people.",
           "My friend has a white cat.",
           "Cats and dogs are not friends.",
           "I live with a dog and a white cat.")

corpus <- process_texts(textos, stem= TRUE)

dtm_model <- X_y_dataset(corpus,
                        labels= numeric(),
                        tfidf= FALSE,
                        with.normalize= FALSE,
```

```

word.length= c(2,15),
freq.bounds= c(0, Inf),
with.sparse= FALSE,
sparse= 0.95)

dtm<- dtm_model$X; dtm

```

```

##      Terms
## Docs cat dog friend live love peopl white
##   1  0  2     1   0   1     1     0
##   2  1  0     1   0   0     0     1
##   3  1  1     1   0   0     0     0
##   4  1  1     0   1   0     0     1

```

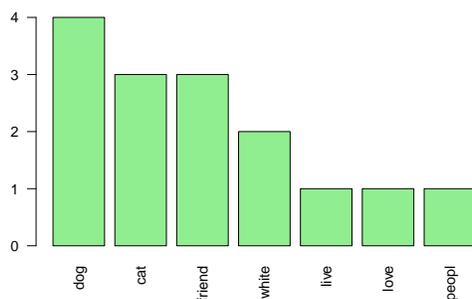


Figura 3.1: Diagrama de barras de la dtm del ejemplo.

3.1.2. Importancia de los términos

Para estudiar la importancia de los términos de un documento en particular, en lugar de utilizar la frecuencia de cada uno de los términos directamente, se pueden utilizar diferentes ponderaciones denominadas TF-IDF (*Term Frequency-Inverse Document Frequency*) que se verán a continuación. Una de las grandes ventajas es que estas ponderaciones se puede aplicar a cualquier idioma del mundo.

Estas ponderaciones se calculan como el producto de dos medidas, la frecuencia de aparición del término (tf) y la frecuencia inversa del documento (idf).

La fórmula matemática para esta métrica es la siguiente:

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D) \quad (3.1)$$

donde t es el término, d denota cada documento, D el espacio total de documentos y tfidf es el peso asignado a ese término en el documento correspondiente [14].

La combinación de los valores de tf e idf da una métrica que permite saber cómo de únicas son las palabras de un documento. La ponderación asigna un alto peso a un término si se produce con

frecuencia en ese documento, pero rara vez en la colección completa. Sin embargo, si el término ocurre pocas veces en el documento, o aparece prácticamente en todos ellos, disminuye el peso asignado por la ponderación tfidf.

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

Frecuencia del término

$tf(t, d)$ se define como frecuencia del término o *term frequency*. Existen distintas formas de medir esta frecuencia, entre las que destacan:

- Recuento: cálculo del número de veces que el término t aparece en el documento d . Se denota como:

$$tf(t, d) = n_{t,d} \quad (3.2)$$

- Forma binaria (o booleana):

$$\begin{cases} tf(t, d) = 1 & \text{si } t \text{ aparece en el documento } d, \\ tf(t, d) = 0 & \text{en otro caso.} \end{cases} \quad (3.3)$$

- Frecuencia de término normalizada:

$$tf(t, d) = \frac{n_{t,d}}{\sqrt{\sum_{t' \in d} n_{t',d}^2}} \quad (3.4)$$

Algunos autores definen esta frecuencia como $tf(t, d) = \frac{n_{t,d}}{\sum_{t' \in d} n_{t',d}}$.

Como cada documento tiene diferente longitud, es más probable que un término aparezca un número mayor de veces en documentos largos que cortos. Por este motivo la frecuencia de término se divide por una medida del número total de términos del documento.

- Frecuencia logarítmicamente escalada:

$$\begin{cases} tf(t, d) = 1 + \log(n_{t,d}) & \text{si } n_{t,d} > 0, \\ tf(t, d) = 0 & \text{en otro caso.} \end{cases}$$

Frecuencia inversa del documento

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

Los idf más utilizados son los que se presentan a continuación:

- Unitario:

$$\text{idf}(t, D) = 1 \quad (3.5)$$

En este caso no se llevaría a cabo ninguna corrección.

- Frecuencia inversa del documento:

$$\text{idf}(t, D) = \log \left(\frac{N}{\text{df}(t)} \right) \quad (3.6)$$

siendo:

- $N = |D|$, número de documentos en el corpus.
- Frecuencia de documentos ($\text{df}(t)$): número de documentos en la colección que contienen el término t . Hay que tener en cuenta que no importa cuantas veces aparece un término en el documento, tanto si es una o varias veces se contará como 1.

$$\text{df}(t) = |\{d \in D : t \in d\}|$$

Si el término no está en el corpus, esto conducirá a una división por cero. Por lo tanto, es común ajustar el denominador a $1 + |\{d \in D : t \in d\}|$.

El término (3.6) es una interpretación estadística de la especificidad, un término ideado por Karen Spärck Jones [24]. Ésta es una medida de cuánta información proporciona una palabra, es decir, si el término es común en todos los documentos el idf es probable que sea bajo, y si es raro será alto.

Los siguientes tipos de idf son modificaciones de éste.

- Frecuencia inversa del documento con suavizado:

$$\text{idf}(t, D) = \log \left(1 + \frac{N}{\text{df}(t)} \right)$$

Tomando $\frac{N}{\text{df}(t)} = 0$ si $\text{df}(t) = 0$.

- Frecuencia inversa del documento utilizando el máximo:

$$\text{idf}(t, D) = \log \frac{\max_{t'} \text{df}_{t'}}{1 + \text{df}(t)}$$

Para trabajar con textos, aparte de la matriz de documentos-términos vista en la sección 3.1.1, las ponderaciones más utilizadas son:

- Binaria:

`weightBin(dtm)`

Se define como el producto de la tf binaria (3.3) y la idf unitaria (3.5).

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D) = \begin{cases} 1 \times 1 = 1 & \text{si } t \text{ aparece en el documento } d, \\ 0 \times 1 = 0 & \text{en otro caso.} \end{cases}$$

- tfidf estándar:

```
weightTfIdf(tdm, normalize = TRUE) o weightTfIdf(tdm, normalize = FALSE)
```

Se define como el producto de tf, el recuento (3.2), o tf normalizada (3.4), con idf (3.6).

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D) = \begin{cases} n_{t,d} \times \log\left(\frac{N}{\text{df}(t)}\right) & \text{sin normalizar,} \\ \frac{n_{t,d}}{\sqrt{\sum_{t' \in d} n_{t',d}^2}} \times \log\left(\frac{N}{\text{df}(t)}\right) & \text{normalizada.} \end{cases}$$

Después de probar varias tfidf posibles, la de la frecuencia de término normalizada e idf suavizada es con la que se han obtenido mejores resultados para el clasificador de textos. El código utilizado se muestra a continuación:

```
# Vector de pesos idf:
idf <- log(nrow(dt)/(colSums(dt!=0))+1)

# Matriz tf-idf:
dtidf <- t(t(dt)*idf)

# Normalización:
if(with.normalize) {
  dtidf <- dtidf/sqrt(rowSums(dtidf^2))

  # En los documentos vacíos aparecen NaN al dividir por 0,
  # por lo que se asigna 0 a los NaN:
  dtidf[is.nan(dtidf)] <- 0
}
```

Además de éstas, se pueden utilizar ponderaciones personalizadas dependiendo de lo que se desee. En [17] se pueden ver más ponderaciones.

Ejemplo:

A continuación, se explicará paso a paso la obtención de esta tfidf a partir del ejemplo de la subsección 3.1.1. Primero se calcula la tfidf sin normalizar.

```
tfidf_model <- X_y_dataset(corpus,
  labels= numeric(),
  tfidf= TRUE,
  with.normalize= FALSE,
  word.length= c(2,15),
  freq.bounds= c(0, Inf),
  with.sparse= FALSE,
  sparse= 0.95)
```

```
idf <- tfidf_model$idf; idf
```

```
##      cat      dog  friend    live    love  peopl  white
## 0.8472979 0.8472979 0.8472979 1.6094379 1.6094379 1.6094379 1.0986123
```

```
tfidf <- tfidf_model$X; tfidf
```

```
##      Terms
## Docs      cat      dog  friend    live    love  peopl  white
## 1 0.0000000 1.6945957 0.8472979 0.0000000 1.609438 1.609438 0.000000
## 2 0.8472979 0.0000000 0.8472979 0.0000000 0.000000 0.000000 1.098612
## 3 0.8472979 0.8472979 0.8472979 0.0000000 0.000000 0.000000 0.000000
## 4 0.8472979 0.8472979 0.0000000 1.609438 0.000000 0.000000 1.098612
```

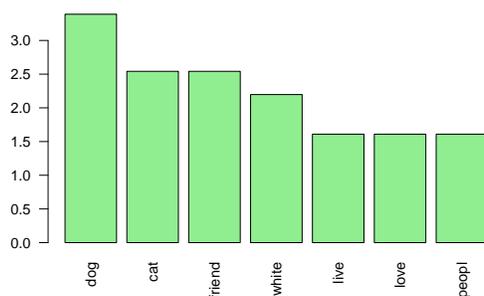


Figura 3.2: Diagrama de barras de la tfidf del ejemplo.

Se pretende obtener la columna de tfidf correspondiente al término *cat*, por ejemplo. La columna de la matriz tf para *cat* es en este caso la columna de la dtm de la subsección 3.1.1, es decir, $(0, 1, 1, 1)^t$. Para construir idf se necesitan:

- N , número de documentos en el corpus. En este caso es 4.
- $df(t)$, frecuencia de documentos. El término aparece en 3 documentos.

$$\begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} \cdot \left(\frac{4}{3} + 1 \right) = \begin{pmatrix} 0 \\ 0,8472979 \\ 0,8472979 \\ 0,8472979 \end{pmatrix}$$

Se procedería de forma similar para cada uno de los términos.

Por último, se lleva a cabo la normalización.

```
tfidfnorm_model <- X_y_dataset(corpus,
                              labels= numeric(),
                              tfidf= TRUE,
                              with.normalize= TRUE,
                              word.length= c(2,15),
                              freq.bounds= c(0, Inf),
                              with.sparse= FALSE,
                              sparse= 0.95)

tfidfnorm <- tfidfnorm_model$X; tfidfnorm

##      Terms
## Docs      cat      dog  friend    live    love    peopl    white
##   1 0.000000 0.5722195 0.2861097 0.0000000 0.5434639 0.5434639 0.0000000
##   2 0.5212018 0.0000000 0.5212018 0.0000000 0.0000000 0.0000000 0.6757939
##   3 0.5773503 0.5773503 0.5773503 0.0000000 0.0000000 0.0000000 0.0000000
##   4 0.3703889 0.3703889 0.0000000 0.7035518 0.0000000 0.0000000 0.4802488
```

Para obtener la segunda fila de *cat* se procedería de la siguiente forma:

$$\frac{0,8472979}{\sqrt{0,8472979^2 + 0,8472979^2 + 1,098612^2}} = 0,5212018$$

El resto de valores se consiguen análogamente.

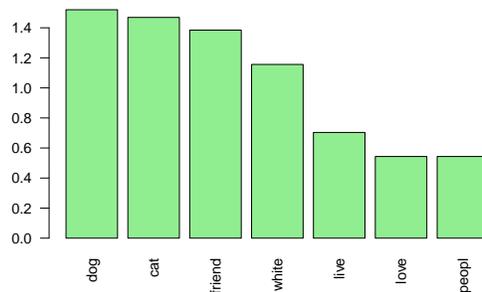


Figura 3.3: Diagrama de barras de la tfidf normalizada del ejemplo.

Cada valor del diagrama de barras es la suma de todos los valores de la columna de cada término en la tabla.

3.2. Métodos gráficos: el diagrama de barras y la nube de palabras

Para poder continuar trabajando con el ejemplo del clasificador, se necesita tener una idea clara de cómo son los textos con los que se trabaja. Para ello, se puede mostrar un diagrama de barras como en el ejemplo anterior o una nube de palabras.

Capítulo 4

Métodos de inferencia

4.1. Comparación de documentos y términos

La función básica de la minería de textos es analizar la existencia de asociaciones entre documentos y términos de una colección. Hay múltiples formas de calcular estas relaciones.

4.1.1. Distancia euclídea

Utilizando la tfidf vista en el capítulo anterior, se pueden crear representaciones vectoriales de documentos. Para ello, cada componente de un vector se corresponde al valor de un término en particular; es decir, cada dimensión del espacio es un término del documento. La componente de un término que no se produce se toma como 0.

Al visualizar esto en un espacio de documento-término, los documentos son puntos y se puede estudiar lo similares que son calculando la distancia euclídea entre dos de ellos.

La distancia euclídea se puede calcular como sigue:

$$dist(A, B) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \dots}$$

donde $A = (a_1, a_2, a_3, \dots)$ y $B = (b_1, b_2, b_3, \dots)$ son los dos documentos y a_i y b_i , las veces que aparece el i -ésimo término en esos documentos, donde i es un número natural que varía entre 0 y el número de términos totales.

Dado que los documentos no suelen tener la misma longitud, el simple cálculo de la diferencia entre dos vectores tiene la desventaja de que los documentos de contenido parecido pero de longitud diferente no se consideran similares en el espacio vectorial. Así mismo, los documentos con muchos términos están muy lejos del origen, por lo tanto, se pueden encontrar documentos pequeños relativamente similares aún cuando no estén relacionados, debido a la corta distancia entre ellos.

4.1.2. Distancia del coseno

Para evitar los problemas ya vistos con documentos de diferentes longitudes, se utiliza la distancia del coseno. Para ello se mide el coseno del ángulo que forman los vectores del origen al punto. Cuanto mayor es ese valor, más similares son los documentos que se están comparando.

En el caso de documentos que no tienen ningún término en común, su valor de similitud es cero, ya que los vectores son ortogonales. Cuantos más términos tengan en común mayor será su similitud.

hasta alcanzar el valor máximo de uno, que ocurre en el caso de comparar dos documentos iguales.

La fórmula para calcular el coseno del ángulo que forman los vectores de extremo los dos puntos y origen el origen de coordenadas es la siguiente:

$$\cos \theta = \cos(A, B) = \frac{\langle A, B \rangle}{\|A\| \|B\|}$$

siendo $\langle A, B \rangle$ el producto interior y $\|A\|$ y $\|B\|$ la normas de los vectores que unen el origen con los puntos A y B .

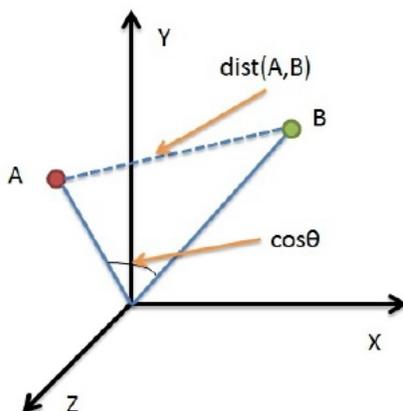


Figura 4.1: Distancias euclídea y del coseno.

Para calcular la distancia entre dos documentos en el ejemplo del clasificador se utilizó la función `distancia`, que utiliza la similitud del coseno. El código necesario para este cálculo se muestra a continuación.

```
# Matriz de documentos-términos:
dtm_DT <- DocumentTermMatrix(corpus_DT)
tfidf_DT <- suppressWarnings(weightTfIdf(dtm_DT))

# Vector de los términos para pasar como nombres de vectores doc1 y doc2:
terms_DT <- tfidf_DT$dimnames$Terms

id1 <- 1
id2 <- 2

# Extracción de los vectores-documentos para los id1 e id2 seleccionados:
doc1 <- as.vector(tfidf_DT[ , id1])
names(doc1) <- terms_DT
doc2 <- as.vector(tfidf_DT[ , id2])
names(doc2) <- terms_DT
```

```
# Comparación de dos documentos:
comparaDocs <- distancia(doc1, doc2)
cat("\nLa similitud entre id1 e id2 es: ", comparaDocs, "\n")
```

```
## La similitud entre id1 e id2 es: 0.3278182
```

Análogamente, se puede obtener la distancia entre dos términos. Para ello se utiliza el término lematizado, como por ejemplo “peopl” y “time”.

```
# Comparación de dos términos:
term1 <- "peopl"
term2 <- "time"

# Extracción de los vectores-documentos para los term1 y term2 seleccionados:
term1vec <- as.vector(tfidf_DT[term1, ])
term2vec <- as.vector(tfidf_DT[term2, ])

comparaTerms <- distancia(term1vec, term2vec)
cat("\nLa similitud entre ", term1, "y", term2, "es: ", comparaTerms, "\n")
```

```
## La similitud entre peopl y time es: 0.1372286
```

4.1.3. Correlación

Otra forma de medir la distancia es calcular la correlación entre un término y todos los demás de la matriz. Para ello, se utiliza la función `findAssocs()`, que calcula todas las asociaciones para un término dado. El parámetro `corlimit` fija la correlación mínima que se desea encontrar.

```
corpus <- process_texts(textos, stem= TRUE)

dtm<-DocumentTermMatrix(corpus)
findAssocs (dtm, "dog", corlimit = 0.6)
```

```
## $dog
## love peopl
## 0.82 0.82
```

Este comando calcula las correlaciones entre todos los términos de la matriz `dtm` y se queda aquellos que superan el límite, en este caso, los que tienen una correlación superior a 0,6. Los lemas “love” y “peopl” son los que tienen una correlación mayor con “dog”.

Si dos palabras siempre aparecen juntas entonces la correlación sería 1 y si nunca lo hacen sería 0. Por lo tanto, ésta es una medida de cuánta asociación hay entre dos términos en el corpus.

Aunque éstas son las distancias que más se utilizan para la comparación de términos o documentos existen muchas otras. Para más información, se puede consultar el capítulo de “*Statistical Linguistics with R*” de [12].

4.2. Técnicas de reducción de dimensión

Las matrices de documentos-términos en grandes colecciones de documentos tienden a tener dimensiones enormes y suelen ser poco densas, ya que la mayoría de las combinaciones de documentos y términos son cero. A pesar de ello, miles de sus componentes son distintas de cero.

```
inspect(dtm)
```

```
## <<DocumentTermMatrix (documents: 7500, terms: 61834)>>
## Non-/sparse entries: 502584/463252416
## Sparsity           : 100%
## Maximal term length: 177
## Weighting          : term frequency (tf)
## Sample            :
##      Terms
## Docs  articl call file god peopl post system time window write
## 1890   1   10   8  0   3   13    1   7    0    0
## 2162   6    5  36  0   9   21   22   4    6    2
## 2662   3    5  14  0   4    4   26   7   25    2
## 3657   8   11  53  0   4    9   26   9    7    4
## 4209   1   18  28  1   8    4   33  14    4    6
## 4457   0   11   2  5   1    0   52  11    0    0
## 5382   1    7  49  0   3    4   32   8   46    5
## 5880   8   11  53  0   4    9   25   9    7    4
## 6980   2    6  15  0   0    3   34   8   10    4
## 7482   1    1   6  0   0    1    0   0    2    3
```

En el ejemplo del clasificador esta matriz es de 7500×61834 . Es necesario, por lo tanto, que se puedan filtrar de alguna manera estas matrices para reducir su tamaño y el ruido de los datos sin perder información significativa, ya que se necesita un conjunto de datos manejables. Así cada documento se podrá representar utilizando menos dimensiones.

La mayoría de los términos son irrelevantes para la tarea y pueden eliminarse sin empeorar el rendimiento del clasificador e incluso algunas veces mejorándolo debido a la eliminación de ruido.

Con el fin de realizar el filtrado, se recurre a la eliminación de palabras *stop*, como ya se ha visto en el preprocesado. Seguidamente, se intentan suprimir muchos de los términos que no tengan significado, para ello se eliminan los términos con una longitud menor de 2 letras y mayor de 15, ya que se supone que los términos de más de 15 letras serán agrupaciones de caracteres sin sentido.

```
dtm <- DocumentTermMatrix(corpus, list(wordLengths= c(2,15)))
```

Seguidamente, se estudiarán otros posibles métodos de filtrado. Estas técnicas que filtran información o agrupan datos relacionados se conocen como post-procesado.

4.2.1. Frecuencias

Filtrando las palabras que no aportan información para la caracterización de los documentos se obtienen las palabras clave o *keywords*. Para identificar estos términos representativos se consideran únicamente las palabras de frecuencia media.

Esto se debe a que los términos de frecuencia alta aparecen en prácticamente todos los documentos, como es el caso de las palabras *stop*, por lo que la información que se obtiene de ellos es prácticamente nula. Con respecto a las palabras de frecuencia muy baja, suele haber muchas y la gran mayoría no aportan información, además, si no se eliminan aumenta considerablemente la dimensión del espacio en el que se trabaja.

Para buscar palabras en un intervalo de frecuencias se utiliza la función `findFreqTerms`.

```
findFreqTerms(dtm, lowfreq = 5, highfreq = Inf)
```

Si se desea filtrar la matriz de documentos-términos por frecuencias se tiene el comando `bounds`.

```
dtm <- DocumentTermMatrix(corpus, list(bounds= list(global= c(5,Inf))))
```

Después de estos filtrados la matriz del clasificador queda de la siguiente forma.

```
inspect(dtm)
```

```
## <<DocumentTermMatrix (documents: 7500, terms: 55505)>>
## Non-/sparse entries: 505764/415781736
## Sparsity           : 100%
## Maximal term length: 15
## Weighting          : term frequency (tf)
## Sample            :
##      Terms
## Docs  articl call file god peopl post system time window write
## 1890   1   10   8  0   3   13    1   7    0    0
## 2162   6    5  36  0   9   21   22   4    6    2
## 2662   3    5  14  0   4    4   26   7   25    2
## 3657   8   11  53  0   4    9   26   9    7    4
## 4209   1   18  28  1   8    4   33  14    4    6
## 4457   0   11   2  5   1    0   52  11    0    0
## 5382   1    7  49  0   3    4   32   8   46    5
## 5880   8   11  53  0   4    9   25   9    7    4
## 6980   2    6  15  0   0    3   34   8   10    4
## 7482   1    1   6  0   0    1    0   0    2    3
```

Como se puede ver, la matriz de documentos-términos ha disminuido el número de términos.

4.2.2. Sparsity

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

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

$$df(t) > N \cdot (1 - \text{sparse}),$$

siendo df la frecuencia de documentos del término t y N el número de vectores. El parámetro `sparse` toma valores entre 0 y 1.

```
dtm <- removeSparseTerms(dtm, sparse= 0.95)
```

Si el umbral de escasez es 0,95, se toman los términos que aparecen en más del 5 % de documentos.

```
inspect(dtm)
```

```
## <<DocumentTermMatrix (documents: 7500, terms: 7695)>>
## Non-/sparse entries: 423560/57288940
## Sparsity           : 99%
## Maximal term length: 15
## Weighting          : term frequency (tf)
## Sample            :
##      Terms
## Docs  articl call file god peopl post system time window write
## 1890    1   10   8  0    3   13    1   7    0    0
## 2162    6    5  36  0    9   21   22  4    6    2
## 2662    3    5  14  0    4    4   26  7   25    2
## 3657    8   11  53  0    4    9   26  9    7    4
## 4209    1   18  28  1    8    4   33 14    4    6
## 4457    0   11   2  5    1    0   52 11    0    0
## 5382    1    7  49  0    3    4   32  8   46    5
## 5880    8   11  53  0    4    9   25  9    7    4
## 6528    1    7  31  0    2    7   29  3   10    2
## 6980    2    6  15  0    0    3   34  8   10    4
```

En la dtm ya solo se tienen 7695 términos.

4.2.3. Asignación de Dirichlet latente (LDA)

Otra técnica utilizada para reducir la dimensión de la matriz de documentos-términos es la Asignación de Dirichlet Latente o, como se suele abreviar, LDA (*Latent Dirichlet Allocation*). Ésta asume que existe un número fijo de temas o categorías que se distribuyen sobre los documentos de toda la colección, supone que cada documento del corpus trata varios temas y a cada término le asigna una probabilidad de pertenecer a un tema en particular.

Para este modelo, tanto el orden de las palabras como el de los documentos no importa. Saber los términos que se utilizaron en cada documento y sus frecuencias ya aporta un resultado suficientemente bueno para tomar decisiones acerca de a qué tema pertenece cada uno de ellos.

En vez de trabajar con la matriz de documentos-términos, se cambia a una matriz de documentos-temas y con ello se reduce la dimensión. Sin embargo, este modelo presenta varios problemas:

- No es capaz de incluir relaciones entre distintos temas.
- No da buenos resultados si los temas son parecidos.
- Necesita muchos documentos y que estos tengan una gran longitud.

Una implementación del LDA se proporciona en el paquete `text2vec` [22]. Como el corpus del ejemplo del clasificador consta de 20 carpetas, se ha propuesto como 20 el número de temas a ser estimados.

```
library("tm")
library("text2vec")

load("../data/DT_sample.RData")
source("../functionScripts/process_texts.R")
```

```

# Preprocesado:
corpus <- process_texts(DT_sample$Content, stem= TRUE)

# Extracción de los textos preprocesados del corpus:
corpus_texts <- unlist(sapply(corpus, '[', "content"))
names(corpus_texts) <- NULL

# Tokenizado de los textos por palabra utilizando el paquete text2vec:
tokens <- corpus_texts%>% word_tokenizer

iterator <- itoken(tokens, ids= DT_sample$id, progressBar= FALSE)

# create_vocabulary: recopila términos
# prune_vocabulary: filtra términos

vocabulary = create_vocabulary(iterator)%>%
  prune_vocabulary(term_count_min = 10, doc_proportion_max = 0.2)

# vocab_vectorizer: crea un vectorizador de texto que se utiliza
# a continuación en la creación de dtm

vectorizer <- vocab_vectorizer(vocabulary)

dtm = create_dtm(iterator, vectorizer, type = "lda_c")

# LDA
if(TRUE) {
  lda_model <- LDA$new(n_topics= 20, vocabulary= vocabulary,
                      doc_topic_prior= 0.1, topic_word_prior= 0.01)

  # Matriz documento-tema
  doc_topic_distr <- lda_model$fit_transform(dtm, n_iter= 1000,
                                           convergence_tol= 0.01,
                                           check_convergence_every_n= 10)

  # Matriz palabra-tema
  word_topic_dist <- lda_model$get_word_vectors()

  library(LDAvis)
  library(servr)
  lda_model$plot() # gráfico
}

```

En la Figura 4.2 se puede apreciar la distribución de los 20 temas que ha elegido el LDA y, en la Figura 4.3, se muestra el grupo 3, donde se encuentran muchas de las palabras relacionadas con la religión.

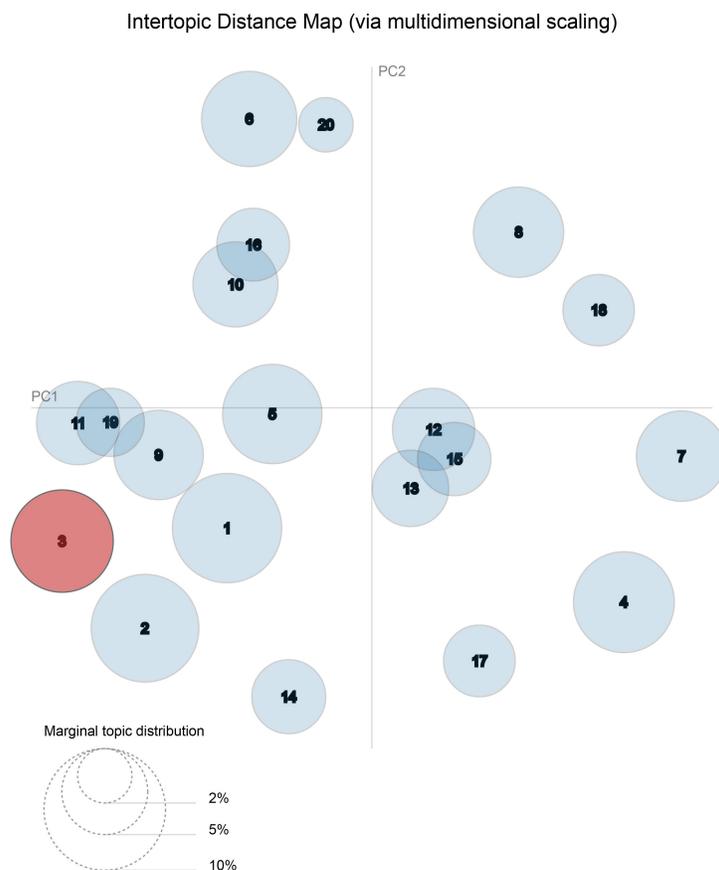


Figura 4.2: Gráfico en el que se muestra la asignación de temas del LDA.

Si se desea tener más información acerca del LDA se puede consultar la referencia [2] de la bibliografía.

Para el ejemplo del clasificador se han utilizado todos los métodos de filtrado explicados exceptuando el LDA, debido a que, para estos datos, no daba resultados suficientemente buenos.

4.3. Clasificación de un nuevo documento

En apartados anteriores se han visto los diversos pasos previos a la clasificación de un texto nuevo. En este apartado, se explicarán los tipos de clasificadores que hasta ahora han obtenido mejores resultados, prestando especial atención a los que se han utilizado para el ejemplo del clasificador.

Existen dos tipos de clasificación, la supervisada y la no supervisada. La primera es en la que la respuesta es conocida y en la segunda, también llamada *cluster*, no se tienen las posibles respuestas. Cada una de ellas tiene sus ventajas e inconvenientes, por lo que dependiendo de cuál sea el objetivo, se emplea una u otra.

La clasificación supervisada proporciona a los investigadores la oportunidad de especificar las categorías para el algoritmo de clasificación. El inconveniente es que estos clasificadores requieren mano

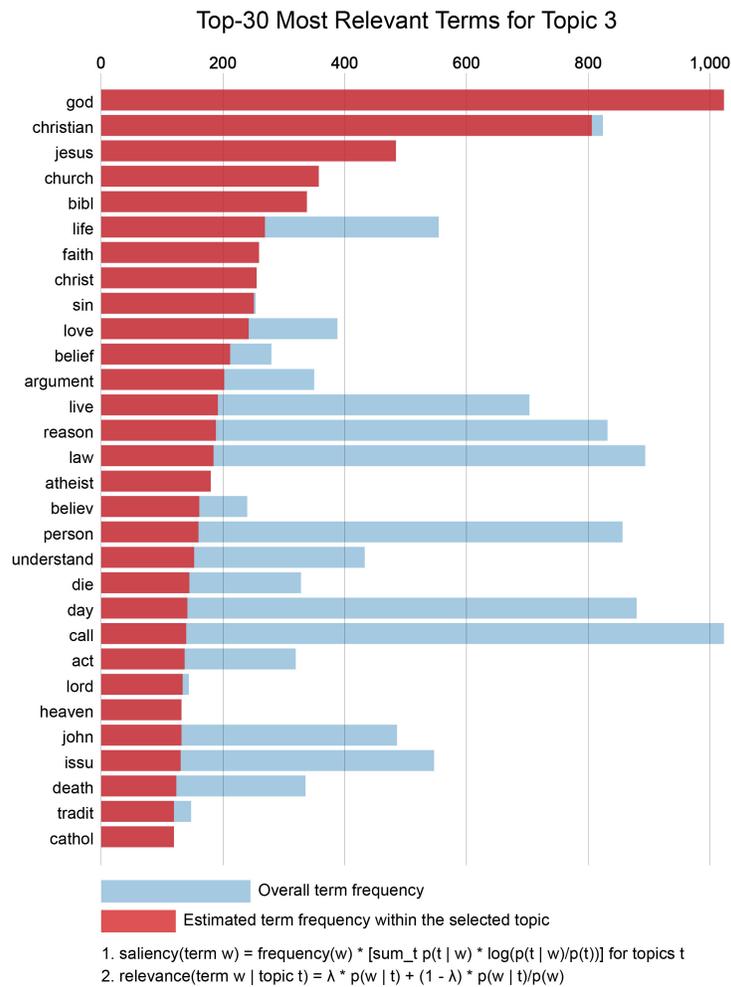


Figura 4.3: Gráfico en el que se muestran las palabras más frecuentes del tema 3.

de obra, debido a las grandes cantidades de datos de entrenamiento necesarias. El concepto de datos de entrenamiento se explicará en el próximo apartado.

Los métodos de clasificación no supervisada generan ellos mismos categorías. Esto es interesante para el estudio de las principales líneas de división en un corpus lingüístico. Además, no necesita una codificación manual de los datos. Dos de los contras principales de esta clasificación son que se requiere que los investigadores especifiquen el número de categorías en las que se va a agrupar el corpus o que establezcan un criterio que determine ese número, y, además, la interpretación a posteriori de los resultados de la estimación suele ser muy compleja. El LDA explicado anteriormente se utiliza como método de clasificación no supervisada en muchas ocasiones.

A continuación, se citarán los principales clasificadores supervisados ya que para el ejemplo se han utilizado estos solamente. Esto es debido a que se sabía que las categorías eran 20 y los textos ya habían sido asignados cada uno a su categoría correspondiente.

4.3.1. Particionamiento de los datos

Antes de la explicación de los métodos de clasificación supervisada es necesario presentar el particionamiento de los datos.

Este particionamiento es muy importante ya que permite estimar la precisión de las predicciones de un modelo antes de aplicarlo a un conjunto de datos no observado. Por este motivo, no se recomienda utilizar el total de datos como entrenamiento, porque de esa forma no se sabría cómo va a actuar el modelo en la práctica con datos nuevos.

Ésta es una técnica que ayuda a validar el modelo ajustado. Para ello, se toman los textos que ya están clasificados y se dividen en dos grupos, que serán denominados textos de entrenamiento (*training*) y textos de prueba (*test*). Con el primer grupo de textos se estiman los parámetros del modelo y se utiliza este modelo ajustado para predecir la clasificación de los textos del segundo grupo. Así, como los textos de prueba ya estaban clasificados, se puede estudiar si el modelo entrenado se equivoca en un alto porcentaje de textos o no. Si el modelo tiene pocos errores de predicción en los textos de entrenamiento, pero no predice bien en los de prueba, es un caso claro de *overfitting*. Esto significa que el clasificador no aprendió suficiente de los datos y que no es capaz de generalizar en los textos de prueba.

En el ejemplo se dividió el 80 % de los datos para entrenamiento y el 20 % restante para prueba.

4.3.2. Métodos de clasificación basados en árboles

Árbol de decisión

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

Generalmente se presenta el árbol boca abajo, es decir, con la raíz en la parte superior y las hojas, o nodos terminales, en la inferior. A partir de la raíz, el árbol se divide en ramas y éstas en más ramas hasta llegar a las hojas. Cada una de estas divisiones en ramas, la raíz y las hojas reciben el nombre de nodos. Asociada a cada una de las divisiones se tiene un criterio, que determina por qué rama seguir, hasta llegar a las hojas, que contienen las decisiones finales. Para ilustrar este clasificador se utilizarán los datos de `vino.csv`, donde aparecen distintos vinos y una serie de características de cada uno y se busca clasificar un nuevo vino como “bueno” o “malo”.

Como los posibles árboles para un conjunto de datos pueden llegar a ser muchos y no se podrían analizar todos, se utilizan criterios que en cada paso buscan la mejor opción, denominados criterios avariciosos, para tratar de encontrar el mejor árbol posible en un tiempo razonable. Para ello, en cada nodo se busca un punto de corte en una variable explicativa que haga cada una de las ramas lo más homogénea posible, es decir, para el ejemplo del vino, se trataría de encontrar una partición binaria que contuviese en la rama izquierda una mayor proporción de vino bueno, y en la derecha, una mayor proporción de vino malo. Para ello se suele utilizar el índice de Gini, que es una medida de la varianza total en los grupos. Este índice toma valores cercanos a 0 cuando hay mucha igualdad dentro de los grupos y aumenta hasta 100 según va creciendo la desigualdad.

El proceso se repite para las dos ramas por separado, y así sucesivamente hasta que se cumple un criterio de parada. El más frecuente es que se llegue a una rama que no tenga un número mínimo de observaciones.

Si el árbol es muy grande se suele “podar” para disminuir su complejidad.

Ejemplo:

```
vino<-read.csv(file="vino.csv",sep=";",header=T)
vino<-data.frame(vino)
head(vino)
```

```
##   fixed.acidity volatile.acidity citric.acid residual.sugar chlorides
## 1           7.0           0.27         0.36           20.7      0.045
## 2           6.3           0.30         0.34            1.6      0.049
## 3           8.1           0.28         0.40            6.9      0.050
## 4           7.2           0.23         0.32            8.5      0.058
## 5           7.2           0.23         0.32            8.5      0.058
## 6           8.1           0.28         0.40            6.9      0.050
##   free.sulfur.dioxide total.sulfur.dioxide density   pH sulphates alcohol  quality
## 1                   45                   170 1.0010 3.00     0.45     8.8      6
## 2                   14                   132 0.9940 3.30     0.49     9.5      6
## 3                   30                    97 0.9951 3.26     0.44    10.1      6
## 4                   47                   186 0.9956 3.19     0.40     9.9      6
## 5                   47                   186 0.9956 3.19     0.40     9.9      6
## 6                   30                    97 0.9951 3.26     0.44    10.1      6
```

```
barplot(table(vino$quality))
```

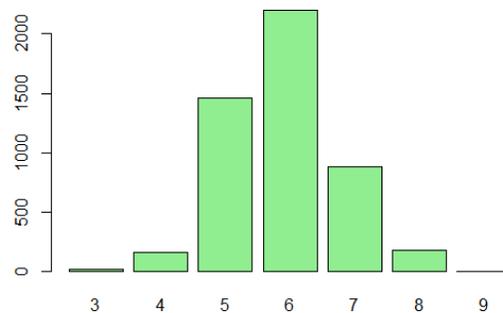


Figura 4.4: Diagrama de barras que muestra los diferentes vinos según su calidad.

Se crea la variable `sabor` que clasifica los vinos en “bueno” o “malo” y se elimina la variable `quality`.

```
vino$sabor <- ifelse(vino$quality < 6, 'malo', 'bueno')
vino$sabor <- as.factor(vino$sabor)
table(vino$sabor)
```

```
##
## bueno malo
## 3258 1640
```

```
vino <- vino[ ,!colnames(vino)=="quality"]
```

Para obtener los árboles de decisión es común utilizar la interfaz gráfica `rattle` [27] de R.

```
library(rattle)

library(magrittr) # Para los operadores %>% y %<>%.
building <- TRUE
scoring <- ! building

seed <- 42
dataset <- vino
```

A continuación, se construyen los conjuntos de datos de entrenamiento (80%) y prueba (20%).

```
set.seed(seed)
nobs <- nrow(dataset) # 4898 observaciones
sample <- train <- sample(nrow(dataset), 0.8*nobs) # 3918 observaciones
validate <- NULL
test <- setdiff(setdiff(seq_len(nrow(dataset)), train), validate) # 980 observaciones

# Se han anotado las siguientes selecciones de variable.

input <- c("fixed.acidity", "volatile.acidity", "citric.acid", "residual.sugar",
           "chlorides", "free.sulfur.dioxide", "total.sulfur.dioxide", "density",
           "pH", "sulphates", "alcohol")

numeric <- c("fixed.acidity", "volatile.acidity", "citric.acid", "residual.sugar",
             "chlorides", "free.sulfur.dioxide", "total.sulfur.dioxide", "density",
             "pH", "sulphates", "alcohol")

categoric <- NULL

target <- "sabor"
risk <- NULL
ident <- NULL
ignore <- NULL
weights <- NULL
```

Para la construcción del árbol también se puede utilizar la función `rpart` que se encuentra en el paquete homónimo [25]. El principal parámetro de esta función es la complejidad, `cp`. Éste permite simplificar los modelos ajustados mediante la poda de las divisiones que no merecen la pena. Otros parámetros importantes son `minsplit`, número mínimo de observaciones que debe haber en un nodo para que se intente una partición, y `minbucket`, número mínimo de observaciones de un nodo terminal. Por defecto `minsplit=20`, `minbucket=round(minsplit/3)` y `cp=0,01`.

Para este ejemplo se han tomado `minsplit=30`, `minbucket=10` y `cp=0,01` ya que se está trabajando con bastantes observaciones.

```

library(rpart, quietly=TRUE)
set.seed(seed)

rpart <- rpart(sabor ~ ., data=dataset[train, c(input, target)],
              method="class",
              parms=list(split="information"),
              control=rpart.control(minsplit=30,
                                    minbucket=10,
                                    cp=0.01,
                                    usesurrogate=0,
                                    maxsurrogate=0))

# Árbol de decisión:
print(rpart)

## n= 3918
##
## node), split, n, loss, yval, (yprob)
##      * denotes terminal node
##
## 1) root 3918 1322 bueno (0.6625830 0.3374170)
## 2) alcohol>=10.85 1470 185 bueno (0.8741497 0.1258503) *
## 3) alcohol< 10.85 2448 1137 bueno (0.5355392 0.4644608)
## 6) volatile.acidity< 0.2475 1051 304 bueno (0.7107517 0.2892483) *
## 7) volatile.acidity>=0.2475 1397 564 malo (0.4037223 0.5962777)
## 14) alcohol>=9.85 493 227 bueno (0.5395538 0.4604462)
## 28) free.sulfur.dioxide>=18.5 394 157 bueno (0.6015228 0.3984772) *
## 29) free.sulfur.dioxide< 18.5 99 29 malo (0.2929293 0.7070707) *
## 15) alcohol< 9.85 904 298 malo (0.3296460 0.6703540) *

```

Para que el resultado sea más visual se crea el gráfico del árbol de decisión, como se puede ver en la Figura 4.5.

```

# Gráfico del árbol de decisión:
fancyRpartPlot(rpart, main="Decision Tree vino $ sabor")

```

Se considera un nuevo vino con los siguientes parámetros:

```

## fixed.acidity volatile.acidity citric.acid residual.sugar chlorides
## 1 7.0 0.27 0.36 20.7 0.045

## free.sulfur.dioxide total.sulfur.dioxide density pH sulphates alcohol
## 1 17 170 1.0010 3.00 0.45 10

```

Para su clasificación se estudia a qué nodo terminal del árbol pertenece. El parámetro alcohol es menor que 11, por lo que se escoge la rama de la derecha. A continuación se considera el parámetro volatile.acidity, que es mayor que 0,25, por lo que se sigue de nuevo por la rama derecha. Como alcohol es mayor que 9,9, se continua por la rama izquierda, y, por último, al ser free.sulfur.dioxide menor que 18, se deduce que el nuevo vino pertenece al cuarto nodo terminal. Este nodo tiene una probabilidad del 71% de que los vinos que contiene sean malos, por lo que se cataloga el vino como malo.

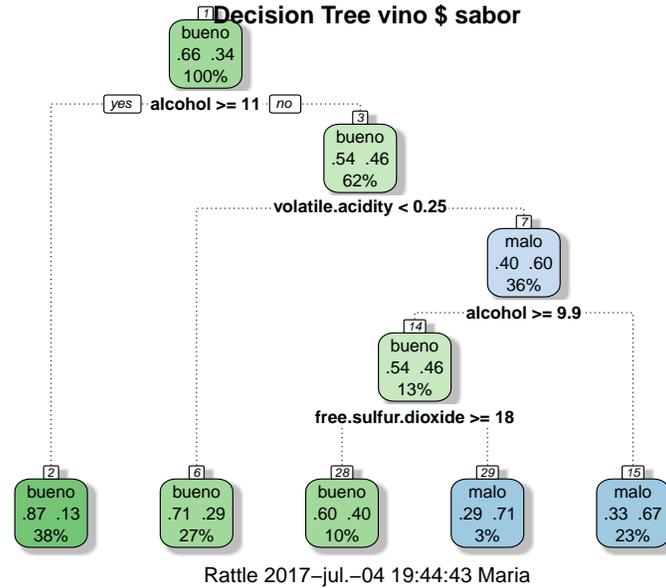


Figura 4.5: Árbol de decisión.

Determinar parámetros

El procedimiento habitual para determinar los parámetros del modelo consiste en dejar que el árbol crezca hasta que cada nodo terminal tenga menos de un número mínimo de observaciones, ya que en algunas divisiones puede que apenas mejore y en las siguientes sí lo haga. Para determinar `cp` se considera como 0, para que el árbol crezca lo máximo posible.

```

set.seed(seed)

rpart <- rpart(sabor ~ ., data=dataset[train, c(input, target)],
  method="class",
  parms=list(split="information"),
  control=rpart.control(minsplit=30,
    minbucket=10,
    cp=0.0,
    usesurrogate=0,
    maxsurrogate=0))
  
```

Como el árbol que se obtiene es muy grande, se podría considerando una función de coste basada en la complejidad. Para esto, se calcula el error de la validación cruzada (Figura 4.6) y se elige el menor.

```
plotcp(rpart)
```

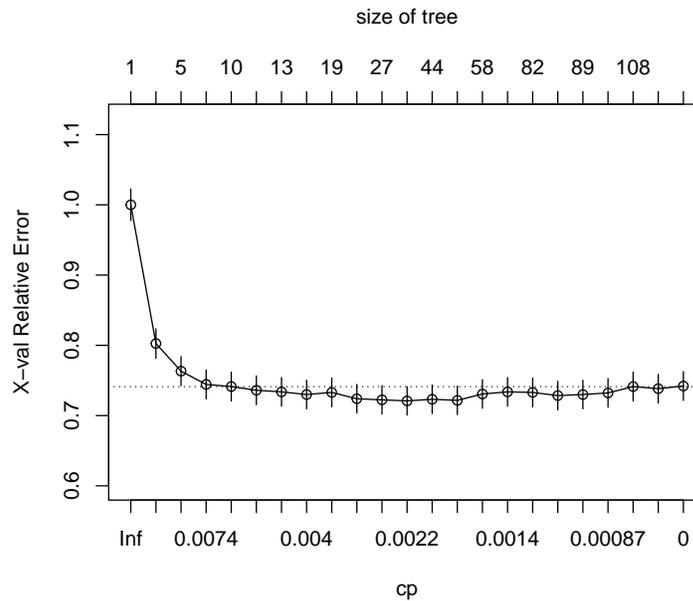


Figura 4.6: Gráfico del error de la validación cruzada.

```
printcp(rpart)
```

```
##
## Classification tree:
## rpart(formula = sabor ~ ., data = dataset[train, c(input, target)],
## method = "class", parms = list(split = "information"), control =
## rpart.control(minsplit = 30, minbucket = 10, cp = 0, usesurrogate = 0,
## maxsurrogate = 0))
##
## Variables actually used in tree construction:
## [1] alcohol          chlorides          citric.acid
## [4] density          fixed.acidity      free.sulfur.dioxide
## [7] pH              residual.sugar     sulphates
## [10] total.sulfur.dioxide volatile.acidity
##
## Root node error: 1322/3918 = 0.33742
##
## n= 3918
##
##          CP nsplit rel error  xerror   xstd
## 1  0.10173979      0  1.00000  1.00000  0.022387
## 2  0.03025719      2  0.79652  0.80257  0.021040
## 3  0.00794251      4  0.73601  0.76324  0.020704
## 4  0.00680787      6  0.72012  0.74433  0.020534
## 5  0.00605144      9  0.69970  0.74130  0.020506
## 6  0.00453858     11  0.68759  0.73601  0.020457
## 7  0.00416036     12  0.68306  0.73374  0.020436
```

```
## 8 0.00378215 17 0.66188 0.72995 0.020400
## 9 0.00353001 18 0.65809 0.73298 0.020428
## 10 0.00302572 25 0.62405 0.72390 0.020343
## 11 0.00226929 26 0.62103 0.72239 0.020328
## 12 0.00208018 29 0.61422 0.72088 0.020314
## 13 0.00189107 43 0.55825 0.72315 0.020336
## 14 0.00166415 52 0.54085 0.72163 0.020321
## 15 0.00151286 57 0.53253 0.73071 0.020407
## 16 0.00126072 78 0.49622 0.73374 0.020436
## 17 0.00113464 81 0.49244 0.73298 0.020428
## 18 0.00105900 83 0.49017 0.72844 0.020386
## 19 0.00100857 88 0.48487 0.72995 0.020400
## 20 0.00075643 91 0.48185 0.73222 0.020421
## 21 0.00050429 107 0.46520 0.74130 0.020506
## 22 0.00037821 110 0.46369 0.73828 0.020478
## 23 0.00000000 112 0.46293 0.74206 0.020513
```

Tal y como se puede ver en la tabla anterior, el mínimo error se alcanza en el nodo 12.

```
##          CP nsplit rel error  xerror    xstd
## 12 0.00208018    29  0.61422 0.72088 0.020314
```

Típicamente se considera que hasta la línea discontinua de la Figura 4.6 no hay diferencias significativas. Esta línea es la suma del mínimo error y la desviación típica, es decir:

```
0.72088 + 0.020314
```

```
## [1] 0.741194
```

Se toma el modelo más simple, que en este caso es el 6, de $cp = 0.00453858$ y se construye el árbol final que se puede ver en la Figura 4.7.

```
# Modelo árbol de decisión:
set.seed(seed)

rpart <- rpart(sabor ~ ., data=dataset[train, c(input, target)],
              method="class",
              parms=list(split="information"),
              control=rpart.control(minsplit=30,
                                   minbucket=10,
                                   cp=0.0046,
                                   usesurrogate=0,
                                   maxsurrogate=0))

# Vista textual del modelo árbol de decisión:

print(rpart)

## n= 3918
##
## node), split, n, loss, yval, (yprob)
##      * denotes terminal node
```

```
##
## 1) root 3918 1322 bueno (0.66258295 0.33741705)
## 2) alcohol>=10.85 1470 185 bueno (0.87414966 0.12585034)
## 4) free.sulfur.dioxide>=11.5 1379 140 bueno (0.89847716 0.10152284) *
## 5) free.sulfur.dioxide< 11.5 91 45 bueno (0.50549451 0.49450549)
## 10) alcohol>=11.75 43 12 bueno (0.72093023 0.27906977) *
## 11) alcohol< 11.75 48 15 malo (0.31250000 0.68750000) *
## 3) alcohol< 10.85 2448 1137 bueno (0.53553922 0.46446078)
## 6) volatile.acidity< 0.2475 1051 304 bueno (0.71075167 0.28924833)
## 12) volatile.acidity< 0.2075 591 135 bueno (0.77157360 0.22842640) *
## 13) volatile.acidity>=0.2075 460 169 bueno (0.63260870 0.36739130)
## 26) residual.sugar< 17.7 442 152 bueno (0.65610860 0.34389140) *
## 27) residual.sugar>=17.7 18 1 malo (0.05555556 0.94444444) *
## 7) volatile.acidity>=0.2475 1397 564 malo (0.40372226 0.59627774)
## 14) alcohol>=9.85 493 227 bueno (0.53955375 0.46044625)
## 28) free.sulfur.dioxide>=18.5 394 157 bueno (0.60152284 0.39847716)
## 56) total.sulfur.dioxide< 140.5 152 37 bueno (0.75657895 0.24342105) *
## 57) total.sulfur.dioxide>=140.5 242 120 bueno (0.50413223 0.49586777)
## 114) residual.sugar>=1.75 199 88 bueno (0.55778894 0.44221106)
## 228) volatile.acidity< 0.285 94 31 bueno (0.67021277 0.32978723) *
## 229) volatile.acidity>=0.285 105 48 malo (0.45714286 0.54285714) *
## 115) residual.sugar< 1.75 43 11 malo (0.25581395 0.74418605) *
## 29) free.sulfur.dioxide< 18.5 99 29 malo (0.29292929 0.70707071) *
## 15) alcohol< 9.85 904 298 malo (0.32964602 0.67035398) *
```

```
printcp(rpart)
```

```
##
## Classification tree:
## rpart(formula = sabor ~ ., data = dataset[train, c(input, target)],
## method = "class", parms = list(split = "information"), control =
## rpart.control(minsplit = 30, minbucket = 10, cp = 0.0046,
## usesurrogate = 0, maxsurrogate = 0))
##
## Variables actually used in tree construction:
## [1] alcohol free.sulfur.dioxide residual.sugar
## [4] total.sulfur.dioxide volatile.acidity
##
## Root node error: 1322/3918 = 0.33742
##
## n= 3918
##
## CP nsplit rel error xerror xstd
## 1 0.1017398 0 1.00000 1.00000 0.022387
## 2 0.0302572 2 0.79652 0.80257 0.021040
## 3 0.0079425 4 0.73601 0.76324 0.020704
## 4 0.0068079 6 0.72012 0.74433 0.020534
## 5 0.0060514 9 0.69970 0.74130 0.020506
## 6 0.0046000 11 0.68759 0.73601 0.020457
```

```
fancyRpartPlot(rpart, main="Árbol de decisión hbat $ sabor")
```

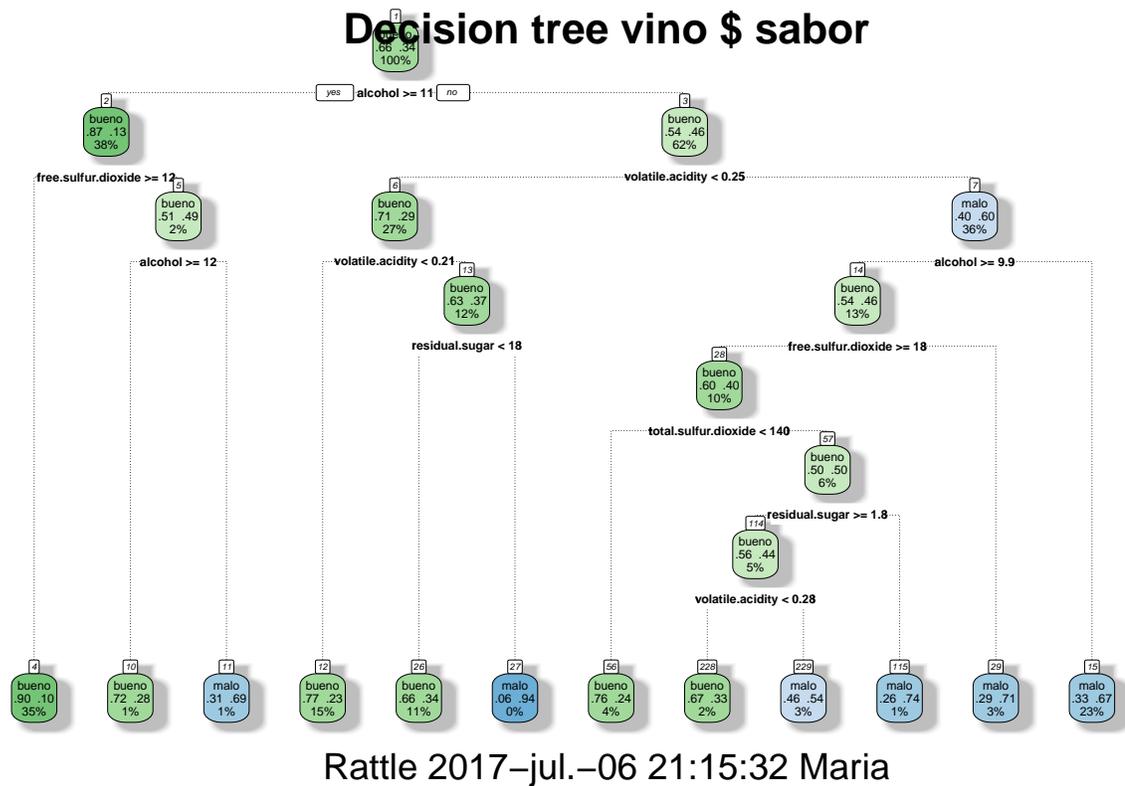


Figura 4.7: Árbol de decisión resultante.

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

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

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

```
pred <- predict(rpart, newdata=dataset[test, c(input, target)], type="class")

# Matriz de confusión
tabla <- table(dataset[test, c(input, target)]$sabor, pred, useNA="ifany",
               dnn=c("Real", "Predicho"))
```

```
tabla
```

```
##          Predicho
## Real     bueno malo
##  bueno   543  119
##  malo   102  216
```

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

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

```
## [1] 0.7744898
```

La precisión de este modelo es del 77,45% aproximadamente.

El poder de predicción de los árboles de decisión no suele ser muy bueno, pero el algoritmo es sencillo y los modelos resultantes tienen una fácil interpretación. Con el fin de mejorar esta predicción, se puede seguir la idea del *bagging* de combinar muchos métodos sencillos, como se verá en el método de bosques aleatorios.

Bosques aleatorios o *random forest*

Utilizando un árbol se observa que muchas veces dos variables tienen una capacidad similar de particionar en conjuntos de datos más homogéneos. Para contemplar varias posibilidades se utiliza el clasificador de bosque aleatorio, que genera múltiples árboles de decisión utilizando el mismo algoritmo ya visto con unas pequeñas modificaciones.

Este método utiliza un concepto denominado *bagging*, en el que el conjunto de observaciones de entrenamiento se selecciona al azar en una cesta con reemplazo, es decir, un mismo dato tiene la oportunidad de salir varias veces en la misma cesta. Con cada una de ellas se entrena un modelo.

Por defecto, en cada modelo se utiliza un conjunto aleatorio de variables diferente, de tamaño generalmente pequeño (\sqrt{v} , siendo v el número de variables). Como se considera solamente una fracción del número total de variables, la cantidad de cálculo requerida se reduce significativamente, con esto se evita la dependencia entre las variables explicativas y no hay la necesidad de podar los árboles, a diferencia del algoritmo del árbol de decisión.

Cada árbol tiene el mismo peso por lo que la clasificación de un nuevo documento será la predicción más frecuente de los estos árboles. Una de las ventajas es que este método permite estimar el error de predicción empleando las observaciones que no están en el conjunto de entrenamiento, lo que usualmente son denominados “datos fuera de la cesta” (*Out of bag*, OOB).

El algoritmo tiende a producir modelos bastante precisos, generalmente muy competitivos con clasificadores no lineales tales como redes neuronales artificiales y máquinas de soporte vectorial. Sin embargo, este rendimiento depende de los datos, siendo mejor cuando hay muchas variables y no tantas observaciones.

Para trabajar con este modelo en R se utiliza el paquete `randomForest` [13] que proporciona acceso directo a la implementación original del algoritmo.

Ejemplo:

Antes de construir el modelo, se separa el 80 % de los datos originales como training y el 20 % como test.

```
set.seed(42)
samp <- sample(nrow(vino), 0.8 * nrow(vino))
train <- vino[samp, ]
test <- vino[-samp, ]

library(randomForest)

modelo <- randomForest(sabor ~., data = train); modelo

##
## Call:
## randomForest(formula = sabor ~ ., data = train)
##           Type of random forest: classification
##           Number of trees: 500
## No. of variables tried at each split: 3
##
##           OOB estimate of error rate: 16.18%
## Confusion matrix:
##           bueno malo class.error
## bueno  2358  238  0.09167951
## malo   396  926  0.29954614
```

Se construyeron 500 árboles, que es el valor por defecto, y el modelo muestreó 3 predictores aleatorios en cada división. El error de estimación OOB es del 16,18 %. También muestra una matriz que contiene predicción vs. real, así como el error de clasificación para cada clase.

```
# Comprobación del modelo con los datos del test:
pred <- predict(modelo, newdata = test)

# Matriz de confusión:
tabla <- table(pred, test$sabor); tabla

##
## pred      bueno malo
## bueno    596   85
## malo     66  233
```

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

```
## [1] 0.8459184
```

La precisión es aproximadamente de un 84,59 %.

4.3.3. Regresión logística

Sea una variable respuesta que solamente puede tomar dos valores que se codifican como 0 o 1. El modelo de regresión logística busca conocer la relación entre esta variable dicotómica y las variables explicativas independientes, que pueden ser cualitativas o cuantitativas. A partir del entrenamiento del modelo, se puede predecir la probabilidad de pertenencia a una de las dos categorías.

Para crear la función de regresión logística se parte de la base de que el propósito es construir un modelo para la probabilidad de éxito condicionada a cada valor de la variable explicativa, es decir:

$$\pi(x) = P(Y = 1 | X = x)$$

Con el fin de considerar un modelo lineal, se le aplica a $\pi(x)$ una función g que transforma el intervalo $[0, 1]$ en toda la recta real. A esta g se le conoce como función enlace o *link*.

$$g(\pi(x, \beta)) = x'\beta \quad (4.1)$$

siendo β el vector de coeficientes de la regresión.

En el caso que se está estudiando, donde la variable respuesta es dicotómica, se suele considerar como función *link* la siguiente:

$$g(p) = \log \frac{p}{1-p} \quad \forall p \in [0, 1]$$

Esta función *link* recibe el nombre de función logística o *logit*.

Finalmente, de la ecuación (4.1) se deduce que el modelo logístico consiste en expresar la probabilidad de éxito como sigue:

$$\pi(x, \beta) = g^{-1}(x'\beta) = \frac{e^{x'\beta}}{1 + e^{x'\beta}}$$

Una generalización de este modelo se utiliza en el caso de que la variable respuesta tenga más categorías. En el ejemplo del clasificador se trata de estimar la pertenencia a una de las veinte categorías distintas, para ello se puede utilizar el método conocido como “*one vs. all*”.

Este método entrena un clasificador de regresión logística para cada grupo de clasificación, i . Una vez entrenado, se trata de predecir la probabilidad de que la respuesta sea i , es decir, de que un nuevo dato pertenezca a la categoría i . Para ello, se contrasta la probabilidad de pertenecer al grupo i frente a no hacerlo como se puede ver a continuación.

Sea un nuevo dato x , se calculan los modelos de regresión logística:

$$h_{\theta}^{(i)} = P(y = i | x; \theta)$$

Después de haber obtenido las $h_{\theta}^{(i)}$, se escoge la i que las maximice, es decir, $\max_i h_{\theta}^{(i)}(x)$.

Seguidamente, se muestra el código del modelo para el ejemplo del clasificador.

Ejemplo:

En este caso, se decidió usar código propio debido a que el código que está implementado en R para el modelo logístico daba problemas para matrices tan dispersas. Al probar el modelo logístico “*one vs. all*” funcionó sin problema y se obtuvo muy buena precisión.

La función `logist_ml` utiliza el método “*one vs. all*” para entrenar el clasificador. Para la optimización de la función de coste de cada regresión logística se usa el método `optim()`. La función devuelve la matriz de thetas, donde se guardan los parámetros θ de cada clasificador.

La función `predict.logist`, lleva a cabo la clasificación de la regresión logística. Utiliza para ello la matriz thetas y los documentos de los que se desea hacer la predicción.

```
#####
# CLASIFICACIÓN POR "ONE vs ALL".
#####
if(LOGISTIC_CLASSIF) {
  source("./functionScripts/logist_ml.R")
  source("./functionScripts/predict.logist.R")

  ## ENTRENAMIENTO DEL MODELO POR ONE vs ALL:

  logist_model <- logist_ml(Xtrain, ytrain,
                           maxit= 50,      # Iteraciones máximas
                           lambda= 1)     # Parámetro de regularización

  ## ACCURACIES DEL MODELO POR LOGISTIC REGRESSION:

  pred_train <- predict.logist(logist_model$weights, Xtrain)
  pred_test  <- predict.logist(logist_model$weights, Xtest)

  accu_logist_train <- signif(sum(pred_train$prediction == ytrain)/
                              length(ytrain), digits= 3)
  accu_logist_test  <- signif(sum(pred_test$prediction == ytest)/
                              length(ytest), digits= 3)

  # Matriz de confusión
  # confMat_logist <- table(pred_test, ytest)
}
```

Con este modelo se obtuvo una precisión en la clasificación de 78,5% trabajando con 7500 documentos y de 81,4% utilizando toda la base de datos. Para ello se fijó una *sparsity* del 99,9%, es decir, que se eliminaron los términos que aparecían en menos de un 0,1% de los documentos.

4.3.4. Los k vecinos más próximos (k-nearest neighbour classification, kNN)

El algoritmo del vecino más próximo (Nearest Neighbour, NN) calcula la similitud entre el documento que se desea clasificar y todos los documentos que pertenecen al *training*. Una vez localizado el documento de entrenamiento más similar, se le asigna la misma categoría a este nuevo documento.

Una de las variantes más utilizadas de este algoritmo es la de los k vecinos más próximos (k-nearest neighbour, kNN). En este caso, se toman los k documentos más parecidos y se estudian las categorías a las que pertenecen. La categoría que tenga más representantes será la asignada al nuevo documento.

Pese a su sencillez, el método es muy eficaz, especialmente cuando el número de categorías posibles es alto y los documentos son heterogéneos.

Esta clasificación está implementada en el paquete `class` [26] tomando como similitud la distancia euclídea.

Ejemplo:

La función `knn` entrena el método de los k vecinos más próximos. Para ello, toma el conjunto de entrenamiento y de test sin la variable respuesta y, en `trainClass`, la clasificación verdadera del *training*. Devuelve la predicción de clasificación del *test*.

Por defecto toma $k = 1$.

```
library("class")

set.seed(42)
samp <- sample(nrow(vino), 0.8 * nrow(vino))
train <- vino[samp, ]
test <- vino[-samp, ]

trainClass<-train[,"sabor"]
trueClass<-test[,"sabor"]

knnClass <- knn (train[, -12], test[, -12], trainClass)

# Matriz de confusión:
nnTabla <- table ("1-NN" = knnClass, Reuters = trueClass); nnTabla

##           Reuters
## 1-NN      bueno malo
## bueno    544  116
## malo     118  202

sum(diag(nnTabla))/nrow(test)

## [1] 0.7612245
```

La precisión obtenida es aproximadamente del 76,12%.

4.3.5. SVM o máquina de soporte vectorial

Las máquinas de soporte vectorial (*support vector machine*, SVM), son métodos de clasificación desarrolladas en los años 90 que tienen mucha importancia actualmente debido a su capacidad para trabajar con datos de alta dimensión. Aunque comenzaron únicamente resolviendo problemas de clasificación binaria, actualmente se utilizan para múltiples tipos de problemas, como por ejemplo la clasificación de documentos.

Para entender bien su funcionamiento, se explicará para la situación en la que se tengan dos dimensiones y dos categorías.

Sea un conjunto de datos como el que se puede ver en la Figura 4.8 (a). Estos datos están representados como un vector p -dimensional de la misma forma que se ha visto en los métodos anteriores. El caso más sencillo del SVM consiste en tratar de encontrar un hiperplano lineal que separe las dos

categorías, aquí representadas como cuadrados y círculos. En este caso el hiperplano sería una recta y la dimensión $p = 2$.

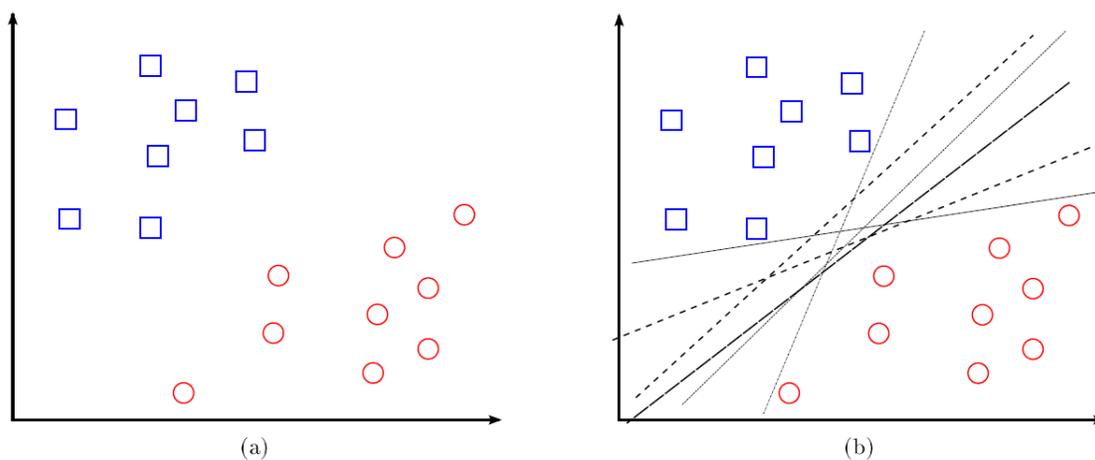


Figura 4.8: (a) Conjunto de datos del ejemplo. (b) Posibles líneas con las que dividir las dos categorías.

En este ejemplo, existen infinitos hiperplanos lineales que consiguen separar las categorías. Algunos de ellos se pueden apreciar en la Figura 4.8 (b).

Lo que se busca es seleccionar un hiperplano de separación óptima, es decir, aquella solución que permita un margen lo más amplio posible entre las categorías, debido a esto, el método también se denomina clasificador de máximo margen. El hiperplano que cumple esa condición es el equidistante a los ejemplos más cercanos de cada clase. Estos puntos determinan el margen y el hiperplano y se denominan vectores de soporte. En la Figura 4.9, son los que tienen el color compacto.

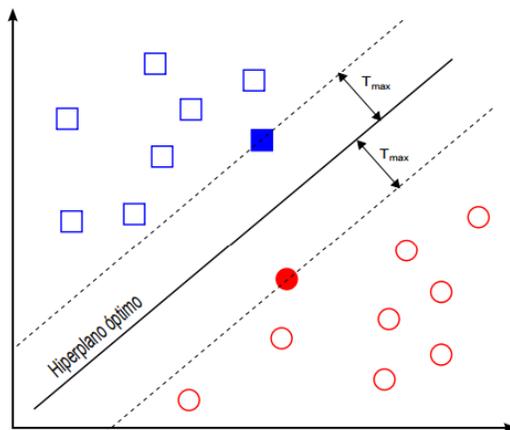


Figura 4.9: Imagen donde se muestra el hiperplano óptimo y los vectores de soporte para este ejemplo.

En este caso, se clasificaría como cuadrado cualquier nuevo documento que estuviese por encima del hiperplano, y como círculo, cualquiera que estuviese por debajo.

Desafortunadamente los problemas que se desean clasificar no acostumbran a ser tan sencillos. Hay casos en los que las dimensiones y las categorías son más de dos y donde los conjuntos de datos no son separables por un hiperplano lineal, como es el caso de la Figura 4.10.

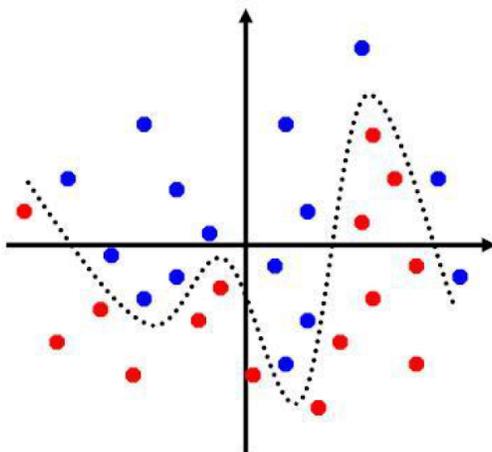


Figura 4.10: Ejemplo en el que no existe ningún hiperplano lineal que divida las dos categorías.

En el caso de que las categorías no sean linealmente separables, se busca un clasificador de “margen débil”, que permite que algunos puntos no estén en su lado del hiperplano. Para ello se tiene una restricción de que el total de las distancias de los errores de entrenamiento son menores que una constante C , mayor que 0. Este tipo de clasificador también se puede utilizar en el caso de que las categorías sean separables con el fin de permitir cierta flexibilidad y evitar un sobreajuste.

Si los datos no son separables linealmente en el espacio original se puede llevar a cabo una transformación de estos a través de una función núcleo o *kernel*.

Definición 4.1 (Función núcleo). Sea X el espacio de entrada, H el de características dotado de un producto interno y una función $F : X \rightarrow H$, con H el espacio inducido de Hilbert, se define la función núcleo como $K : X \times X \rightarrow \mathbb{R}$ como:

$$K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$$

Estas funciones proyectan la información a un espacio de características de mayor dimensión, lo que aumenta la capacidad de clasificación de las máquinas de aprendizaje lineal. Los núcleos más utilizados son el lineal, el polinómico, el radial y el sigmooidal, aunque se puede crear uno propio si se desea.

- Lineal: $K(x_i, x_j) = x_i \cdot x_j$
- Polinómico: $K(x_i, x_j) = (x'_i \cdot x_j + c)^d$, siendo d el grado del polinomio y c un parámetro.
- Radial: $K(x_i, x_j) = \exp\left(\frac{-\|x_i - x_j\|^2}{2\sigma^2}\right)$, siendo σ un parámetro.

- Sigmoidal: $K(x_i, x_j) = \tanh(\gamma \cdot x'_i \cdot x_j + \delta)$, siendo γ y δ parámetros.

La búsqueda de un hiperplano óptimo es un problema de optimización cuadrática con restricciones lineales que se resuelve utilizando técnicas estándar. La propiedad de convexidad que se exige garantiza una única solución.

Cuando ya se tiene el hiperplano, se puede clasificar cualquier nuevo documento simplemente comprobando en qué lado de los vectores está, y con ello, estimar la categoría a la que pertenece.

Para su implementación en R se utiliza el paquete `kernlab` [11] o el `e1071` [15]. Primero se mostrará el proceso con el ejemplo del vino, ya que es más sencillo, y luego se explicará el ejemplo de la clasificación de documentos.

```
library("kernlab")

# Se aplica el SVM a los datos de entrenamiento:
ksvmTrain <- ksvm (sabor ~., data = train)

# Se clasifica el conjunto de datos de prueba a partir del SVM entrenado,
# para ello, se elimina la columna donde está la clasificación original:
svmClass <- predict (ksvmTrain, test [, -12])

# Matriz de confusión:
svmTabla <- table(SVM = svmClass, Reuters = trueClass); svmTabla

##           Reuters
## SVM      bueno malo
## bueno    575  113
## malo     87  205

sum(diag(svmTabla))/nrow(test)
```

```
## [1] 0.7959184
```

La precisión de este método es aproximadamente del 79,59%.

A continuación se muestra el SVM para el problema del clasificador, comenzando por la librería `e1071`.

```
#####
# CLASIFICACION POR "SUPPORT VECTOR MACHINES" library(e1071).
#####
if(SVM_CLASSIF) {

  library(e1071)

  ## ENTRENAMIENTO DEL MODELO POR SVM:

  svm_model <- svm(Xtrain, ytrain,
```

```

        type= "C-classification",
        probability= TRUE,
        kernel= "radial",
        cost= 1) # Parámetro de regularización (1 por defecto)

## ACCURACIES DEL MODELO POR SVM:

pred_train <- as.numeric(predict(svm_model, Xtrain))
pred_test  <- as.numeric(predict(svm_model, Xtest))

accu_svm_train <- signif(sum(pred_train==ytrain)/length(ytrain), digits= 3)
accu_svm_test  <- signif(sum(pred_test==ytest)/length(ytest), digits= 3)
}

```

Utilizando la librería `kernlab`, se procede a definir un núcleo basado en la similaridad del coseno para el problema del clasificador.

```

#=====  

# CLASIFICACION POR "SUPPORT VECTOR MACHINES" library(kernlab). Coseno Kernel  

#=====
if(KSVM_CLASSIF) {

  library(kernlab)

  ## ENTRENAMIENTO DEL MODELO POR SVM:

  # Núcleo basado en la similaridad por coseno:
  coseno_kern <- function(x,y) {sum(x*y)/sqrt(sum(x^2)*sum(y^2))}
  class(coseno_kern) <- "kernel"

  # SVM:
  ksvm_model <- ksvm(Xtrain, ytrain,
                    type= "C-svc",
                    kernel= coseno_kern)

  ## ACCURACIES DEL MODELO POR SVM:

  pred_train <- as.numeric(predict(svm_model, Xtrain))
  pred_test  <- as.numeric(predict(svm_model, Xtest))

  accu_svm_train <- signif(sum(pred_train==ytrain)/length(ytrain), digits= 3)
  accu_svm_test  <- signif(sum(pred_test==ytest)/length(ytest), digits= 3)
}

```

Con este modelo se obtuvieron resultados muy prometedores para un número de documentos bajo. No se pudo comprobar para más documentos debido a que requería mucha capacidad de cómputo, por lo tanto para el clasificador se utiliza el SVM radial.

Con el SVM radial se obtuvo una precisión en la clasificación de 71% trabajando con 7500 documentos y de 77,7% utilizando toda la base de datos. Para ello se fijó una *sparsity* del 99,5%.

4.3.6. Resumen de los métodos para el ejemplo del clasificador.

Los métodos explicados en esta sección son los más utilizados para la clasificación de documentos, no obstante, para el ejemplo inicial de los 19997 documentos, con el *random forest* se obtuvieron peores resultados que con los otros métodos y el *k-nearest neighbour* presenta un tiempo de computación muy elevado, debido al alto número de dimensiones. Por estos motivos, a partir de aquí, el estudio se centra en las distintas variaciones posibles dentro del modelo logístico y el SVM.

Después de comparar estos modelos utilizando la ponderación tfidf y solamente la tf, se ha llegado a la conclusión de que el uso de la tfidf mejora la precisión. En la Figura 4.11 se ven los métodos SVM y logístico y la diferencia entre utilizar una u otra ponderación. tomando un *training* del 60 % y un *test* del 40 %.

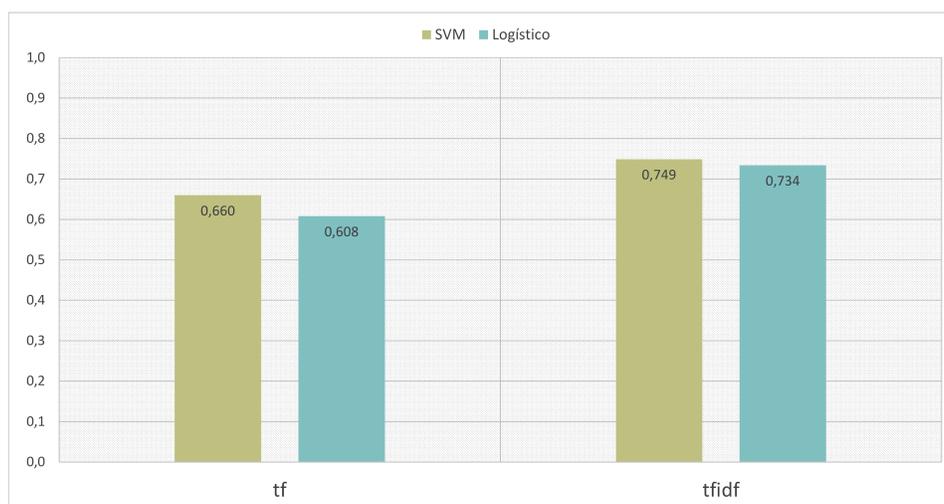


Figura 4.11: Porcentaje de documentos bien clasificados en el conjunto de prueba por los modelos SVM y logístico utilizando tf y tfidf, con 19997 documentos.

En la Figura 4.12 y la Figura 4.13, se puede observar que la precisión de los dos métodos aumenta según se va aumentando el tamaño de documentos con los que se trabaja (5000, 7500 o el total, 19997). Tomando un 80 % de los textos como entrenamiento y un 20 % de prueba.

El mejor resultado obtenido con el SVM presenta una precisión del 77,7 %, con 7500 documentos. Para el caso de 19997 documentos no ha sido posible calcular su precisión, debido a que el método requería de una mayor capacidad de procesado.

El mejor resultado obtenido para el clasificador se ha conseguido utilizando la tfidf y el modelo logístico, con 19997 documentos y una *sparsity* del 99,9 %. Cuanto más filtrado por frecuencia se ha obtenido peor precisión, por lo que no se ha filtrado por frecuencia. Este modelo presenta una precisión del 81,4 %.

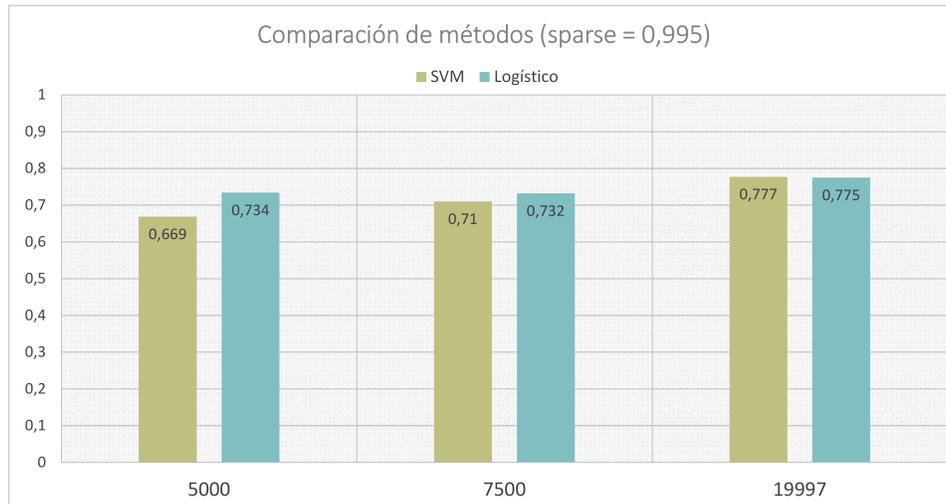


Figura 4.12: Porcentaje de documentos bien clasificados en el conjunto de prueba por los modelos SVM y logístico con sparse = 0,995.

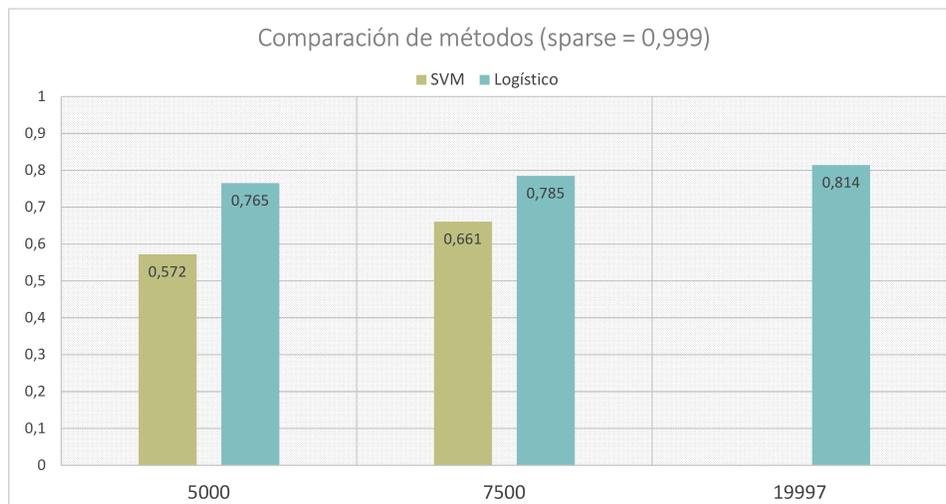


Figura 4.13: Porcentaje de documentos bien clasificados en el conjunto de prueba por los modelos SVM y logístico con sparse = 0,999.

En este capítulo se han utilizado principalmente las referencias [3], [5], [12], [26], [27] y [28].

Capítulo 5

Aplicación en *Shiny*

Las aplicaciones gráficas se utilizan a menudo para mostrar más claramente resultados de la minería de textos. Han ido avanzando a lo largo de estos últimos años, llegando a dejar al usuario ejecutar un cierto número de consultas preprogramadas y fijas. Actualmente, se ha conseguido que estos sistemas de minería de textos puedan exponer gran parte de su funcionalidad al usuario, mediante un acceso a las líneas de comando del programa.

Con la finalidad de que el trabajo realizado se pudiese ver e interactuar con él con la máxima sencillez posible se creó una aplicación *Shiny* del clasificador de textos utilizando RStudio.

Esta aplicación es una carpeta que contiene los siguientes dos archivos (y alguno extra de manera opcional):

- La interfaz de usuario (`ui.R`): controla el diseño y aspecto de la aplicación.
- El servidor (`server.R`): contiene las instrucciones que necesita el equipo para construir la aplicación.

El *Shiny* se puede correr en cualquiera de los dos archivos mediante el botón “Run App”.

Esta aplicación consta de tres pestañas donde cada una proporciona información útil acerca de los documentos con los que se está trabajando.

En la primera pestaña (Figura 5.1), se permite elegir una de las 20 carpetas de textos y ver su diagrama de barras o su nube de palabras, pudiendo seleccionar el número de términos deseados para realizar este gráfico. Con esta pestaña se pretende que el usuario se familiarice con el conjunto de datos con los que se trabaja y pueda comparar gráficamente varias carpetas que sean de su interés.

En la segunda pestaña (Figura 5.2), se muestra una tabla donde están los términos pertenecientes a cada una de las carpetas. Cada término está acompañado de su número correspondiente y su frecuencia. Se puede elegir la carpeta de la que se desea tener información, el intervalo de frecuencia y el número de términos que se muestran por pantalla. Además, a la derecha hay un buscador rápido que se puede utilizar tanto para encontrar un término como los términos con una frecuencia determinada.

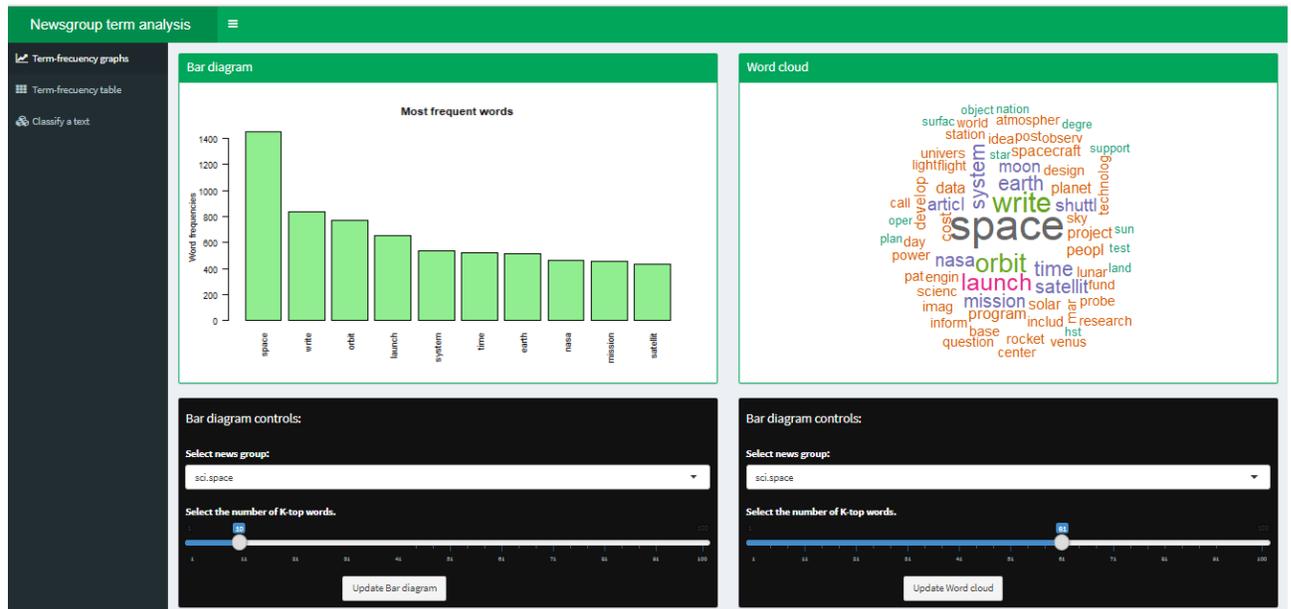


Figura 5.1: Primera pestaña de la aplicación *Shiny*.

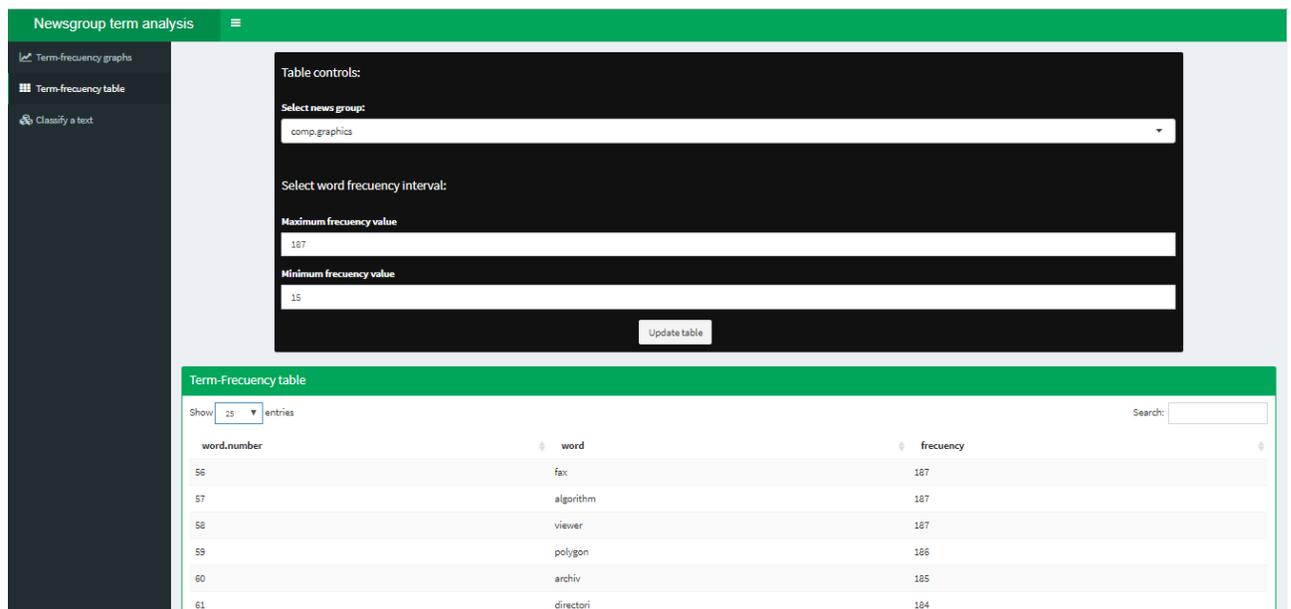


Figura 5.2: Segunda pestaña de la aplicación *Shiny*.

La última pestaña de la aplicación es el clasificador propiamente dicho. El usuario introduce, bien a mano o bien adjuntando un archivo de texto, el documento que se desea clasificar. Este texto aparece en una ventana, como se puede apreciar en la Figura 5.3. Pulsando el botón “Classify”, el modelo elegido hace una predicción de a qué carpeta pertenece el texto.

Si se desea borrar el texto con el que se está trabajando es posible, simplemente hay que pulsar el botón “Reset”.

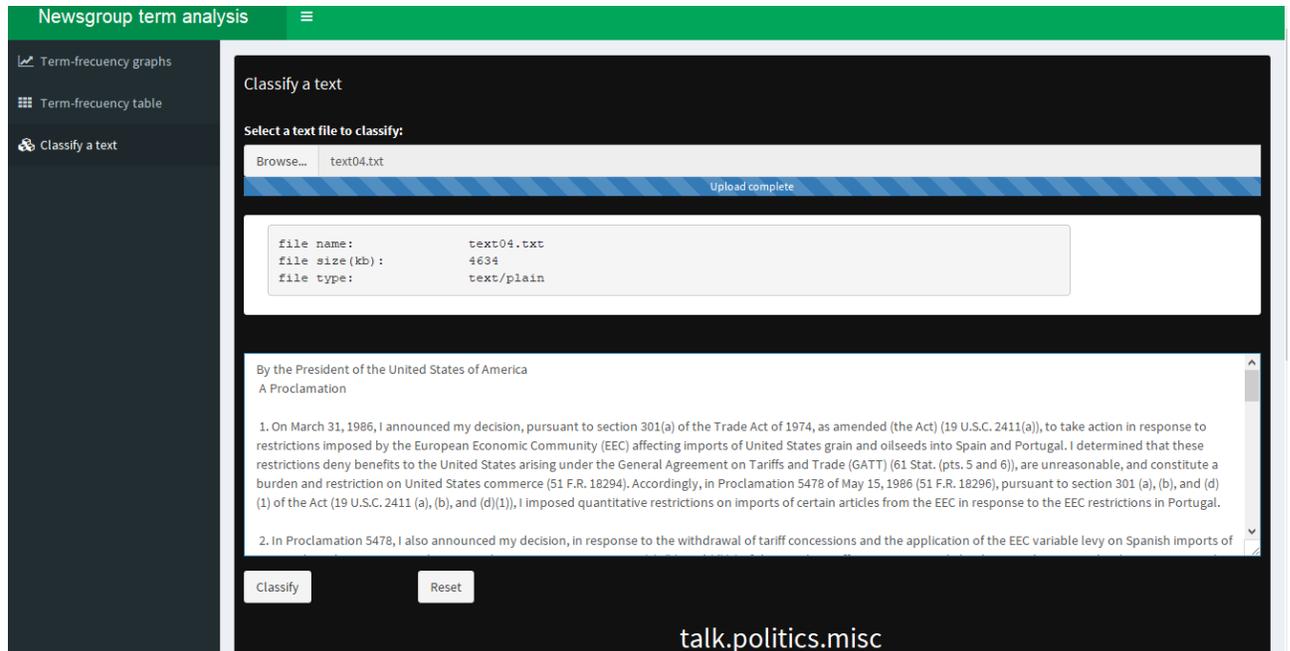


Figura 5.3: Tercera pestaña de la aplicación *Shiny*.

Si se continúa bajando en la pestaña, se muestran las probabilidades que tiene el texto de pertenecer a cada una de las carpetas, según el modelo elegido.

Esta aplicación tiene implementados los dos modelos con mejores resultados, es decir, el modelo logístico y el SVM. Para elegir el modelo SVM es necesario cambiarlo en el código dentro de una función denominada `tratayPredice`, ya que por defecto está seleccionado el logístico, debido a que fue el que proporcionó mejores resultados.

```
tratayPredice <- function(texto){
  logistic_classification <- TRUE # si no, hace svm_classification
  if(logistic_classification) {
    # Cargamos el modelo entrenado.
    load("./modelos/WS_logist_19997_81point4.RData")
  } else {
    # Cargamos el modelo entrenado.
    load("./modelos/WS_svm_19997_77point7.RData")
  }
  ...
}
```

Se puede observar en la Figura 5.4 que el modelo logístico predice como más probables la carpeta de *talk.politics.misc* y *talk.politics.guns*, lo que tiene sentido, puesto que el texto escogido es una declaración del presidente de los Estados Unidos, Ronald Reagan, el 22 de enero de 1987.

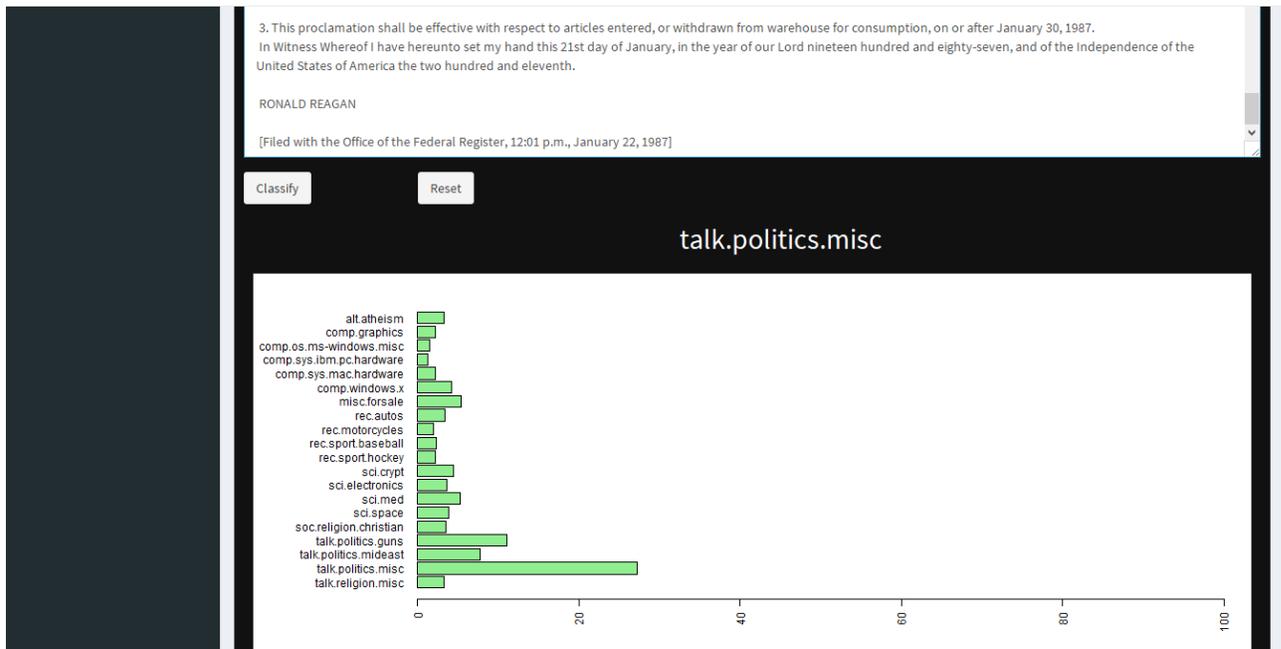


Figura 5.4: Continuación de la tercera pestaña de la aplicación *Shiny*.



Figura 5.5: Logotipo de Oesía.

Este trabajo ha sido realizado con la ayuda de la empresa Oesía. Ésta es un filial fundada en el año 2000 de Oesía Networks S.L., una consultora multinacional especializada en tecnología, presente en España y Latinoamérica, que desarrolla proyectos para clientes en todo el mundo.

Oesía Networks S.L. está especializada en:

1. Prestación de servicios de tecnologías de la información y de las telecomunicaciones, incluidos, entre otros, la captura de información por medios electrónicos, informáticos y telemáticos.
2. Diseño, desarrollo, producción, implantación, integración, operación, mantenimiento, reparación y comercio.

Para la realización de este capítulo se han utilizado principalmente los enlaces [30] y [31].

Bibliografía

- [1] Allaire J. J., Cheng J., Xie Y., McPherson J., Chang W., Allen J., Wickham H., Atkins A., Hyndman R., Arslan R. (2017) rmarkdown: Dynamic Documents for R. R package version 1.5. <https://CRAN.R-project.org/package=rmarkdown>
- [2] Blei D. M., Ng A. Y., Jordan M. I. (2003) Latent Dirichlet Allocation. *Journal of Machine Learning Research* 3, 993-1022, USA.
- [3] Carmona Suárez E. J. (2014) Tutorial sobre Máquinas de Vectores Soporte (SVM). UNED
- [4] Dowle M., Srinivasan A. (2017) data.table: Extension of ‘data.frame’. R package version 1.10.4. <https://CRAN.R-project.org/package=data.table>
- [5] Faraway J. J. (2005) Extending the linear model with R: generalized linear, mixed effects and nonparametric regression models. Chapman and Hall/CRC
- [6] Feinerer I. (2008) An introduction to text mining in R. *R News*, 8(2):19-22, <http://CRAN.R-project.org/doc/Rnews/>
- [7] Feinerer I., Hornik K. (2016) _wordnet: WordNet Interface_. R package version 0.1-11. <https://CRAN.R-project.org/package=wordnet>
- [8] Feinerer I., Hornik K. (2017) tm: Text Mining Package. R package version 0.7-1. <https://CRAN.R-project.org/package=tm>
- [9] Feinerer I., Hornik K., Meyer D. (2008) Text mining infrastructure in R. *Journal of Statistical Software*, 25(5): 1-54, ISSN 1548-7660. <http://www.jstatsoft.org/v25/i05>
- [10] Fellbaum C. (1998) _WordNet: An Electronic Lexical Database_. Bradford
- [11] Karatzoglou A., Smola A., Hornik K., Zeileis A. (2004) kernlab - An S4 Package for Kernel Methods in R. *Journal of Statistical Software* 11(9), 1-20. <http://www.jstatsoft.org/v11/i09/>
- [12] Kumar A., Paul A. (2016) Mastering. Text Mining with R.
- [13] Liaw A., Wiener M. (2002) Classification and Regression by randomForest, *R News*, 2(3), 18–22.
- [14] Manning C. D., Raghavan P., Schütze H. (2009) An Introduction to Information Retrieval. Cambridge University Press, Cambridge, England, ISBN 978-0-511-80907-1. <https://doi.org/10.1017/CBO9780511809071>
- [15] Meyer D., Dimitriadou E., Hornik K., Weingessel A., Leisch F. (2017) e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien. R package version 1.6-8. <https://CRAN.R-project.org/package=e1071>
- [16] Munzert S., Rubba C., Meißner P., Nyhuis D. (2015) Automated Data Collection with R. A Practical Guide to Web Scraping and Text Mining. Ed. Wiley

- [17] Myint Z. M., OO M. Z. (2016) Analysis of modified inverse document frequency variants for word sense disambiguation. Department of Computer Engineering and Information Technology, Mandalay Technological University, Myanmar
- [18] Paltoglou G., Thelwall M. (2010) A study of Information Retrieval weighting schemes for sentiment analysis. University of Wolverhampton, Wolverhampton, United Kingdom. <http://www.aclweb.org/anthology/P10-1141>
- [19] Porter M. F. (1980) An algorithm for suffix stripping, Program, Vol. 14 Issue: 3, pp.130-137, Cambridge. <https://doi.org/10.1108/eb046814>
- [20] R Core Team. (2013) R: A language and environment for statistical computing. R Foundation for Statistical Computing, Viena, Austria. <http://www.R-project.org/>
- [21] Rosas M. V., Errecalde M. L., Rosso P. (2010) Un framework de Ingeniería del Lenguaje para el Pre-procesado Semántico de Textos. XVI Congreso Argentino de Ciencias de la Computación. (Objeto de conferencia)
- [22] Selivanov D. (2016) text2vec: Modern Text Mining Framework for R. R package version 0.4.0. <https://CRAN.R-project.org/package=text2vec>
- [23] Sievert C., Shirley K. (2015) LDAvis: Interactive Visualization of Topic Models. R package version 0.3.2. <https://CRAN.R-project.org/package=LDAvis>
- [24] Spärck Jones K. (1972) A statistical interpretation of term specificity and its application in retrieval. Computer Laboratory, University of Cambridge, Cambridge, UK
- [25] Therneau T., Atkinson B., Ripley B. (2017) rpart: Recursive Partitioning and Regression Trees. R package version 4.1-11. <https://CRAN.R-project.org/package=rpart>
- [26] Venables W. N., Ripley B. D. (2002) Modern Applied Statistics with S. Fourth Edition. Springer, New York. ISBN 0-387-95457-0
- [27] Williams G. (2011) Data Mining with Rattle and R: The Art of Excavating Data for Knowledge Discovery. Use R!, Springer, New York. <http://dx.doi.org/10.1007/978-1-4419-9890-3>
- [28] Williams G. (2016) Hands-On Data Science with R. Text Mining <http://handsondatascience.com/TextMiningO.pdf>
- [29] Xie Y. (2016) servr: A Simple HTTP Server to Serve Static Files or Dynamic Documents. R package version 0.5. <https://CRAN.R-project.org/package=servr>

Páginas web:

- [30] <https://shiny.rstudio.com/>
- [31] <https://rstudio.github.io/shinydashboard/>