Universidad de Valladolid

FACULTAD DE CIENCIAS

TRABAJO FIN DE GRADO

Grado en Estadística

Análisis de texto con R

Autor: **Alfonso Cabrero de Diego**

Tutor: **Luis Ángel García Escudero**

2025

Resumen

El análisis estadístico de texto es una especialización del Análisis de Datos que se enfrenta al reto de tratar y procesar textos con el objetivo de obtener información útil. Esta tarea presenta importantes desafíos, ya que el texto es un tipo de dato no estructurado, frecuentemente muy disperso, y el cual, además, suele representarse en espacios de elevada dimensionalidad. Estas características provocan que las técnicas tradicionales de Análisis de Datos obtengan resultados poco satisfactorios, o que, en algunos casos, no sean viables debido a su muy elevado coste computacional.

Este Trabajo de Fin de Grado proporciona una revisión, tanto a nivel teórico como aplicado, de algunas de las técnicas estadísticas aplicadas en análisis de texto. En concreto, el trabajo se centra en tres de sus técnicas más representativas, que son: el Análisis de Sentimiento, el Clústering y la Clasificación. Para cada una de ellas, se realiza una breve introducción que describe las principales adaptaciones orientadas al tratamiento de texto, así como una demostración práctica con R, utilizando un conjunto de datos reales. Todo el código desarrollado en este trabajo se encuentra disponible en un repositorio público de GitHub.

Palabras clave: Análisis de texto, minería de texto, análisis de sentimiento, clústering de texto, clasificación de texto, procesamiento del lenguaje natural.

Abstract

Statistical text analysis is a specialization within Data Analysis that addresses the challenge of handling and processing textual data in order to extract useful information. This task poses significant challenges, as text is an unstructured and often highly sparse type of data, which is typically represented in high-dimensional spaces. These characteristics lead traditional Data Analysis techniques to yield unsatisfactory results or, in some cases, to be infeasible due to their high computational cost.

This Bachelor's Thesis offers both a theoretical and applied review of several statistical techniques used in text analysis. Specifically, it focuses on three of the most representative methods: Sentiment Analysis, Clustering, and Classification. For each of these techniques, a brief introduction is provided, outlining the main adaptations required for text processing, along with a practical demonstration using R and a real-world dataset. All the code developed in this project is available in a public GitHub repository.

Keywords: Text analysis, text mining, sentiment analysis, text clustering, text classification, natural language processing.

Agradecimientos

Me gustaría agradecer profundamente a mi familia todo su apoyo durante este camino. Sin vuestra confianza y apoyo incondicional, no habría podido llegar hasta aquí.

También quiero expresar mi agradecimiento a todos los profesores del grado, por despertar mi curiosidad por la estadística y por transmitirme su pasión por una forma tan especial de entender la realidad. Pero ante todo, me gustaría agradecer a mi tutor, Luis Ángel, por sus consejos, enseñanzas y valiosa guía a lo largo de este trabajo.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto

El Análisis de Datos se ha convertido en una herramienta fundamental para orientar la toma de decisiones, permitiendo extraer información útil de datos obtenidos en cualquier tipo de entorno. Sectores como la salud, las finanzas o la ingeniería utilizan a diario estas herramientas para beneficiarse de los grandes volúmenes de datos que se generan durante sus operativas, especialmente en entornos digitales.

En este contexto, los datos no estructurados adquieren una importancia creciente, dado que suponen una proporción mayoritaria y en continuo aumento de los datos generados a nivel global. Por ejemplo, los datos no estructurados se estima ya representan entre el 80 % y el 90 % de los datos manejados por las empresas a nivel global [1].

Entre estos datos no estructurados destaca el texto, en forma de documentos, correos electrónicos o comentarios en redes sociales, entre otros. A diferencia de los datos estructurados, los textos carecen de una organización tabular, lo que dificulta su análisis mediante técnicas estadísticas tradicionales.

En respuesta a este problema, el análisis de texto ofrece métodos y herramientas diseñadas específicamente para procesar el lenguaje natural, adaptando las técnicas de Análisis de Datos a un contexto de datos dispersos y en alta dimensionalidad, como es habitualmente el caso en esta disciplina.

1.2. Objetivos

El objetivo de este trabajo es presentar metodológicamente y aplicar en R algunas de las principales técnicas de análisis de texto. Específicamente, se abordarán las siguientes técnicas: Análisis de Sentimiento, Clústering, y Clasificación.

Aunque el trabajo se centra en la aplicación práctica de algunas técnicas, se mencionarán también otros modelos disponibles, sin realizar una demostración de su uso. El fin es ofrecer una revisión de algunas técnicas muy utilizadas en este campo, junto a su debida motivación y justificación, si bien no se pretende realizar una revisión exhaustiva del estado del arte.

1.3. Herramientas

Durante el desarrollo del trabajo, las herramientas más importantes que se han utilizado han sido las siguientes:

- *Lenguaje de programación R con el IDE RStudio*: utilizados para desarrollar el código de demostración para las técnicas de análisis de texto. Algunos de los paquetes de procesamiento de texto más relevantes que se han utilizado son:
  - `tm` – utilizado para crear y limpiar el corpus.
  - `textclean` – utilizado para preprocesamiento del corpus (normalización).
  - `textstem` – utilizado para preprocesamiento del corpus (lematización).
  - `text2vec` – utilizado para crear matrices término-documento (BoW, TF-IDF), y para las representaciones con reducción de dimensión LSA y GloVe.
  - `word2vec` – utilizado para representación del texto con el modelo Word2Vec.
- *git y GitHub*: Utilizados para el control de versiones del código desarrollado, y para publicarlo una vez completado. Se puede encontrar en el siguiente repositorio: https://github.com/acabrerod/text-analysis (en inglés).
- LaTeX *con el IDE TeXstudio*: utilizados para redactar la presente memoria.
- *Google Scholar*: utilizado para buscar referencias y literatura académica.

1.4. Estructura de la memoria

Capítulo 1 – Introducción: Presentación del trabajo, objetivos, contexto y relevancia.

Capítulo 2 – Análisis de texto: Visión general de la disciplina. Se introducen conceptos como técnicas de preprocesamiento, métodos de representación numérica, distancias, y formas de visualización de texto.

Capítulo 3 – Análisis de Sentimiento, **Capítulo 4 – Clústering**, y **Capítulo 5 – Clasificación supervisada**: Introducen brevemente las técnicas de análisis de texto correspondientes, y muestran su aplicación en R con el conjunto de datos *IMDb Movie Reviews*.

Capítulo 6 – Conclusiones y trabajo futuro: Resumen de los resultados del trabajo, evaluación de los objetivos, aportaciones realizadas y líneas de trabajo futuro propuestas.

Bibliografía Referencias bibliográficas.

Apéndice A, **Apéndice B**, y **Apéndice C**: Contienen el código en R utilizado para los análisis del presente trabajo, respectivamente para las técnicas de Análisis de Sentimiento, Clústering y Clasificación, que también se puede consultar en el siguiente repositorio: enlace a GitHub.

Capítulo 2

Análisis de texto

El análisis de texto es el proceso de extraer información útil de alto nivel de texto no estructurado, identificando y explorando patrones relevantes [2]. A diferencia del Análisis de Datos convencional, donde se tratan datos estructurados, el análisis de texto se enfrenta al reto de adaptar estas técnicas analizando datos dispersos, en altas dimensiones y no estructurados [3].

La era de la información se caracteriza por un gran crecimiento en la disponibilidad de los datos, principalmente no estructurados [1, 4]. Las técnicas de análisis de texto pretenden aprovecharlos, analizando, por ejemplo, noticias, artículos, publicaciones en redes sociales, transcripciones de audios y vídeos, reseñas de productos, correos electrónicos, documentos e informes empresariales o académicos [3, 5, 6].

Para adaptar al texto las técnicas convencionales de Análisis de Datos, el principal reto del análisis de texto es transformarlo a un formato estructurado. Para esta tarea, se apoya en varias técnicas de preprocesamiento como las que se describen en este capítulo.

El análisis de texto tiene aplicaciones en muchos sectores, y puede servir, por ejemplo, para mejorar la atención al cliente, midiendo la satisfacción de los consumidores, para las autoridades de supervisión y operadores financieros, midiendo la incertidumbre en la economía, anticipando volatilidad y mejorando la gestión de riesgos, o para mejorar el acceso a la información, con técnicas más sofisticadas de representación, visualización, filtrado y de generación de resúmenes [5, 7].

En el contexto de este trabajo, se consideran como sinónimos los términos análisis de texto y minería de texto (*text mining*), dado que ambos persiguen el mismo objetivo: extraer información útil a partir de datos textuales. No obstante, algunos autores las distinguen por utilizar metodologías diferentes [8].

Este capítulo se centra en analizar y resumir las principales diferencias entre el análisis de texto y el análisis de datos estructurados, así como las técnicas fundamentales para su tratamiento. Su estructura es la siguiente: en primer lugar, en la Sección 2.1, se presentan las técnicas de preprocesamiento más comunes, cuyo objetivo es la reducción de ruido y facilitar la representación posterior. La Sección 2.2 expone las principales metodologías para la representación numérica del texto, requisito indispensable para su uso en modelos de aprendizaje automático. A continuación, la Sección 2.3 describe diversas métricas de distancia, fundamentales para estimar la similitud entre textos o documentos. La Sección 2.4 aborda las tareas más relevantes dentro del análisis de texto, junto con algunas de sus aplicaciones prácticas. Finalmente, la Sección 2.5 describe las técnicas de visualización más utilizadas en el ámbito del análisis de texto.

2.1. Preprocesamiento

Cualquier tarea de análisis de texto debe basarse en metodologías sofisticadas de preprocesamiento para ser efectiva. De hecho, el análisis de texto depende tanto de las distintas técnicas de preprocesamiento que se utilicen que, en cierta medida, podría decirse, que está *definido* por ellas [2]. Esta sección está dedicada a describir el proceso de preprocesamiento de texto y las tareas que lo componen, aunque naturalemente, no todas las metodologías o técnicas de análisis aplican el mismo tratamiento.

El preprocesamiento de texto es la etapa inicial en la que se limpia, normaliza y transforma el texto a un formato estructurado y útil, que pueda ser procesado por algoritmos de análisis o modelos de aprendizaje automático [2, 9].

Según Palmer [10], el proceso de preprocesamiento se puede dividir en dos fases: cribado de documentos y segmentación y normalización del texto. Ambas fases se describen en las Subsecciones 2.1.1 y 2.1.2, que definen brevemente estos conceptos y los principales retos a los que se enfrentan.

2.1.1. Cribado de documentos

El cribado se refiere al proceso de convertir archivos digitales en documentos de texto. En esta fase se debe identificar el idioma, con el objetivo de determinar el tipo de algoritmos de preprocesado que se debe aplicar[1], y extraer el texto de los documentos, convirtiendo, por ejemplo, documentos guardados como pdf a txt. Normalmente se ignoran otros elementos como figuras, tablas, encabezados o enlaces.

El producto resultante de la fase de cribado es un corpus, que en análisis de texto se define como una colección de textos en un formato legible por una máquina [11]. Desde la perspectiva del procesamiento de lenguaje natural, este término se emplea además para referirse a una muestra representativa de una lengua natural o de una variedad lingüística particular [12]. En el contexto de este trabajo, el término corpus se utilizará conforme a la primera definición.

2.1.2. Segmentación y normalización

La fase de segmentación tiene como objetivo dividir el corpus obtenido en la fase de cribado en sus componentes básicos, que normalmente serán palabras u oraciones [10]. A este proceso también se lo conoce como tokenización.

Cada metodología de análisis de texto adapta estas técnicas de segmentación y normalización en función de sus objetivos específicos. Dependiendo del tipo de texto, ciertas operaciones pueden ser más o menos importantes. Por ejemplo, en textos de redes sociales, es importante la normalización por la presencia de errores, abreviaturas, o ruido en general.

[1] Las convenciones ortográficas de los lenguajes escritos son muy cambiantes dependiendo del idioma. Por ejemplo, considerando los separadores entre palabras y oraciones. En ambos extremos, el arameo utiliza separadores explícitos para ambos, mientras que el tailandés no utiliza ninguno. A mitad de camino, por ejemplo, está el español, que utiliza espacios para separar las palabras y signos de puntuación para separar las oraciones, aunque ninguno de ellos son suficientes para segmentar el texto sin ambigüedades [10].

Al final de esta fase, se debe abordar la representación del texto en un formato numérico. Debido a su relevancia, este proceso se desarrolla en detalle en la Sección 2.2, y sería la última fase del preprocesamiento.

- **Tokenización**: Consiste en separar el texto en tokens, que dependiendo del método, se pueden utilizar desde letras o partes de palabras[2], hasta palabras u oraciones enteras [11].

  El objetivo principal de la tokenización es identificar los separadores entre los tokens, algo que no es siempre sencillo. Por ejemplo, no todos los idiomas utilizan espacios para separar palabras, como son el chino, el tailandés, o el japonés [10, 11]. En estos casos, es necesaria información léxica o morfológica adicional.

  Otro problema común para la tokenización de oraciones, es identificar los signos de puntuación, ya que en algunos idiomas se utilizan también para indicar abreviaturas, entre otros usos.

  Una cuestión adicional que también hay que tener en cuenta para la tokenización de palabras, es que algunos conceptos se expresan utilizando varias de ellas, lo que en ocasiones lleva a considerar como palabras largas términos como *Reino Unido*, o *Unión Europea*, aunque esta distinción no sea siempre aplicable[3].

- **Normalización**: Convierte el texto a una forma canónica normalizada. La normalización reduce la variabilidad del lenguaje, facilitando el análisis y la comparación entre textos [10]. Algunas de las tareas relacionadas con la normalización son las siguientes:

  1. Conversión a minúsculas.

  2. Eliminación o sustitución de símbolos (acentos, diéresis, exclamaciones, guiones, signos de puntuación, etc).

  3. Expansión de contracciones (abreviaturas, fechas, cantidades numéricas, etc).

  4. **Lematización**: Reduce las palabras a su forma en el diccionario (lema). Esta técnica es especialmente relevante para idiomas morfológicamente complejos como el árabe o el polaco [11].

  5. ***Stemming***: Es una forma simplificada de la lematización, en la que se reduce las palabras a su raíz (*stem*), eliminando afijos [11]. Por ejemplo, la siguiente oración: *La estadística me parece fascinante*, tras aplicar *stemming*, quedaría como: *La estadíst me parec fascin*.

- **Eliminación de palabras vacías** (*stop words*): Se filtran del texto palabras sin valor analítico o con poco valor semántico. Normalmente son palabras que aparecen con mucha frecuencia como *el*, *la*, *de*, *que*, etc. Algunos métodos de ponderación como TF-IDF (ver Subsección 2.2.1) solucionan esta problemática de forma automatizada [11].

[2] El modelo FastText (2015) [13], divide las palabras en n-gramas de caracteres. Por ejemplo, usando 3-gramas, la palabra `azúcar` quedaría dividida en `az`, `azú`, `zúc`, `úca`, `car` y `ar`.

[3] No siempre es sencillo tokenizar el texto teniendo en cuenta palabras largas como nombres de países, organizaciones, o términos técnicos, como expone [7]. Considerando los ejemplos *Promover el bienestar del pueblo del Reino Unido*, y *La unión matrimonial de Isabel de Castilla con Fernando de Aragón creó un reino unido en España*, está claro que no siempre sería apropiado considerarlas una única palabra larga [7].

2.2. Representación

La representación es un paso previo necesario para cualquier modelo estadístico o de aprendizaje automático, que consiste en adaptar el texto a un formato numérico y estructurado. La representación es un paso clave en el análisis de texto, ya que determina cuánta información útil se puede extraer y utilizar para tareas posteriores [14].

La representación de texto se enfrenta al reto de intentar capturar de la forma más adecuada el significado semántico del texto. Para esta tarea, no solo es suficiente con representar cada palabra de forma individual, sino que también se debe considerar el contexto en el que aparece.

Muchas palabras tienen varios significados dependiendo del contexto (polisemia), pero otras también pueden ser modificadas por su entorno, por ejemplo, mediante adjetivos o adverbios que introducen matices. Un factor clave para valorar los métodos de representación es la medida en que son capaces de capturar el contexto. Por ejemplo, la palabra *ruido* suele asociarse a un sonido molesto o no deseado, mientras que la expresión *ruido blanco* es un concepto técnico que hace referencia a una señal aleatoria, un significado completamente diferente.

En esta sección se distinguen dos categorías de representación de texto: simbólica y distribuida. Las técnicas simbólicas son las más clásicas, aunque siguen siendo útiles para algunas tareas sencillas. En la Subsección 2.2.1 se describen los principios de esta representación, así como algunas técnicas que se basan en este modelo. En la Subsección 2.2.2 se aborda la representación distribuida, que representa el significado semántico del texto en espacios de menor dimensionalidad, ganando así mucha eficiencia, aunque sacrificando la interpretabilidad. Las representaciones distribuidas son superiores a las simbólicas en la mayoría de tareas de procesamiento y análisis de texto [14].

2.2.1. Representación simbólica (vectores dispersos)

La representación simbólica es una codificación del texto que utiliza caracteres, donde las palabras o frases se toman como símbolos (términos) distintos. Cada término de esta representación se corresponde directamente con un determinado concepto. Es una representación natural e intuitiva, que aún siendo simple, es muy efectiva para tareas como filtrado de spam, clasificación y recuperación de información [14].

El principal problema de la representación simbólica es que no es muy eficiente, por la alta dimensionalidad a la que se llega habitualmente. Cada término ocupa una dimensión propia, por lo que tenemos un espacio vectorial tan grande como el tamaño del vocabulario. Por ejemplo, para representar adecuadamente una colección modesta de documentos (15.000 noticias de Reuters), se necesitarían al menos 25.000 palabras [2].

Otro problema persistente de la representación simbólica es la *dispersión* o *escasez* (*sparsity*) de los datos. Debido a que las palabras siguen una distribución del tipo "*power law*", el funcionamiento de los modelos es muy pobre para las palabras muy poco comunes [14].

Las técnicas más utilizadas son las siguientes, que se utilizan según el caso para representar palabras individuales (one-hot), o textos más amplios (BoW, TF-IDF).

One-Hot Encoding

Cada palabra se representa como un vector de 0s, con un único 1. No es un método muy efectivo, ya que no es capaz de capturar el contexto de cada palabra, y tampoco es eficiente porque es una representación en muy altas dimensiones (una por palabra).

Por ejemplo, usando una representación one-hot (o cualquier otra simbólica), la diferencia entre *vidrio* y *ventana* es la misma que entre *vidrio* y *deporte*.

Bag of Words (BoW)

Se ignora el orden de los términos y se considera cada texto como un conjunto (bolsa) no ordenado de palabras. Representa cada texto como las frecuencias de aparición de cada palabra. Con este método se pierde el contexto de cada palabra, y tampoco se tiene en cuenta ni la sintaxis ni la gramática.

Por ejemplo, usando BoW, las oraciones *Pedro contrata a Juan*, y *Juan contrata a Pedro* son indistinguibles. La Tabla 2.1 muestra un ejemplo de una representación con BoW. Los textos son 4 obras de Shakespeare (columnas), en las que se ha contado el número de apariciones de 4 palabras (filas).

	As You Like It	***Twelfth Night***	***Julius Caesar***	***Henry V***
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Tabla 2.1: Matriz término-documento para 4 obras de Shakespeare con BoW [11]

TF-IDF (Term Frequency-Inverse Document Frequency)

Es una mejora de BoW, que pondera más a las palabras poco comunes en promedio, restándole importancia a palabras comunes que no ayudan a discriminar entre textos. Existen muchas variantes, pero la definición más común es la siguiente. Sea D el conjunto de documentos del corpus, $|d : d \in D|$ el número total de documentos, $f_{t,d}$ el número de veces que aparece el término t en el documento d, y $|\{d \in D : t \in d\}|$ el número total de documentos en los que aparece el término t. Entonces, la ponderación TF-IDF para un término t y un documento d se define como:

$$\text{TF-IDF}_{t,d} = \overbrace{f_{t,d}}^{\text{Term Frequency}} \cdot \overbrace{\log\left(\frac{|d : d \in D|}{|\{d \in D : t \in d\}|}\right)}^{\text{Inverse Document Frequency}}.$$

En la Tabla 2.2 se muestran las representaciones de las mismas 4 obras de Shakespeare con TF-IDF. Como se puede apreciar, la ponderación ha eliminado la importancia de la palabra *good*, y reducido mucho el peso de la palabra *fool*, por ser demasiado comunes.

	As You Like It	*Twelfth Night*	*Julius Caesar*	*Henry V*
battle	0.246	0	0.454	0.520
good	0	0	0	0
fool	0.030	0.033	0.0012	0.0019
wit	0.085	0.081	0.048	0.054

Tabla 2.2: Matriz término-documento para 4 obras de Shakespeare con TF-IDF [11]

2.2.2. Representación distribuida (vectores densos)

La representación distribuida se inspira en la cognición humana, representando cada concepto como un determinado patrón de activación de neuronas. De esta manera, el significado no se concentra en ningún lugar concreto, sino que está *distribuido* en muchas neuronas, que a su vez están relacionadas en la representación de otros conceptos similares [14].

De esta manera, la representación distribuida es capaz de capturar la similaridad entre conceptos y su significado semántico, representándolos como un patrón a lo largo de varias dimensiones [11, 14]. Los vectores que representan estos conceptos se denominan *embeddings*.

Estos *embeddings* se encuentran en un espacio vectorial denso en el que las distancias entre conceptos reflejan su similitud semántica. Es decir, el significado de un concepto no lo determina su posición absoluta, sino su cercanía relativa a otros conceptos. Esta representación es más eficiente, ya que utiliza una dimensionalidad menor que los métodos simbólicos. Además, es capaz de capturar mejor las relaciones semánticas entre los conceptos y soluciona de paso el problema de dispersión de los datos [14].

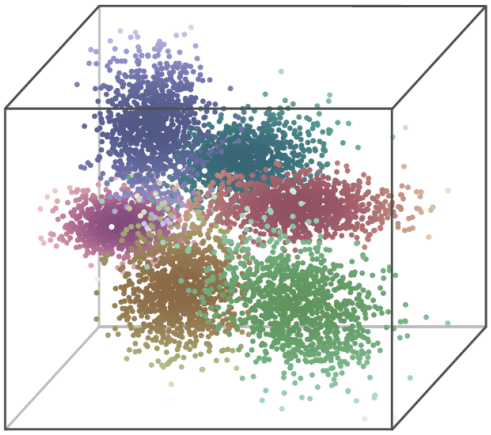


Figura 2.1: Ilustración de un espacio vectorial semántico [14]

Métodos basados en factorización de matrices

Aunque la idea de la representación distribuida se inspira en las redes neuronales, esta no es la única manera de "*repartir*" el significado semántico. Otras técnicas de reducción de dimensión también se han utilizado tradicionalmente para representar texto con vectores densos. Una de estas técnicas es *Latent Semantic Analysis (LSA, 1990)*, que utiliza descomposición en valores singulares [14, 15].

El LSA descompone una matriz término-documento (como las de las Tablas 2.1 y 2.2) aproximándola por una matriz de rango más bajo, es decir, comprimiendo la información de la matriz inicial a un espacio vectorial reducido. La matriz M se descompone en:

$$M = E\,\Sigma\,D^{\mathsf{T}}.$$

En esta descomposición, Σ es una matriz diagonal con k elementos, correspondientes al número de dimensiones seleccionado, y que se pueden interpretar como los temas latentes en el conjunto del texto, escalados y ordenados por importancia.

Por otra parte, la matriz E contiene los *embeddings* para todos los términos del vocabulario utilizado, que se pueden interpretar como la proyección de cada término en el espacio semántico de los temas latentes. Estos *embeddings* pueden servir para comparar los términos entre sí y, por ejemplo, identificar sinónimos.

Y por último, la matriz D representa a los documentos en el espacio semántico de los temas latentes. Esta matriz puede servir para comparar los documentos en un espacio dimensional más reducido, muy útil, por ejemplo, para tareas de clasificación o clústering. Otra utilidad es en el campo de la recuperación de información, convirtiendo preguntas (búsquedas) al espacio dimensional reducido, y encontrando documentos relacionados.

Una buena propiedad de LSA es que es capaz de reducir el ruido relacionado con la sinonimia y polisemia [16]. Sin embargo, y de acuerdo con [17], aunque el LSA es capaz de aprovechar de forma efectiva la información estadística del corpus, su rendimiento en tareas como las analogías de palabras es limitado, lo que sugiere que la estructura del espacio vectorial que genera es subóptima. Otros métodos también basados en factorización de matrices son los mencionados en [17], como *Hyperspace Analogue to Language (HAL, 1996)*, métodos *COALS (2006)* o *Hellinger PCA (HPCA, 2014)*.

La Figura 2.2, obtenida de [18], muestra una visualización de un espacio vectorial basado en temas explícitos, que son: *vehículo*, *movimiento* y *color*. La figura permite visualizar como se pueden representar palabras como *rápido* o *aeronave* como una superposición entre de temas, aunque los generados por LSA no se puedan asociar tan fácilmente con conceptos directos como los de la figura.

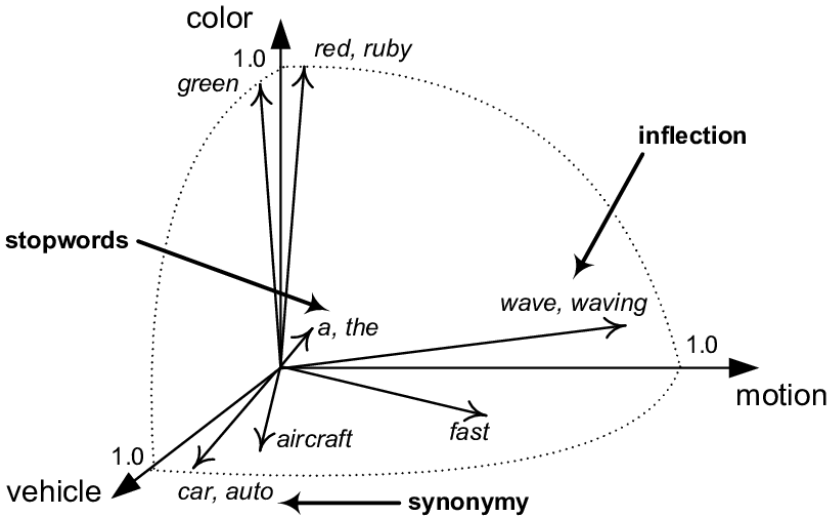


Figura 2.2: Visualización de varias palabras en un espacio vectorial basado en temas [18]

Embeddings estáticos

Los *embeddings* estáticos relacionan cada palabra con un vector único, independientemente del contexto en el que aparezcan. Modelos como *Word2Vec (2013)* [19], *GloVe (2014)* [17] o *FastText (2015)* [13] producen *embeddings* que pertecenen a esta categoría.

Como evidencian los resultados obtenidos en tareas de rendimiento como las analogías entre palabras, estos modelos son significativamente mejores a los simbólicos, y además, cada uno mejora progresivamente los resultados del anterior [19, 17, 13].

Word2Vec Word2Vec pertenece al grupo de modelos basados en ventanas de contexto local, que tratan de aprender el significado de las palabras observando su contexto. De esta familia, los modelos *skip-gram* y *continuous bag-of-words* (CBOW), introducidos en 2013, son quizás los más conocidos [11].

La intuición detrás de Word2Vec es entrenar un clasificador logístico para la tarea binaria: ¿Es probable que la palabra w_1 aparezca cerca de la palabra w_2? Aunque en realidad la tarea de predicción no nos importa. En su lugar, nos interesan parámetros de la tarea de clasificación, que tomamos como *embeddings*. La ventaja de este enfoque es que podemos utilizar texto como entrenamiento supervisado de este clasificador, y sin tener que utilizar redes neuronales [11].

GloVe *Global Vectors for Word Representation (GloVe, 2014)* es un modelo que se basa en utilizar información global del corpus. GloVe utiliza una matriz de coocurrencia (que cuenta cuántas veces aparece cada palabra junto a otras) para aprender el significado de las palabras utilizando los ratios de probabilidades de coocurrencia [17].

Por ejemplo, los autores de esta propuesta utilizan el ejemplo de la Tabla 2.3 para mostrar como se puede extraer significado de las probabilidades de coocurrencia. Tomando las palabras *hielo* y *vapor* como referencia, esperamos que palabras como *sólido* sean más probables de aparecer en el contexto de *hielo*, que de *vapor*, como queda reflejado en el ratio de sus probabilidades (8.9). De la misma manera pasa con *gas*, pero a la inversa (0.085). Para palabras que están relacionadas con ambas o con ninguna, como *agua*, o *moda*, el ratio de probabilidades debería ser cercano a 1.

Probabilidad y Ratio	k = sólido	k = gas	k = agua	k = moda
$P(k \mid \text{hielo})$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k \mid \text{vapor})$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k \mid \text{hielo})/P(k \mid \text{vapor})$	8.9	8.5×10^{-2}	1.36	0.96

Tabla 2.3: Probabilidad de coocurrencia para las palabras objetivo *hielo* y *vapor* [17]

Embeddings contextuales

En el lenguaje natural, el significado de las palabras normalmente depende del contexto, como es el caso de las palabras polisémicas. Los *embeddings* contextuales son capaces de representar adecuadamente estas palabras utilizando vectores diferentes dependiendo del contexto [11, 14].

Uno de los primeros modelos exitosos en generar *embeddings* contextuales fue ELMo (*Embeddings from Language Models*) [20], presentado en 2018. ELMo, al igual que los modelos basados en GPT *(OpenAI GPT, 2018)*, utilizan modelos de lenguaje unidireccionales en su fase de preentrenamiento [21], en el sentido de solo aprovechan el contexto anterior o posterior.

Para superar esta limitación, Google presentó en 2019 BERT (*Bidirectional Encoder Representations from Transformers*), que es capaz de aprovechar el contexto a ambos lados. BERT utiliza un modelo de lenguaje enmascarado, que oculta aleatoriamente términos del corpus para después intentar predecirlos usando el contexto a ambos lados [21].

Estos modelos han supuesto una revolución en la representación del lenguaje, consiguiendo los mejores resultados para la mayoría de tareas de rendimiento disponibles [14]. Todos estos modelos pertenecen a la familia de modelos de lenguaje preentrenados, que se ajustan posteriormente (*fine tuning*) para tareas concretas. La Figura 2.3 representa algunos de los modelos de esta familia y sus interrelaciones.

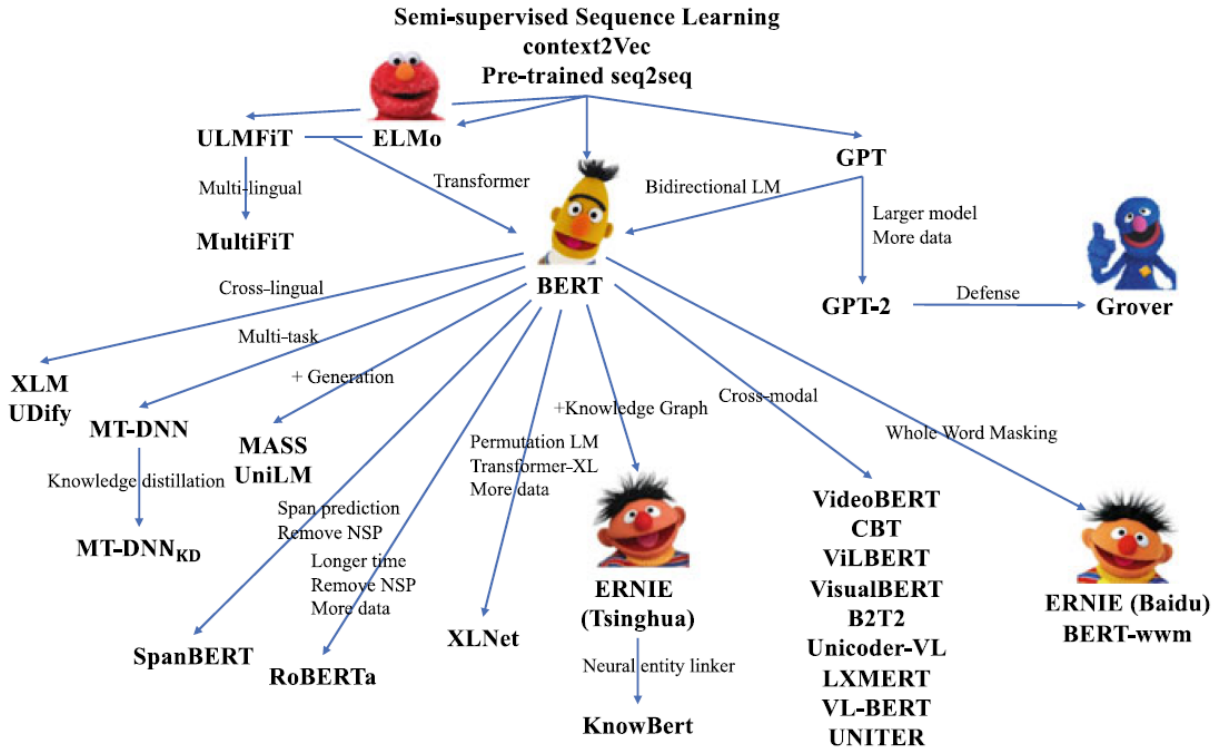


Figura 2.3: Familia de modelos de lenguaje preentrenados [14]

2.3. Distancias

La distancia es fundamental como medida de similitud o disimilitud. En el análisis de texto, la más utilizada es la distancia del coseno, aunque también son aplicables las otras medidas estándar de distancia entre vectores como la euclídea o manhattan.

Si no se ha representado numéricamente el texto, también se puede tener una noción de distancia utilizando otras medidas aplicables directamente a texto. Algunas medidas muy utilizadas son la distancia de edición y la de Jaccard (aplicable a conjuntos).

2.3.1. Distancia de edición

La distancia de edición cuantifica las diferencias entre dos cadenas de caracteres, calculando el número mínimo de operaciones de edición necesarias para transformar una cadena en la otra. Entre las operaciones de edición están: inserción, borrado y sustitución. También le podemos asignar diferentes costes a cada una de estas operaciones. Por ejemplo, la distancia Levenshtein le asigna a todas las operaciones el coste de 1 [11].

Otras variaciones de la distancia de edición consideran otro tipo de operaciones de edición, como por ejemplo, las distancias Hamming y Damerau–Levenshtein:

- **Hamming**: Solo permite las operaciones de sustitución, y por tanto solo permite comparar cadenas de igual longitud. Esta distancia es equivalente a contar el número de caracteres que son distintos [22].
- **Damerau–Levenshtein**: Añade a las operaciones permitidas la trasposición de caracteres adyacentes [22].

Por ejemplo, las distancias de edición para las palabras `algo` y `lago`, serían 2 para Levenshtein y para Hamming (sustituir la `a` por la `l`, y viceversa), pero 1 para Damerau–Levenshtein (intercambiar la `a` con la `l`).

2.3.2. Distancia de Jaccard

El índice de Jaccard es una medida bien conocida de similaridad entre dos conjuntos. Basado en este índice, se puede definir la distancia de Jaccard como su complementario [23]. Se puede aplicar esta distancia a textos utilizando el mismo razonamiento del modelo Bag of Words (BoW) (Subsección 2.2.1), que trata a los textos como un conjunto de palabras. La distancia se define de esta manera [23]:

$$\delta_{\mathcal{J}}(A, B) = 1 - \mathcal{J}(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}.$$

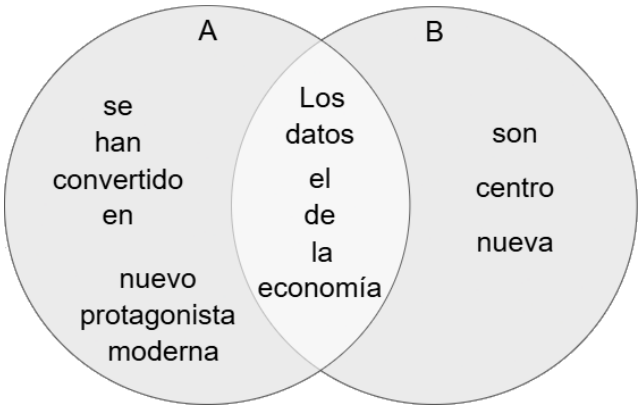
Donde $|A \cap B|$ es el número de palabras coincidentes entre ambos textos, y $|A \cup B|$ el número total de palabras únicas en ellos. Esta distancia toma valores entre 0 y 1, donde 0 indica máxima similitud (ambos textos tienen las mismas palabras), y 1 máxima disimilitud (no tienen ninguna palabra en común). A modo de ejemplo, se puede ilustrar este concepto utilizando las siguientes oraciones:

- *Los datos se han convertido en el nuevo protagonista de la economía moderna*
- *Los datos son el centro de la nueva economía*

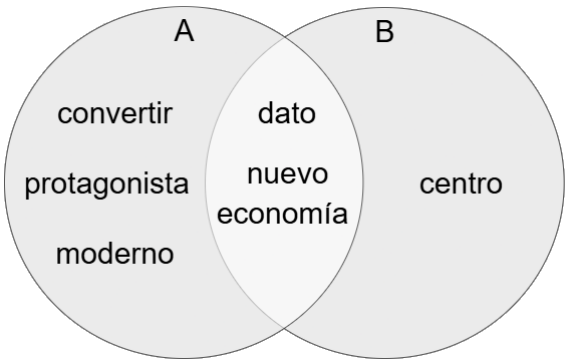
Opcionalmente, también se pueden preprocesar ambas oraciones. Por ejemplo, con lematización y eliminación de *stop words*, que permite que las palabras *nuevo* y *nueva* se consideren como coincidentes. El resultado del preprocesamiento sería el siguiente:

- *dato convertir nuevo protagonista economía moderno*
- *dato centro nuevo economía*

La distancia de Jaccard sin preprocesamiento para estas oraciones es de $0.625 = 1 - {}^{6}/_{16}$, y de $0.571 = 1 - {}^{3}/_{7}$ con preprocesamiento. Los conjuntos utilizados para este cálculo se muestran como diagramas de Venn en la Figura 2.4.



(a) Oraciones sin preprocesar



(b) Con lematización y eliminación de *stop words*

Figura 2.4: Conjuntos para calcular la distancia de Jaccard en dos oraciones de ejemplo

2.3.3. Distancia del coseno

La similaridad del coseno es la métrica más utilizada para medir la relación entre palabras y textos en análisis de texto [7, 11, 14]. De igual manera que para la distancia de Jaccard, se puede definir la distancia del coseno como el complementario de esta métrica. La distancia del coseno se define de la siguiente manera [11]:

$$\delta_{\cos}(w, v) = 1 - \frac{w \cdot v}{\|w\| \, \|v\|} = 1 - \frac{\sum_{i=1}^{n} w_i v_i}{\sqrt{\sum_{i=1}^{n} w_i^2} \cdot \sqrt{\sum_{i=1}^{n} v_i^2}}.$$

Esta métrica mide el ángulo que forman dos vectores v y w, que pueden representar documentos o tokens individuales, por ejemplo, con alguno de los métodos de la Sección 2.2. Una importante distinción es que esta distancia no tiene en cuenta la magnitud global de v y w, ya que se centra en las direcciones de los vectores, que es una mejor representación del significado semántico frente a su posición absoluta.

Del otro lado está la distancia euclídea, que también se puede usar como medida de similaridad, y que mide la distancia entre los puntos. Esta característica nos puede llevar a conclusiones indeseadas como en el ejemplo que muestra la Tabla 2.4.

	Frecuencia de los términos	
Documento	t_1	t_2
A	10	1
B	20	1
C	1	3

(a) Matriz término-documento para los documentos A, B y C y los términos t_1 y t_2

Documentos	Euclídea	Coseno
A – B	10,00	0,0012
A – C	9,22	0,5909
B – C	19,10	0,6368

(b) Distancias entre documentos

Tabla 2.4: Ejemplo para distancia coseno

En la Tabla 2.4 se puede apreciar como para la distancia euclídea, el documento A es más cercano a C que a B, mientras que para la distancia coseno los documentos A y B son muy cercanos entre sí y lejanos a C. Desde el punto de vista del significado de los documentos, es más razonable la segunda conclusión, ya que ambos parecen estar más relacionados con el término t_1 y no tanto con t_2.

2.4. Técnicas de análisis de texto

El contexto de las tareas de análisis de texto es amplio y depende en gran medida de la definición que se utilice del término. El análisis de texto, al igual que la minería de texto tiene como objetivo extraer información útil de los textos. No obstante, un matiz relevante es si se considera esta información como novedosa en algún sentido.

La frontera entre considerar una información como nueva también es ambigua. Para ilustrar esta diferencia, se puede considerar el siguiente ejemplo: detectar el fraude en el uso de tarjetas de crédito (información nueva), o extraer de currículums: nombres, direcciones y habilidades (información existente) [24].

En esta sección se describen las técnicas orientadas principalmente a la generación de información nueva a partir del descubrimiento de patrones, tendencias o atípicos usando texto. Estas serán: análisis de sentimiento, clasificación (supervisada y no supervisada), y modelado de temas (*topic modeling*).

Por otro lado, existen otras tareas que también son importantes, aunque quizás más en el contexto de procesamiento de lenguaje natural. Estas tareas no se centran propiamente en crear nueva información, sino en transformar de alguna manera la existente, facilitando su acceso o presentación. Algunas tareas de esta categoría son la recuperación de información, que trata de buscar información o documentos relevantes para una determinada consulta [3], la generación de resúmenes (*summarization*), cuyo objetivo es condensar el significado de grandes cantidades de texto [3], o el reconocimiento de entidades nombradas (*named entity recognition*), que trata de identificar y clasificar nombres propios en categorías como personas, organizaciones, ubicaciones, etc [3].

Análisis de Sentimiento El análisis de sentimiento (*sentiment analysis*), también conocido como minería de opinión (*opinion mining*), extrae información sobre las emociones u opiniones de un determinado texto. La forma más simple de análisis de sentimiento clasifica a los textos como positivos o negativos. También se puede extraer información sobre otras emociones como miedo, enfado, o alegría [25].

El análisis de sentimiento es muy utilizado en atención al cliente, para analizar las opiniones de los usuarios como reseñas o respuestas a encuestas [3], en los mercados financieros para identificar tendencias, anticipar volatilidad e incertidumbre, o predecir precios de acciones [26], o para la política, opinión pública y campañas electorales [27].

A esta técnica se le ha dedicado el Capítulo 3, donde se muestra su implementación en R.

Reseña	*Categoría*
Simplemente aburrido	Negativo
Totalmente predecible y sin energía	Negativo
Sin sorpresas y muy pocas risas	Negativo
Muy impactante	Positivo
La película mas divertida del verano	Positivo

Tabla 2.5: Análisis de Sentimiento aplicado a reseñas de películas [11]

Modelado de temas (*topic modeling*) El modelado de temas es una técnica no supervisada que intenta identificar los temas latentes en una colección de documentos [8]. El modelado de temas está muy relacionado con el problema de reducción de dimensiones, donde cada tema puede representar una dimensión conceptual [3], como ya se ha mostrado en la Subsección 2.2.2 con LSA y la Figura 2.2.

La Tabla 2.6 muestra a modo de ejemplo la aplicación de modelado de temas a las actas de las reuniones sobre las decisiones de los tipos de interés del Banco Central de Israel [6]. En la tabla se pueden ver algunas de las palabras que tienen más probabilidad de aparecer para cada uno de los temas, que se podrían interpretar de la siguiente manera:

- *Tema 1*: Cambios en la tasa de interés y objetivo de mantener la inflación baja.
- *Tema 2*: Datos de inflación y expectativas futuras.
- *Tema 3*: Discusión general de política monetaria.
- *Tema 4*: Mercado de la vivienda.

Tema 1	*Tema 2*	*Tema 3*	*Tema 4*
expectativa	aumento	interés	mes
continuo	descenso	tasa	aumento
tasa	continuo	estabilidad	tasa
inflación	tasa	israel	previsión
interés	expectativa	banco	banco
alcance	permanecer	inflación	indicador
israel	crecimiento	mercado	crecimiento

Tabla 2.6: Palabras asociadas con 4 temas extraídos de las actas del Banco Central de Israel [6]

Clústering Los métodos de clústering buscan crear grupos (clústers) de objetos similares de manera automática y no supervisada. Los algoritmos de clústering de texto más habituales los clasifican en categorías utilizando criterios de similaridad como los vistos en la Sección 2.3 [6, 16].

El clústering puede ser muy útil para organizar grandes cantidades de documentos de manera automática, facilitando la búsqueda y recuperación de información, o para facilitar la generación de resúmenes [16].

A esta técnica se le ha dedicado el Capítulo 4, donde se muestra su implementación en R.

Clasificación La clasificación de texto, al igual que el clústering, también busca separarlos de forma automática, con la diferencia que se hace de manera supervisada, es decir, entrenando un modelo de aprendizaje automático para esta tarea [28].

La clasificación necesita un conjunto de entrenamiento que haya sido etiquetado previamente con estas categorías, de forma que un modelo pueda aprender a predecirlas, detectando patrones subyacentes que ayuden a diferenciarlas.

La clasificación es aplicable para tareas como el filtrado de noticias, la organización de documentos, el análisis de sentimiento, la clasificación de correos electrónicos, el filtrado de spam y el diagnóstico médico [28].

A esta técnica se le ha dedicado el Capítulo 5, donde se muestra su implementación en R.

2.5. Visualización

La visualización es una herramienta fundamental para resumir y comunicar ideas de forma efectiva, que ayuda tanto durante el proceso de análisis, como para la comunicación posterior de los resultados obtenidos. Una buena visualización es capaz de comunicar grandes cantidades de información con una baja carga cognitiva, facilitando la identificación de patrones, temas relevantes y relaciones ocultas [29].

La visualización también es clave en la comunicación de resultados, facilitando la comprensión de los mismos tanto para expertos como audiencias no técnicas. Algo especialmente importante teniendo en cuenta que la comunicación visual es mucho más efectiva que la basada en texto, como bien refleja el dicho popular "*una imagen vale más que mil palabras*".

La visualización de texto es un campo de creciente importancia dentro de la visualización de información, y normalmente se centra en representar datos textuales crudos o también en los resultados de las técnicas de análisis de texto [30].

En esta sección se describen 3 tipos de visualizaciones que son especialmente útiles para tratar con datos de texto, en concreto:

- **Nubes de palabras**: Visualizan las palabras más frecuentes de un corpus, donde el tamaño de cada palabra indica su importancia.
- **Redes de coocurrencia**: Muestra las relaciones entre diferentes términos, lo que permite explorar el contexto de ciertas palabras o conceptos.
- **Diagramas de frecuencia**: Permite comparar la frecuencia de aparición de términos o etiquetas de clasificación, comparando su distribución entre textos o representando su evolución a lo largo de un texto.

2.5.1. Nubes de palabras

Una nube de palabras es una imagen que nos permite representar una lista de términos con distintos pesos relativos. Aplicado a un corpus, se pueden representar las palabras según su frecuencia de aparición, o incluso a un clúster (o tema latente) para representar su distribución subyacente. En R se puede usar el paquete `wordcloud`[4] para crear nubes de palabras.

A modo de ejemplo, se ha utilizado precisamente este paquete para elaborar las nubes de palabras de la Figura 2.5. Como texto se ha utilizado la Ley de la Función Estadística Pública (LFEP)[5], convenientemente preprocesada.

La Figura 2.5a representa a toda el texto de la ley, mientras que las otras dos nubes representan la distribución de palabras del segundo y tercer temas latentes extraídos con LSA. Se ha omitido el primer tema latente por ser muy parecido a la distribución de palabras de toda la ley.

[4] https://CRAN.R-project.org/package=wordcloud
[5] https://www.boe.es/buscar/doc.php?id=BOE-A-1989-10767

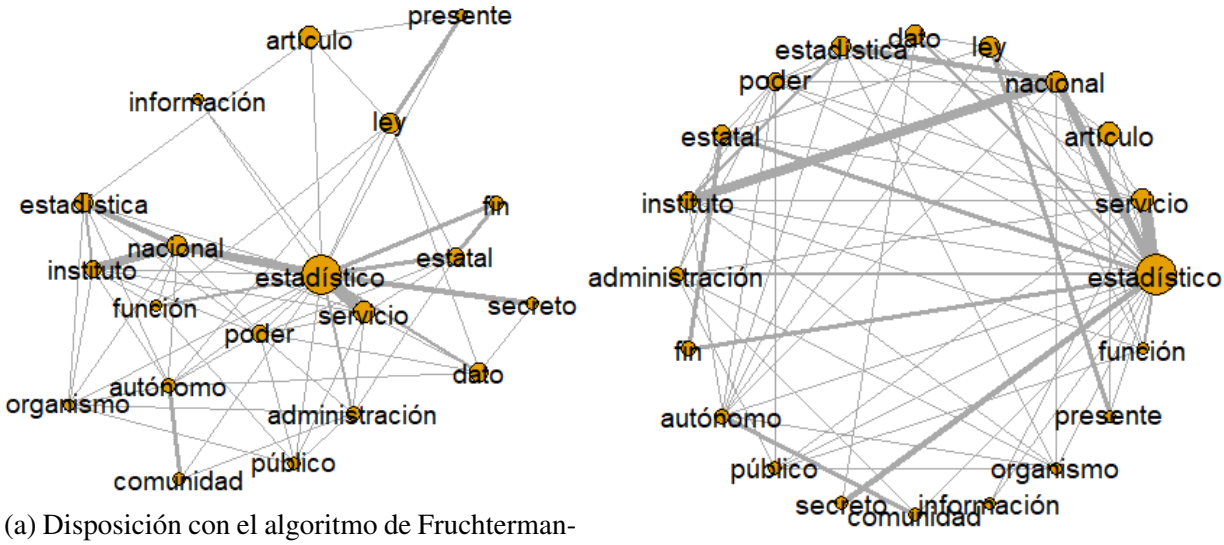
(a) Conjunto de la ley (b) Segundo tema latente (LSA) (c) Tercer tema latente (LSA)

Figura 2.5: Nubes de palabras de la Ley de la Función Estadística Pública. Para aplicar LSA se ha considerado cada párrafo como un documento. Se visualizan las 60 palabras más frecuentes

2.5.2. Redes de coocurrencia

Las redes de coocurrencia son muy utilizadas para representar las relaciones entre distintas entidades. Las redes se representan como un grafo, donde los nodos normalmente son palabras, y los ejes reflejan las relaciones de contexto entre ellas. Opcionalmente, el tamaño de los nodos puede representar la importancia relativa de las palabras, y el grosor de los ejes una relación más fuerte o débil de coocurrencia [8].

A modo de ejemplo, la Figura 2.6 representa la red de coocurrencia de las 20 palabras más frecuentes de la LFEP, que muestra las relaciones de contexto entre ellas. Los gráficos se han elaborado usando el paquete `text2vec`[6] para calcular la matriz de coocurrencia y el paquete `igraph`[7] para representar el grafo.



(a) Disposición con el algoritmo de Fruchterman-Reingold

(b) Disposición circular

Figura 2.6: Red de coocurrencia de las 20 palabras más frecuentes de la LFEP

[6] https://text2vec.org/index.html
[7] https://r.igraph.org/

2.5.3. Diagramas de frecuencia

Los diagramas de frecuencia describen la distribución de diferentes términos, analizando su aparición a lo largo de uno o varios documentos. También podemos representar la frecuencia de aparición de los términos más comunes en uno o en varios documentos. Los diagramas de frecuencia se pueden representar como un diagrama de barras, como el de la Figura 2.7, o como un gráfico de líneas, como el de la Figura 2.8.

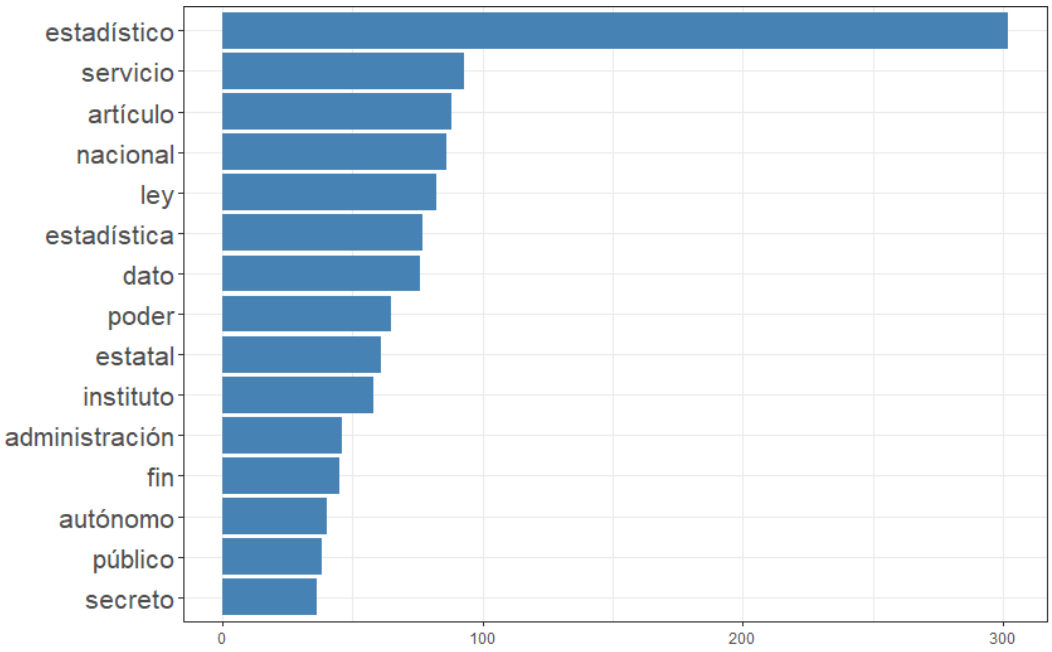


Figura 2.7: Frecuencias de aparición de las 15 palabras más comunes en la LFEP

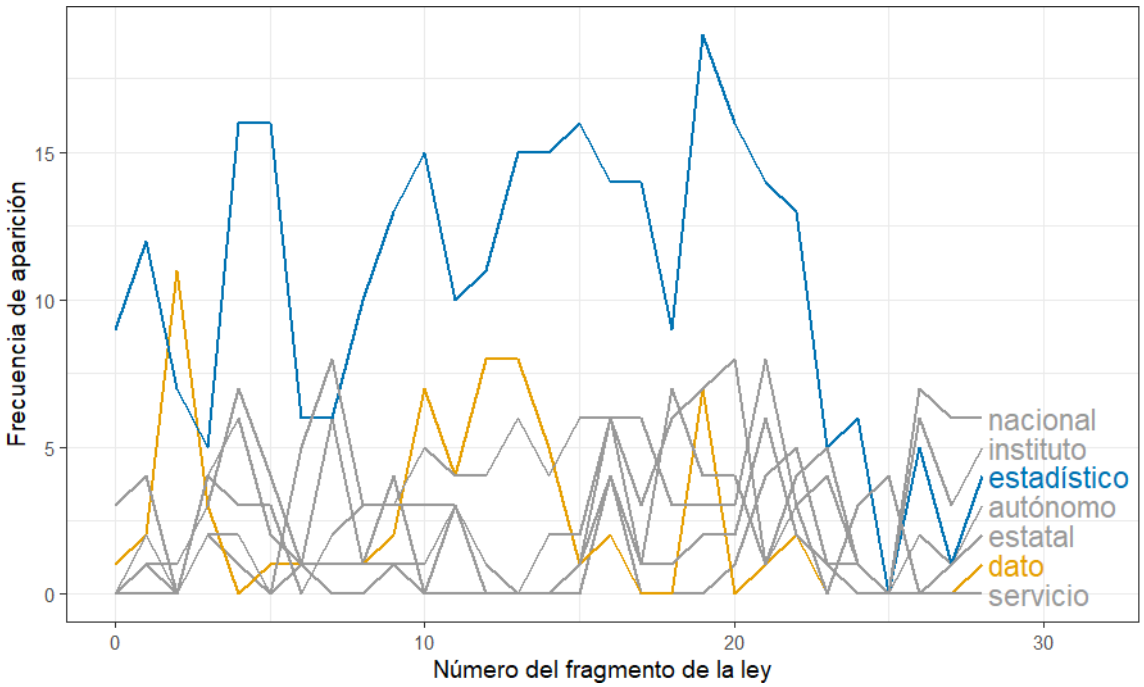


Figura 2.8: Frecuencias de aparición de 7 palabras a lo largo de la LFEP, dividida en 29 fragmentos de 200 palabras

Capítulo 3

Análisis de Sentimiento

Este capítulo aborda la aplicación de la técnica de Análisis de Sentimiento (*Sentiment Analysis*) en R. Para ello, se utilizarán las capacidades y paquetes que ofrece este lenguaje, utilizando una familia de métodos basados en diccionarios anotados con el valor sentimental de las palabras. Además, cabe señalar que el Análisis de Sentimiento también se puede plantear como un problema de clasificación, perspectiva que se desarrollará en el Capítulo 5.

La organización del capítulo es la siguiente: la Sección 3.1 ofrece una breve definición del Análisis de Sentimiento y proporciona el contexto necesario para comprender los métodos que se aplican en este capítulo, así como otras alternativas que quedan fuera de su alcance. Además, se presentan los principales paquetes disponibles en R para llevar a cabo este tipo de análisis. La Sección 3.2 detalla la metodología utilizada, la Sección 3.3 describe y visualiza el conjunto de datos utilizado para en análisis, y la Sección 3.4 expone los resultados obtenidos.

Adicionalmente, todo el código desarrollado para el análisis de este capítulo se puede encontrar en el Apéndice A, además de a través del siguiente enlace: https://github.com/acabrerod/text-analysis/tree/main/sentiment-analysis, lo cual permite reproducir el análisis desarrollado en este capítulo.

3.1. Introducción

El Análisis de Sentimiento es el estudio computacional de determinados textos con el objetivo de extraer opiniones, emociones o actitudes con respecto a algún tema, individuo o entidad [27].

Para cuantificar estas valoraciones, este análisis se basa en la polaridad, una medida que determina la orientación sentimental de un texto, clasificándolo como *positivo*, *negativo* o *neutro*. La polaridad también se puede representar en una escala numérica, donde valores negativos indican polaridad negativa, valores cercanos a 0 polaridad neutra, y valores positivos polaridad positiva. La magnitud de la polaridad expresa la intensidad del sentimiento.

El Análisis de Sentimiento se puede entender como un problema de clasificación, en el que se determina la polaridad o sentimientos con diferentes niveles de detalle. Por ejemplo, analizando en su conjunto documentos u oraciones, o buscando la polaridad con respecto a entidades concretas [27].

3.1.1. Métodos para Análisis de Sentimiento

Según [26], hay dos grandes familias de métodos para abordar el Análisis de Sentimiento: los basados en aprendizaje automático y los basados en léxicos, como muestra la Figura 3.1.

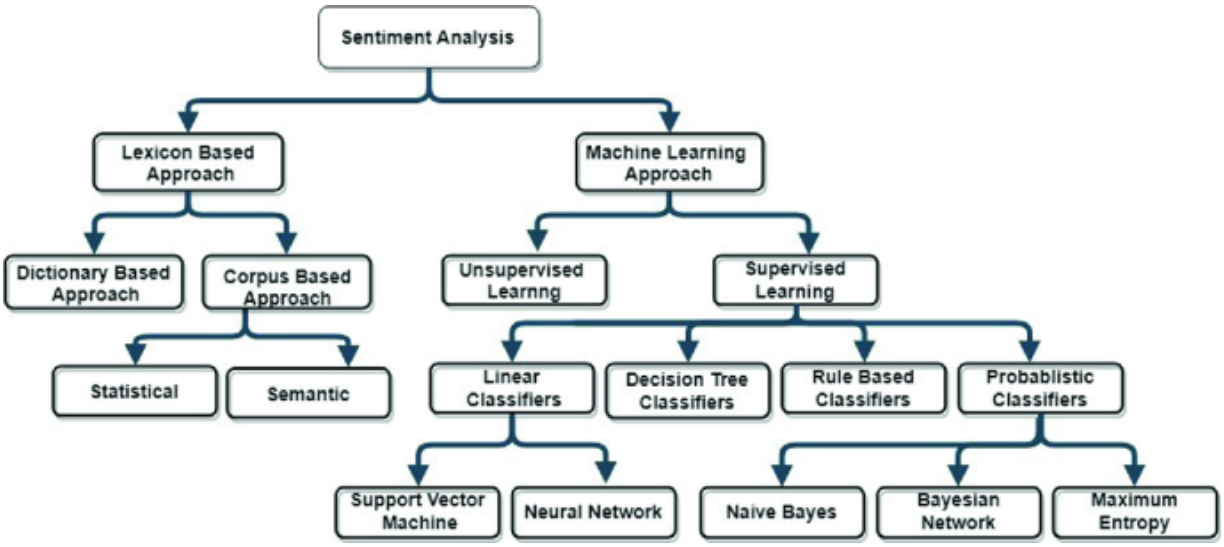


Figura 3.1: Técnicas para afrontar el Análisis de Sentimiento [31]

Para poder aplicar alguno de estos métodos, se debe tener en cuenta lo siguiente:

- **Aprendizaje supervisado**: Se pueden usar cuando la polaridad se conoce de antemano, ya sea porque se mencione en los propios datos (por ejemplo, estrellas en una reseña), o porque se clasifiquen de forma manual.
- **Aprendizaje no supervisado**: Se pueden usar cuando no se dispone de unos datos etiquetados, o si las etiquetas no son confiables (por ejemplo, los números de "*me gusta"* vs "*no me gusta"* en una red social).
- **Basado en léxicos**: Esta es la forma más sencilla y común de afrontar el Análisis de Sentimiento, y es no supervisada. Se basa en componer de diferentes maneras los sentimientos que expresan cada una de las palabras, recurriendo a diccionarios anotados donde se especifican las emociones o polaridad de cada palabra. Este tipo de métodos se describen con mayor detalle en la Subsección 3.1.2, dado que son los utilizados en este capítulo.

3.1.2. Métodos basados en léxicos

Los métodos basados en léxicos son los más utilizados por su simplicidad y rapidez. Estos métodos utilizan una lista de palabras (diccionarios) que expresan emociones, y las cuantifican. Por ejemplo, uno de estos diccionarios (AFINN), cuantifica así la polaridad sentimental de las siguientes palabras:

- **Negativas**: *fraude* (-4), *catástrofe* (-4), *soledad* (-2), *dolor* (-2), *ataque* (-1), *disculpa* (-1).
- **Positivas**: *divertido* (+4), *milagro* (+4), *humor* (+2), *logro* (+2), *sueño* (+1), *perdonar* (+1).

El principal inconveniente de los métodos basados en léxicos es que no tienen en cuenta el contexto específico de las palabras. Por ejemplo, *impuesto*, *gasto* o *deuda* tienen connotaciones negativas en general, pero serían neutras en un texto económico o financiero. Por este motivo es importante elegir un diccionario apropiado para el texto que se analice [6].

La forma más simple de calcular la polaridad de un texto con estos métodos es aplicar Bag of Words (BoW), es decir, contar la frecuencia de cada palabra, multiplicarla por su polaridad y sumar los resultados. Este enfoque, sin embargo, ignora por completo la estructura gramatical, que es esencial para la construcción del significado en cualquier lenguaje.

Un ejemplo muy evidente de esta limitación se observa en el caso de los adverbios para cambiar de sentido (no, nunca, tampoco, etc), intensificar (muy, mucho, demasiado, etc) o atenuar (poco, algo, menos, etc) el significado de una oración. Estas diferencias son relevantes, por ejemplo, en el caso de analizar textos financieros, como se justifica en [26]. Es común que los autores de textos financieros limiten su uso de palabras negativas, reescribiendo los contenidos negativos usando palabras más neutras [26].

Algunos de los diccionarios anotados más relevantes son los siguientes:

- *General Inquirer (1966)*, específicamente, el diccionario Harvard-IV-4: Clasifica palabras en inglés en diferentes categorías, entre las que se encuentran positivo/negativo[1].
- *Bing Liu (2004)*: Lista de palabras en inglés clasificadas como positivas y negativas[2].
- *SentiWordNet (2006), SentiWordNet 3.0 (2010)*: Basados en la base de datos léxica inglesa WordNet, le asignan a cada conjunto de sinónimos tres puntuaciones numéricas en base a su *positividad*, *negatividad* y *neutralidad*[3].
- *AFINN (2011)*: Lista de palabras en inglés puntuadas en una escala de -5 a $+5$[4].
- *NRC Word-Emotion Association Dictionary (2013)*: Lista de palabras en inglés y sus asociaciones con 8 emociones básicas y su polaridad[5].

Algunos de estos diccionarios también se han traducido a otras lenguas como es el caso de NRC, que está disponible en más de 100 idiomas. También hay diccionarios adaptados a otros contextos específicos, como ya hemos mencionado para el caso de las finanzas, uno muy conocido de este campo es el de Loughran-McDonald (2011)[6].

3.1.3. Métodos de preprocesamiento

Para el Análisis de Sentimiento pueden aplicarse las técnicas de preprocesamiento descritas en la Sección 2.1. La elección de unas u otras dependerá tanto del objetivo como del método utilizado. En particular, para los métodos basados en léxicos es especialmente importante la normalización, ya que permite localizar las palabras en los diccionarios. Por otro lado, en métodos como Bag of Words (BoW), donde se ignora el orden de las palabras, podrán eliminarse

[1] General Inquirer: https://inquirer.sites.fas.harvard.edu/.
[2] Bing Liu: https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html.
[3] SentiWordNet: https://github.com/aesuli/SentiWordNet.
[4] AFINN: https://darenr.github.io/afinn/.
[5] NRC: https://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm.
[6] Loughran-McDonald: https://sraf.nd.edu/loughranmcdonald-master-dictionary/.

también las *stop words*, ya que solo serán relevantes las palabras que expresan sentimientos. Sin embargo, las listas de *stop words* normalmente incluyen palabras como *no* o *nunca*, por lo que si se quieren tener en cuenta no se podrán eliminar.

3.1.4. Paquetes de R

Para aplicar Análisis de Sentimiento en R se pueden usar los siguientes paquetes, todos basados en métodos léxicos. El paquete `sentimentr` es el único que tiene algo en cuenta el contexto, procesando algunos adverbios y conjunciones para detectar cambios de polaridad.

1. `syuzhet` (2015): Paquete para Análisis de Sentimiento. Usa un diccionario propio, junto a AFINN, Bing y NRC[7].

2. `tidytext` (2016): Paquete para minería y análisis de texto desde el enfoque del tidyverse. Permite aplicar Análisis de Sentimiento y está integrado con diccionarios como AFINN, Bing, y NRC[8].

3. `sentimentr` (2018): Paquete para Análisis de Sentimiento que mejora el soporte a negaciones y otros modificadores de la polaridad[9].

4. `SentimentAnalysis` (2021): Paquete para Análisis de Sentimiento que contiene los diccionarios General Inquirer y QDAP (del paquete `qdap` de R), y dos específicos para finanzas: Henry y Loughran-McDonald. También permite generar diccionarios propios y compararlos con los estándar[10].

3.2. Metodología

El objetivo de este capítulo es aplicar los paquetes descritos en la Subsección 3.1.4 para ilustrar su funcionamiento, sus capacidades y los resultados que proporcionan en tareas de Análisis de Sentimiento. Para ello, se empleará el conjunto de datos *IMDb Movie Reviews*, que se presentará en la Sección 3.3, concretamente se utilizarán las 25.000 reseñas del conjunto de *test*. Esta elección permitirá comparar los resultados con los análisis de los Capítulos 4 y 5, que se validarán con esas mismas reseñas.

Se utilizarán todos los diccionarios disponibles para cada uno de los paquetes, con los que se aplicará Análisis de Ssentimiento para obtener la polaridad de las reseñas. A partir de esta puntuación, las reseñas se clasificarán como positivas si la polaridad es mayor que 0, y como negativas en caso contrario. También se clasificarán como negativas las reseñas neutras, que obtengan una puntuación exactamente igual a 0, que podría suceder en el caso de que la polaridad de sus palabras se contrarreste, o si ninguna de sus palabras expresa sentimientos ni positivos ni negativos.

[7] `syuzhet`: https://github.com/mjockers/syuzhet, https://CRAN.R-project.org/package=syuzhet.
[8] `tidytext`: https://www.tidytextmining.com, https://CRAN.R-project.org/package=tidytext.
[9] `sentimentr`: https://github.com/trinker/sentimentr, https://CRAN.R-project.org/package=sentimentr.
[10] `SentimentAnalysis`: https://CRAN.R-project.org/package=SentimentAnalysis.

Una vez realizada esta clasificación binaria, el rendimiento de cada método se evaluará utilizando las métricas de exactitud y *F1-Score*, tratándolos como algoritmos de clasificación. En la Subsección 3.2.1 se definen estas métricas.

El preprocesamiento de las reseñas se llevará a cabo con los paquetes `tm`, `textstem` y `textclean`, e incluirá los siguientes pasos: (1) conversión a minúsculas mediante la función `tolower`, (2) expansión de contracciones con la función `replace_contraction`, y (3) lematización con la función `lemmatize_strings`.

3.2.1. Validación

Para validar el rendimiento de cada uno de los métodos de Análisis de Sentimiento se utilizarán las métricas de exactitud y *F1-Score*. Ambas se pueden derivar de la matriz de confusión y que se definen se la siguiente manera:

- **Matriz de confusión**: Representa en forma de tabla los resultados de una clasificación, donde las filas representan las categorías reales y las columnas las categorías predichas. De esta forma, cada celda representa una de las posibles combinaciones de los resultados de la clasificación. Por ejemplo, la Tabla 3.1 es una matriz de confusión para 2 categorías.

		Categoría predicha	
		Positivo	**Negativo**
Categoría real	**Positivo**	Verdadero positivo (VP)	Falso negativo (FN)
	Negativo	Falso positivo (FP)	Verdadero negativo (VN)

Tabla 3.1: Matriz de confusión con dos categorías

- **Exactitud**: Mide la proporción de aciertos sobre el total de observaciones.

$$\textit{Exactitud} = \frac{VP + VN}{VP + VN + FP + FN}$$

- ***F1-Score***: Promedio harmónico entre las métricas de precisión y de sensibilidad.
  - **Precisión** (precisión positiva, VPP): También conocido como valor predictivo positivo, mide la proporción de observaciones identificadas como positivas que realmente son positivas.
  - **Sensibilidad** (*recall*, TPR): Mide la proporción de observaciones realmente positivas que han sido identificadas correctamente.

$$\textit{Precisión} = \frac{VP}{VP + FP} \qquad \textit{Sensibilidad} = \frac{VP}{VP + FN}$$

$$\textit{F1-Score} = 2 \cdot \frac{\textit{Precisión} \cdot \textit{Sensibilidad}}{\textit{Precisión} + \textit{Sensibilidad}}$$

3.3. Conjunto de datos *IMDb Movie Reviews*

Esta sección describe el conjunto de datos que será empleado a lo largo de este trabajo, no solo para el análisis de este capítulo, sino también para los Capítulos 4 y 5. El conjunto de datos utilizado, el *IMDb Movie Reviews dataset*, constituye una referencia ampliamente utilizada en la literatura académica para evaluar el rendimiento de métodos de procesamiento de texto en tareas como Análisis de Sentimiento y Clasificación. En la Subsección 3.3.1 se describe brevemente el conjunto de datos, y en la Subsección 3.3.2 se realiza una exploración preliminar del mismo, mediante algunos gráficos y ejemplos.

3.3.1. Descripción

El *IMDb Movie Reviews dataset* es un conjunto de datos compuesto por 50.000 reseñas de películas en inglés, extraídas de la página web *Internet Movie Database (IMDb)*. Este recurso fue recopilado y puesto a disposición pública en [32], de donde se ha obtenido.

Todas las reseñas han sido clasificadas automáticamente como positivas y negativas, a partir sus puntuaciones en la web IMDb. Las reseñas con una puntuación menor o igual que 4 se han clasificado como negativas, mientras que las reseñas con una nota mayor o igual que 7 se han clasificado como positivas. Las reseñas con puntuaciones intermedias no se incluyen en el conjunto de datos, con el objetivo de excluir las ambiguas.

El conjunto de datos contiene un máximo de hasta 30 reseñas por cada película, y está equilibrado a partes iguales entre reseñas positivas y negativas (25.000 de cada tipo). Además, está dividido en dos subconjuntos para entrenamiento y prueba, con un 50 % de las reseñas cada uno.

3.3.2. Exploración

Al analizar la longitud de las reseñas, se puede apreciar que son en general bastante largas, siendo su longitud promedio de 231 palabras y su mediana de 171 palabras. La distribución de estas longitudes se muestra en la Figura 3.2, en forma de un histograma. La figura se ha cortado a partir de las 1.000 palabras, por lo que se ocultan las 79 reseñas cuya longitud se encuentra entre las 1.000 y 2.470 palabras (el máximo). Las longitudes extensas favorecen la clasificación, ya que será raro que no contengan ninguna palabra con valor sentimental.

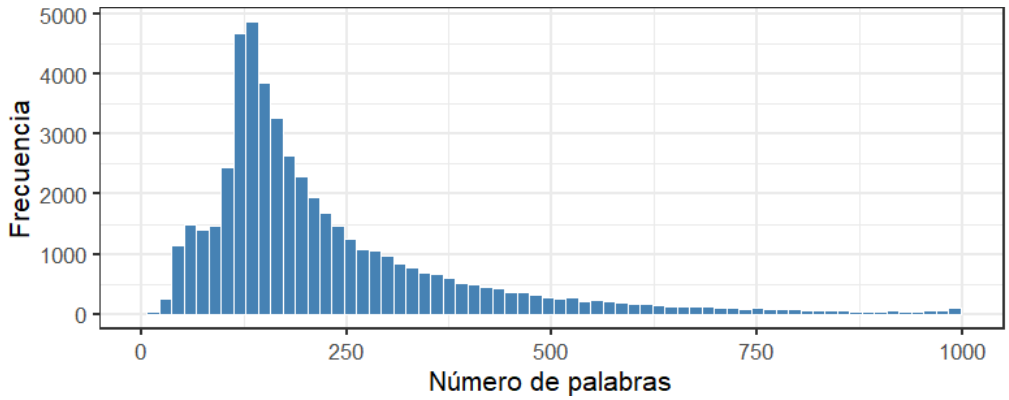


Figura 3.2: Histograma de la longitud de las reseñas del *IMDb dataset* en número de palabras

A continuación se muestran dos figuras que permiten visualizar el conjunto de datos, y algunos ejemplos de reseñas breves.

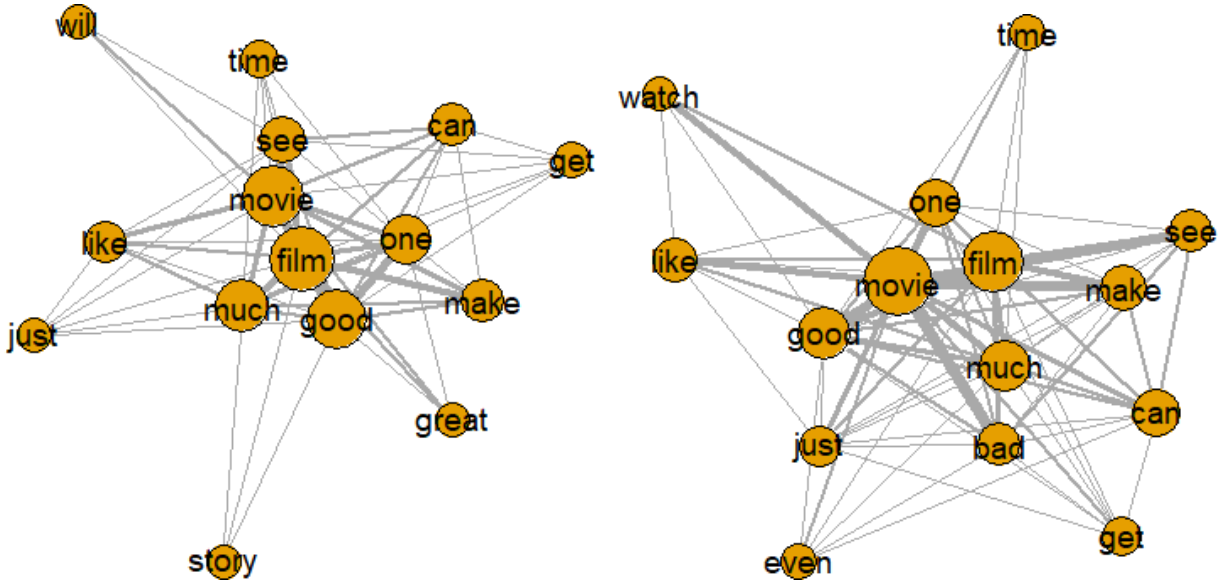
Visualización

Las Figuras 3.3 y 3.4 ofrecen una representación visual del *IMDb dataset*, diferenciando entre reseñas positivas y negativas. Ambas figuras permiten identificar las palabras y patrones léxicos más comunes de cada grupo. Para su elaboración, se han utilizado las 50.000 las reseñas, las cuales se han preprocesado con conversión a minúsculas, expansión de contracciones, lematización y eliminación de *stop words*.

(a) Reseñas positivas

(b) Reseñas negativas

Figura 3.3: Nubes de palabras para las 60 más frecuentes en el *IMDb dataset*



(a) Reseñas positivas

(b) Reseñas negativas

Figura 3.4: Redes de coocurrencia de las 15 palabras más frecuentes en el *IMDb dataset*

Ejemplos

La Tabla 3.2 muestra 6 reseñas breves a modo de ejemplo. Como evidencian las de la última fila, no todas serán sencillas de clasificar en los análisis. Por ejemplo, la tercera reseña positiva no contiene palabras con valor sentimental positivo, y sin embargo, puede entenderse que es positiva. Por otro lado, la negativa desafía a los modelos que no consideren el orden de las palabras, que al incluir términos como *divertida* e *interesante*, la clasificarán erróneamente como positiva.

Reseñas positivas	**Reseñas negativas**
Creo que es una de las mejores películas jamás hechas, y he visto muchas... El libro es mejor, pero sigue siendo una muy buena película.	*Los personajes son antipáticos y el guion es horrible. Es un desperdicio del talento de Deneuve y Auteuil.*
Una de mis escenas favoritas es al principio, cuando los invitados en un yate privado deciden darse un baño improvisado... ¡en ropa interior! ¡Bastante atrevido para 1931!	*Mala película. Es demasiado complicada para los niños pequeños y demasiado infantil para los adultos. Solo la vi porque soy fan de Robin Williams y me decepcionó mucho.*
La primera vez que ves The Second Renaissance puede parecerte aburrida. Vela al menos un par de veces y, sobre todo, no te pierdas la segunda parte. Te cambiará la forma de ver Matrix. ¿Fueron los humanos quienes empezaron la guerra? ¿Es la IA algo malo?	*No es divertida, no es interesante, no está bien filmada, no te importan los personajes, ni uno solo de ellos. No hay nada que te atrape en el desarrollo de la narrativa, realmente te da igual lo que ocurra. Una gran, gran pérdida de tiempo y talento.*

Tabla 3.2: Seis reseñas de ejemplo del *IMDb dataset* traducidas al castellano

3.4. Resultados

Los resultados obtenidos para los diferentes métodos de Análisis de Sentimiento se recogen en las Tablas 3.3 y 3.5. En ambas se ha resaltado en negrita la combinación que ofrece el mejor rendimiento en cada métrica. Tanto para la exactitud como para *F1-Score*, la mejor puntuación se alcanza utilizando el paquete `sentimentr` junto con el diccionario de Bing Liu.

Es importante destacar que, dado que las reseñas positivas y negativas están distribuidas por igual, una clasificación al azar establece un límite inferior para la exactitud en el 50 %. El mejor método probado alcanza un 75,83 %, en comparación con el estado del arte, que es capaz de obtener un 96,80 % mediante XLNet (2019) [33], o un 88,89 % de los propios autores del conjunto de datos [32]. La matriz de confusión del mejor método probado se presenta en la Tabla 3.6.

También se puede apreciar que los resultados obtenidos con el paquete sentimentr son superiores a syuzhet y tidytext a igualdad de diccionario, aunque con un significativo aumento del coste computacional. En particular, para analizar las 25.000 reseñas con el equipo utilizado para los análisis, sentimentr requirió del orden de 100 segundos, mientras que syuzhet y tidytext necesitaron del orden de 10 y 1 segundos, respectivamente.

La Tabla 3.4 muestra a mayores algunas reseñas de hasta 60 palabras que han sido mal clasificadas por el mejor método (sentimentr con Bing), y que han obtenido las polaridades contrarias más altas (los falsos negativos con menor polaridad y los falsos positivos con mayor polaridad).

Como se puede apreciar, la errores en estos ejemplos se deben a comentarios positivos o negativos que no hacen referencia a la película en sí. El sarcasmo también perjudica el análisis como se puede ver en la primera reseña de los falsos positivos.

Diccionario \ Paquete	**syuzhet**	**tidytext**	**sentimentr**	**SentimentAnalysis**
General Inquirer	-	-	-	64,10 %
Bing Liu	72,79 %	73,13 %	**75,83 %**	-
AFINN	71,03 %	69,04 %	-	-
NRC	65,67 %	65,84 %	69,25 %	-
Syuzhet	68,87 %	-	72,67 %	-
QDAP	-	-	-	66,26 %
SentiWordNet 3.0	-	-	68,53 %	-
Henry (Financial)	-	-	-	57,18 %
Loughran-McDonald	-	-	68,55 %	65,14 %

Tabla 3.3: Resultados del análisis: métrica exactitud

Polaridad	**Falsos positivos**
1,73	*¡Oh, vaya! ¡Solo fue un sueño! ¡Qué gran idea! [...] Echa un vistazo a [...] para ver una mezcla mucho más inteligente de fantasía y realidad.*
1,13	*La trama parecía interesante, pero esta película es una gran decepción. [...] Si existieran las películas de categoría C, esta sería un ejemplo perfecto. Un punto a favor por una carátula de DVD bonita y [...]*
0,98	*Esta película fue increíblemente aburrida, Michael J. Fox podría haberlo hecho mucho mejor. Lo siento, pero es la verdad para todos los que os gustó la película.*

(a) Ejemplos de falsos positivos

Polaridad	**Falsos negativos**
-1,21	*Esta es la película más difícil que he visto en mi vida... el contenido emocional es horrible, pero inolvidable. [...] Alan Rickman y Madeleine Stowe [...] merecen un Oscar por sus actuaciones.*
-0,48	*Una película extremadamente poderosa que ciertamente no está lo suficientemente valorada. [...] La reciente adaptación para la televisión británica fue vergonzosa: demasiado ordinaria y sosa. [...]*
-0,22	*Es una gran película. Qué lástima que no esté disponible en vídeo doméstico.*

(b) Ejemplos de falsos negativos

Tabla 3.4: Ejemplos de reseñas de hasta 60 palabras mal clasificadas aplicando Análisis de Sentimiento con el paquete `sentimentr` y el diccionario Bing

Diccionario \ Paquete	**syuzhet**	**tidytext**	**sentimentr**	**SentimentAnalysis**
General Inquirer	-	-	-	60,83 %
Bing Liu	74,92 %	73,89 %	**75,54 %**	-
AFINN	66,83 %	62,62 %	-	-
NRC	58,40 %	58,18 %	63,45 %	-
Syuzhet	63,56 %	-	68,57 %	-
QDAP	-	-	-	57,62 %
SentiWordNet 3.0	-	-	57,62 %	-
Henry (Financial)	-	-	-	43,32 %
Loughran-McDonald	-	-	68,84 %	70,28 %

Tabla 3.5: Resultados del análisis: métrica *F1-Score*

Real \ Predicho	**Positivo**	**Negativo**	**Suma**
Positivo	9.331	3.169	12.500
Negativo	2.874	9.626	12.500
Suma	12.205	12.795	25.000

Tabla 3.6: Matriz de confusión obtenida sobre *IMDb dataset* (train) aplicando Análisis de Sentimiento con el paquete `sentimentr` y el diccionario Bing

Capítulo 4

Clústering

Este capítulo aborda la aplicación de la técnica de Clústering de texto en R. Para ello, se introducen las principales diferencias con respecto al clústering con datos estructurados.

La organización del capítulo es la siguiente: la Sección 4.1 ofrece el contexto necesario, definiendo brevemente la técnica y describiendo los algoritmos de clústering más comunes, los métodos de preprocesamiento no supervisados específicos para texto y algunos de los paquetes más relevantes en R para esta tarea. La Sección 4.2 describe la metodología para las pruebas realizadas, que se han hecho con el conjunto de datos *IMDb Movie Reviews*. Y por último, la Sección 4.3 describe los resultados obtenidos.

Adicionalmente, todo el código desarrollado para el análisis de este capítulo se puede encontrar en el Apéndice B, además de a través del siguiente enlace: https://github.com/acabrerod/text-analysis/tree/main/clustering.

4.1. Introducción

El clústering es una técnica de clasificación no supervisada en el que diferentes individuos son asignados automáticamente a grupos llamados clústers. El clústering busca que estos individuos sean lo más homogéneos posibles, de forma que los individuos de un mismo clúster sean similares entre sí y diferentes a los de otros clústers [2, 16].

El clústering no necesariamente asigna un individuo solo a un clúster (*hard clustering*), sino que también puede hacer una asignación blanda o difusa (*soft clustering*). En este esquema, a cada individuo se le asigna una probabilidad de pertenecer a cada clúster. El clústering blando está íntimamente relacionado con el *topic modeling*, que interpreta los clústers como temas latentes [2, 16].

Como técnica no supervisada, no es necesario que los individuos que se agrupan estén etiquetados. Por tanto, no siempre es posible medir la *calidad* de los clústers que se obtienen usando estas etiquetas (validación externa). Sin embargo, existen métricas de validación interna que son no supervisadas, en [34] se puede consultar una revisión de las mismas. Algunas de ellas son: el índice de silueta, el índice de Dunn o la varianza intra-clúster.

En el resto de esta sección se introducen los algoritmos más comunes para aplicar clústering (Subsección 4.1.1), algunos de los métodos de preprocesamiento que se pueden utilizar, y en especial métodos de extracción de características no supervisados (Subsección 4.1.2), y por último paquetes de R para clústering (Subsección 4.1.3).

4.1.1. Algoritmos de clústering

Según [16], las dos principales categorías de algoritmos de clústering son los basados en distancias y los basados en modelos probabilísticos. Una forma alternativa de categorizarlos es las mencionada por [2], y que los distingue en base a las 3 siguientes propiedades: (1) la estructura del resultado, que puede ser plana (particiones) o jerárquica; (2) el tipo de pertenencia, que puede ser dura (*hard*) o blanda (*soft*); y (3) la estrategia de agrupamiento, que puede ser aglomerativa, divisiva o redistributiva (*aglomerative*, *divisive* o *shuffling*).

Basados en distancias Los algoritmos de clústering basados en distancias utilizan métricas de similaridad para medir lo cercanos que son los individuos entre sí. Estos algoritmos suelen seguir un esquema de asignación duro. Las dos principales variantes de estos algoritmos son los jerárquicos y particionales. También existen variantes híbridas que combinan ambas.

1. **Jerárquicos**: Construyen los clústers dividiendo o fusionando recursivamente los grupos en un esquema de "*arriba a abajo"*, o de "*abajo a arriba"* (*divisive* vs *aglomerative*). El criterio para dividir o fusionar los clústers depende de la forma de cálculo de la similaridad entre grupos de individuos, por ejemplo, *single linkage*, *group-average linkage* o *complete linkage* [35].

2. **Particiones**: Empiezan con una partición inicial y redistribuyen iterativamente los individuos entre los clústers. Normalmente requieren que el usuario fije previamente el número de clústers. El algoritmo más conocido de esta categoría es k-medias [35].

3. **Híbridos**: Los algoritmos jerárquicos tienden a ser más robustos que los basados en particiones, pero son computacionalmente más costosos. Los métodos del tipo *Scatter/Gather* combinan ambos algoritmos reduciendo su coste total. Primero utilizan métodos jerárquicos en un subconjunto de los datos, con lo que obtienen una distribución inicial robusta. Después aplican algoritmos como k-medias sobre el resto de los datos [2, 16].

Basados en modelos Los algoritmos de clústering basados en modelos asumen que los datos han sido generados por un modelo probabilístico subyacente, como por ejemplo, una mixtura de distribuciones de probabilidad.

A diferencia de los basados en distancias, estos métodos establecen una asignación blanda de cada individuo con los clústers. En el caso del texto, este problema es también estudiado desde el *topic modeling*, donde se pretende estimar la estructura subyacente de un corpus en forma de los temas que lo componen.

Por ejemplo, dado un documento, la probabilidad de pertenencia a cada tema subyacente (clústers) sería la asignación blanda. Para asignar un documento a un único clúster, basta con elegir el que le asigne la probabilidad más alta. Dentro del *topic modeling* hay muchos algoritmos, aunque los más básicos según [16] son los siguientes:

- **Probabilistic Latent Semantic Analysis** (PLSA): Considera a cada documento como una composición de temas latentes, y a los temas latentes como una distribución de probabilidad sobre palabras. Es decir, se modelan los documentos como observaciones de una mixtura de temas latentes.
- **Latent Dirichlet Allocation** (LDA): Versión bayesiana del PLSA, en la que las distribuciones palabra-tema y tema-documento se modelan con una distribución a priori Dirichlet.

Los métodos descritos hasta ahora son de tipo completamente no supervisado, y tratan de descubrir los agrupamientos sin ningún tipo de información previa. Aunque otro enfoque especialmente relevante para algunos datos de texto es el *semisupervisado*. Si se dispone de datos previos en cuanto a la estructura subyacente de los clústers, como por ejemplo, categorías en un foro (ciencia, deportes, salud, etc), *hastags* en redes sociales o etiquetas de documentos, se pueden aprovechar para mejorar la calidad de los clústers obtenidos. Este tipo de técnicas se ubican entre medias de la clasificación supervisada y el clústering. La forma más simple de incorporar la información conocida es utilizarla como punto de partida para el algoritmo de k-medias, aunque existen gran variedad de métodos como se explica en [16].

4.1.2. Métodos de preprocesamiento

Para aplicar clústering sobre datos de texto, pueden utilizarse las mismas técnicas de preprocesamiento descritas en la Sección 2.1, como la tokenización, la normalización y la eliminación de *stop words*, así como las técnicas de representación detalladas en la Sección 2.2. En esta subsección se describen métodos no supervisados adicionales, que son especialmente válidos para tareas como clústering, y que se han tomado de [16] y [36].

Extracción de características El problema más recurrente de las representaciones simbólicas es la alta dimensionalidad, que puede afectar negativamente tanto al rendimiento computacional como a la calidad del agrupamiento. Para mitigar este problema, se emplean técnicas de extracción de características, basadas en seleccionar un subconjunto de términos (palabras, n-gramas u otras unidades) que sean especialmente representativos e informativos para la tarea en cuestión. Estas técnicas permiten reducir la dimensionalidad del problema, descartando términos ruidosos, irrelevantes o poco importantes.

La extracción de características resulta más sencilla en problemas de clasificación supervisada, donde se conoce de antemano la información relevante que se tiene que preservar. A pesar de ello, existen diversos métodos no supervisados para la extracción de características, tales como:

- **Document Frequency** (DF): Número de documentos en el que aparece un término.
- **Term Strength** (TS): Mide cómo de informativo es un término para identificar dos documentos que están relacionados[1]. Para calcular el TS de un término t, se toman todas las parejas de documentos relacionados y se realiza el siguiente cálculo:

$$TS(t) = \frac{\text{Número de parejas donde } t \text{ está en ambos documentos}}{\text{Número de parejas en las que } t \text{ está en el primer documento}}.$$

[1] En contextos no supervisados, se puede considerar que dos documentos están relacionados si su distancia del coseno es menor a un determinado umbral.

- **Entropy-based Ranking** (EBR): Mide la información proporcionada por un término teniendo en cuenta la reducción de la entropía al eliminarse.

$$E(t) = -\sum_{i=1}^{N}\sum_{j=1}^{N}(S_{ij} \cdot logS_{ij} + (1 - S_{ij}) \cdot log(1 - S_{ij})).$$

  Donde $S_{ij} \in (0, 1)$ es la similaridad entre los documentos i y j después de eliminar el término t: $S_{ij} = 2^{-\frac{d(i,j)}{\overline{d}}}$, siendo $\overline{d}$ la distancia promedio entre todos los pares de documentos.

- **Term Contribution** (TC): Mide la contribución de cada término a la similaridad de los documentos.

$$TC(t) = \sum_{\substack{i,j \\ i \neq j}} \text{TF-IDF}_{t,d_i} \cdot \text{TF-IDF}_{t,d_j}. \tag{4.1}$$

  Donde $\text{TF-IDF}_{t,d_i}$ es la ponderación TF-IDF del término t en el documento d_i.

Los métodos *TS* y *EBR* requieren calcular las distancias entre los textos que se clasifican, para lo cual habrá que aplicar técnicas de representación númerica como cualquiera de las vistas en la Sección 2.2, aunque en principio están pensadas solo para representaciones simbólicas.

Transformación de características Estos métodos no preservan la interpretabilidad de las características originales, sino que las transforman reduciendo la dimensionalidad. Además de los métodos ya vistos en la Subsección 2.2.2, como LSA, Word2Vec o BERT, cabe destacar la factorización no negativa de matrices. Como justifica [16], es especialmente apropiada para clústering.

- **Non-negative Matrix Factorization** (NMF): A diferencia de LSA, los vectores resultado de la factorización no son ortonormales. Esto significa que se puede obtener un clúster (o tema latente) mirando la componente más grande de cada vector de características. Es decir, que NMF es un método de reducción de dimensión que además sirve para hacer *topic modeling* y extraer clústers de forma directa.

  La Figura 4.1 muestra la descomposición de rango k con NMF. Si A es una matriz término-documento (m términos, n documentos), entonces la matriz W una representación de los m términos en el espacio latente de los k temas, y H los vectores de características que representan a los documentos.

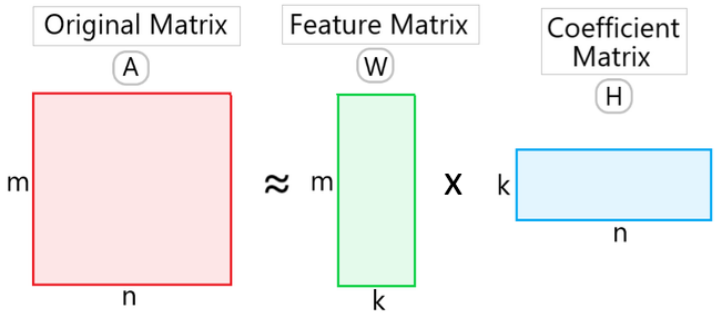


Figura 4.1: Esquema de la factorización no negativa de matrices [37]

4.1.3. Paquetes de R

Para aplicar las técnicas de clústering en R se pueden usar paquetes habituales para clústering, previo preprocesamiento y representación numérica del corpus. Algunos muy comunes son los siguientes:

- `stats` (base de R): Paquete de estadística básica que tiene funciones para clústering jerárquico (hclust) y k-medias (kmeans).
- `cluster`: Paquete con métodos clásicos y avanzados de clústering que tiene funciones para clústering jerárquico aglomerativo (agnes) y divisivo (diana), para la versión robusta de k-medias, *k-medoids* (pam, clara) y para evaluar calidad del clústering (silhouette)[2].
- `mclust`: Paquete para clústering basado en modelos con mixturas de normales[3].
- `factoextra`: Paquete para visualizar los resultados de análisis multivariante, entre los que está el clústering. Tiene funciones para la representación de clústers en dos dimensiones (fviz_cluster), para representar dendogramas (fviz_dend) y para obtener un número razonable de clústers (fviz_nbclust)[4].

4.2. Metodología

El objetivo de este capítulo es mostrar la aplicación en R de diversos métodos de clústering, como los expuestos en la Sección 4.1. Para ello, se ha utilizado el conjunto de datos *IMDb Movie Reviews* descrito en la Sección 3.3, en concreto las 25.000 reseñas del conjunto de *test*. Esta elección permitirá comparar los resultados con los análisis de los Capítulos 3 y 5. También por este motivo se ha buscado identificar 2 clústers, para comprobar la coincidencia con los grupos de reseñas positivas y negativas.

Para poder aplicar las técnicas de clústering sobre estos datos, hay que incidir en su elevado coste computacional, que supone una importante restricción tanto en tiempo como en memoria. En primer lugar, dos de las métricas de extracción de características que se han utilizado han resultado tener un coste demasiado alto, y en lugar de calcularlas de manera exacta, se ha optado por calcularlas sobre una muestra de reseñas. La Subsección 4.2.1 describe en detalle el procedimiento que se ha seguido con cada métrica de extracción de características.

Y en segundo lugar, también se tiene que considerar la complejidad de los diferentes algoritmos de clústering. Los métodos jerárquicos necesitan utilizar una matriz de distancias, que para los datos que se usarán en este análisis (25.000), tiene un tamaño en términos de memoria de 2,33 GB. Es decir, que tanto calcular como almacenar esta matriz implica un alto coste, al que se debería sumar el coste del propio algoritmo, que tiene un orden de complejidad cuadrático con el número de reseñas. Por este motivo, en las pruebas se utilizará k-medias, que tiene un coste lineal con el número de reseñas para cada iteración[5], y por tanto mucho menor que los jerárquicos.

[2] `cluster`: https://cran.r-project.org/package=cluster
[3] `mclust`: https://cran.r-project.org/package=mclust
[4] `factoextra`: https://cran.r-project.org/package=factoextra
[5] Si n es el número de reseñas, m el número de características, k el número de clústers y t el número de iteraciones, el coste de k-medias es $O(t \cdot k \cdot n \cdot m)$ [35].

Teniendo en cuenta estas consideraciones, las pruebas se han realizado variando los métodos de procesamiento, para ser posteriormente evaluados con las métricas de validación que se detallan en la Subsección 4.2.2. Las condiciones de cada prueba se pueden resumir a decidir sobre las siguientes 4 cuestiones:

1. **Preprocesamiento**: Se aplica el mismo para todas las pruebas, que consiste en: conversión a minúsculas, eliminación de números, eliminación de *stop words*, expansión de contracciones, lematización y eliminación de términos menos frecuentes (menos de 4 apariciones en total).

2. **Representación**: Se han probado los siguientes métodos:

   - Simbólicas: BoW y TF-IDF (con el paquete `text2vec`).

   - Distribuidas: LSA, Word2Vec y GloVe[6]. Se han utilizado respectivamente los paquetes: `text2vec`, `word2vec` y `text2vec`. Para todos ellos se han extraído 5, 10, 25, 50, 100, 250 y 500 dimensiones, excepto para LSA, para el cual se han tomado hasta un máximo de 100 por problemas en los embeddings generados.

3. **Extracción de características**: Para las representaciones simbólicas, se han probado los 4 métodos de extracción de características mencionados en la Subsección 4.1.2, a saber, DF, TS, EBR y TC. Para todos los métodos, se han extraído las 10, 25, 50, 100, 250, 500, 1.000 y 2.500 mejores características (palabras). No se han considerado métodos de extracción de características supervisados.

4. **Algoritmo de clústering**: Se aplica la misma configuración para todas las pruebas, para las que se ha utilizado el algoritmo de k-medias con $k = 2$. Además, se han empleado 10 inicializaciones aleatorias y un límite máximo de 100 iteraciones.

4.2.1. Extracción de características

Como ya se ha mencionado, el cálculo de dos de estas métricas es muy elevado, en concreto para TS y EBR. Las otras dos métricas (DF y TC) sí se han podido calcular de manera exacta. Es importante hacer notar que la implementación para el cálculo de estas métricas es propia, a excepción de DF, ya que no se ha encontrado un paquete de R que las implementara. Este código se puede encontrar en la Sección B.5.

Los cálculos se han paralelizado para reducir el tiempo de cómputo (excepto DF), y se ha fijado la semilla para que los resultados sean reproducibles. En el equipo que se ha utilizado para estas pruebas, los cálculos han tardado aproximadamente lo siguiente: DF (1 segundo), TS (5 minutos), EBR (10 minutos), TC (10 segundos). A continuación se describe el proceso seguido para cada métrica:

Document Frecuency (DF) Se ha calculado de manera exacta. La función `create_vocabulary` del paquete `text2vec` indica el número de reseñas en el que aparece cada término, es decir, DF.

[6]Los métodos Word2Vec y GloVe producen emebddings para las palabras que aparecen en las reseñas. Para obtener embeddings que representen a las reseñas, se ha tomado para cada una el vector promedio de las palabras ponderadas por su frecuencia.

Term Strength (TS) Se ha calculado mediante una estimación. El coste de TS es cuadrático con el número de reseñas. Para todas las palabras, hay que comprobar todas las parejas de reseñas relacionadas.

Se ha tomado una muestra de 5.000 reseñas, que contiene aproximadamente 20.000 palabras únicas. De estas reseñas, se han tomado 500.000 emparejamientos (el 4 % de las menores distancias entre reseñas) para calcular el TS.

Entropy-based Ranking (EBR) Se ha calculado mediante una estimación. El coste de EBR es cuadrático con el número de reseñas. Para todas las palabras, hay que calcular la matriz de distancias de las reseñas sin esa palabra.

Este método es con diferencia el más costoso computacionalmente, debido a que calcular una matriz de distancias es una operación especialmente costosa. Para calcular la EBR se ha tomado una muestra de 200 reseñas, que contienen unas 5.000 palabras únicas.

Term Contribution (TC) Se ha calculado de manera exacta. Partiendo de la Ecuación 4.1, se ha simplificado esta expresión, que es una suma de productos cruzados, calculando a partir del vector de características en la matriz TF-IDF de cada término t, $\text{TF-IDF}_{t,\cdot}$, la siguiente operación para obtener TC:

$$\sum_{i \neq j} \text{TF-IDF}_{t,d_i} \cdot \text{TF-IDF}_{t,d_j} = \left(\sum_{i=1}^{n} \text{TF-IDF}_{t,d_i} \right)^2 - \sum_{i=1}^{n} \text{TF-IDF}_{t,d_i}^2.$$

4.2.2. Validación

Para validar el rendimiento de cada prueba se han utilizado dos métricas de validación interna (índice de Dunn e índice de silueta promedio), y una métrica de validación externa (exactitud). Esta última permite comparar los resultados con los análisis de los Capítulos 3 y 5. La definición de las métricas de validación interna es la siguiente:

- **Índice de Dunn**: Se define como la mínima separación entre clústers dividida por la máxima dispersión dentro de un clúster. Sea k el número de clústers, C_i el clúster i-ésimo, $\delta(C_i, C_j)$ la distancia mínima entre los clústers C_i y C_j, y $\Delta(C_i) = \max_{x,y \in C_i} d(x, y)$ la máxima distancia entre dos puntos del clúster C_i. El índice de Dunn se define como:

$$D = \frac{\min\limits_{1 \leq i < j \leq k} \delta(C_i, C_j)}{\max\limits_{1 \leq l \leq k} \Delta(C_l)}.$$

  Este índice toma valores positivos, y mejora cuanto más alto sea.

- **Índice de Silueta**: Mide lo similar que es un individuo con su clúster en comparación con otros clústers. Sea $x \in C_i$ un individuo en el clúster i, $a(x) = \frac{1}{|C_i|-1} \sum_{y \in C_i, x \neq y} d(x, y)$ la distancia promedio de x con el resto de los individuos de su clúster, y $b(x) = \min_{C_j \neq C_i} (\frac{1}{|C_j|} \cdot \sum_{y \in C_j} d(x, y))$ la distancia mínima promedio de x con los individuos de cualquier otro clúster. El índice de silueta para x se define como:

$$s(x) = \frac{b(x) - a(x)}{\max \{a(x), b(x)\}}.$$

Después se toma el índice de silueta promedio: $\bar{s} = \frac{1}{n}\sum_{i=1}^{n} s(i)$, donde n es igual al número total de individuos. Este índice toma valores entre -1 y $+1$, y es mejor cuanto más alto sea.

Ambas métricas de validación interna requieren el cálculo de una matriz de distancias, por lo que también se ha optado por estimarlas utilizando muestras aleatorias. En concreto, para cada prueba se han tomado 10 muestras de 2.000 reseñas cada una, y se ha tomado el valor promedio de los indicadores. Para calcularlas se ha utilizado la función `cluster.stats` del paquete `fpc`. Las matrices de distancias (coseno) se han calculado con el paquete `proxy`.

4.3. Resultados

Los resultados obtenidos en las pruebas se presentan organizados en dos subsecciones: la primera, la Subsección 4.3.1, expone los obtenidos con las representaciones simbólicas, que se basan en extracción de características. Y en la segunda, la Subsección 4.3.2, se exponen los correspondientes a las representaciones distribuidas.

En ambos casos se muestran los resultados en forma de gráficos de líneas, donde el eje X representa en escala logarítmica el número de dimensiones de la representación, y el eje Y la métrica de validación correspondiente.

Por último, la Subsección 4.3.3, realiza una pequeña comparación de los mejores métodos, resumiendo sus resultados en la Tabla 4.1.

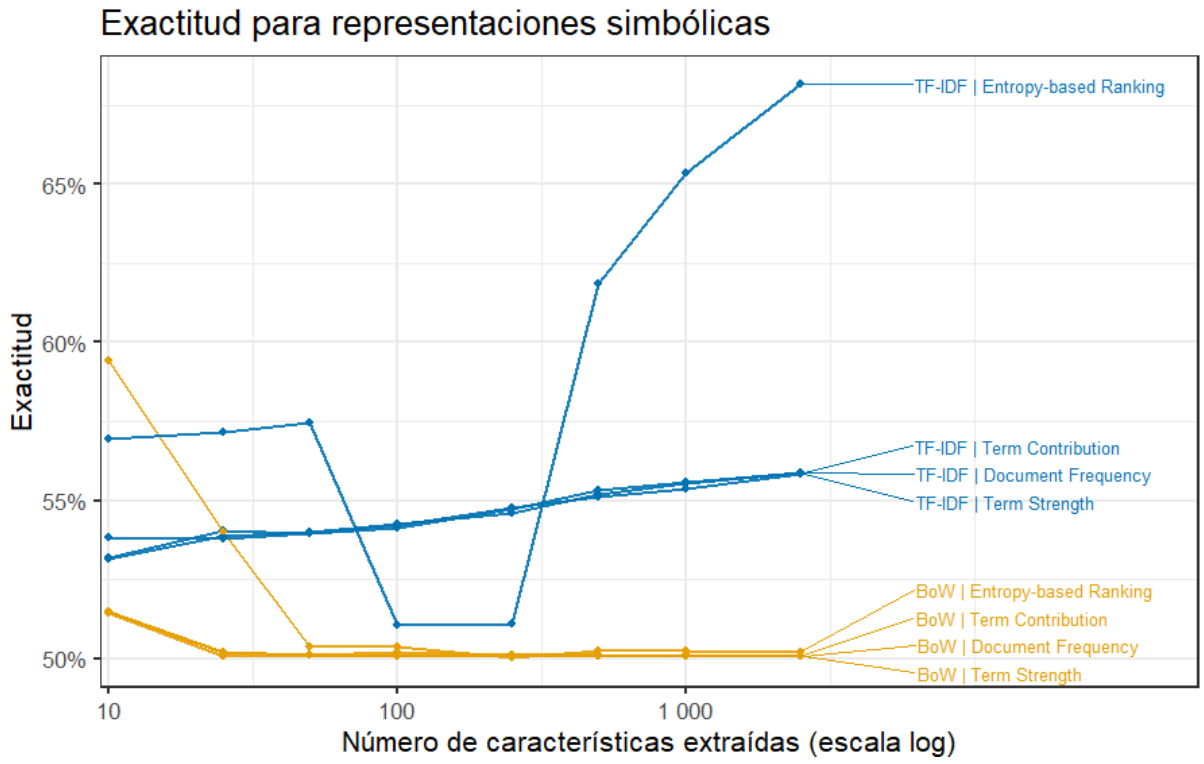
4.3.1. Representaciones simbólicas (BoW y TF-IDF)

Los resultados obtenidos se presentan en la Figura 4.2, la cual se divide en tres subfiguras, correspondientes a cada una de las métricas de validación empleadas. Como se puede observar, la representación TF-IDF es superior a BoW para las tres métricas consideradas. No obstante, el mejor método de extracción de características varía en función de la métrica evaluada (EBR para exactitud, DF para Dunn y TS para Silueta).

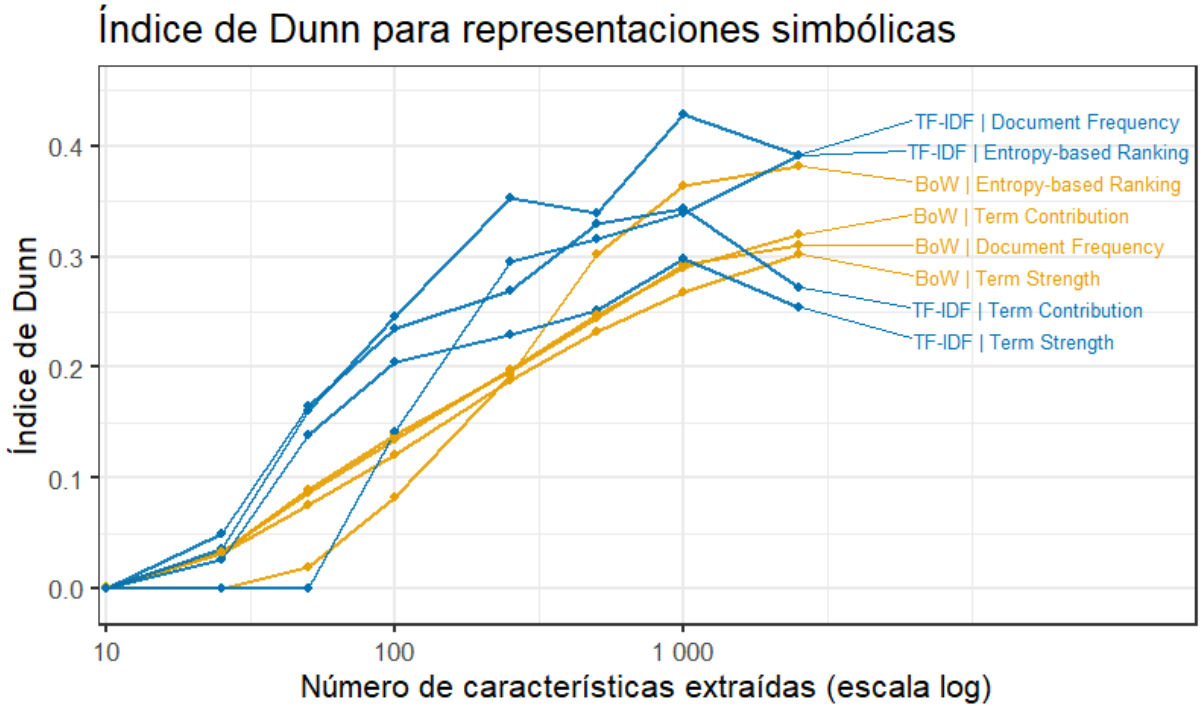
En relación con la exactitud, se observa que BoW no logra capturar una estructura clara en cuanto a reseñas positivas o negativas, dado que su rendimiento se mantiene en torno al 50 %. En cambio, TF-IDF sí muestra una mejora general al aumentar el número de características extraídas. También destaca EBR, cuyos rendimientos se desvían notablemente del resto de métodos de extracción de características, que son parecidos entre sí. Esta mayor variabilidad puede deberse al reducido tamaño de la muestra que se ha usado para calcularlo (200 reseñas)[7].

En lo que respecta al índice de Dunn, las diferencias entre BoW y TF-IDF son menos pronunciadas, si bien TF-IDF vuelve a ser superior. Además, se aprecia una tendencia creciente en el rendimiento a medida que se incluyen más características. Por último, en lo referente al índice de silueta promedio, lo más llamativo es que disminuye al aumentar las características. Esto es un efecto lógico, ya que al aumentar las dimensiones las reseñas se vuelven más dispersas, incluso dentro de un mismo clúster. También se puede apreciar como todos los métodos tienen un resultado casi idéntico, excepto de nuevo con EBR.

[7] Como muestra la Sección B.5, las palabras extraídas por EBR están descorrelacionadas del resto de métodos.

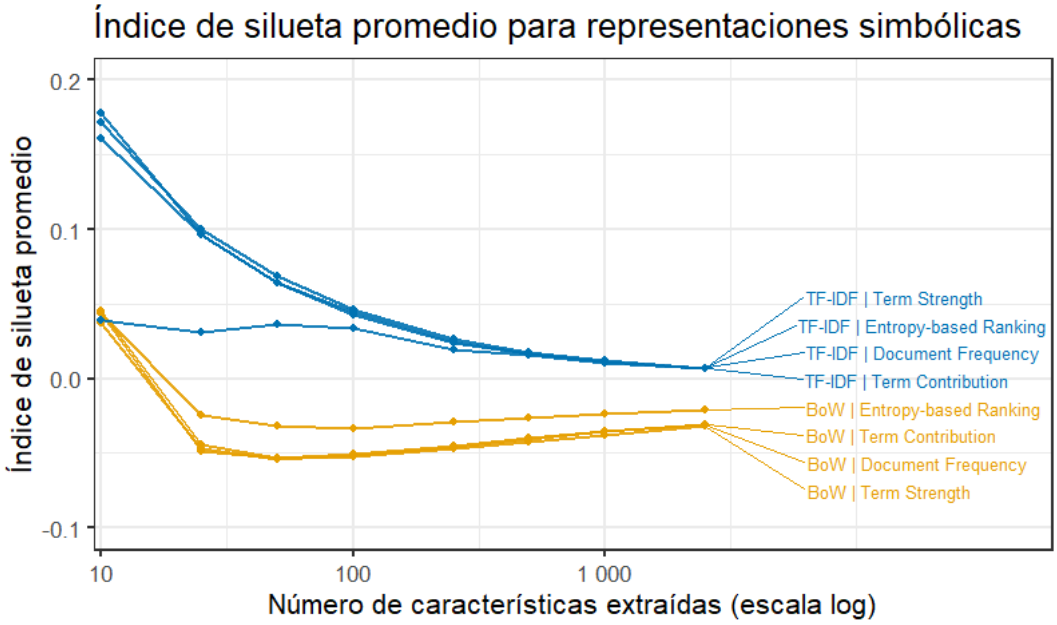


(a) Exactitud



(b) Índice de Dunn

Figura 4.2: Resultados para representaciones simbólicas. Exactitud, Índice de Dunn e Índice de silueta promedio vs el número de características extraídas en escala semilogarítmica. Se muestra una línea para cada método de representación (BoW y TF-IDF) y de extracción de características (DF, TC, EBR y TS)



(c) Índice de silueta promedio

Figura 4.2: Resultados para representaciones simbólicas (continuación)

4.3.2. Representaciones distribuidas (LSA, Word2Vec y GloVe)

Los resultados obtenidos se muestran en la Figura 4.3, la cual vuelve a estar dividida en 3 gráficos, uno por cada métrica de validación. En este caso no hay ninguna representación que sea claramente superior a las demás, aunque sí podría decirse que hay una peor, que es GloVe.

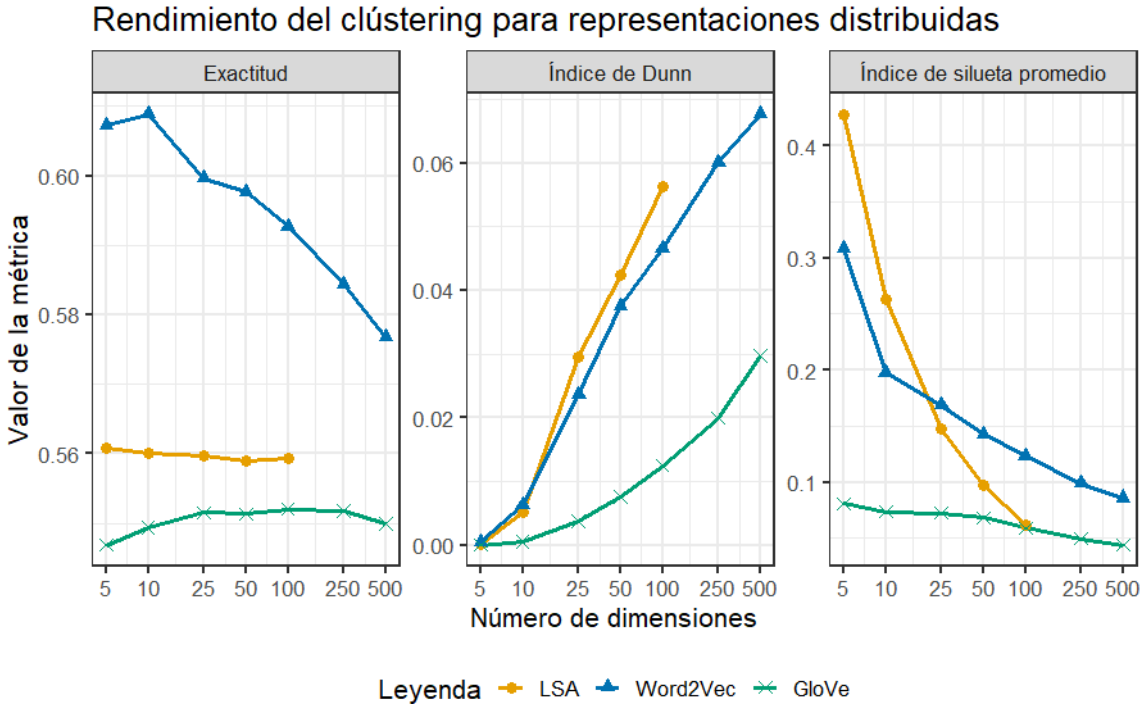


Figura 4.3: Resultados para representaciones distribuidas. Exactitud, Índice de Dunn e Índice de silueta promedio vs el número de dimensiones en escala semilogarítmica, dependiendo del método de representación (LSA, Word2Vec y GloVe)

En términos de exactitud, la mejor representación es Word2Vec, que supera a LSA a igualdad de dimensiones con entre un 3 y un 5 %. También supera a GloVe con entre un 3 y un 6 %. Llama la atención que el rendimiento de Word2Vec disminuye al aumentar las dimensiones, lo que indica que hay un número óptimo de dimensiones relativamente bajo. Por su parte, LSA y GloVe, logran capturar cierta estructura subyacente en las reseñas en términos de polaridad, con unos resultados estables entre el 55 y el 56 % para todas las dimensiones probadas.

En lo que respecta a las métricas de validación interna, se aprecia una evolución similar a las obtenidas con las representaciones simbólicas, si bien los clústers obtenidos con LSA y Word2Vec tienen una mayor cohesión que los obtenidos con GloVe.

4.3.3. Comparación

La Tabla 4.1 reúne una pequeña selección de los mejores métodos probados en este capítulo, donde se han resaltado en negrita las mejores métricas. El método que alcanza la mayor exactitud es TF-IDF con 2.500 características extraídas con EBR (68,17 %), seguido por Word2Vec con 10 dimensiones (61,10 %).

Método		**Rendimiento**		
Rep.	**Dimensiones**	**Exactitud**	**Dunn**	**Silueta**
BoW	10 (EBR)	59,41 %	0,0000	0,0440
TF-IDF	2.500 (EBR)	**68,17 %**	**0,3919**	0,0071
TF-IDF	2.500 (DF)	55,86 %	**0,3922**	0,0071
LSA	5	56,08 %	0,0001	**0,4300**
Word2Vec	5	60,82 %	0,0006	0,2986
Word2Vec	10	61,10 %	0,0065	0,2214
GloVe	25	55,22 %	0,0038	0,0738

Tabla 4.1: Comparación general de resultados de métodos de clústering

Capítulo 5

Clasificación supervisada

Este capítulo aborda la aplicación de clasificación de texto en R. Para ello, se introducen las principales diferencias con respecto a la clasificación con datos estructurados.

La organización del capítulo es la siguiente: la Sección 5.1 ofrece una breve introducción a la clasificación, definiendo brevemente la técnica y describiendo los algoritmos más utilizados, algunas técnicas supervisadas de preprocesamiento de texto y algunos de los paquetes más relevantes en R para esta tarea. La Sección 5.2 detalla la metodología utilizada para las demostraciones realizadas, y la Sección 5.3 describe los resultados obtenidos.

Adicionalmente, todo el código desarrollado para el análisis de este capítulo se puede encontrar en el Apéndice C, además de a través del siguiente enlace: https://github.com/acabrerod/text-analysis/tree/main/classification.

5.1. Introducción

La clasificación es una técnica de aprendizaje supervisado que consiste en asignar a individuos no etiquetados, la etiqueta más adecuada de un conjunto finito de categorías, basándose en un modelo entrenado con individuos previamente etiquetados. En el contexto de clasificación de texto, los individuos pueden ser documentos, noticias, correos electrónicos, reseñas o cualquier otro tipo de texto genérico.

Formalmente, el problema se puede definir de la siguiente manera: dado un conjunto de entrenamiento de N documentos $D = \{d_1, d_2, \ldots, d_N\}$, donde cada documento d_i está asociado a una etiqueta correspondiente a un conjunto finito y predefinido de categorías $C = \{c_1, \ldots, c_k\}$, el objetivo consiste en construir un modelo de clasificación que aproxime una función de asignación a clases $F : D \to C$. Las funciones que definen F se denominan clasificadores. Una vez entrenado el modelo, este puede ser utilizado para predecir la categoría de nuevos documentos cuya etiqueta sea desconocida.

Existen múltiples variantes del problema de clasificación, como la categorización multiclase, donde cada instancia se asigna simultáneamente a varias categorías, o la categorización dura o blanda, que difieren en si la asignación es determinista o probabilística, respectivamente, de igual manera que sucede para el clústering duro o blando.

En el resto de esta sección se introducen los tipos de algoritmos más utilizados para clasificación (Subsección 5.1.1), los métodos supervisados de preprocesamiento más relevantes para texto (Subsección 5.1.2), y algunos paquetes de R para clasificación (Subsección 5.1.3).

5.1.1. Algoritmos de clasificación

Existe una gran variedad de técnicas de clasificación que son estudiadas en el campo del aprendizaje automático. Estos algoritmos son de propósito general y también se pueden utilizar para la clasificación de texto. A continuación, se presentan las principales familias de algoritmos de clasificación, siguiendo como referencia [2] y [16].

Clasificadores probabilísticos Se basan en un modelo para calcular la probabilidad de que cada documento pertenezca a una clase. Para ello, cada clase se modela como una dsitribución condicional, y se estima la probabilidad de observar cada término (palabras, n-gramas, etc) dentro de dicha clase. La probabilidad de que un documento d pertenezca a una clase c se calcula aplicando el teorema de Bayes:

$$P(c \mid d) = \frac{P(d \mid c)\, P(c)}{P(d)}$$

- **Naïve bayes**: Asume que los términos son condicionalmente independientes dada la clase, lo que permite calcular la probabilidad del documento como el producto de las probabilidades condicionadas a la clase c de los términos: $P(d \mid c) = \prod_i P(t_i \mid c)$.

Clasificadores lineales El predictor utilizado en los clasificadores lineales consiste en una combinación lineal de las características de cada documento, que dependen del método elegido de representación. Por ejemplo, con BoW sin preprocesamiento adicional, las características son las frecuencias de cada palabra en el documento. Estos predictores se pueden interpretar geométricamente como un hiperplano separador en el espacio de características.

Entre los clasificadores lineales más relevantes se encuentran las máquinas de soporte vectorial (SVMs), los basados en regresión, como la logística, o las redes neuronales de una capa (perceptrón). Tanto los SVMs como las redes neuronales pueden adaptarse para problemas de separación no lineal utilizando respectivamente el *kernel trick* o múltiples capas de neuronas.

Basados en árboles Los árboles de decisión son modelos con estructura jerárquica que utilizan condiciones lógicas para dividir recursivamente el espacio de características en regiones cada vez más homogéneas. Estas condiciones se eligen buscando optimizar criterios de pureza como el índice de Gini o la entropía.

Basados en reglas Al igual que los árboles de decisión, los clasificadores basados en reglas utilizan condiciones lógicas para fragmentar el espacio de características. Pero a diferencia de los árboles, lo hacen sin una estructura jerárquica, que permite el solapamiento entre reglas. Estas reglas pueden construirse mediante aprendizaje inductivo, o también manualmente incorporando conocimiento experto. El principal beneficio de estos clasificadores, al igual que los árboles, es su muy elevada interpretabilidad.

Basados en proximidad Utilizan métricas de distancia, bajo la suposición de que documentos que son cercanos pertenecen a la misma categoría. Dos ejemplos de algoritmos de esta clase son kNN (k vecinos más próximos), que calcula la categoría de un nuevo documento observando la clase mayoritaria de los k documentos más cercanos, o los métodos de Rocchio, que clasifican en base a las distancias con los representantes prototípicos de cada clase.

Meta-algoritmos Los meta-algoritmos son estrategias que combinan múltiples clasificadores para mejorar el rendimiento de la clasificación. Para generar la clase final, estos algoritmos pueden utilizar alguna estrategia de agregación, como los esquemas de votación o como el *stacking*. Los dos meta-algoritmos más conocidos son *boosting* y *bagging*:

- **Boosting**: Entrena clasificadores secuencialmente, donde los nuevos modelos se concentran en los errores de los anteriores.
- **Bagging** (Bootstrap Aggregating): Entrena clasificadores independientes sobre diferentes muestras bootstrap (muestreo con reemplazamiento).

Los meta-algoritmos permiten combinar clasificadores de diferentes tipos, e incluso con preprocesamientos y representaciones distintas.

De acuerdo con [16], los SVMs son particularmente apropiados para la clasificación de texto por sus buenas propiedades, como la simplicidad e interpretabilidad, y por los robustos resultados que obtienen en muchas de las tareas de clasificación de texto.

5.1.2. Métodos de preprocesamiento

Para la clasificación de texto, se pueden utilizar las técnicas de preprocesamiento descritas en la Sección 2.1, como la tokenización, la normalización y la eliminación de *stop words*, así como las técnicas de representación detalladas en la Sección 2.2. Sin embargo, en el contexto del aprendizaje supervisado, resulta especialmente importante incorporar la información de las categorías durante el proceso de reducción de dimensiones, con el objetivo de conservar de forma más efectiva la información relevante.

En esta subsección se presentan algunos métodos de extracción y transformación de características supervisados, para los cuales se ha tomado como referencia [2] y [16].

Extracción de características No todas las palabras son igual de relevantes para la clasificación. Mientras que algunas pueden tener una mayor relación con la distribución de las categorías, otras pueden ser irrelevantes o hasta ruidosas[1]. Con el objetivo de seleccionar los términos mas informativos, se pueden emplear métodos de extracción de características supervisados, como los siguientes:

- **Índice de Gini**: Mide la desigualdad de la distribución de las clases para cada término. Un valor más alto indica un mayor poder informativo del término, alcanzando su máximo si todos los documentos que lo contienen pertenecen a una única clase.
- **Ganancia de información** (Information Gain), o entropía: Cuantifica la información aportada por cada término para la predicción de las categorías.

[1] Por ejemplo, en un filtro de spam, es fácilmente comprobable si términos como *dinero gratis*, *barato* o *urgente* son relevantes para detectar si un correo es o no spam.

- **Información mutua**: Evalúa la cantidad de información compartida entre los términos y las clases. Para cada término, se puede calcular la información mutua tomando la información promedio sobre todas las clases, o su máximo.

- **Estadístico** χ^2: Evalúa la falta de independencia entre los términos y las categorías. Considerando la tabla de contingencia para un término t y una categoría c, donde A representa el número de documentos de categoría c que contienen t, y C los que no contienen t, B el número de documentos de categorías distintas a c que contienen t, y D los que no contienen t, se calcula:

$$\chi^2(t,c) = \frac{(AD - BC)^2 \cdot N}{(A+C)(B+D)(A+B)(C+D)}.$$

  El valor de este estadístico puede compararse con una distribución χ^2 con 1 grado de libertad (sin ajustar por comparaciones múltiples). Para cada término se debe agregar el valor del estadístico sobre todas las clases, ya sea mediante su promedio o tomando su valor máximo.

En la Subsección 4.1.2 pueden consultarse otros métodos de extracción de características no supervisados.

Transformación de características Todos los métodos de reducción de dimensión presentados hasta ahora en el presente trabajo, como los expuestos en la Subsección 2.2.2, o como los descritos en la Subsección 4.1.2, corresponden a técnicas no supervisadas. Esto implica que la generación de las dimensiones no considera la distribución de las clases de los documentos.

En contraposición, las técnicas supervisadas de transformación de características incorporan esta información durante el proceso de reducción de dimensiones, generando unas representaciones con mayor poder discriminatorio. Entre estas técnicas destacan el LSA supervisado (SLSA) y el Análisis Discriminante Lineal (LDA).

- **LSA supervisado** (SLSA): Es una adaptación del LSA convencional que incorpora la información de las clases. La variante del SLSA más habitual consiste en aplicar LSA por separado para cada clase, para luego identificar y combinar, mediante un proceso iterativo, los autovectores que mejor discriminan las clases.

- **Análisis Discriminante Lineal** (LDA): Proyecta las características originales en un espacio dimensional más reducido, donde las dimensiones son combinaciones lineales de las características y maximizan el cociente entre la varianza interclase y la varianza intraclase. Es decir, el LDA calcula las combinaciones lineales de características que mejor discriminan entre las categorías.

5.1.3. Paquetes de R

Para clasificación de texto se pueden utilizar los paquetes habituales de clasificación en R, previo preprocesamiento y representación numérica de los documentos. Algunos de los más significativos son los siguientes:

- `e1071`: Ofrece una implementación para SVMs a través de la librería de C++ `libsvm`. También implementa el clasificador Naïve Bayes[2].
- `glmnet`: Implementa modelos lineales generalizados (GLMs) con regularización de red elástica, entre las que se incluyen lasso y ridge[3].
- `rpart`: Implementa y permite visualizar árboles de decisión y regresión (CART)[4].
- `randomForest`: Implementa el algoritmo de clasificación *random forest*[5].

5.2. Metodología

El objetivo de este capítulo es mostrar la aplicación en R de algunos métodos de clasificación de texto. Para ello, se ha utilizado el conjunto de datos *IMDb Movie Reviews* descrito en la Sección 3.3, empleando el conjunto de *train* para entrenar los clasificadores, y el conjunto de *test* para evaluarlos.

Aunque la clasificación puede aplicarse a múltiples tareas, como el filtrado de spam o la moderación de contenido en redes sociales, en este capítulo se utilizará para análisis de sentimiento, es decir, con las categorías de *positivo* y *negativo*. Con el objetivo de realizar varias demostraciones que permitan comparar el rendimiento de diversos métodos de clasificación de texto, se han establecido las siguientes condiciones para cada prueba:

1. **Preprocesamiento**: Se aplica el mismo para todas las pruebas, y el mismo que en el Capítulo 4, que consiste en: conversión a minúsculas, eliminación de números, eliminación de *stop words*, expansión de contracciones, lematización y eliminación de términos menos frecuentes (menos de 4 apariciones en total).
2. **Representación**: No se han considerado representaciones distribuidas supervisadas por suponer un coste computacional demasiado alto, y se han probado los siguientes métodos, que coinciden con los del Capítulo 4:
   - Simbólicas: BoW y TF-IDF (con el paquete `text2vec`).
   - Distribuidas: LSA, Word2Vec y GloVe[6]. Se han utilizado respectivamente los paquetes: `text2vec`, `word2vec` y `text2vec`. Para todos ellos se han extraído 5, 10, 25, 50, 100, 250 y 500 dimensiones, excepto para LSA, para el cual se han tomado hasta un máximo de 100 por problemas en los embeddings generados.
3. **Extracción de características**: Para las representaciones simbólicas, se han extraído las 10, 25, 50, 100, 250, 500, 1.000, 2.500, 5.000 y 10.000 mejores características (palabras) con el método del estadístico χ^2.

[2] `e1071`: https://cran.r-project.org/package=e1071
[3] `glmnet`: https://cran.r-project.org/package=glmnet
[4] `rpart`: https://cran.r-project.org/package=rpart
[5] `randomForest`: https://cran.r-project.org/package=randomForest
[6] Los métodos Word2Vec y GloVe producen emebddings para las palabras que aparecen en las reseñas. Para obtener embeddings que representen a las reseñas, se ha tomado para cada una el vector promedio de las palabras ponderadas por su frecuencia.

4. **Algoritmo de clasificación**: Se han utilizado SVM y regresión logística, ambas con regularización L2. Para el entrenamiento de los clasificadores se ha utilizado el paquete `LiblineaR`, que es una interfaz de la librería con el mismo nombre de C++, y que soporta grandes conjuntos de datos. Para estimar el parámetro de coste de la regularización se ha empleado validación cruzada 10-fold.

Para validar los resultados obtenidos, se han utilizado las métricas de exactitud y *F1-Score*, que ya han sido definidas en la Subsección 3.2.1.

5.3. Resultados

Los resultados obtenidos se presentan en las dos siguientes subsecciones: en la primera (Subsección 5.3.1) se exponen los obtenidos con las representaciones simbólicas (con extracción de características χ^2), y en la segunda (Subsección 5.3.2) los correspondientes a las representaciones distribuidas.

Los resultados se presentan en las Figuras 5.1 y 5.2 respectivamente para cada subsección, mediante gráficos de líneas donde el eje X representa en escala logarítmica el número de dimensiones de la representación, y el eje Y la métrica de validación correspondiente.

También se muestran en las Tablas 5.1 y 5.2 los rendimientos de los dos mejores métodos para todas las representaciones (el mejor para exactitud, y el mejor para *F1-Score*). En caso de que el mismo método sea óptimo simultáneamente para ambas métricas de validación, se extrae el siguiente mejor en términos de exactitud. Si varios métodos coinciden con el mismo mejor rendimiento, se indica en la tabla y se extrae el siguiente mejor.

Por último, en la Subsección 5.3.3, se realiza una pequeña comparación general de los mejores métodos, en las Tablas 5.3 y 5.4.

5.3.1. Representaciones simbólicas (BoW y TF-IDF)

Los resultados obtenidos se muestran en la Figura 5.1 y en la Tabla 5.1. Como se puede observar, todos los métodos obtienen un rendimiento muy parecido y creciente hasta las primeras 1.000 características extraídas.

Además, se puede apreciar como el rendimiento alcanza su máximo en el entorno de las 1.000 características, alrededor del 87,5 %. Después, comienza a disminuir excepto para TF-IDF, cuyo rendimiento se mantiene en la misma zona.

En la Tabla 5.1 se muestran los resultados de los dos mejores métodos para cada representación redondeados a 3 cifras decimales. El mejor método en términos absolutos es TF-IDF con 5.000 características, aunque todos los basados en TF-IDF tienen un rendimiento muy similar entre las 1.000 y 10.000 características, siendo superior al 87,4 % en todos los casos (para ambas métricas y para ambos algoritmos de clasificación).

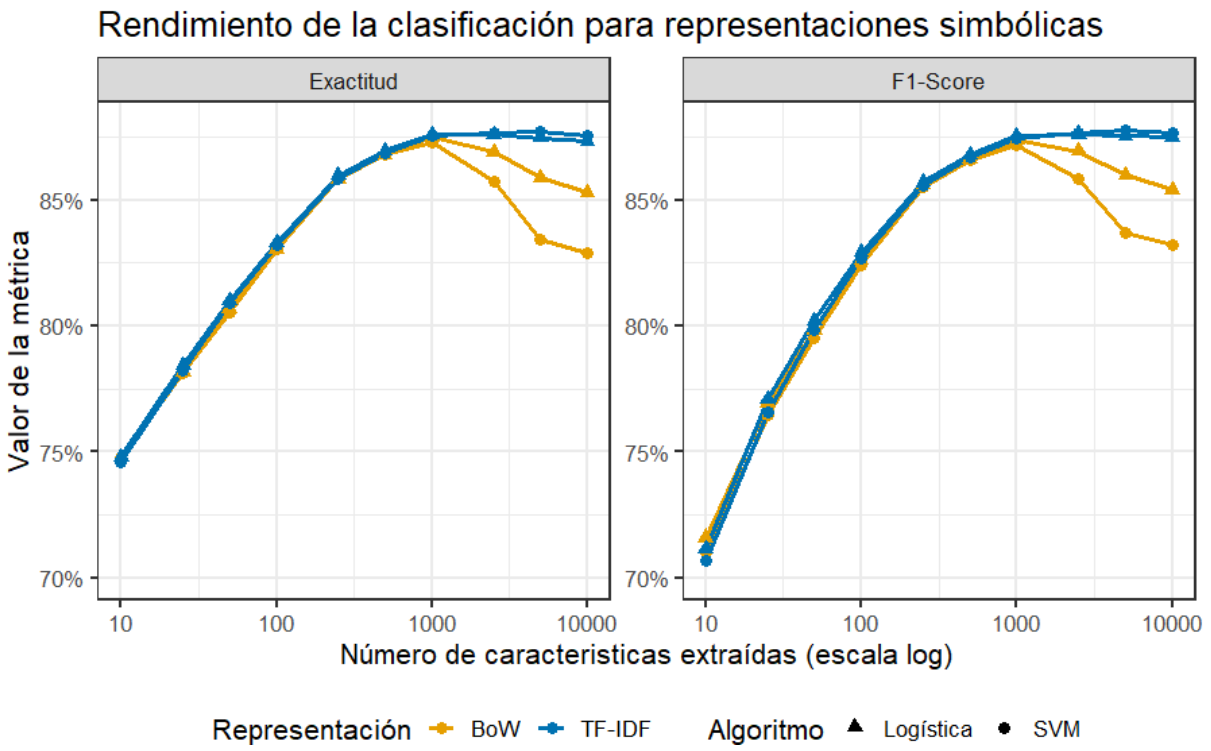


Figura 5.1: Resultados para representaciones simbólicas. Exactitud y *F1-Score* vs el número de características extraídas con el estadístico χ^2, en escala semilogarítmica. Se muestra una línea para cada combinación de los métodos de representación BoW y TF-IDF, y los algoritmos de clasificación SVM y regresión logística, ambos con regularización L2 y selección del parámetro de coste con validación cruzada 10-fold

Metodología de clasificación			**Rendimiento**	
Rep.	**Nº carac.**	**Algoritmo**	**Exactitud**	***F1-Score***
TF-IDF	5.000	SVM	87,7 %	87,8 %
TF-IDF	2.500	SVM	87,7 %	87,7 %
BoW	1.000	Logística	87,5 %	87,4 %
BoW	1.000	SVM	87,3 %	87,2 %

Tabla 5.1: Rendimiento de los 2 mejores clasificadores para cada uno de los métodos de representación simbólicos con extracción de características χ^2

5.3.2. Representaciones distribuidas (LSA, Word2Vec y GloVe)

Los resultados obtenidos se presentan en la Figura 5.2 y en la Tabla 5.2. Como se puede apreciar, el algoritmo de clasificación no tiene una gran influencia sobre el rendimiento final en comparación con el método de representación o del número de dimensiones elegido.

En la Figura 5.2 se puede observar como las representaciones basadas en Word2Vec parecen alcanzar su rendimiento máximo en el entorno de las 100 - 250 dimensiones, sin ninguna mejora adicional hasta las 500 dimensiones. En contraste, GloVe parece poder beneficiarse de una dimensionalidad mayor a 500. Por su parte, LSA destaca como el mejor método de representación en bajas dimensiones, pero tiende a igualarse con Word2Vec a partir de las 25 dimensiones.

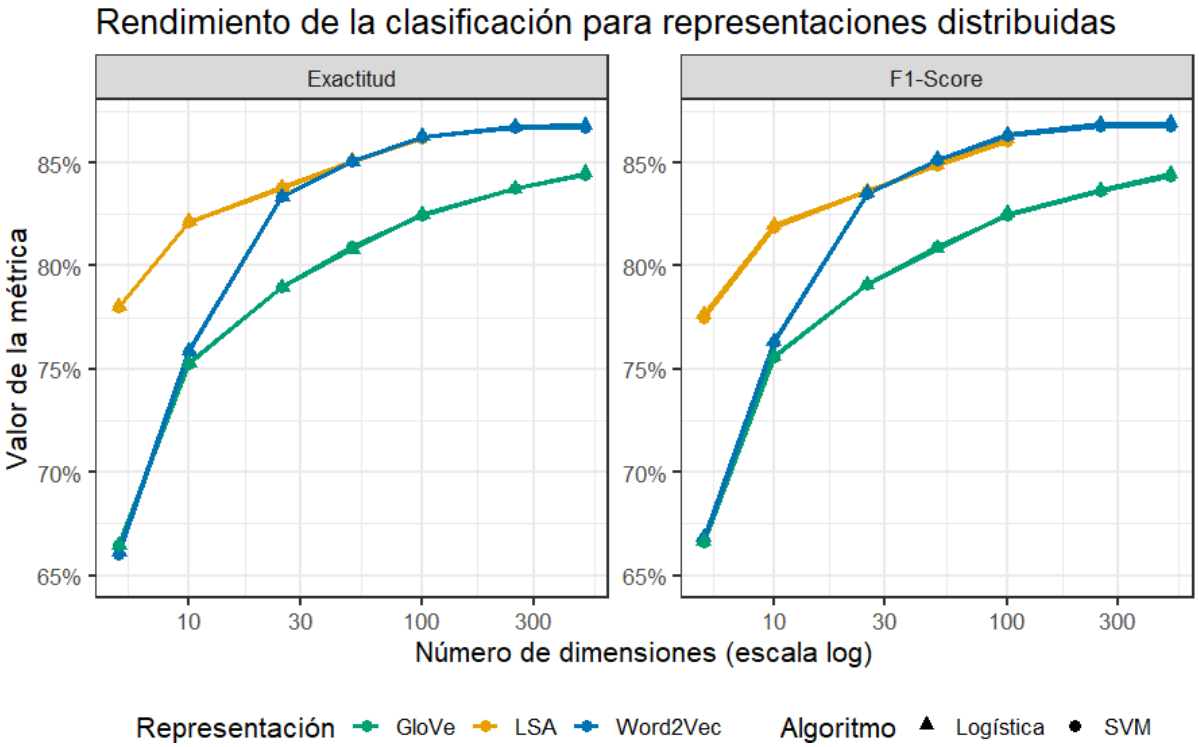


Figura 5.2: Resultados para representaciones distribuidas. Exactitud y *F1-Score* vs el número de dimensiones, en escala semilogarítmica. Se muestra una línea para cada combinación de los métodos de representación LSA, Word2Vec y GloVe, y los algoritmos de clasificación SVM y regresión logística, ambos con regularización L2 y selección del parámetro de coste con validación cruzada 10-fold

La Tabla 5.2 recoge los resultados correspondientes a los dos mejores métodos para cada representación distribuida, redondeados a 3 cifras decimales.

Metodología de clasificación			**Rendimiento**	
Rep.	**Dimensiones**	**Algoritmo**	**Exactitud**	***F1-Score***
Word2Vec	250 y 500	Logística	86,8 %	86,9 %
Word2Vec	250 y 500	SVM	86,7 %	86,8 %
LSA	100	Logística	86,3 %	86,2 %
LSA	100	SVM	86,2 %	86,1 %
GloVe	500	Logística	84,5 %	84,5 %
GloVe	500	SVM	84,4 %	84,3 %

Tabla 5.2: Rendimiento de los 2 mejores clasificadores para cada uno de los métodos de representación distribuidos

5.3.3. Comparación

Las Tablas 5.3 y 5.4 ofrecen una visión global y comparativa de los resultados presentados en este capítulo. La Tabla 5.3 muestra los mejores clasificadores para cada tipo de representación probado, mientras que la Tabla 5.4 hace lo propio a igualdad de dimensiones (100).

Tal y como se observa en la Tabla 5.3, el clasificador SVM aplicado sobre representaciones TF-IDF con 5.000 características extraídas es el que alcanza los mejores rendimientos en términos absolutos, que estaría seguido muy de cerca por el mismo clasificador utilizando 2.500 y 1.000 características. Sin embargo, para un número de dimensiones reducido, los mejores métodos de representación son LSA y Word2Vec, como se puede apreciar en la Tabla 5.4.

Metodología de clasificación			**Rendimiento**	
Rep.	**Dim.**	**Algoritmo**	**Exactitud**	***F1-Score***
TF-IDF	5.000	SVM	87,7 %	87,8 %
BoW	1.000	Logística	87,5 %	87,4 %
Word2Vec	250 y 500	Logística	86,8 %	86,9 %
LSA	100	Logística	86,3 %	86,2 %
GloVe	500	Logística	84,5 %	84,5 %

Tabla 5.3: Resultados de clasificación y mejores parámetros para cada representación probada

Representación	**Rendimiento con logística**		**Rendimiento con SVM**	
	Exactitud	***F1-Score***	**Exactitud**	***F1-Score***
Word2Vec	86,28 %	86,36 %	86,25 %	86,34 %
LSA	86,27 %	86,15 %	86,20 %	86,05 %
TF-IDF	83,32 %	82,91 %	83,19 %	82,66 %
BoW	83,05 %	82,61 %	83,03 %	82,43 %
GloVe	82,50 %	82,54 %	82,44 %	82,45 %

Tabla 5.4: Resultados de clasificación con 100 dimensiones para cada método de representación probado y para los algoritmos de clasificación SVM y regresión logística

Capítulo 6

Conclusiones y trabajo futuro

Este capítulo final está dedicado a recapitular los resultados del trabajo, evaluando los objetivos planteados inicialmente, y destacando las principales conclusiones obtenidas durante el mismo. Además, se proponen posibles líneas de trabajo futuro orientadas a ampliar y profundizar los análisis desarrollados en este trabajo.

6.1. Conclusiones

A lo largo de este trabajo se han expuesto diversas metodologías que permiten aplicar muchas de las técnicas clásicas de análisis de datos al campo de análisis de texto, entre las que se encuentran los métodos de preprocesamiento y representación que se han presentado en este trabajo. Se ha realizado una revisión de las mismas, describiendo sus fundamentos y utilidades, así como sus respectivas limitaciones.

Además, se ha ofrecido una demostración práctica de su uso, mediante implementaciones en R, realizadas de forma previa a tareas como el Análisis de Sentimiento, el Clústering y la Clasificación. Entre las técnicas utilizadas destacan la lematización, el modelo de representación simbólico BoW ponderado por TF-IDF, y los modelos de representación distribuidos LSA y Word2Vec.

También se han explorado técnicas específicas para el análisis de texto en cada una de las tareas realizadas, como los métodos léxicos para Análisis de Sentimiento, o las técnicas de extracción de características supervisadas y no supervisadas para la Clasificación y Clústering, respectivamente.

Adicionalmente, se han presentado diversas técnicas de visualización específicas para datos textuales, ilustradas mediante su aplicación al conjunto de datos *IMDb Movie Reviews*, y al texto de la Ley de la Función Estadística Pública.

Por otro lado, y como se ha mostrado para un problema sencillo como la clasificación de reseñas, una metodología aparentemente poco sofisticada como TF-IDF puede llegar a ser tan efectiva como algunos métodos más complejos como Word2Vec. No obstante, en términos generales, los modelos de representación contextuales siguen siendo los más eficaces, aunque con un significativo aumento del coste computacional.

En conclusión, con la puesta en práctica de las metodologías de procesamiento descritas en este trabajo, y con las ilustraciones realizadas para tres disciplinas muy relevantes de análisis de texto, se han cumplido de forma satisfactoria los objetivos planteados inicialmente. Los resultados obtenidos evidencian la importancia de mantener el equilibrio entre un coste computacional asequible y un funcionamiento adecuado, seleccionando de forma apropiada los métodos de análisis de texto.

6.2. Trabajo futuro

El objetivo de este trabajo ha sido explorar y demostrar la aplicación práctica de un número amplio de técnicas de análisis de texto, lo que ha impedido concentrar los esfuerzos excesivamente en cualquiera de ellas. Es decir, que se han excluido también algunas técnicas, que a pesar de su interés y novedad, pudieran resultar más complejas de presentar. Por tanto, una de las líneas de trabajo futuro de interés es profundizar en estas técnicas, entre las que se encuentran las siguientes:

1. Para clústering y clasificación, explorar el uso de representaciones basadas en *embeddings* contextuales. Estos modelos integran el contexto de las palabras en sus representaciones, lo que resulta esencial para capturar adecuadamente el significado real del texto.

   En R se podría utilizar, por ejemplo, la función `textEmbed()` del paquete `text`[1], que permite utilizar los modelos preeentenados de *HuggingFace*, entre los que se encuentran BERT y cualquiera de sus variantes.

2. Para análisis de sentimiento, explorar el uso de métodos léxicos basados en árboles de derivación gramatical, como los utilizados en [26]. Estas estructuras son capaces de tratar el contexto, imputando adecuadamente los cambios en la polaridad dependiendo de las estructuras gramaticales y otros modificadores, como conjunciones, adjetivos y adverbios.

Otra línea de trabajo futuro adicional sería ampliar las técnicas tratadas en este trabajo, abordando otras como el *topic modeling*, mediante métodos como Latent Dirichlet Allocation (LDA) o Non-negative Matrix Factorization (NMF), u otras más relacionadas al procesamiento de lenguaje natural como la generación de resúmenes y la recuperación de información.

[1] https://cran.r-project.org/web/packages/text/index.html

Bibliografía

[1] Robert Heeg. Possibilities and limitations of unstructured data. https://researchworld.com/articles/possibilities-and-limitations-of-unstructured-data, 2023. Visitado el 6 de julio de 2025.

[2] Ronen Feldman and James Sanger. *The Text Mining Handbook: Advanced Approaches to Analyzing Unstructured Data*. Cambridge university press, 2007.

[3] Charu C. Aggarwal and ChengXiang Zhai. An Introduction to Text Mining. In *Mining Text Data*, pages 1–10. Springer, 2012.

[4] Holly Muscolino, Amy Machado, John Rydning, and Dan Vesset. Untapped Value: What Every Executive Needs to Know About Unstructured Data. White paper US51128223, IDC Research, Inc., 2023.

[5] Amazon Web Services. What is Text Analysis? https://aws.amazon.com/what-is/text-analysis/. Visitado el 6 de junio de 2025.

[6] Jonathan Benchimol, Sophia Kazinnik, and Yossi Saadon. Text Mining methodologies with R: An application to central bank texts. *Machine Learning with Applications*, 8:100286, 2022.

[7] David Bholat, Stephen Hansen, Pedro Santos, and Cheryl Schonhardt-Bailey. Text mining for central banks: handbook. *Centre for Central Banking Studies Handbook*, (33):1–19, 2015.

[8] Martin Schweinberger. *Introduction to Text Analysis*. The Language Technology and Data Analysis Laboratory (LADAL), Brisbane, 2025.04.01 edition, 2025. https://ladal.edu.au/tutorials/introta/introta.html.

[9] IBM. What is Text Mining? https://www.ibm.com/think/topics/text-mining. Visitado el 6 de junio de 2025.

[10] David D. Palmer. Text Preprocessing. In Nitin Indurkhya and Fred J. Damerau, editors, *Handbook of Natural Language Processing*, pages 9–30. Chapman & Hall/CRC, 2 edition, 2010.

[11] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 3rd edition, 2025. Online manuscript released January 12, 2025. https://web.stanford.edu/~jurafsky/slp3/.

[12] Tony McEnery, Richard Xiao, and Yukio Tono. *Corpus-based language studies: An advanced resource book*. Taylor & Francis, 2006.

[13] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching Word Vectors with Subword Information. *arXiv preprint arXiv:1607.04606*, 2016.

[14] Zhiyuan Liu, Yankai Lin, and Maosong Sun. *Representation Learning for Natural Language Processing*. Springer Nature, 2023.

[15] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by Latent Semantic Analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[16] Charu C. Aggarwal and ChengXiang Zhai. A Survey of Text Clustering Algorithms. In *Mining Text Data*, pages 77–128. Springer, 2012.

[17] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[18] Artem Polyvyanyy and Dominik Kuropka. *A Quantitative Evalution of the Enhanced Topic-based Vector Space Model*. 2007.

[19] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781*, 2013.

[20] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of NAACL-HLT, Volume 1*, pages 2227–2237, 2018.

[21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL-HLT, Volume 1*, pages 4171–4186, 2019.

[22] Gang Wu and Eric Martin. String Similarity Metrics – Edit Distance. https://www.baeldung.com/cs/string-similarity-edit-distance, 2024. Visitado el 9 de junio de 2025.

[23] Mushfeq-Us-Saleheen Shameem and Raihana Ferdous. An efficient k-means algorithm integrated with Jaccard distance measure for document clustering. In *2009 First Asian Himalayas International Conference on Internet*, pages 1–6. IEEE, 2009.

[24] Marti Hearst. What is Text Mining. *SIMS, UC Berkeley*, 5, 2003.

[25] Martin Schweinberger. *Sentiment Analysis in R*. The University of Queensland, School of Languages and Cultures, Brisbane, 2025.04.02 edition, 2025. https://ladal.edu.au/tutorials/sentiment/sentiment.html.

[26] Samuel WK Chan and Mickey WC Chong. Sentiment Analysis in financial texts. *Decision Support Systems*, 94:53–64, 2017.

[27] Walaa Medhat, Ahmed Hassan, and Hoda Korashy. Sentiment Analysis algorithms and applications: A survey. *Ain Shams engineering journal*, 5(4):1093–1113, 2014.

[28] Charu C. Aggarwal and ChengXiang Zhai. A Survey of Text Classification Algorithms. In *Mining Text Data*, pages 163–222. Springer, 2012.

[29] Nan Cao and Weiwei Cui. *Introduction to Text Visualization*, volume 1. Springer, 2016.

[30] Kostiantyn Kucher and Andreas Kerren. Text Visualization Techniques: Taxonomy, Visual Survey, and Community Insights. In *2015 IEEE Pacific visualization symposium (pacificVis)*, pages 117–121. IEEE, 2015.

[31] Dipti Sharma, Munish Sabharwal, Vinay Goyal, and Mohit Vij. Sentiment Analysis Techniques for Social Media Data: A Review. In *First International Conference on Sustainable Technologies for Computational Intelligence: Proceedings of ICTSCI 2019*, pages 75–90. Springer, 2020.

[32] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, 2011. Conjunto de datos obtenido de https://ai.stanford.edu/~amaas/data/sentiment/.

[33] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. XLNet: Generalized Autoregressive Pretraining for Language Understanding. *Advances in neural information processing systems*, 32, 2019.

[34] Julia Handl, Joshua Knowles, and Douglas B Kell. Computational Cluster Validation in Post-Genomic Data Analysis. *Bioinformatics*, 21(15):3201–3212, 2005.

[35] Lior Rokach and Oded Maimon. Clustering Methods. In *Data Mining and Knowledge Discovery Handbook*, pages 321–352. Springer, 2005.

[36] Tao Liu, Shengping Liu, Zheng Chen, and Wei-Ying Ma. An Evaluation on Feature Selection for Text Clustering. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 488–495, 2003.

[37] GeeksforGeeks. Non-Negative Matrix Factorization. https://www.geeksforgeeks.org/machine-learning/non-negative-matrix-factorization/, 2025. Visitado el 21 de junio de 2025.

Apéndice A

Código para Análisis de Sentimiento

En el presente anexo se presenta el código desarrollado en este trabajo para aplicar Análisis de Sentimiento. La metodología empleada para el análisis, junto con los resultados obtenidos, puede consultarse en el Capítulo 3. Este capítulo, en la Sección 3.3, incluye adicionalmente la descripción del conjunto de datos utilizado. Siguiendo las convenciones estándar en el desarrollo de software, el código ha sido escrito en inglés, a excepción de los comentarios y cadenas de texto. Para mayor comodidad, también se puede descargar el código a través del repositorio público de GitHub del trabajo (https://github.com/acabrerod/text-analysis), en su versión en inglés.

Este anexo está dividido en seis secciones, con la siguiente estructura:

- Sección A.1 – Listado y carga de los paquetes utilizados.
- Sección A.2 – Lectura de datos.
- Sección A.3 – Preprocesamiento de datos.
- Sección A.4 – Aplicación del Análisis de Sentimiento, dividida en cuatro subsecciones, una por cada paquete de análisis de sentimiento utilizado.
- Sección A.5 – Evaluación del rendimiento de las polaridades obtenidas.
- Sección A.6 – Tablas de resultados finales.

A.1. Paquetes utilizados

```
# Paquetes para lectura, manipulacion de datos y evaluacion
library(here)
library(readr)
library(dplyr)
library(tidyr)
library(yardstick)

# Paquetes de preprocesamiento y analisis de texto
library(tm)        # Para crear corpus y limpiarlos
library(textstem)  # Para lematizacion
library(textclean) # Para normalizacion

```

```r
# Paquetes de manipulacion de texto
library(stringi)
library(stringr)

# Paquetes de Analisis de Sentimiento
library(syuzhet)
library(tidytext)
library(sentimentr)
library(SentimentAnalysis)
```

A.2. Lectura de datos

Los datos se han descargado los datos del siguiente enlace: https://ai.stanford.edu/~amaas/data/sentiment/, aunque también se pueden encontrar en el repositorio de código.

Para importar los datos al entorno de R, se ha definido la función `read.imdb`. Y con ella se han creado los siguientes dos objetos:

- `imdb`: Vector de caracteres con las 25.000 reseñas del conjunto de *test*.
- `imdb.true`: Factor con la categoría real de las reseñas (positivas/negativas).

```r
read.imdb <- function(path, polarity) {
   # Funcion para leer todos los ficheros de texto de un directorio
   #     base 'path' al que se le anade un subdirectorio 'polarity'
   # path - string con la ruta a los datos
   # polarity - string con "pos" o "neg"
   files.list <- list.files(
      path = paste(path, "/", polarity, sep = "") %>% here(),
      pattern = "\\.txt$", full.names = TRUE
   )
   files.vector <- sapply(files.list, read_file, USE.NAMES = FALSE)
   return(files.vector)
}

# Ubicacion de los datos
files.test <- "datos/imdb-test"

# Lectura del conjunto de test
imdb.test.pos <- read.imdb(files.test, "pos")
imdb.test.neg <- read.imdb(files.test, "neg")

imdb <- c(imdb.test.pos, imdb.test.neg)
imdb.true <- c(
rep("positive", length(imdb.test.pos)),
rep("negative", length(imdb.test.neg))
) %>% as.factor()
```

A.3. Preprocesamiento

Para preprocesar las reseñas, se convierten a un objeto de clase `Corpus` del paquete `tm`, y se le aplican las transformaciones que se indican en el código. El resultado es un nuevo vector

de caracteres con las reseñas preprocesadas. A continuación del código se muestra un ejemplo de una reseña breve antes y después del preprocesamiento.

```
imdb.preprocess <- function(imdb) {
   # Funcion de preprocesamiento
   imbd.corpus <- Corpus(VectorSource(imdb))

   # 1. Eliminar etiquetas HTML
   imbd.corpus <- tm_map(imbd.corpus, content_transformer(function(x) gsub("
      <.*?>", " ", x)))

   # 2. Conversion a minusculas
   imbd.corpus <- tm_map(imbd.corpus, content_transformer(tolower))

   # 3. Expansion de contracciones
   imbd.corpus <- tm_map(imbd.corpus, content_transformer(function(x)
      replace_contraction(x)))

   # 4. Lematizacion
   imbd.corpus <- tm_map(imbd.corpus, content_transformer(function(x)
      lemmatize_strings(x)))

   return(sapply(imbd.corpus, as.character))
}
```

Ejemplo de una reseña breve (la vigésimo octava positiva del conjunto *test*) antes y después del preprocesamiento:

```
# Antes del preprocesamiento:
> imdb[28]
[1] "Felix in Hollywood is a great film. The version I viewed was very well
   restored, which is sometimes a problem with these silent era animated
   films. It has some of Hollywood's most famous stars making cameo animated
    appearances. A must for any silent film or animation enthusiast."

# Despues del preprocesamiento:
> imdb.preprocess(imdb[28])
[1] "felix in hollywood be a great film. the version i view be very good
   restore, which be sometimes a problem with this silent era animate film.
   it have some of hollywood's much famous star make cameo animate
   appearance. a must for any silent film or animation enthusiast."
```

```
imdb <- imdb.preprocess(imdb)
```

A.4. Análisis de Sentimiento

Se han utilizado los siguientes paquetes de Análisis de Sentimiento: `syuzhet`, `tidytext`, `sentimentr` y `SentimentAnalysis`. Para cada uno de ellos, se crean los siguientes objetos, donde la notación "{paquete}", indica que se sustituye por el nombre del paquete correspondiente:

- {paquete}`.dicts`: Vector o lista de los diccionarios que se usarán de ese paquete.

- {paquete}.`results`: Data frame que almacena las puntuaciones sentimentales de las reseñas con cada diccionario de ese paquete.
- {paquete}.`imdb` (solo para tidytext y sentimentr): Preprocesamiento adicional de las reseñas. Para tidytext conversión a tibble con numeración de reseña, y para sentimentr conversión a oraciones individuales.

Después, se utilizan las respectivas funciones de Análisis de Sentimiento de cada paquete para todos los diccionarios, y se almacenan para su posterior evaluación (en la Sección A.5).

A.4.1. `syuzhet`

```
syuzhet.dicts <- c("Bing", "AFINN", "NRC", "Syuzhet")
syuzhet.results <- matrix(
   0, nrow = length(imdb), ncol = length(syuzhet.dicts),
   dimnames = list(NULL, syuzhet.dicts)
) %>% data.frame()

# Se calcula el sentimiento con la funcion get_sentiment
for(dictionary in syuzhet.dicts) {
   syuzhet.results[[dictionary]] <- get_sentiment(
      imdb, method = dictionary %>% tolower()
   )
}
```

A.4.2. `tidytext`

```
tidytext.dicts <- list(
   "Bing" = get_sentiments("bing"),
   "AFINN" = get_sentiments("afinn"),
   "NRC" = get_sentiments("nrc") %>%
   filter(sentiment %in% c("positive", "negative"))
)
tidytext.imdb <- tibble(
      review.number = 1:length(imdb),
      review = imdb
   ) %>%
   unnest_tokens(word, review)
tidytext.results <- matrix(
      0, nrow = length(imdb), ncol = length(tidytext.dicts),
      dimnames = list(NULL, names(tidytext.dicts))
   ) %>% data.frame()

# Se calcula el sentimiento con un left join. Las reviews tokenizadas
#     en palabras se emparejan con las palabras presentes en los
#     respectivos diccionarios. El calculo final de la polaridad
#     depende del diccionario utilizado.
#     - Para Bing y NRC: Recuento del numero de palabras positivas
#         menos las negativas
#     - Para AFINN: Suma de las polaridades de cada palabra
for(dictionary in c("Bing", "NRC")) {
   tidytext.results[[dictionary]] <- tidytext.imdb %>%
      left_join(tidytext.dicts[[dictionary]], by = "word") %>%
```

```
      count(review.number, sentiment) %>%
      pivot_wider(names_from = sentiment, values_from = n, values_fill = 0)
          %>%
      mutate(sentiment.score = positive - negative) %>%
      pull(sentiment.score)
}
tidytext.results$AFINN <- tidytext.imdb %>%
   left_join(tidytext.dicts[["AFINN"]], by = "word") %>%
   group_by(review.number) %>%
   summarise(sentiment.score = sum(value, na.rm = T)) %>%
   pull(sentiment.score)
```

A.4.3. sentimentr

```
sentimentr.imdb <- imdb %>% get_sentences()
sentimentr.dicts <- list(
   "Syuzhet" = lexicon::hash_sentiment_jockers,
   "Bing" = lexicon::hash_sentiment_huliu,
   "Loughran" = lexicon::hash_sentiment_loughran_mcdonald,
   "NRC" = lexicon::hash_sentiment_nrc,
   "SentiWordNet" = lexicon::hash_sentiment_sentiword,
   "Jockers-Rinker" = lexicon::hash_sentiment_jockers_rinker
)
sentimentr.results <- matrix(
   0, nrow = length(imdb), ncol = length(sentimentr.dicts),
   dimnames = list(NULL, names(sentimentr.dicts))
) %>% data.frame()

# Se calcula el sentimiento con la funcion sentiment, pero
#     como este paquete calcula la polaridad por cada oracion
#     individual, la polaridad final se calcula sumando las
#     polaridades de todas las oraciones de cada review
for(dictionary in names(sentimentr.dicts)) {
   sentimentr.results[[dictionary]] <-
      sentiment(sentimentr.imdb, sentimentr.dicts[[dictionary]]) %>%
      group_by(element_id) %>%
      summarise(sentiment = sum(sentiment)) %>%
      pull(sentiment)
}
```

A.4.4. SentimentAnalysis

```
# Los numeros indican el indice de cada diccionario en el data frame
#     que devuelve el paquete al llamar a analyzeSentiment
sa.dicts <- c("GI" = 2, "Henry" = 5, "Loughran" = 8, "QDAP" = 12)

# Se calcula el sentimiento con la funcion analyzeSentiment, aunque debido
#     a que el paquete utiliza demasiada memoria, se ha fraccionado el
#     calculo en 4 llamadas secuenciales.
#     Esta funcion devuelve un dataframe con las polaridades, ya calculadas
#     para todos los diccionarios.
sa.results <- rbind(
   analyzeSentiment(imdb[1:6250]),
```

```
    analyzeSentiment(imdb[6251:12500]),
    analyzeSentiment(imdb[12501:18750]),
    analyzeSentiment(imdb[18751:25000])
)
sa.results[is.na(sa.results)] <- 0
```

A.5. Evaluación del rendimiento

Para almacenar las medidas de validación, se crean dos tablas, una para la exactitud, y otra para *F1-Score*. Para el cálculo se han utilizado las funciones `accuracy_vec` y `f_meas_vec` del paquete `yardstick`.

```
# Columnas de las tablas
dictionaries <- c(
    "GI", "Bing", "AFINN", "NRC", "Syuzhet",
    "QDAP", "SentiWordNet", "Henry", "Loughran",
    "Jockers-Rinker"
)
# Filas de las tablas
packages <- c(
    "syuzhet", "tidytext",
    "sentimentr", "SentimentAnalysis"
)

# Creacion de las tablas
f1 <- accuracy <- matrix(NA,
    nrow = length(dictionaries),
    ncol = length(packages),
    dimnames = list(dictionaries, packages)
)

# Calculo del rendimiento para syuzhet
for(dictionary in syuzhet.dicts) {
    pred <- ifelse(syuzhet.results[[dictionary]] > 0, "positive", "negative")
        %>% as.factor()

    accuracy[dictionary, "syuzhet"] <- accuracy_vec(imdb.true, pred)
    f1[dictionary, "syuzhet"] <- f_meas_vec(imdb.true, pred)
}

# Calculo del rendimiento para tidytext
for(dictionary in names(tidytext.dicts)) {
    pred <- ifelse(tidytext.results[[dictionary]] > 0, "positive", "negative"
        ) %>% as.factor()

    accuracy[dictionary, "tidytext"] <- accuracy_vec(imdb.true, pred)
    f1[dictionary, "tidytext"] <- f_meas_vec(imdb.true, pred)
}

# Calculo del rendimiento para sentimentr
for(dictionary in names(sentimentr.dicts)) {
    pred <- ifelse(sentimentr.results[[dictionary]] > 0, "positive", "
        negative") %>% as.factor()

    accuracy[dictionary, "sentimentr"] <- accuracy_vec(imdb.true, pred)
```

```
41    f1[dictionary, "sentimentr"] <- f_meas_vec(imdb.true, pred)
42 }
43
44 # Calculo del rendimiento para SentimentAnalysis
45 for(dictionary in names(sa.dicts)) {
46    pred <- ifelse(sa.results[[sa.dicts[dictionary]]] > 0, "positive", "
          negative") %>% as.factor()
47
48    accuracy[dictionary, "SentimentAnalysis"] <- accuracy_vec(imdb.true, pred
          )
49    f1[dictionary, "SentimentAnalysis"] <- f_meas_vec(imdb.true, pred)
50 }
```

A.6. Resultados

A continuación se muestran los resultados obtenidos, que coinciden con los que se han presentado en el Capítulo 3, en las Tablas 3.3 y 3.5. Adicionalmente, se muestran al final los resultados para el diccionario Jockers-Rinker, cuyos resultados se omitieron de dichas tablas.

```
> round(accuracy, 4)
syuzhet tidytext sentimentr SentimentAnalysis
GI             NA     NA       NA         0.6410
Bing         0.7279 0.7313  0.7583           NA
AFINN        0.7103 0.6904     NA            NA
NRC          0.6567 0.6584  0.6925           NA
Syuzhet      0.6887    NA    0.7267           NA
QDAP            NA     NA       NA         0.6626
SentiWordNet    NA     NA    0.6331           NA
Henry           NA     NA       NA         0.5718
Loughran        NA     NA    0.6853         0.6514
Jockers-Rinker NA      NA    0.7355           NA

> round(f1, 4)
syuzhet tidytext sentimentr SentimentAnalysis
GI             NA     NA       NA         0.5336
Bing         0.7492 0.7389  0.7554           NA
AFINN        0.6683 0.6262     NA            NA
NRC          0.5840 0.5818  0.6345           NA
Syuzhet      0.6356    NA    0.6857           NA
QDAP            NA     NA       NA         0.5762
SentiWordNet    NA     NA    0.5762           NA
Henry           NA     NA       NA         0.4332
Loughran        NA     NA    0.6864         0.7028
Jockers-Rinker NA      NA    0.7084           NA
```

Apéndice B

Código para Clústering

En el presente anexo se presenta el código desarrollado en este trabajo para aplicar Clústering La metodología empleada para el análisis, junto con los resultados obtenidos, puede consultarse en el Capítulo 4. Siguiendo las convenciones estándar en el desarrollo de software, el código ha sido escrito en inglés, a excepción de los comentarios y cadenas de texto. Para mayor comodidad, también se puede descargar el código a través del repositorio público de GitHub del trabajo (https://github.com/acabrerod/text-analysis), en su versión en inglés.

Este anexo está dividido en ocho secciones, con la siguiente estructura:

- Sección B.1 – Listado y carga de los paquetes utilizados.
- Sección B.2 – Lectura de datos.
- Sección B.3 – Preprocesamiento de datos.
- Sección B.4 – Cálculo de las representaciones simbólicas.
- Sección B.5 – Cálculo de las métricas de extracción de características. En la Tabla B.1 se muestran las 10 palabras con mejor puntuación en cada una de las métricas.
- Sección B.6 – Cálculo de las representaciones distribuidas.
- Sección B.7 – Aplicación del Clústering, organizada en cinco subsecciones, en las cuatro primeras se configuran y preparan funciones, y en la última se muestra el código para ejecutar las pruebas.
- Sección B.8 – Código para obtener los gráficos de resultados.

B.1. Paquetes utilizados

```
# Paquetes para lectura, manipulacion de datos y evaluacion
library(here)
library(readr)
library(dplyr)
library(tidyr)
library(yardstick)
```

```r
library(parallel)
library(fpc)
library(lubridate)
library(Matrix)


# Paquetes de preprocesamiento y analisis de texto
library(tm)         # Para crear y limpiar corpus
library(textstem)   # Para lematizacion
library(textclean)  # Para normalizacion
library(text2vec)   # Para representaciones BoW, TF-IDF, LSA y GloVe
library(word2vec)   # Para representacion Word2Vec
library(proxy)      # Para calcular distancias coseno

# Paquetes de manipulacion de texto
library(stringi)
library(stringr)


# Paquetes para graficos de resultados
library(ggplot2)
library(ggrepel)
library(scales)
```

B.2. Lectura de datos

Los datos se han descargado los datos del siguiente enlace: https://ai.stanford.edu/~amaas/data/sentiment/, aunque también se pueden encontrar en el repositorio de código.

Para importar los datos al entorno de R, se ha definido la función `read.imdb` en la Sección A.2, con la que se se crean los objetos `imdb` e `imdb.true`.

```r
# Ubicacion de los datos
files.test <- "datos/imdb-test"

# Lectura del conjunto de test
imdb.test.pos <- read.imdb(files.test, "pos")
imdb.test.neg <- read.imdb(files.test, "neg")

imdb <- c(imdb.test.pos, imdb.test.neg)
imdb.true <- c(
   rep("positive", length(imdb.test.pos)),
   rep("negative", length(imdb.test.neg))
) %>% as.factor()
```

B.3. Preprocesamiento

Para el preprocesamiento se sigue el mismo procedimiento que en la Sección A.3, eliminando adicionalmente números y *stop words*. A continuación del código se muestra un ejemplo de una reseña antes y después del preprocesamiento.

```
imdb.preprocess <- function(imdb) {
   # Funcion de preprocesamiento
   imbd.corpus <- Corpus(VectorSource(imdb))

   # 1. Eliminar etiquetas HTML
   imbd.corpus <- tm_map(imbd.corpus, content_transformer(function(x) gsub("
      <.*?>", " ", x)))

   # 2. Conversion a minusculas
   imbd.corpus <- tm_map(imbd.corpus, content_transformer(tolower))

   # 3. Eliminacion de numeros
   imbd.corpus <- tm_map(imbd.corpus, removeNumbers)

   # 4. Eliminacion de stopwords
   imbd.corpus <- tm_map(imbd.corpus, removeWords, stopwords("en"))

   # 5. Expansion de contracciones
   imbd.corpus <- tm_map(imbd.corpus, content_transformer(function(x)
      replace_contraction(x)))

   # 6. Lematizacion
   imbd.corpus <- tm_map(imbd.corpus, content_transformer(function(x)
      lemmatize_strings(x)))

   return(sapply(imbd.corpus, as.character))
}
```

Ejemplo de una reseña breve (la vigésimo octava positiva del conjunto *test*) antes y después del preprocesamiento:

```
# Antes del preprocesamiento:
> imdb[28]
[1] "Felix in Hollywood is a great film. The version I viewed was very well
   restored, which is sometimes a problem with these silent era animated
   films. It has some of Hollywood's most famous stars making cameo animated
    appearances. A must for any silent film or animation enthusiast."

# Despues del preprocesamiento:
> imdb.preprocess(imdb[28])
[1] "felix hollywood great film. version view good restore, sometimes
   problem silent era animate film. hollywood's famous star make cameo
   animate appearance. must silent film animation enthusiast."
```

```
imdb <- imdb.preprocess(imdb)
```

B.4. Representaciones simbólicas: BoW y TF-IDF

Se obtienen las matrices `tcm` (*term co-occurrence matrix*), `dtm` (*document-term matrix*) y `tfidf` utilizando el paquete `text2vec`. La matriz de coocurrencia es necesaria para GloVe.

```
# Tokeniza el corpus, creando una lista de vectores de palabras
tokens <- word_tokenizer(imdb)

# Crea un iterador a partir de los tokens
it <- itoken(tokens, progressbar = TRUE)

# Construye el vocabulario, que es una tabla con todas las
#     palabras que aparecen junto a su frecuencia en el corpus.
#     Se utiliza para crear las matrices termino-documento
vocab <- create_vocabulary(it)

# Se eliminan los terminos menos frecuentes
vocab <- prune_vocabulary(vocab, term_count_min = 4)

# Crea un vectorizador, que sirve para crear las matrices
#     de representacion con el vocabulario dado
vectorizer <- vocab_vectorizer(vocab)

# Matriz de coocurrencia (Term Co-occurrence Matrix), con
#     una ventana de contexto de 5 palabras. Se necesita
#     para GloVe posteriormente
tcm <- create_tcm(it, vectorizer, skip_grams_window = 5L)

# Crea la matriz documento-termino (Document-Term Matrix), que
#     es la base para la representacion BoW
dtm <- create_dtm(it, vectorizer)

# Crea y ajusta un modelo TF-IDF, que calcula las ponderaciones
#     para el corpus, y devuelve la matriz para la representacion
#     TF-IDF
tfidf.model <- TfIdf$new()
tfidf <- tfidf.model$fit_transform(dtm)
```

B.5. Extracción de características

Al no haber encontrado un paquete que calculase estas métricas, se ha optado por implementarlas manualmente, excepto para *Document Frequency* (DF). El cálculo se realiza en paralelo, y se almacenan ordenadas por importancia en los siguientes 4 vectores. El cálculo para DF y TC es exacto, mientras que el de TS y EBR se ha estimado tomando una muestra de reseñas, de tamaños 5.000 y 200, respectivamente.

- `document.frecuency`
- `term.strength`
- `entropy.based.ranking`
- `term.contribution`

Además, en la Tabla B.1 se muestran las 10 palabras más relevantes según cada métrica de extracción de características. Se puede observar como DF, TS y TC identifican de manera similar las palabras más relevantes, lo cual es coherente con los resultados correlacionados que obtienen. Asimismo, esta coincidencia explica la divergencia de EBR, que presenta una baja correlación con las demás.

DF	TS	EBR	TC
good	*movie*	*monster*	*movie*
movie	*good*	*man*	*film*
film	*see*	*ever*	*good*
one	*one*	*scene*	*see*
see	*film*	*life*	*one*
make	*make*	*play*	*like*
like	*like*	*great*	*make*
just	*just*	*bad*	*bad*
get	*get*	*horror*	*watch*
time	*watch*	*act*	*just*

Tabla B.1: Las 10 palabras más relevantes para cada métrica de extracción de características

B.5.1. Document Frecuency (DF)

```
# Se obtiene la document frequency del vocabulario calculado
#     anteriormente con create_vocabulary
document.frequency <- vocab$doc_count
names(document.frequency) <- vocab$term
document.frequency <- document.frequency[order(document.frequency,
    decreasing = TRUE)]
```

B.5.2. Term Strength (TS)

```
# Para calcular term strength, se extrae una muestra de 5.000 reviews
# Para medir la similaridad entre las reviews, se utiliza la similaridad
#     del coseno a partir de la matriz TF-IDF
set.seed(456)
ts.sample <- sample(1:25000, 5000)
tfidf.ts.sample <- tfidf[ts.sample, ]
tfidf.similarity <- sim2(tfidf.ts.sample, method = "cosine", norm = "l2")


# Se crea una nueva matriz documento-termino para la muestra
ts.tokens <- word_tokenizer(imdb[ts.sample])
ts.it <- itoken(ts.tokens, progressbar = TRUE)
ts.vocab <- create_vocabulary(ts.it)
# Le quitamos los terminos al vocabulario que no se incluyen en
#     el vocabulario general de BoW y TF-IDF
ts.vocab <- ts.vocab[ts.vocab$term %in% vocab$term, ]
ts.vectorizer <- vocab_vectorizer(ts.vocab)
ts.dtm <- dtm[ts.sample, ]
length(ts.vocab$term) # -> 21.186


# Para el calculo de TS se extraen las 80.000 parejas de reviews
#     mas proximas, correspondientes al 4% de las menores
#     distancias
ts.cutoff <- quantile(tfidf.similarity, probs = 0.96)
ts.related.reviews <- as.matrix(tfidf.similarity) > ts.cutoff
ts.related.reviews <- which(ts.related.reviews, arr.ind = TRUE)
```

```
ts.related.reviews <- ts.related.reviews[ts.related.reviews[,1] < ts.related
    .reviews[,2], ]


# Funcion para calcular Term Strength (TS)
calculate.ts.for.term <- function(term, ts.dtm, ts.related.reviews) {
    # Calcula TS para term, dadas las parejas de reviews relacionadas
    #     en ts.related.reviews. Para ello, calcula cociente entre el
    #     numero de reviews donde term esta en ambas, y el numero de
    #     reviews donde esta solo en la primera de las parejas
    term.index <- which(colnames(ts.dtm) == term)
    term.present <- ts.dtm[, term.index] > 0

    in.first <- term.present[ts.related.reviews[, 1]]
    in.second <- term.present[ts.related.reviews[, 2]]

    count.in.first <- sum(in.first)
    count.in.both <- sum(in.first & in.second)

    return(count.in.both / count.in.first)
}

# Vector para almacenar los resultados
term.strength <- numeric(length(ts.vocab$term))
names(term.strength) <- ts.vocab$term


# Creacion de un cluster para ejecucion en paralelo
cl <- makeCluster(detectCores() - 1)
clusterExport(cl, varlist = c("ts.dtm", "ts.related.reviews", "calculate.ts.
    for.term"), envir = environment())

# Calculo de TS para todos los terminos de la muestra
term.strength <- parSapply(cl, ts.vocab$term,
    function(term) calculate.ts.for.term(term, ts.dtm, ts.related.reviews)
)
stopCluster(cl)

term.strength[is.nan(term.strength)] <- 0
term.strength <- term.strength[order(term.strength, decreasing = TRUE)]
```

B.5.3. Entropy-based Ranking (EBR)

```
# Para calcular EBR se extrae una muestra de 200 reviews. No se ha
#     podido extraer una muestra mas grande ya que este calculo es
#     muy caro. Se necesita calcular matrices de distancias tantas
#     veces como palabras haya en el vocabulario.
#
# En la muestra de 200 reviews hay 5.000 palabras unicas, por lo que
#     se calculan 5.000 matrices de distancia de dimension 199 x 199
#     Al incluir mas reviews en la muestra, el numero de palabras
#     aumentaria pregoresivamente hasta las 25.000, con un crecimiento
#     mas rapido al principio y mas lento despues

set.seed(4567)
ebr.sample <- sample(1:25000, 200)
```

```
ebr.tokens <- word_tokenizer(imdb[ebr.sample])
ebr.it <- itoken(ebr.tokens, progressbar = TRUE)
ebr.vocab <- create_vocabulary(ebr.it)
# Le quitamos los terminos al vocabulario que no se incluyen en
#     el vocabulario general de BoW y TF-IDF
ebr.vocab <- ebr.vocab[ebr.vocab$term %in% vocab$term, ]
ebr.vectorizer <- vocab_vectorizer(ebr.vocab)
ebr.dtm <- create_dtm(ebr.it, ebr.vectorizer)
length(ebr.vocab$term) # -> 5.360


# Funcion para calcular EBR
calculate.ebr.for.term <- function(term, ebr.dtm) {
   # Calcula EBR para term, para lo cual lo elimina de la matriz
   #     documento-termino (BoW) y calcula una matriz de distancias
   #     con la similaridad del coseno.
   term.index <- which(colnames(ebr.dtm) == term)
   dtm.without.term <- ebr.dtm[, -term.index, drop=FALSE]

   dist.mat <- proxy::dist(as.matrix(dtm.without.term), method = "cosine")
   dist.mat <- as.matrix(dist.mat)

   mean.dist <- mean(dist.mat[lower.tri(dist.mat)])
   S <- 2^(-dist.mat / mean.dist)
   eps <- 1e-10
   S <- pmin(pmax(S, eps), 1- eps)

   entropy <- S * log2(S) + (1 - S) * log2(1 - S)
   return(-sum(entropy))
}

# Vector para almacenar los resultados
entropy.based.ranking <- numeric(length(ebr.vocab$term))
names(entropy.based.ranking) <- ebr.vocab$term


# Creacion de un cluster para ejecucion en paralelo
cl <- makeCluster(detectCores() - 1)
clusterExport(cl, varlist = c("ebr.dtm", "calculate.ebr.for.term"), envir =
   environment())
clusterEvalQ(cl, library(proxy))
clusterEvalQ(cl, library(Matrix))

# Calculo de EBR para todos los terminos de la muestra
entropy.based.ranking <- parSapply(cl, ebr.vocab$term,
   function(t) calculate.ebr.for.term(t, ebr.dtm)
)
stopCluster(cl)

# Se cambia de signo para tenerlas en el mismo orden que las demas
#     metricas de extraccion de caracteristicas, o sea, decrecientes
#     de mejor a peor
entropy.based.ranking <- -entropy.based.ranking
entropy.based.ranking <- entropy.based.ranking[order(entropy.based.ranking,
   decreasing = TRUE)]
```

B.5.4. Term Contribution (TC)

```
# Funcion para calcular TC
calculate.tc.for.term <- function(term, tfidf) {
   # Calcula TC para term, a partir de su vector en la matriz de
   #     ponderaciones TF-IDF. Se aprovecha que el siguiente
   #     producto escalar: sum(vec[i] * vec[j]) para i != j,
   #     es equivalente a: (sum(vec)^2 - sum(vec^2))
   term.index <- which(colnames(tfidf) == term)
   term.vector <- tfidf[, term.index]
   return(sum(term.vector)^2 - sum(term.vector^2))
}

# Vector para almacenar los resultados
term.contribution <- numeric(length(vocab$term))
names(term.contribution) <- vocab$term


# Creacion de un cluster para ejecucion en paralelo
cl <- makeCluster(detectCores() - 1)
clusterExport(cl, varlist = c("tfidf", "calculate.tc.for.term"), envir =
   environment())

# Calculo de TC para todos los terminos del corpus
term.contribution <- parSapply(cl, vocab$term, function(t) calculate.tc.for.
   term(t, tfidf))
stopCluster(cl)

term.contribution <- term.contribution[order(term.contribution, decreasing =
    TRUE)]
```

B.6. Representaciones distribuidas: LSA, Word2Vec y GloVe

Las representaciones se calculan utilizando las dimensiones especificadas por el vector de `dimensions`, excepto en el caso de LSA, cuya representación está limitada a un máximo de 100. No es necesario almacenar los modelos, ya que solo se utilizan una vez como preprocesamiento.

```
dimensions <- c(5, 10, 25, 50, 100, 250, 500)
```

Las matrices resultantes de cada método de representación se almacenarán en las siguientes tres listas, además de en objetos `RData` en disco.

- `lsa.embeddings.list`
- `word2vec.embeddings.list`
- `glove.embeddings.list`

Latent Semantic Analysis (LSA)

Se utiliza el paquete `text2vec`, que implementa LSA con un algoritmo del tipo de *soft impute*, diseñado para factorizar matrices dispersas como las de término-documento, y que permite que el cálculo tenga un coste razonable.

La factorización SVD tendría un coste de $O(k \cdot n \cdot m)$, que para este caso valen lo siguiente: $n = 25.000$, $m = 25.371$, y k es variable con el número de dimensiones seleccionado, desde 5 hasta 100.

```
# Funcion para calcular los embeddings con LSA
get.lsa.embeddings <- function(d) {
   # Calcula la factorizacion SVD mediante un algoritmo del
   #     tipo soft impute a traves del paquete text2vec, lo que
   #     posibilita que se pueda hacer el analisis, ya que una
   #     factorizacion al uso tendria un coste muy elevado
   lsa.model <- LatentSemanticAnalysis$new(n_topics = d)
   lsa.embeddings <- lsa.model$fit_transform(tfidf)
   return(lsa.embeddings)
}

# LSA (aprox 1 minuto)
set.seed(456)
lsa.embeddings.list <- sapply(dimensions[dimensions < 250], get.lsa.
   embeddings)
save(lsa.embeddings.list, file = "representations-distributed-LSA.RData") #
   -> 35 MB
```

Word2Vec y GloVe

Se utiliza el paquete `text2vec` para calcular los *embeddings* de las palabras (*word vectors*) con GloVe, y `word2vec` para el modelo *Continuous-bag of words* (CBOW), es decir, Word2Vec. Para calcular los *embeddings* de las reseñas a partir de los *word vectors*, se toma el vector promedio de las palabras de las reseñas.

```
# Tokens es una lista de vectores de caracteres, una para cada review y
#     que contiene un vector con las palabras. Se deben eliminar las
#     palabras que no se han considerado para las representaciones
#     simbolicas (frecuencia total < 4)
tokens <- word_tokenizer(imdb)
valid.words <- colnames(dtm)
tokens.filtered <- lapply(tokens, function(x) x[x %in% valid.words])


# Calcula los wordvectors con Word2Vec, es decir, las representaciones
#     vectoriales de todos los terminos del corpus. Utiliza la funcion
#     word2vec, que permite ajustar el modelo en paralelo
get.word2vec.wordvectors <- function(d) {
   word2vec.model <- word2vec(
      tokens.filtered, type = "cbow", dim = d,
      iter = 20, threads = detectCores() - 1
   )
   word2vec.wv <- as.matrix(word2vec.model)
   return(word2vec.wv)
```

```
}
# Calcula los wordvectors con GloVe, con el paquete text2vec
get.glove.wordvectors <- function(d) {
   glove.model <- GlobalVectors$new(
      rank = d, x_max = 10, learning_rate = 0.05
   )
   glove.fit <- glove.model$fit_transform(
      tcm, n_iter = 20, n_threads = detectCores() - 1
   )
   glove.wv = glove.fit + t(glove.model$components)
   return(glove.wv)
}

# Calcula una matriz de embeddings para todas las reviews con un numero
#    de dimensiones d, promediando los wordvectors de cada palabra de la
#    review, para los modelos Word2Vec y GloVe
get.embeddings <- function(d, model) {
   wv <- if (model == "Word2Vec") { get.word2vec.wordvectors(d) }
   else if (model == "GloVe") { get.glove.wordvectors(d) }

   # Filtra las palabras validas, que tienen representacion
   #     tanto en simbolicas como para el modelo correspondiente
   valid.words <- rownames(wv)[rownames(wv) %in% colnames(dtm)]

   # Multiplicacion de la matriz documento-termino por wordvectors,
   #     que obtiene para cada review, la suma total de los
   #     wordvectors ponderada por su frecuencia en la review
   embeddings <- as.matrix(dtm[, valid.words] %*% wv[valid.words, ])

   # Division por el numero total de palabras en cada review
   return(sweep(embeddings, 1, rowSums(dtm), FUN = "/"))
}

set.seed(456)

# Word2Vec (aprox 5 minutos)
word2vec.embeddings.list <- sapply(
   dimensions,
   function(dim) get.embeddings(dim, "Word2Vec")
)
save(
   word2vec.embeddings.list,
   file = "representations-distributed-Word2Vec.RData"
) # -> 162 MB


# GloVe (aprox 10 minutos)
glove.embeddings.list <- sapply(
   dimensions,
   function(dim) get.embeddings(dim, "GloVe")
)
save(
   glove.embeddings.list,
   file = "representations-distributed-GloVe.RData"
) # -> 173 MB
```

B.7. Clústering

Para realizar las pruebas de clústering, se han predefinido todas las configuraciones de los parámetros de las pruebas en dos tablas de resultados, que definen el modelo de representación, su dimensionalidad, y el de extracción de características (si aplica). Se define una tabla distinta para cada tipo de representación (simbólicas y distribuidas), en la Subsección B.7.1.

Después, se establecen los parámetros fijos del clústering, como el número de inicializaciones aleatorias de k-medias (Subsección B.7.2), y se definen tres funciones para dividir las responsabilidades del proceso (Subsección B.7.3). A continuación se define una función para evaluar el rendimiento del clústering, proceso que en sí mismo ha resultado ser más costoso que el clústering (Subsección B.7.4). Por último, se ejecutan en paralelo todas las pruebas (Subsección B.7.5).

B.7.1. Tablas de resultados

Para cada tipo de representación, se crea una tabla `performance` que contiene los parámetros de cada prueba y permite almacenar las medidas de rendimiento tras su ejecución.

Tabla `performance` para representaciones simbólicas

```
feature.extraction.method <- c(
   "Document Frequency", "Term Strength",
   "Entropy-based Ranking", "Term Contribution"
)
feature.extraction.amount <- c(10, 25, 50, 100, 250, 500, 1000, 2500)
representation <- c("BoW", "TF-IDF")
performance <- expand.grid(
   clustering.algorithm = "k-medias",
   representation = representation,
   feature.extraction.method = feature.extraction.method,
   feature.extraction.amount = feature.extraction.amount,
   stringsAsFactors = FALSE
)
performance$feature.extraction.amount <- as.numeric(performance$feature.
   extraction.amount)
performance$dunn <- NA
performance$silhouette <- NA
performance$accuracy <- NA
dim(performance) # -> 64, 7
```

Tabla `performance` para representaciones distribuidas

```
representation <- c("LSA", "Word2Vec", "GloVe")
performance <- expand.grid(
   dimensions = dimensions,
   representation = representation,
   stringsAsFactors = TRUE
)
```

```
7
8  # Se eliminan de la tabla las pruebas con LSA para dimensiones
9  #     superiores a 100 (superiores o iguales a 250)
10 performance <- performance[
11    -which(performance$representation == "LSA" & performance$dimensions >= 25
          0),
12 ]
13 performance$dimensions <- as.numeric(performance$dimensions)
14 performance$dunn <- NA
15 performance$silhouette <- NA
16 performance$accuracy <- NA
17 dim(performance) # -> 19, 5
```

B.7.2. Parámetros

Se establecen como parámetros la semilla aleatoria, el número de arranques aleatorios y el número máximo de iteraciones del algoritmo k-medias, así como el tamaño y la cantidad de muestras empleadas para la validación de los resultados.

```
1  # Semilla aleatoria que se fija antes de cada ejecucion de k-medias
2  clustering.seed <- 456
3
4  # Numero de arranques aleatorios de k-medias
5  clustering.nstart <- 10
6
7  # Numero maximo de iteraciones de k-medias
8  clustering.max.iter <- 100
9
10 # Tamano de la muestra para validacion
11 clustering.validation.sample.size <- 2000
12
13 # Numero de repeticiones para extraer muestras de validacion
14 clustering.validation.sample.amount <- 10
```

B.7.3. Funciones

Para fragmentar el proceso de análisis clúster, y facilitar la depuración de código, se definen las siguientes tres funciones:

- `clustering.get.data`: Definida de forma diferente para cada representación, devuelve la matriz de características correspondiente según los parámetros de la prueba de clústering definidos.
- `clustering.log.results`: Imprime en un fichero los resultados parciales cada vez que termina una prueba, y que permite ver el progreso general de todas las pruebas a medida que se ejecutan. También se usa para imprimir los resultados de validación. Es necesaria porque ambos procesos (clústering, validación de los clústers) han tardado cada uno del orden de una hora en el equipo que se ha utilizado para ejecutarlas.
- `clustering.with.kmeans`: Realiza el clústering con el algoritmo k-medias, utilizando a las dos funciones anteriores.

Función `clustering.get.data` para representaciones simbólicas

```
clustering.get.data <- function(performance.index) {
  # Devuelve la matriz BoW o TF-IDF con extraccion de caracteristicas
  # dependiendo del metodo indicado en performance.index
  representation <- as.character(performance$representation[performance.
     index])
  fe.method <- as.character(performance$feature.extraction.method[
     performance.index])
  fe.amount <- performance$feature.extraction.amount[performance.index]

  # Extraer caracteristicas
  fe.criterion <- switch(
    fe.method,
    "Document Frequency" = document.frequency,
    "Term Strength" = term.strength,
    "Entropy-based Ranking" = entropy.based.ranking,
    "Term Contribution" = term.contribution
  )
  extracted.features <- names(fe.criterion)[1:fe.amount]

  x <- if (representation == "BoW") {
    dtm[, extracted.features]
  } else {
    tfidf[, extracted.features]
  }
  return(x)
}
```

Función `clustering.get.data` para representaciones distribuidas

```
clustering.get.data <- function(performance.index) {
  representation <- performance$representation[performance.index]
  dim <- performance$dimensions[performance.index]

  list.index <- which(sapply(glove.embeddings.list, ncol) == dim)

  x <- if (representation == "LSA") {lsa.embeddings.list[[list.index]]}
  else if (representation == "Word2Vec") {word2vec.embeddings.list[[list.
     index]]}
  else if (representation == "GloVe") {glove.embeddings.list[[list.index]]}

  return(x)
}
```

Función `clustering.log.results`

```
# Imprime los resultados en un fichero al haber terminado una ejecucion
#     de una de las pruebas de clustering. Es necesaria al ejecutar las
#     pruebas en paralelo
clustering.log.results <- function(performance.index, start, symbolic,
    validation) {
    # performance.index - el indice de la tabla de parametros, que
```

```
    #              indica toda la configuracion de la prueba
    # start - marca de tiempo en la que empezo la prueba
    # symbolic - booleano que marca si/no si la representacion es simbolica
    # validation - en caso de que el calculo que haya terminado sea de
    #           validacion, es una lista con mas metricas de resultados
    end <- Sys.time()
    elapsed <- as.numeric(difftime(end, start, units = "secs"))
    hours <- floor(elapsed / 3600)
    minutes <- floor((elapsed %% 3600) / 60)
    seconds <- round(elapsed %% 60)


    # Cabecera generica con los parametros de la prueba de clustering,
    #     y marcas de tiempo
    representation <- as.character(performance$representation[performance.
        index])
    log.entry <- paste(
        "Start: ", format(start, "%H:%M:%S"),
        " End: ", format(end, "%H:%M:%S"), "\n",
        "Elapsed: ", sprintf("%02d:%02d:%02d", hours, minutes, seconds),
        "\n", "Representation: ", representation, "\n", sep = ""
    )

    # Anade el metodo de extraccion de caracteristicas, o el numero de
    #     dimensiones, dependiendo de si el metodo de representacion
    #     es simbolico o distribuido
    if(symbolic) {
        fe.method <- as.character(performance$feature.extraction.method[
            performance.index])
        fe.amount <- performance$feature.extraction.amount[performance.index]
        log.entry <- paste(
            log.entry,
            "Feature extraction: ", fe.method, " (", fe.amount, ")\n", sep = ""
        )
    } else {
        dimensions <- performance$dimensions[performance.index]
        log.entry <- paste(
            log.entry,
            "Dimensions: ", dimensions, "\n", sep = ""
        )
    }

    # Anade las metricas de validacion obtenidas al final, en caso de que
    #     corresponda
    if(length(validation) == 0) {
        write(
            paste(log.entry, "\n\n"),
            file = "clustering-log.txt",
            append = TRUE
        )
        return()
    }
    for(metric in names(validation)){
        log.entry <- paste(
            log.entry,
            metric, ": ", validation[[metric]], "\n", sep = ""
        )
    }
```

```
62    write(
63       paste(log.entry, "\n\n"),
64       file = "clustering-validation-log.txt",
65       append = TRUE
66    )
67 }
```

Función `clustering.with.kmeans`

```
# Ejecuta k-medias
clustering.with.kmeans <- function(performance.index, symbolic = TRUE, log =
    TRUE) {
   # Llama a la funcion clustering.get.data para obtener la matriz de
   #     caracteristicas, ejecuta el clustering con kmeans, y registra
   #     que ha terminado la ejecucion si el parametro log es TRUE. Devuelve
   #     el objeto de k-medias juanto al el indice de la tabla performance
   #
   #  symbolic -- indicador para pasarle a clustering.log.results
   #  log    -- flag para indicar si se quiere que se registren en un
   #              fichero los resultados parciales
   start <- Sys.time()
   x <- clustering.get.data(performance.index)

   # Clustering
   set.seed(clustering.seed)
   clustering.result <- kmeans(
      x, centers = 2,
      nstart = clustering.nstart, iter.max = clustering.max.iter
   )

   # Registrar que ha terminado la ejecucion
   if(log) {
      clustering.log.results(performance.index, start, symbolic, list())
   }

   return(list(
      index = performance.index,
      result = clustering.result
   ))
}
```

B.7.4. Evaluación

Se define la función `clustering.validation` para validar los clústers obtenidos con la función `clustering.with.kmeans`. Calcula de forma exacta la métrica de validación externa exactitud, y de forma estimada las métricas de validación interna índice de Dunn e índice de silueta promedio. Toma como parámetro la lista devuelta por `clustering.with.kmeans`, que indica el índice de la tabla `performance` y el objeto de k-medias, que a su vez contiene los clústers.

```
# Calcula las metricas de validacion
clustering.validation <- function(res, symbolic = TRUE, log = TRUE) {
   # Calcula metricas de validacion internas y externas para clustering
   # - metricas internas: dunn, silueta (muestreados)
   # - metricas externas: exactitud
   #
   # Toma muestras de reviews y calcula una matriz de distancias, con la que
   #     se calculan las metricas de validacion. El proceso se repite
   #     varias veces, dependiendo del valor del parametro global fijado
   #     en la seccion de parametros (clustering.validation.sample.amount)
   #     El tamano de la muestra es definido por
   #     clustering.validation.sample.size
   #
   # 'res' es una lista con los siguientes elementos:
   # - index: indice de la tabla performance (para parametros de clustering)
   # - result: objeto devuelto por k-medias

   index <- res$index
   n <- length(res$result$cluster)

   # Para almacenar los resultados de cada muestra aleatoria
   validation.dunn <- numeric(clustering.validation.sample.amount)
   validation.silhouette <- numeric(clustering.validation.sample.amount)

   start <- Sys.time()
   x <- clustering.get.data(index)

   set.seed(clustering.seed)
   # Calculo de metricas internas: dunn y silueta
   for(sample.index in 1:clustering.validation.sample.amount) {
      validation.sample <- sample(1:n, clustering.validation.sample.size)

      # Calculo de la matriz de distancias
      validation.dist.matrix <- proxy::dist(
         as.matrix(x[validation.sample, ]), method = "cosine"
      )

      # Calculo de las metricas de validacion
      validation.stats <- cluster.stats(
         validation.dist.matrix,
         res$result$cluster[validation.sample],
         wgap = FALSE, sepindex = FALSE,
      )

      # Guardar
      validation.dunn[sample.index] <- validation.stats$dunn
      validation.silhouette[sample.index] <- validation.stats$avg.silwidth
   }

   # Calculo de metrica externa: exactitud
   validation.accuracy <- accuracy_vec(
      imdb.true,
      factor(ifelse(res$result$cluster == 1, "positive", "negative"))
   )
   if (validation.accuracy < 0.5)
      validation.accuracy <- 1- validation.accuracy


```

```
    # Registra que ha terminado la ejecucion
    validation <- list(
        "Dunn index" = round(median(validation.dunn), 4),
        "Silhouette index" = round(median(validation.silhouette), 4),
        "Accuracy" = validation.accuracy
    )
    if(log) { clustering.log.results(index, start, symbolic, validation) }

    return(list(
        index = index,
        dunn = validation.dunn,
        silhouette = validation.silhouette,
        accuracy = validation.accuracy
    ))
}
```

B.7.5. Ejecución

La ejecución se realiza en paralelo, con los agrupamientos en primer lugar y las validaciones a continuación. Como ambos procesos son muy costosos, se almacenan los resultados parciales en disco en objetos `RData`.

Ejecución para representaciones simbólicas

```
# Filtros opcionales que permiten seleccionar las pruebas que se quieren
#     ejecutar
performance.indexes <- which(
    performance$clustering.algorithm == "k-medias" # &
    # performance$feature.extraction.method == "Entropy-based Ranking" &
    # performance$feature.extraction.amount < 100
)


# Creacion de un cluster para ejecucion en paralelo
cl <- makeCluster(8)
clusterExport(cl, varlist = c(
    "clustering.get.data", "clustering.log.results",
    "performance", "dtm", "tfidf",
    "document.frequency", "term.strength", "entropy.based.ranking", "term.
        contribution",
    "clustering.nstart", "clustering.seed", "clustering.max.iter",
    "clustering.validation.sample.size", "clustering.validation.sample.amount
        ", "imdb.true"
))
clusterEvalQ(cl, library(Matrix))
clusterEvalQ(cl, library(yardstick))
clusterEvalQ(cl, library(proxy))
clusterEvalQ(cl, library(fpc))


# Lanzar el clustering
file.create("clustering-log.txt")
```

```
clustering.results <- parLapply(
   cl, performance.indexes, clustering.with.kmeans
)
save(clustering.results, file = "representations-symbolic-cluster.RData")

load("representations-symbolic-cluster.RData")
sapply(clustering.results, function(res) table(res$result$cluster))



# Lanzar la validacion
file.create("clustering-validation-log.txt")
clustering.validation.results <- parLapply(
   cl, clustering.results, clustering.validation
)

stopCluster(cl)


# Guardar los resultados en la tabla performance
for (res in clustering.validation.results) {
   performance$dunn[res$index] <- mean(res$dunn)
   performance$silhouette[res$index] <- mean(res$silhouette)
   performance$accuracy[res$index] <- res$accuracy
}
save(performance, clustering.validation.results, file = "representations-
   symbolic-performance.RData")
```

Ejecución para representaciones distribuidas

```
# Se seleccionan todas las pruebas
performance.indexes <- 1:nrow(performance)

# Creacion de un cluster para ejecucion en paralelo
cl <- makeCluster(8)
clusterExport(cl, varlist = c(
   "clustering.with.kmeans", "clustering.validation", "clustering.get.data",
       "clustering.log.results",
   "performance", "lsa.embeddings.list", "word2vec.embeddings.list", "glove.
      embeddings.list",
   "clustering.nstart", "clustering.seed", "clustering.max.iter",
   "clustering.validation.sample.size", "clustering.validation.sample.amount
      ", "imdb.true"
))
clusterEvalQ(cl, library(Matrix))
clusterEvalQ(cl, library(yardstick))
clusterEvalQ(cl, library(proxy))
clusterEvalQ(cl, library(fpc))


# Lanzar el clustering
file.create("clustering-log.txt")
clustering.results <- parLapply(
   cl, performance.indexes,
   function(idx) clustering.with.kmeans(idx, FALSE, TRUE)
)
```

```
save(clustering.results, file = "representations-distributed-cluster.RData")

load("representations-distributed-cluster.RData")


# Lanzar la validacion
file.create("clustering-validation-log.txt")
clustering.validation.results <- parLapply(
   cl, clustering.results,
   function(res) clustering.validation(res, FALSE, TRUE)
)

stopCluster(cl)


# Guarda los resultados en la tabla performance
for (res in clustering.validation.results) {
   performance$dunn[res$index] <- mean(res$dunn)
   performance$silhouette[res$index] <- mean(res$silhouette)
   performance$accuracy[res$index] <- res$accuracy
}
save(
   performance,
   clustering.validation.results,
   file = "representations-distributed-performance.RData"
)
```

B.8. Gráficos de resultados

Los resultados del clústering se han expuesto en la Sección 4.3, así como las conclusiones que se pueden obtener de los mismos. En esta sección se muestra el código para generar los gráficos mostrados en dicha sección (Figuras 4.2 y 4.3), además de código para un gráfico adicional que muestra los resultados de las representaciones simbólicas de forma conjunta para todas las métricas, y que se puede ver en la Figura B.1.

Gráficos para representaciones simbólicas

Gráfico conjunto con *facet wrap*

```
performance.long <- pivot_longer(
  performance,
  cols = c(silhouette, dunn, accuracy),
  names_to = "metric",
  values_to = "score"
)

# Grafico con facet wrap
performance.long <- performance.long %>%
  mutate(
    group = paste(representation, feature.extraction.method, sep = " | "),
```

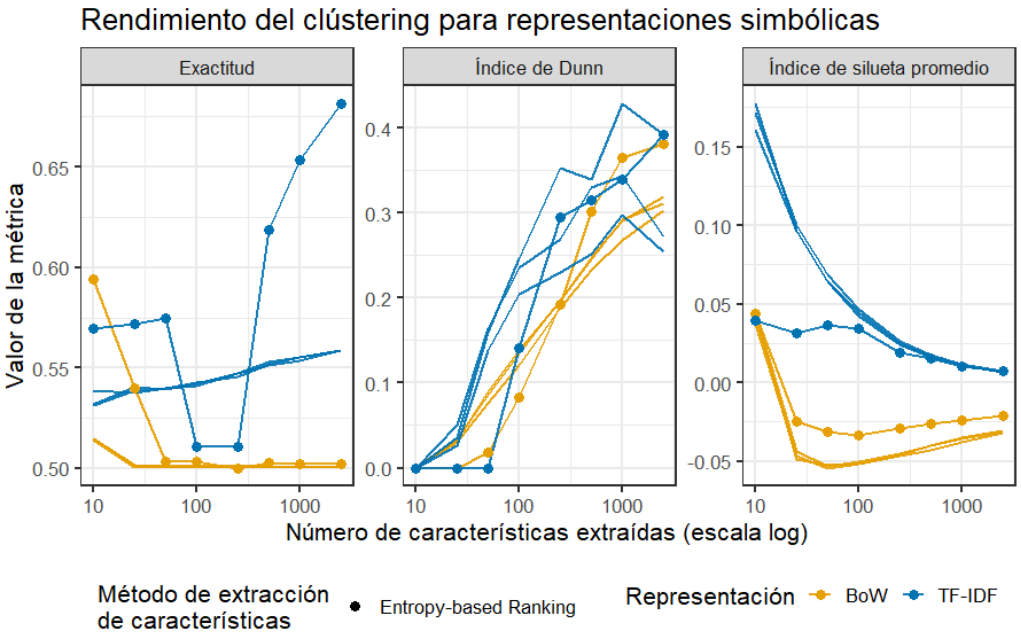


Figura B.1: Resultados para representaciones simbólicas (con facet wrap)

```
    is.ebr = ifelse(feature.extraction.method == "Entropy-based Ranking", "
        EBR", "Resto")
  )

ggplot(performance.long, aes(
    x = feature.extraction.amount, y = score,
    color = representation, group = group,
  )) +
  geom_line(aes(color = representation), linewidth = 1, alpha = 0.9) +
  geom_point(aes(shape = is.ebr), size = 3) +
  scale_color_manual(
    name = "Representacion",
    values = c("TF-IDF" = "#0072B2", "BoW" = "#E69F00")
  ) +
  scale_shape_manual(
    name = "Metodo de extraccion\nde caracteristicas",
    values = c("EBR" = 16),
    labels = c("Entropy-based Ranking")
  ) +
  scale_x_log10() +
  facet_wrap(
    ~ metric, scales = "free_y",
    labeller = labeller(metric = c(
      silhouette = "Indice de silueta promedio",
      dunn = "Indice de Dunn",
      accuracy = "Exactitud"
    ))
  ) +
  labs(
    title = "Rendimiento del clustering para representaciones simbolicas",
    x = "Numero de caracteristicas extraidas (escala log)",
    y = "Valor de la metrica"
  ) +
  theme_bw(base_size = 16) +
  theme(legend.position = "bottom")
```

Gráficos individuales para cada métrica

```
# Graficos individuales
# Se crea la lista graph.params.list, que contiene los parametros
#    que permiten personalizar cada grafico. Estos parametros
#    controlan los titulos, las etiquetas, y el eje Y
graph.params.list <- list(
  list(
    metric = "accuracy", name = "Exactitud",
    scale.y.limits = c(0.50, max(performance$accuracy)),
    scale.y.labels = scales::percent_format(accuracy = 1)
  ),
  list(
    metric = "dunn", name = "Indice de Dunn",
    scale.y.limits = c(-0.01, 0.45),
    scale.y.labels = waiver()
  ),
  list(
    metric = "silhouette", name = "Indice de silueta promedio",
    scale.y.limits = c(-0.1, 0.2),
    scale.y.labels = waiver()
  )
)
for (graph.params in graph.params.list) {
  # Dataframe para las etiquetas que se colocan con ggrepel,
  #    para lo que se filtra la tabla performance para
  #    quedarse con solo los datos de las representaciones con
  #    mayor numero de caracteristicas extraidas
  label.data <- performance.long %>%
    filter(metric == graph.params$metric) %>%
    group_by(group) %>%
    filter(feature.extraction.amount == max(feature.extraction.amount)) %>%
    ungroup()

  set.seed(456)
  p <- performance.long %>%
    filter(metric == graph.params$metric) %>%
    ggplot(aes(
      x = feature.extraction.amount, y = score,
      color = representation, group = group
    )) +
    geom_line(aes(color = representation), linewidth = 1, alpha = 0.9) +
    geom_point(size = 1.75) +
    geom_text_repel(
      data = label.data,
      aes(label = group, color = representation),
      size = 3.5, direction = "y", hjust = 0,
      nudge_x = 0.4, nudge_y = 0
    ) +
    scale_x_log10(
      breaks = c(10, 100, 1000),
      labels = scales::label_number(),
      expand = expansion(mult = c(0.01, 0.35))
    ) +
    scale_y_continuous(
      limits = graph.params$scale.y.limits,
      labels = graph.params$scale.y.labels
    ) +
```

```
    scale_color_manual(
      name = "Representacion",
      values = c("TF-IDF" = "#0072B2", "BoW" = "#E69F00")
    ) +
    theme_bw(base_size = 16) +
    theme(legend.position = "none") +
    labs(
      x = "Numero de caracteristicas extraidas (escala log)",
      y = graph.params$name,
      color = "Line Type",
      size = "Line Type",
      title = paste(graph.params$name, "para representaciones simbolicas"),
    )

  print(p)
}
```

Gráfico para representaciones distribuidas

```
performance.long <- pivot_longer(
  performance,
  cols = c(silhouette, dunn, accuracy),
  names_to = "metric",
  values_to = "score"
)

ggplot(performance.long, aes(x = dimensions, y = score, group =
    representation)) +
  geom_line(aes(color = representation), linewidth = 1.2) +
  geom_point(aes(color = representation, shape = representation), size = 3)
     +
  scale_color_manual(name = "Leyenda",
    values = c("LSA" = "#E69F00", "Word2Vec" = "#0072B2", "GloVe" = "#009E73"
       )
  ) +
  scale_shape_manual(name = "Leyenda",
    values = c("LSA" = 16, "Word2Vec" = 17, "GloVe" = 4)
  ) +
  scale_x_log10(breaks = unique(performance.long$dimensions)) +
  facet_wrap(
    ~ metric, scales = "free_y",
    labeller = labeller(metric = c(
      silhouette = "Indice de silueta promedio",
      dunn = "Indice de Dunn",
      accuracy = "Exactitud"
    ))
  ) +
  labs(
    title = "Rendimiento del clustering para representaciones distribuidas",
    x = "Numero de dimensiones", y = "Valor de la metrica"
  ) +
  theme_bw(base_size = 16) +
  theme(legend.position = "bottom")
```

Apéndice C

Código para Clasificación supervisada

En el presente anexo se presenta el código desarrollado en este trabajo para aplicar Clasificación. La metodología empleada para el análisis, junto con los resultados obtenidos, se puede consultar en el Capítulo 5. Siguiendo las convenciones estándar en el desarrollo de software, el código ha sido escrito en inglés, a excepción de los comentarios y cadenas de texto. Para mayor comodidad, también se puede descargar el código a través del repositorio público de GitHub del trabajo (https://github.com/acabrerod/text-analysis), en su versión en inglés.

Este anexo está dividido en ocho secciones, con la siguiente estructura:

- Sección C.1 – Listado y carga de los paquetes utilizados.
- Sección C.2 – Lectura de datos.
- Sección C.3 – Preprocesamiento de datos.
- Sección C.4 – Cálculo de las representaciones simbólicas.
- Sección C.5 – Cálculo de la métrica de extracción de características χ^2. En la Tabla C.1 se muestran las 15 palabras más relevantes según esta métrica.
- Sección C.6 – Cálculo de las representaciones distribuidas.
- Sección C.7 – Aplicación de la Clasificación, organizada en cuatro subsecciones, en las tres primeras se configuran y preparan funciones, y en la última se muestra el código para ejecutar las pruebas.
- Sección C.8 – Código para obtener los gráficos de resultados.

C.1. Paquetes utilizados

```
# Paquetes para lectura, manipulacion de datos y evaluacion
library(here)
library(readr)
library(dplyr)
library(tidyr)
library(yardstick)
```

```r
library(parallel)
library(Matrix)


# Paquetes de preprocesamiento y analisis de texto
library(tm)         # Para crear y limpiar corpus
library(textstem)   # Para lematizacion
library(textclean)  # Para normalizacion
library(text2vec)   # Para representaciones BoW, TF-IDF, LSA y GloVe
library(word2vec)   # Para representacion Word2Vec

# Paquetes de manipulacion de texto
library(stringi)
library(stringr)


# Paquete para clasificacion
library(LiblineaR)


# Paquetes para graficos de resultados
library(ggplot2)
library(scales)
```

C.2. Lectura de datos

Los datos se han descargado los datos del siguiente enlace: https://ai.stanford.edu/~amaas/data/sentiment/, aunque también se pueden encontrar en el repositorio de código.

Para importar los datos al entorno de R, se ha definido la función `read.imdb` en la Sección A.2, con la que se se crean los objetos `imdb.train`, `imdb.test` e `imdb.true`.

```r
# Ubicacion de los datos
files.train <- "datos/imdb-train"
files.test <- "datos/imdb-test"

# Lectura de los datos
imdb.train.pos <- read.imdb(files.train, "pos")
imdb.train.neg <- read.imdb(files.train, "neg")
imdb.test.pos <- read.imdb(files.test, "pos")
imdb.test.neg <- read.imdb(files.test, "neg")

# Organizacion de los subconjuntos de datos en entrenamiento y prueba
imdb.train <- c(imdb.train.pos, imdb.train.neg)
imdb.test <- c(imdb.test.pos, imdb.test.neg)
imdb.true <- c(
rep("positive", length(imdb.test.pos)),
rep("negative", length(imdb.test.neg))
) %>% as.factor()
```

C.3. Preprocesamiento

Se sigue el mismo procedimiento de preprocesamiento que en la Sección B.3, con la función `imdb.preprocess`, aplicada tanto al conjunto de *train* como al de *test*. En dicha sección se puede ver también una reseña de ejemplo antes y después del preprocesamiento.

```
imdb.train <- imdb.preprocess(imdb.train)
imdb.test <- imdb.preprocess(imdb.test)
```

C.4. Representaciones simbólicas: BoW y TF-IDF

Se utiliza el mismo procedimiento que en la Sección B.4, donde se pueden ver comentarios más detallados. A diferencia del código de dicha sección, para clasificación se aplican las representaciones a ambos conjuntos de entrenamiento y validación. Se obtienen las matrices `dtm.train` y `dtm.test`, correspondientes a BoW, y las matrices `tfidf.train` y `tfidf.test` correspondientes a TF-IDF. Además, se obtiene la matriz de coocurrencia `tcm.train` necesaria para GloVe.

```
# Representacion del conjunto de entrenamiento
tokens.train <- word_tokenizer(imdb.train)
it.train <- itoken(tokens.train, progressbar = FALSE)
vocab <- create_vocabulary(it.train)
vocab <- prune_vocabulary(vocab, term_count_min = 4) # -> 25,788 terminos
vectorizer <- vocab_vectorizer(vocab)

tcm.train <- create_tcm(it.train, vectorizer, skip_grams_window = 5)
dtm.train <- create_dtm(it.train, vectorizer)
tfidf.model <- TfIdf$new()
tfidf.train <- tfidf.model$fit_transform(dtm.train)


# Representacion del conjunto de validacion
tokens.test <- word_tokenizer(imdb.test)
it.test <- itoken(tokens.test, progressbar = FALSE)

dtm.test <- create_dtm(it.test, vectorizer)
tfidf.test <- tfidf.model$transform(dtm.test)
```

C.5. Extracción de características

En esta sección se calculan los estadísticos χ^2 para cada palabra del vocabulario, que miden la falta de independencia de las palabras con la distribución de las clases. Este cálculo se hace con la función `chisq.test` sobre las tablas de contingencia 2×2 de las etiquetas de las clases con la presencia binaria de cada palabra en las reseñas.

```
# Funcion para calcular los estadisticos chi-2
calculate.chi2.for.term <- function(bow.term.column) {
   # Calcula la tabla de contingencia para la palabra dada, a partir de su
   #     vector en la matriz documento-termino. Para ello, cruza la
   #     presencia de cada palabra en cada review (bow.term.column > 0)
   #     con la distribucion de las clases (imdb.true)
   class.term.contigency.table <- table(bow.term.column > 0, imdb.true)

   # Comprobacion de que el termino aparece en mas de una clase
   if(!all(dim(class.term.contigency.table) == c(2, 2))) return(0)

   return(chisq.test(class.term.contigency.table)$statistic)
}

# Calculo de los estadisticos chi-2 para todas las palabras
feature.extraction.chi2 <- apply(dtm.train, 2, calculate.chi2.for.term)

# Ordenacion de mas relevante a menos
feature.extraction.chi2 <- feature.extraction.chi2[order(feature.extraction.
   chi2, decreasing = TRUE)]
```

La Tabla C.1 muestra las 15 palabras con los estadísticos χ^2 más altos, es decir, las palabras con mayor evidencia de estar relacionadas con la distribución de las clases. Como se puede apreciar, todas tienen un elevado valor sentimental, ya sea positivo o negativo. A modo de referencia, el quantil 0,99999 de la distribución χ^2 con un grado de libertad es aproximadamente 20 (sin ajustar por comparaciones múltiples).

Palabra	**Estadístico χ^2**	**Palabra**	**Estadístico χ^2**	**Palabra**	**Estadístico χ^2**
bad	3.126	*bore*	678	*nothing*	521
waste	1.312	*terrible*	642	*horrible*	503
great	1.038	*love*	642	*poor*	489
awful	953	*stupid*	627	*crap*	448
excellent	704	*wonderful*	585	*perfect*	434

Tabla C.1: Las 15 palabras con el estadístico χ^2 más elevado

C.6. Representaciones distribuidas: LSA, Word2Vec y GloVe

Para obtener las representaciones distribuidas se ha seguido el mismo procedimiento descrito en la Sección B.6. La única diferencia entre el código de ambas, es que en este caso los modelos se entrenan con el conjunto de *train*, y posteriormente se son utilizados para representar la totalidad de las reseñas. Los resultados generados se almacenan en listas anidadas con los mismos nombres que en la Sección B.6, a saber:

- `lsa.embeddings.list`
- `word2vec.embeddings.list`
- `glove.embeddings.list`

```
dimensions <- c(5, 10, 25, 50, 100, 250, 500)
```

Latent Semantic Analysis (LSA)

```
# Funcion para calcular los embeddings con LSA
get.lsa.embeddings <- function(d) {
   # Calcula la factorizacion SVD mediante un algoritmo del
   #     tipo soft impute a traves del paquete text2vec, lo que
   #     posibilita que se pueda hacer el analisis, ya que una
   #     factorizacion al uso tendria un muy elevado coste
   lsa.model <- LatentSemanticAnalysis$new(n_topics = d)
   lsa.embeddings.train <- lsa.model$fit_transform(tfidf.train)
   lsa.embeddings.test <- lsa.model$transform(tfidf.test)
   return(list(train = lsa.embeddings.train, test = lsa.embeddings.test))
}

set.seed(456)
lsa.embeddings.list <- lapply(dimensions[dimensions < 250], get.lsa.
   embeddings)
save(lsa.embeddings.list, file = "representations-distributed-LSA.RData") #
   -> 71 MB
```

Word2Vec y GloVe

```
# Filtrado de palabras eliminadas en las representaciones simbolicas
valid.words <- colnames(dtm.train)
tokens.train.filtered <- lapply(tokens.train, function(x) x[x %in% valid.
   words])


# Word vectors con Word2Vec
get.word2vec.wordvectors <- function(d) {
  word2vec.model <- word2vec(
    tokens.train.filtered, type = "cbow", dim = d,
    iter = 20, threads = detectCores() - 1
  )
  word2vec.wv <- as.matrix(word2vec.model)
  return(word2vec.wv)
}
# Word vectors con GloVe
get.glove.wordvectors <- function(d) {
  glove.model <- GlobalVectors$new(rank = d, x_max = 10, learning_rate = 0.0
     5)
  glove.fit <- glove.model$fit_transform(tcm.train, n_iter = 20, n_threads =
      detectCores() - 1)
  glove.wv = glove.fit + t(glove.model$components)
  return(glove.wv)
}

# Calcula una matriz de embeddings para todas las reviews con un numero
#    de dimensiones d, promediando los wordvectors de cada palabra de la
#    review, para los modelos Word2Vec y GloVe
```

```
get.embeddings <- function(d, model) {
  wv <- if (model == "Word2Vec") { get.word2vec.wordvectors(d) }
  else if (model == "GloVe") { get.glove.wordvectors(d) }

  # Las matrices dtm.train y dtm.test tienen las mismas columnas
  valid.words <- rownames(wv)[rownames(wv) %in% colnames(dtm.train)]

  embeddings.train.sum <- as.matrix(dtm.train[, valid.words] %*% wv[valid.
     words, ])
  embeddings.test.sum <- as.matrix(dtm.test[, valid.words] %*% wv[valid.
     words, ])

  return(list(
    train = sweep(embeddings.train.sum, 1, rowSums(dtm.train), FUN = "/"),
    test = sweep(embeddings.test.sum, 1, rowSums(dtm.test), FUN = "/")
  ))
}

set.seed(456)
word2vec.embeddings.list <- lapply(dimensions, function(dim) get.embeddings(
    dim, "Word2Vec"))
save(word2vec.embeddings.list, file = "representations-distributed-Word2Vec.
    RData") # -> 325 MB

glove.embeddings.list <- lapply(dimensions, function(dim) get.embeddings(dim
    , "GloVe"))
save(glove.embeddings.list, file = "representations-distributed-GloVe.RData"
    ) # -> 347 MB
```

C.7. Clasificación

Al igual que en la Sección B.7, se fijan los parámetros para todas las pruebas que se van a realizar en una tabla de resultados: `performance`, cuya definición se muestra en la Subsección C.7.1, y que también servirá para almacenar los resultados obtenidos. A continuación, se establecen los parámetros fijos (Subsección C.7.2), y se definen dos funciones para facilitar la lectura y depuración del código (Subsección C.7.3). Por último, se ejecutan en paralelo todas las pruebas (Subsección C.7.4), llamando a las funciones definidas anteriormente.

C.7.1. Tabla de resultados

```
classification.algorithm <- c("SVM", "Logistic")
feature.extraction.amount <- c(10, 25, 50, 100, 250, 500, 1000, 2500, 5000,
    10000)

performance <- rbind(
  expand.grid(
    classification.algorithm = classification.algorithm,
    representation = c("BoW", "TF-IDF"),
    dimensions = feature.extraction.amount,
    stringsAsFactors = FALSE
  ),
```

```
  expand.grid(
    classification.algorithm = classification.algorithm,
    representation = c("LSA", "Word2Vec", "GloVe"),
    dimensions = dimensions,
    stringsAsFactors = FALSE
  )
)
performance <- performance[
  -which(performance$representation == "LSA" & performance$dimensions >= 250
      ),
]
performance$dimensions <- as.numeric(performance$dimensions)
performance$accuracy <- NA
performance$f1.score <- NA

dim(performance) # -> 78, 5
```

C.7.2. Parámetros

Se fija la semilla aleatoria para obtener los mismos clasificadores, y el parámetro k para la validación cruzada k-fold.

```
classification.seed <- 456
classification.kfold.k <- 10
```

C.7.3. Funciones

Se definen las siguientes 2 funciones:

- `classification.get.data`: Devuelve la matriz de características correspondiente según los parámetros definidos. Soporta todas las representaciones, tanto simbólicas como distribuidas, y además también devuelve las matrices de características dependiendo del tipo de subconjunto de datos (entenamiento y validación).
- `train.and.eval.classifier`: Entrena el clasificador correspondiente y evalúa su rendimiento con las métricas exactitud y *F1-Score*. Si se ejecuta en el mismo hilo, también imprime por pantalla los resultados de cada prueba.

Función `classification.get.data`

```
# Devuelve la matriz de caracteristicas
classification.get.data <- function(performance.index, train = TRUE) {
   # Devuelve la matriz de caracteristicas dependiendo del metodo
   #    indicado en performance.index
   representation <- as.character(performance$representation[performance.
      index])
   d <- performance$dimensions[performance.index]

   # Extraccion de caracteristicas
   if(representation %in% c("BoW", "TF-IDF")) {
```

```
      extracted.features <- names(feature.extraction.chi2)[1:d]
   } else {
      list.index <- which(sapply(glove.embeddings.list, function(l) ncol(
         l$train)) == d)
   }

   # Obtencion de la matriz de caracteristicas
   if(train) {
      x.train <- if (representation == "LSA") {lsa.embeddings.list[[list.
         index]]$train}
      else if (representation == "Word2Vec") {word2vec.embeddings.list[[list
         .index]]$train}
      else if (representation == "GloVe") {glove.embeddings.list[[list.index
         ]]$train}
      else if (representation == "BoW") {dtm.train[, extracted.features]}
      else if (representation == "TF-IDF") {tfidf.train[, extracted.features
         ]}
      else { stop("Error: non-valid representation method") }
      return(x.train)
   }
   x.test <- if (representation == "LSA") {lsa.embeddings.list[[list.index]]
      $test}
   else if (representation == "Word2Vec") {word2vec.embeddings.list[[list.
      index]]$test}
   else if (representation == "GloVe") {glove.embeddings.list[[list.index]]
      $test}
   else if (representation == "BoW") {dtm.test[, extracted.features]}
   else if (representation == "TF-IDF") {tfidf.test[, extracted.features]}
   else { stop("Error: non-valid representation method") }
   return(x.test)

}
```

Función `train.and.eval.classifier`

```
# Entrena y evalua al clasificador
train.and.eval.classifier <- function(performance.index) {
   # Utiliza el paquete LiblineaR para realizar la clasificacion.
   #     Soporta los algoritmos de regresion logistica y SVM.
   #     Devuelve el clasificador y las metricas de evaluacion
   classification.algorithm <- as.character(performance$classification.
      algorithm[performance.index])
   train.data <- classification.get.data(performance.index, train = TRUE)
   test.data <- classification.get.data(performance.index, train = FALSE)

   # Entrenamiento
   set.seed(classification.seed)

   # Codigo del clasificador en el paquete LiblineaR
   classifier.type <- c("Logistic" = 0, "SVM" = 2)[classification.algorithm]

   # Estimacion del parametro de coste con k-fold
   best.cost <- LiblineaR(
      as.matrix(train.data), imdb.true, type = classifier.type,
      cost = 1, cross = classification.kfold.k, findC = TRUE
   )
```

```
21
22    # Entrenamiento del clasificador
23    classifier <- LiblineaR(
24       as.matrix(train.data), imdb.true,
25       type = classifier.type, cost = best.cost
26    )
27
28    # Evaluacion
29    pred.train <- predict(classifier, train.data)
30    pred.test <- predict(classifier, test.data)
31    accuracy.test <- accuracy_vec(imdb.true, pred.test[[1]])
32    accuracy.train <- accuracy_vec(imdb.true, pred.train[[1]])
33    f1.score.test <- f_meas_vec(imdb.true, pred.test[[1]])
34    f1.score.train <- f_meas_vec(imdb.true, pred.train[[1]])
35
36    # Imprime en la consola los resultados al haber terminado
37    representation <- as.character(performance$representation[performance.
          index])
38    dimensions <- performance$dimensions[performance.index]
39    cat(
40       "** Classifier **\n",
41       "Algorithm: ", classification.algorithm, "\n",
42       "Representation: ", representation, " (", dimensions, ")", "\n",
43
44       "Accuracy (train): ", round(accuracy.test, 3),
45       " (", round(accuracy.train, 3), ")", "\n",
46
47       "F1-Score (train): ", round(f1.score.test, 3),
48       " (", round(f1.score.train, 3), ")", "\n",
49       "\n\n", sep = ""
50    )
51    return(list(
52       index = performance.index,
53       classifier = classifier,
54       accuracy = accuracy.test,
55       f1.score = f1.score.test
56    ))
57 }
```

C.7.4. Ejecución

Se ejecutan todas las pruebas en paralelo y se almacenan los resultados.

```
1  # Filtros opcionales que permiten seleccionar las pruebas
2  #     que se quieren ejecutar
3  performance.indexes <- which(
4     # performance$representation %in% c("BoW", "TF-IDF") &
5     # performance$representation %in% c("LSA", "Word2Vec", "GloVe")
6     performance$dimensions > 0
7  )
8
9  # Esta linea de codigo ejecuta la clasificacion en monohilo, donde
10 #      ademas se imprimen en la pantalla los resultados a medida
11 #      que avanza el proceso
12 # train.results <- sapply(performance.indexes, train.and.eval.classifier)
13
```

```
14
15 # Creacion de un cluster para ejecucion en paralelo
16 cl <- makeCluster(3) # Solo se pueden 3 hilos por limite de memoria
17 clusterExport(cl, varlist = c(
18    "classification.get.data",
19    "performance", "feature.extraction.chi2",
20    "dtm.train", "dtm.test", "tfidf.train", "tfidf.test",
21    "lsa.embeddings.list", "word2vec.embeddings.list", "glove.embeddings.list
          ",
22    "classification.seed", "classification.kfold.k", "imdb.true"
23 ))
24 clusterEvalQ(cl, library(Matrix))
25 clusterEvalQ(cl, library(yardstick))
26 clusterEvalQ(cl, library(LiblineaR))
27
28
29 # Lanzar la clasificacion
30 classification.results <- parLapply(
31    cl, performance.indexes, train.and.eval.classifier
32 )
33 save(classification.results, file = "classification-results.RData")
34
35 stopCluster(cl)
36
37
38 # Guarda los resultados en la tabla performance
39 for (res in classification.results) {
40    performance$accuracy[res$index] <- res$accuracy
41    performance$f1.score[res$index] <- res$f1.score
42 }
43 save(performance, file = "performance.RData")
```

C.8. Gráficos de resultados

Los resultados de la clasificación se han expuesto en la Sección 5.3, así como las conclusiones que se pueden obtener de los mismos. En esta sección se muestra el código para generar los gráficos mostrados en dicha sección (Figuras 5.1 y 5.2).

```
1 performance.long <- pivot_longer(
2   performance,
3   cols = c(accuracy, f1.score),
4   names_to = "metric",
5   values_to = "score"
6 )
7
8
9 # La lista graph.params.list contiene los parametros que permiten
10 #    personalizar cada grafico, ya que se hace uno para cada tipo de
11 #    representacion: simbolicas y distribuidas. Estos parametros controlan
12 #    los titulos, los colores, las etiquetas y el eje Y
13 graph.params.list <- list(
14   list(
15     representations.filter = c("BoW", "TF-IDF"),
```

```
    title = "Rendimiento de la clasificacion para representaciones simbolicas
        ",
    x.label = "Numero de caracteristicas extraidas (escala log)",
    y.limits = c(0.7, 0.88),
    manual.color.values = c("TF-IDF" = "#0072B2", "BoW" = "#E69F00")
  ),
  list(
    representations.filter = c("LSA", "Word2Vec", "GloVe"),
    title = "Rendimiento de la clasificacion para representaciones
        distribuidas",
    x.label = "Numero de dimensiones (escala log)",
    y.limits = c(0.65, 0.87),
    manual.color.values = c("LSA" = "#E69F00", "Word2Vec" = "#0072B2", "GloVe
        " = "#009E73")
  )
)
for (graph.params in graph.params.list) {
  p <- performance.long %>%
    filter(representation %in% graph.params$representations.filter) %>%
    mutate(group = paste(representation, classification.algorithm, sep = "_")
        ) %>%
    ggplot(aes(
      x = dimensions, y = score,
      color = representation, group = group
    )) +
    geom_line(aes(color = representation), linewidth = 1.2) +
    geom_point(aes(shape = classification.algorithm), size = 3) +
    scale_color_manual(
      name = "Representacion",
      values = graph.params$manual.color.values
    ) +
    scale_shape_manual(
      name = "Algoritmo",
      values = c("SVM" = 16, "Logistic" = 17),
      labels = c("Logistica", "SVM")
    ) +
    scale_x_log10() +
    scale_y_continuous(
      labels = percent_format(accuracy = 1),
      limits = graph.params$y.limits
    ) +
    facet_wrap(
      ~ metric, scales = "free_y",
      labeller = labeller(metric = c(
        accuracy = "Exactitud", f1.score = "F1-Score"
      ))
    ) +
    labs(
      title = graph.params$title,
      x = graph.params$x.label,
      y = "Valor de la metrica"
    ) +
    theme_bw(base_size = 16) +
    theme(legend.position = "bottom")

  print(p)
}
```