UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Creación de corpus de artículos de prensa y categorización de noticias

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Vicent Ahuir Esteve
Cotutor: Ferran Pla Santamaria
Cotutor: Lluís Felip Hurtado Oliver

Curso 2018-2019

Resum

En l'actualitat, gràcies als avanços de la tecnologia, ha augmentat exponencialment el nombre de documents digitals en format text que es generen diàriament. Aquests documents poden arribar des d'una gran diversitat de fonts, tals com periòdics, blogs, xarxes socials, etc. Tots aquests documents han de ser classificats de manera que es facilite el filtrat i cerca d'informació concreta entre tota l'amalgama de documents de text disponibles. A causa de la gran quantitat de documents de text generada, aquest procés s'ha de realitzar de manera automàtica, mitjançant l'ús d'alguns dels diferents paradigmes d'aprenentatge automàtic existents.

Aquest treball es centrarà en la classificació de textos noticiaris provinents de fonts periodístiques en castellà i català. En ell es realitza un estudi de diferents models d'aprenentatge automàtic i mètodes d'extracció de característiques per a text, en el qual s'analitza com influeix en el rendiment del classificador l'ús d'un cert model d'aprenentatge automàtic en combinació amb diferents mètodes d'extracció de característiques. Així mateix, s'analitza l'efecte obtingut en el comportament del classificador en triar l'una o l'altra mesura de rendiment.

Per a poder realitzar l'estudi de classificació, s'ha creat un procés automàtic de captura de notícies. Amb ell, s'han creat dos *corpus* –un per a castellà i un altre per a català– a partir de diferents fonts periodístiques digitals.

Paraules clau: *corpus* de notícies de premsa, *web-crawling*, classificació automàtica de text, aprenentatge automàtic, extracció de característiques

Resumen

En la actualidad, gracias a los avances de la tecnología, ha aumentado exponencialmente el número de documentos digitales en formato texto que se generan diariamente. Estos documentos pueden llegar desde una gran diversidad de fuentes, tales como periódicos, blogs, redes sociales, etc. Todos estos documentos deben ser clasificados de modo que se facilite el filtrado y búsqueda de información concreta entre toda la amalgama de documentos de texto disponibles. Debido a la gran cantidad de documentos de texto generada, este proceso se debe realizar de manera automática, mediante el uso de algunos de los distintos paradigmas de aprendizaje automático existentes.

Este trabajo se centrará en la clasificación de textos noticiarios provenientes de fuentes periodísticas en castellano y catalán. En él se realiza un estudio de distintos modelos de aprendizaje automático y métodos de extracción de características para texto, en el cual se analiza como influye en el rendimiento del clasificador el uso de un cierto modelo de aprendizaje automático en combinación con distintos métodos de extracción de características. Así mismo, se analiza el efecto obtenido en el comportamiento del clasificador al elegir una u otra medida de rendimiento.

Para poder realizar el estudio de clasificación, se ha creado un proceso automático de captura de noticias. Con él, se han creado dos *corpus* –uno para castellano y otro para catalán– a partir de distintas fuentes periodísticas digitales.

Palabras clave: *corpus* de noticias de prensa, *web-crawling*, clasificación automática de texto, aprendizaje automático, extracción de características

Abstract

Nowadays, thanks to advances in technology, it has increased exponentially the number of digital documents in text format that are generated daily. These documents can come from a great diversity of sources, such as newspapers, blogs, social networks, etc. All these documents must be classified in order to enhance filtering and searching for a specific information among the entire amalgam of text documents available. Due to the large amount of text documents generated, this process must be performed automatically, by using some of the different existing machine learning paradigms.

This work will focus on the classification of news texts coming from Spanish journalistic sources that publish news written in Spanish and/or Catalan languages. It carries out a study of different machine learning models and methods of feature extraction for text, in which it is analysed how it influences the classifier performance using a certain machine learning paradigm in combination with different feature extraction methods. Moreover, it's analysed the impact in the behaviour of the classifier when one or the other metric is chosen.

In order to perform the study of the automatic classification, a web-crawler process has been created for news capture. With it, two corpora have been created – one for Spanish and one for Catalan – from different digital Spanish journalistic sources.

Key words: corpora of press news, web-crawling, automatic classification of text, machine learning, features extraction

Mi más sincero agradecimiento a mis tutores, Ferran y Lluís, por aconsejarme y ayudarme en la elaboración de este trabajo. A mis padres, mi hermano y Natasha; sin su apoyo incondicional nunca habría llegado hasta la redacción de estas páginas. Y por último, a todos mis amigos y compañeros de la universidad, que han hecho de estos cuatro años algo inolvidable.

Tus suposiciones son ventanas en el mundo. Límpialas de vez en cuando, o la luz no entrará.

Isaac Asimov.

Índice general

Índice de figuras

Índice de tablas

Índice de Códigos

1 Introducción

La clasificación automática de documentos –en este trabajo centrado en texto– nos permite categorizar cualquier tipo de documento sin la intervención humana. Para ello existen distintos tipos de modelos de clasificación que, según el tipo de contenido del documento, tendrán mayor o menor efectividad en el momento de la clasificación.

Por lo general, para desarrollar un clasificador automático, se necesita un conjunto de muestras de entrenamiento. Analizando las muestras de entrenamiento se pueden ecntrar aquellos parámetros para el clasificador que maximicen el rendimiento de clasificación de este en el proceso de entrenamiento. Para ello debemos tener un *corpus* invariable –conjunto de documentos– y en el que cada documento esté convenientemente etiquetado.

El presente trabajo se centrará en la creación de dos *corpus* de noticias recogidos de varias fuentes –periódicos digitales en nuestro caso– e idiomas –castellano y catalán– que serán utilizados para realizar un estudio de distintos modelos de clasificación; donde podremos observar las ventajas e inconvenientes de cada uno de ellos y si existen algunos modelos que son más recomendables para la clasificación automática de este tipo de documento.

Debido a que la tarea de diseño y desarrollo de la solución encargada de la creación de los *corpus* ya es suficientemente extensa como para cubrir un trabajo de fin de grado, se decidió realizar esta tarea junto con Fernando Alcina Sanchis [1]. El trabajo conjunto permitió reducir notoriamente el tiempo necesario para la creación de los *corpus* y poder utilizar el tiempo restante para realizar estudios alrededor del aprendizaje automático.

1.1 Motivación

Formalmente definimos la clasificación de documentos como la asignación de una o más categorías predefinidas a un documento según su contenido [2]. La categorización de un documento es una tarea compleja, que recae normalmente en el autor del texto en cuestión, el cual selecciona manualmente las categorías a las que pertenece; típicamente, entre ciertas categorías que se usan comúnmente en todos los periódicos, tales como economía, deportes, sociedad, política, internacional, etc. No obstante, haciendo uso de técnicas de *machine learning* y la tecnología actual, se pueden llegar a crear clasificadores automáticos de texto entrenados a partir de grandes colecciones de texto –etiquetadas anteriormente de forma manual– y que sean capaces de asignar de manera efectiva una categoría a nuevos textos, a priori, sin etiquetar [3].

La clasificación de documentos es importante, ya que permite a los usuarios encontrar más fácilmente aquella información que desean. Al clasificar, el usuario puede restringir el número

de documentos en los que buscar la información deseada y así perder el mínimo tiempo posible en el acceso a la información y evitar una perdida de tiempo navegando por documentos que no están relacionados con la información que el usuario desea obtener. En concreto, la clasificación de noticias es crítica debido a que diariamente se están generando noticias, el vocabulario y la gramática empleada va cambiando a lo largo de los años y puede que aparezcan noticias que pertenezcan a una categoría completamente nueva [4].

Con la evolución de la tecnología en las últimas décadas, ha aumentado de manera exponencial la cantidad de documentos digitales que se crean diariamente. Las expectativas de crecimiento de la cantidad diaria de nuevos documentos digitales, nos alertan de la necesidad de encontrar métodos de organización eficaces y eficientes, de modo que nos faciliten la búsqueda de información y el acceso a la misma. Una solución que implique una clasificación manual –intervención humana– no es una solución factible, debido al ritmo de creación de documentos que hace intratable el problema sin un proceso automático. [5]

Por ello, podemos entrever que existe un problema relacionado con la gran cantidad de datos generados continuamente, por lo que es necesaria una tecnología que pueda reconocer de manera automática el contenido de los documentos, que sea eficaz en esta tarea y además sea eficiente tanto en el proceso de entrenamiento como en la tarea de clasificación en si misma. Esta tecnología, si bien está muy avanzada, sigue distando de considerarse un problema completamente resuelto.

Por último, la tarea de entrenamiento de los clasificadores automático necesita de un *corpus* con el que realizarla. La creación de un *corpus* proveniente de varias fuentes que no contienen un formato estándar, en si mismo, ya es un reto; debido a que podemos encontrarnos diversidad de nomenclaturas para un mismo campo –por ejemplo, las categorías, las fechas, etc–, así como que algunos campos de una fuente sean más fácilmente accesibles que en otra o, incluso, que no existan. Por esta razón, y debido al reto, el autor del presente trabajo también se ve motivado a realizar esta tarea de manera conjunta; de modo que no solo tenga utilidad para el estudio sobre la clasificación automática, sino que pueda ser usado en otras tareas y/o estudios futuros.

1.2 Objetivos

Dado que este trabajo contiene claramente dos partes bien diferenciadas, creación de los *corpus* y el estudio sobre técnicas de categorización de noticias, pasaremos a citar por separado los objetivos que queremos conseguir en cada una de estas partes.

1.2.1 Creación de *corpus*

La primera necesidad/objetivo para la creación de un clasificador de noticias, es el de la recolección de un volumen considerable de noticias de modo que podamos entrenar los clasificadores adecuadamente. Además, con un gran volumen de documentos podemos analizar, entre otras cosas, el impacto –en la eficiencia del clasificador– que tiene el aumento del número de muestras.

Por otro lado, dado que el autor de este trabajo considera la diversidad lingüística y cultural como una virtud, cree necesario no solo ceñirse a la creación de un *corpus* en castellano, sino presentar otro –con características similares– para el idioma del catalán. Generalmente, la mayoría de los *corpus* son en inglés, debido a que la mayoría de estudios se hacen para la lengua inglesa. Es por ello que nos reafirmamos en la importancia de crear *corpus* de ámbito general para el idioma castellano y, con más importancia si cabe, para el catalán donde no existen prácticamente referentes.

De la creación de un corpus en castellano y otro en catalán, podemos derivar los siguientes objetivos que deseamos cumplir en este trabajo:

- **Diversidad fuentes.** Cada uno de los dos *corpus* debe ser creado a partir de una colección de páginas web de periódicos u otras fuentes periodísticas, de modo que contengan la mayor diversidad posible del uso de la lengua, así como capturar distintas variantes dialectales si las hay –en catalán, por ejemplo–.
- **Estandarización de las muestras.** Independientemente de la fuente de la cual provenga, dentro del *corpus*, las noticias deben tener el mismo formato común para fecha, categoría, etc.
- **Corpus reutilizable.** Se desea obtener un/os corpus que puedan ser utilizados para otros propósitos más allá de la necesidades/pretensiones del presente trabajo.
- **Proceso reutilizable.** En la medida que sea posible y dentro de las pretensiones de este trabajo, el proceso de captura de las noticias debe de configurable y reutilizable independientemente de la fuente de la que queremos capturar noticias. Es decir, se debe poder contar con ficheros de configuración para que el código relacionado con la captura de los datos no implemente soluciones *ad-hoc*.
- **Proceso escalable y portable.** La solución debe estar implementada con lenguajes y estándares que nos ayuden a la portabilidad, de modo que puede ser ejecutado en la mayor diversidad de entornos posible. Por otro lado, debe ser escalable, con el fin de que el sistema no reduzca su rendimiento dependiendo de la cantidad de fuentes y/o noticias que deseemos capturar.
- **Análisis de los corpus.** Deseamos tener datos estadísticos sobre los corpus que se han extraído mediante el proceso desarrollado.

1.2.2 Categorización de Noticias

En el apartado de categorización de noticias, en líneas generales, mediante el trabajo se desea estudiar los puntos fuertes y débiles de algunos de los métodos clásicos de clasificación para la tarea de clasificación de noticias.

Es por ello que mediante el desarrollo y estudio de esta parte del trabajo, deseamos conseguir los siguientes objetivos:

- **Análisis del comportamiento de distintos modelos de aprendizaje automático.** Dado que existen distintos paradigmas o modelos de aprendizaje automático, se desea elegir algunos de ellos para poder analizar su eficiencia, tanto a nivel de clasificador como de coste computacional.
- **Comparar el comportamiento de los modelos de aprendizaje con distintas técnicas de representación de texto.** Para poder utilizar las muestras de texto en un clasificador –ya sea durante el entrenamiento o para su clasificación–, los textos de deben pasar a vectores numéricos que representen los rasgos característicos del texto, en base a unos ciertos criterios. Para dicha transformación, existen un abanico de métodos para realizar la conversión de texto a vector de características. Según la técnica, el modelo de aprendizaje puede aumentar o disminuir su eficiencia. Por ello, se quiere estudiar la influencia de una serie de métodos de extracción para los modelos de aprendizaje que se elijan.
- **Analizar la influencia del número de características en cada clasificador y método de extracción.** Dependiendo del método utilizado para clasificación, el método de extracción de características y el número de estas, hará que la eficiencia del clasificador varíe. Es por ello que se quiere analizar –mediante una serie de métricas– la influencia del número de características en el proceso de clasificación de noticias. Así mismo, lograr obtener configuraciones en las cuales se maximice algunas de las métricas y se minimice el número de características, dado que este número está ligado al coste de memoria y en muchos casos también influye en el coste computacional.
- **Estudiar el impacto de la elección de las métricas en el momento de la elección de los parámetros de configuración del clasificador.** Para medir la eficiencia de un clasificador, se suele utilizar una de las siguientes cuatro métricas: *accuracy*, *precision*, *recall* o F_1. Dependiendo de la métrica o métricas que se elija para configurar el clasificador final, se obtendrá un rendimiento distinto en cada una de las clases. Es por ello, que se desea ver el impacto de seleccionar los parámetros de configuración en base al *accuracy* y, por otro lado, si los seleccionamos mediante macro F_1.
- **Analizar el impacto en la eficiencia del clasificador de categorías mayoritarias dentro de un *corpus*.** Cuando un *corpus* contiene categorías que doblan o triplican porcentualmente el resto de categorías, puede aumentar la probabilidad que estas categorías tengan textos más heterogéneos y por tanto, reduzcan (o no) la eficiencia del clasificador para el resto de categorías.

1.3 Estructura de la memoria

El presente trabajo, que constituye la memoria del Trabajo de Fin de Grado, se divide en cuatro capítulos, los cuales pasaremos a describir a continuación; en la descripción de cada capítulo indicaremos de manera escueta aquellos aspectos que se trataran en cada capítulo, de modo que el lector pueda tener una visión global del texto.

- **Capítulo 1, Introducción.** El primer capítulo pretende dar al lector una visión global del trabajo, en donde se indican las motivaciones que han llevado al autor a realizarlo, así como los objetivos que se quieren conseguir al finalizar el TFG.
- **Capítulo 2, Creación de los *corpus*.** En este capítulo hablaremos del estado del arte referente a la creación de distintas colecciones de noticias o texto, analizaremos el problema de extraer noticias de distintas fuentes, describiremos el diseño de la solución desarrollada así como los detalles más importantes en la etapa de desarrollo y por último realizaremos una descripción estadística de los *corpus* obtenidos.
- **Capítulo 3, Categorización de noticias.** En esta parte del trabajo citaremos y describiremos brevemente algunos trabajos relacionados con la clasificación de texto, introduciremos aquellos conceptos esenciales para entender el proceso de clasificación. Así mismo, también describiremos los pasos seguidos para la extracción de las noticias desde los *corpus* creados en el capítulo anterior, así como las transformaciones realizadas; seguidamente detallaremos los distintos aspectos relacionados con la implementación del código encargado de generar los distintos modelos de clasificación y por último analizaremos los resultados obtenidos para los distintos modelos de aprendizaje seleccionados.
- **Capítulo 4, Conclusiones y trabajo futuro.** En el último capítulo del trabajo, expondremos o analizaremos el grado de cumplimiento de los distintos objetivos marcados en el primer capítulo; así mismo también se expondrán algunas propuestas para mejorar o ampliar lo realizado en este trabajo y que no se hayan podido llevar a cabo.

2 Creación de los *corpus*

2.1 Estado del arte

En la actualidad, a raíz de distintos trabajos de investigación y estudios, se han obtenido diferentes *corpus* que contienen textos periodísticos provenientes de una o varias fuentes. Dado que el inglés es el idioma de facto para la elaboración proyectos tecnológicos, citaremos algunos proyectos que produjeron como resultado un *corpus* de noticias provenientes de periódicos de habla inglesa; además, citaremos otros trabajos que han generado una colección de noticias en español.

Entre los *corpus* que están formados por noticias en inglés, podemos destacar:

- ***Document Understanding Conference (DUC).*** Se trata de una colección de noticias de un tamaño relativamente reducido. El *corpus* está centrado en la evaluación de sistemas de resumen automático, por lo que solo está presente el texto de la noticia y un resumen de la misma generado de manera manual[6]. Esta colección de noticias está accesible de manera pública [1].

- ***Gigaword.*** Está formado por un conjunto de 10 millones de noticias. Debido a que está enfocado al estudio del uso de las palabras, no contiene otro tipo de información. Este *corpus* solo está disponible para los miembros del Linguistic Data Consortium (LDC) y para aquellas personas que lo soliciten mediante un formulario [2].

- ***New York Times Corpus.*** Recopila 1,8 millones de noticias y resúmenes, provenientes de una sola fuente, el diario *New York Times*. Está disponible para los miembros del LDC y para cualquier usuario que lo solicite [3].

- ***CNN/Daily Mail Corpus.*** Al igual que la colección anterior, esta recopila noticias provenientes de la CNN y del Daily Mail. Este *corpus* fue creado para el estudio de nuevas técnicas de aprendizaje automático que escalasen mejor con el número de muestras de entrenamiento [7]. Está formado por 300 mil documentos y puede ser generado mediante el código fuente publicado *online* [4].

[1] https://duc.nist.gov/
[2] https://catalog.ldc.upenn.edu/LDC2003T05
[3] https://catalog.ldc.upenn.edu/LDC2008T19
[4] http://www.github.com/deepmind/rc-data/

- ***Newsroom.*** Este *corpus* se creó para entrenar modelos de generación de resúmenes automáticos [8], en él se recopilan alrededor de 1.3 millones pares de noticia-resumen, provenientes de varias fuentes periodísticas. Los autores proporcionan el corpus a cualquier persona que rellene el formulario de petición de uso [5].

Pasaremos a enumerar los *corpus* de noticias en español más destacados:

- ***Aracne.*** El *corpus* recoge noticias desde el 1914 al 2014. Fue creado con la pretensión de ver la evolución del uso de la lengua escrita en los periódicos a lo largo de los años [9]. La colección contiene un número reducido de artículos –5167–, los cuales no contiene ni resúmenes ni categorías. El *corpus* no está disponible públicamente para su estudio.

- ***Spanish News Text.*** Es una colección con un tamaño de 1300MB de textos noticiarios escritos en español, provenientes de países hispanohablantes; no contiene ni resúmenes ni categorías y solo está disponible para los miembros del LDC debido al copyright de los textos o para cualquier persona que realice una petición de uso [6].

- ***Corpus del Español NOW.*** Tiene un corpus muy grande –alrededor de 25 millones de noticias hasta abril del 2019–, no obstante, el acceso está limitado a la plataforma *online* donde solo muestra trozos de texto en los que aparece la palabra que se ha elegido buscar anteriormente [7].

- ***Molino Labs.*** *Now* tiene alrededor 1.7 millones de noticias. Al igual que en el *corpus* del *Español NOW*, el acceso está limitado a su página web y tampoco se muestra el texto completo de las noticias [8].

Después de haber citado tanto los *corpus* en inglés como los de noticias en español, podemos observar que el tamaño del *corpus* varia en relación con el uso que se le vaya a dar al mismo, así como la variedad de información por cada noticia; no vamos a necesitar la misma cantidad de noticias para recoger estadísticas sobre el uso de la lengua, como para utilizarlas como entrenamiento para un clasificador. Como ya hemos comentado, de cada noticia recogeremos los datos necesarios para la tarea que deseemos llevar a cabo; si bien, cuanta más información recojamos de cada noticia, posiblemente, más usos le podremos dar a nuestro *corpus*.

En nuestro caso, y dado que se le va a dar diversos usos, deseamos tener una colección de noticias para español y otra para catalán, las cuales provengan de distintas fuentes; el tamaño de cada *corpus* resultante debe de ser de tamaño medio –al menos 200 mil noticias por idioma– y, como mínimo, debe contener el texto de la noticia, el resumen y la categoría asignada por el periódico.

[5] https://summari.es/download/
[6] https://catalog.ldc.upenn.edu/LDC95T9
[7] https://www.corpusdelespanol.org/now/
[8] http://www.molinolabs.com/corpus.html

2.2 Conceptos previos

A continuación, presentaremos algunos de los conceptos básicos que han sido necesarios para poder diseñar la solución propuesta en este trabajo para la extracción de noticias de distintas fuentes *online*. Estos conceptos nos han ayudado a crear una solución genérica a nivel de código, dependiendo solo de la creación de algunos ficheros de configuración para poder recoger noticias de una fuente concreta.

2.2.1 Estructura de un Documento Web

Para poder extraer información de un documento web, lo primero que debemos conocer es que estructura sigue este y donde y como está guardada la información que después pasa a ser mostrada –normalmente– en el navegador.

Los documentos web están definidos mediante el lenguaje HTML. Dave Ragett –miembro de la World Wide Web Consortium (W3C)– explica en su página web en que consiste el lenguaje HTML [10]:

> Es un tipo especial de documento de texto que es usado por los navegadores Web para presentar texto e imágenes. El documento incluye marcas como que indica el comienzo de un párrafo y que indica el final del mismo. Los documentos HTML comúnmente se conocen como "Páginas Web". El navegador recupera las páginas web del servidor gracias a Internet, y por lo tanto puede estar situado en cualquier parte del planeta

De esta explicación, podemos sacar algunas características básicas del HTML. La primera es que una página web –al final, en su código fuente– es un fichero de texto plano. La segunda es que, mediante marcas o *tags*, se define que tipo de contenido contiene una cierta marca. Las marcas ayudarán al navegador web a conocer qué formato debe aplicar al texto. Por ejemplo, ¿Es un párrafo o la cabecera? ¿El texto va en negrita? etc. En ausencia de estas marcas, el navegador interpretaría todo el texto de la misma forma, sin dar formato alguno.

En el párrafo anterior hemos citado dos de las tres características que definen básicamente el lenguaje HTML: *Text*, ya que el documento contiene texto como ya hemos comentado; y *Markup* porque es un lenguaje de marcas, como se ha explicado. La tercera y última característica sería el concepto de *HyperText*, donde lo que nos indica es que el documento, a parte de texto, puede contener enlaces a otros documentos. Sin esta característica, la Web sería imposible de concebir.

Expuesto el concepto básico de lo que es un documento web o HTML, pasaremos a explicar la estructura básica –a nivel de código– de una página web. Dave Ragett también nos indica mediante un ejemplo el esqueleto principal que todo documento web bien formado debe tener [10]:

Código 2.1: Estructura básica HTML

```
<html>
    <head>
      <title> replace with your document's title </title>
    </head>
    <body>

        replace with your document's content

    </body>
</html>
```

En este ejemplo podemos ver que existen tres marcas principales. La primera es `html` la cual indica al navegador que se está leyendo un documento de tipo HTML. Seguidamente tenemos `head`, en donde se definen cosas como el título de la página que vemos al lado del icono del navegador en la barra superior, los estilos, funciones JavaScript, etc. Y por último, tenemos el `body` en donde se encontrará todo aquel texto que el navegador web va a mostrar al usuario.

2.2.2 Selectores CSS

En las primeras versiones de las páginas web, el formato que presentaban estas –color de la fuente, indentado, tamaño de la letra, etc– venía indicado en cada marca de manera explicita mediante la propiedad `style`. Esto generaba un problema, ya que cambiar el estilo de una página web era arduo y poco eficiente. Para resolver el problema se definió el lenguaje o tipo de ficheros CSS.

Según la World Wide Web Consortium (W3C), un CSS se define como [11]:

> CSS es el lenguaje para describir la presentación de las páginas web, incluidos los colores, el diseño y las fuentes. Permite adaptar la presentación a diferentes tipos de dispositivos, como pantallas grandes, pantallas pequeñas o impresoras. CSS es independiente de HTML y se puede utilizar con cualquier lenguaje de marcado basado en XML. La separación de HTML de CSS hace que sea más fácil mantener los sitios, compartir hojas de estilo en las páginas y adaptar las páginas a diferentes entornos. Esto se conoce como la separación de la estructura (o contenido) de la presentación.

Una propiedad muy interesante de los documentos CSS es que utilizan selectores, para especificar o restringir a quien va destinado un cierto formato. Pongamos un ejemplo sencillo; tenemos el siguiente documento web:

Código 2.2: Documento para ejemplo CSS

```
<html>
    <head>
      <title> My First CSS </title>
    </head>
    <body>
        <p id="one">Hello</p>
        <p id="two">World!</p>

```

```
9          It's me again.
10     </body>
11 </html>
```

Y definimos el siguiente CSS o guía de estilo:

Código 2.3: Ejemplo documento CSS

```
1    body = {color: "black"}
2    p.one = {color: "green"}
3    p.two = {color: "red"}
```

Al aplicar esta guía de estilo al documento HTML en el *Código 2.2*, lo que sucederá es que cualquier texto que no está dentro de los párrafos con `id` *one* y *two*, se mostrará en negro; mientras *Hello* se mostrará en verde y *World!* en rojo. Esto lo hemos indicado en la guía de estilos mediante `p.one` y `p.two`, que se lee en el primer caso como "el párrafo con el id `one` tendrá la propiedad `color` a `green`". En definitiva, decimos que `p.one` y `p.two` son ***Selectores CSS***.

2.2.3 Document Object Model (DOM)

Para definir que es un Document Object Model (DOM) nuevamente acudimos a la página de W3C [12]:

> El Document Object Model (DOM) es una interfaz de programación de aplicaciones (API) para HTML válido y documentos XML bien formados. Define la estructura lógica de los documentos y la forma en que se accede y se manipula un documento. En la especificación DOM, el término "documento" se usa en sentido amplio; cada vez más, XML se usa como una forma de representar muchos tipos diferentes de información que pueden almacenarse en diversos sistemas, y gran parte de esto se vería tradicionalmente como Datos en lugar de documentos. Sin embargo, XML presenta estos datos como documentos, y el DOM se puede utilizar para administrar estos datos.
>
> Con el Modelo de objetos de documento, los programadores pueden crear documentos, navegar por su estructura y agregar, modificar o eliminar elementos y contenido. Cualquier cosa que se encuentre en un documento HTML o XML se puede acceder, cambiar, eliminar o agregar utilizando el Modelo de objetos de documento, con algunas excepciones, en particular, las interfaces DOM para los subconjuntos internos y externos de XML aún no se han especificado.

Por la explicación detallada por W3C podemos darnos cuenta de que el uso de esta Application Programming Interface (API) es totalmente necesaria en la Web tal y como la conocemos. Cargar o borrar elementos de un documento web mientras el usuario interactúa con esta página web, modificar el estilo de ciertas casillas de un formulario o actualizar cierto contenido en tiempo real, sin la necesidad de volver a recargar la página.

El DOM no solo está implementado en los navegadores, si no que también está disponible mediante librerías en un gran abanico de lenguajes de programación, es por ello que los desarrolladores pueden elegir aquel lenguaje que sea mejor para la tarea que desean realizar, dándoles gran flexibilidad.

2.3 Análisis del Problema

Con la finalidad de poder realizar un diseño correcto de la funcionalidad de extracción de datos para la creación del *corpus*, debemos clarificar las características que debe tener cada uno de nuestros *corpus*, así como reconocer las posibles dificultades o retos que es necesario solucionar al diseñar e implementar nuestra solución. Por ello, en este apartado hacemos un análisis sobre todo esto, a fin de que sea entendida la solución planeada.

A continuación se enumeran cuales son los requisitos que deberán cumplir nuestros *corpus*:

- ***Diversidad de fuentes.*** Dado que queremos un *corpus* que sea utilizable para diversas tareas/estudios, deseamos tener noticias de distintas fuentes periodísticas, con el fin de tener un abanico amplio de vocabulario, estilos de redacción, enfoques diversos sobre un mismo tema, etc.
- ***Diversidad de información sobre la noticia.*** Al necesitar un *corpus* de ámbito general, deseamos obtener el máximo de información posible –dentro de unos límites– de cada una de las noticias, de modo que la variedad de usos sea mayor.
- ***Posibilidad de generar diversos corpus.*** Además de obtener noticias de distintas fuentes, también deseamos poder generar más de una base de datos de noticias. En este caso las queremos separar por idioma –castellano, catalán–.
- ***Datos escalables.*** Los datos deben ser escalables, tanto para aumentar la información que almacenamos de cada noticia, sin que afecte a la información que ya teníamos guardada; así como con la cantidad de documentos que podemos guardar, sin que por ello nos lleve a perder rendimiento en la consulta de los *corpus*.

Con todos los requisitos citados anteriormente, a continuación se detallarán los problemas que la solución planteada debe resolver. Un requerimiento que ha de tener nuestra solución, es que debe ser lo más general posible, de modo que no haya –en la medida de lo posible– código *adhoc* para ninguna fuente periodística.

Los documentos/noticias dentro de cada uno de los *corpus* contienen una serie de campos que debemos ser capaces de capturar de las diversas fuentes periodísticas. Estos campos pueden estar guardados a lo largo de la página HTML de la noticia o no estar presentes para esa fuente. Además, una misma información puede que esté en distintos formatos, como por ejemplo la fecha o las categorías, y deberán ser homogeneizados para tener unos datos adecuados para cualquier proceso automático que quiera hacer uso de ellos.

Por otro lado, cada fuente periodística lista las noticias de un modo distinto. Unos periódicos contienen una hemeroteca –listados de noticias por fechas–, mientras que otros periódicos

no contienen ningún tipo de hemeroteca y solo incluyen el listado de todas las noticias. Estos listados pueden ser paginados o no paginados, si bien, los no paginados se pueden ver como un caso particular de los paginados –un listado paginado con una sola página.

También existen periódicos en los que se añaden ”noticias”, que en realidad son vídeos o enlaces a páginas donde se muestra información sobre un evento seguido *online*, que no contiene ningún texto periodístico o incluso enlaces a *blogs* externos al periódico. Todo ello se debe tener en cuenta para ser capaces de detectarlo y evitar que se inserte en el *corpus*, y así solo tener noticias del periódico en cuestión.

Otro requerimiento que no se había citado es qué se debe saber de que fuente proviene esa noticia –su *URL* o enlace– y qué pasos se han seguido para procesarla, de modo que, en caso de error, seamos capaces de conocer al detalle los pasos que ha seguido nuestra solución con esa noticia en concreto, más allá de un simple fichero de *log*, dado que vamos a procesar una gran cantidad de noticias.

Por último, es necesario conocer no solo los detalles de aquellas noticias que se han considerado procesadas correctamente, sino que debemos saber todas aquellos enlaces que no se han considerado válidos; ya que puede ocurrir, que estemos procesando de manera incorrecta ciertos enlaces validos y que el código o la configuración no estén preparados para un tipo de formato de noticias.

Además de todo lo dicho para el contenido del *corpus*, no debemos olvidar que la solución que proponemos debe de ser escalable, tanto en la captura de datos como en la consulta y almacenamiento de los datos. Esto es un aspecto crítico, ya que un aumento considerable del número de noticias que deseamos capturar, puede acarrearnos problemas en caso de no proporcionar una solución que pueda adaptarse a esta demanda; lo cual limitaría la diversidad de fuentes y/o cantidad de noticias que podemos tener de estas.

Con todo lo indicado en esta sección, pasaremos a explicar el diseño de la solución que se ha propuesto para tratar todos estos problemas, así como los detalles de la implementación.

2.4 Diseño de la Solución

En la presente sección se detallará la arquitectura diseñada para la extracción de noticias y creación de cada uno de los *corpus*. Esta arquitectura está compuesta por varios módulos, de los cuales se detallarán las tareas que cada uno de estos lleva a cabo. También se detallará la estructura de carpetas utilizada para la organización del código de creación del *corpus*. Finalmente, se explicará las tecnologías utilizadas para la creación de la solución y su justificación.

2.4.1 Arquitectura del Sistema

Para la creación de cada uno de los *corpus*, se ha definido un sistema que contiene cuatro módulos, los cuales realizan tareas bien diferenciadas entre ellos. Nuestra primera tarea, es

la de extraer de las páginas web de los periódicos un listado de las direcciones de las noticias referentes a un día o una categoría. La segunda tarea a realizar es la de recoger de cada noticia –enlace– los campos que hayamos decido obtener. La tercera tarea que hemos querido diferenciar es la de procesar una serie de enlaces de noticias provenientes de una fuente periodística, utilizando para ello el proceso encargado de la tarea dos. Por último, tenemos el módulo de la base de datos, el cual se encarga de guardar todos los documentos –cada una de las noticias– de manera estructurada y permanente.

Por todo lo comentado anteriormente, se han definido los siguientes módulos:

- ***Recogida de URLs.*** Mediante configuraciones específicas para cada uno de los periódicos, este módulo se encarga de obtener una lista de *urls* de noticias y los almacena en archivos junto con la configuración necesaria para procesar cada una de las noticias.
- ***URL a JSON.*** Procesa o extrae la información requerida de una noticia –enlace– utilizando una configuración específica. Este proceso devuelve, como respuesta, un objeto JSON con la información requerida.
- ***Procesador de URLs.*** Utilizando los ficheros con enlaces generados por el módulo de *Recogida de URLs*, extraerá los datos de cada una de las noticias mediante el uso del módulo *URL a JSON*. Una vez obtenido el JSON de la noticia, esta será guardada en disco duro utilizando para ello el último módulo, el de *Base de Datos*.
- ***Base de Datos.*** Este módulo se encarga de procesar cualquier petición de inserción o consulta a cada uno de los *corpus*.

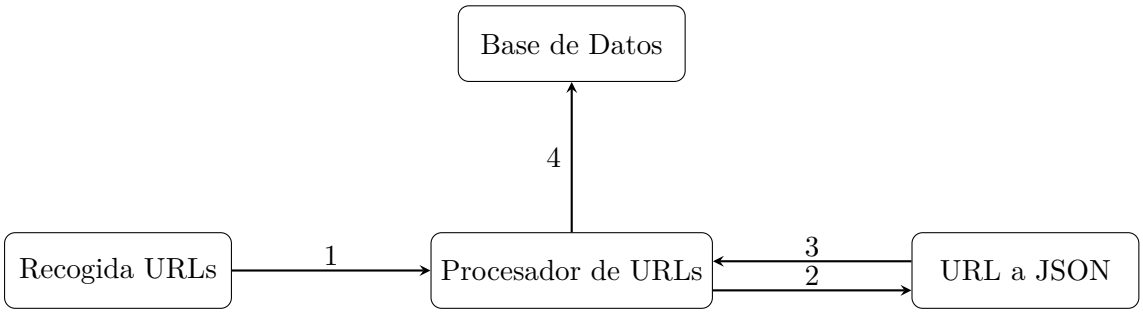

Figura 2.1: Arquitectura para la generación de los *corpus*

En la *Figura 2.1* se muestra una secuencia del proceso seguido para procesar un periódico. El primer paso sería la extracción de los enlaces utilizando el módulo `Recogida URLs`, el cual crearía los ficheros con el listado de enlaces de un periódico. Con estos ficheros el `Procesador de URLs` los iría leyendo y mediante el módulo `URL a JSON`, se recogería toda la información necesaria de cada una de las noticias/enlaces. Finalmente, el `Procesador de URLs` guardaría el resultado obtenido mediante el módulo `Base de Datos`.

2.4.2 Diseño Detallado

En esta sección se explicará en más profundidad el diseño elegido. También se mostrará la estructura del código fuente relacionado con la creación de *corpus*.

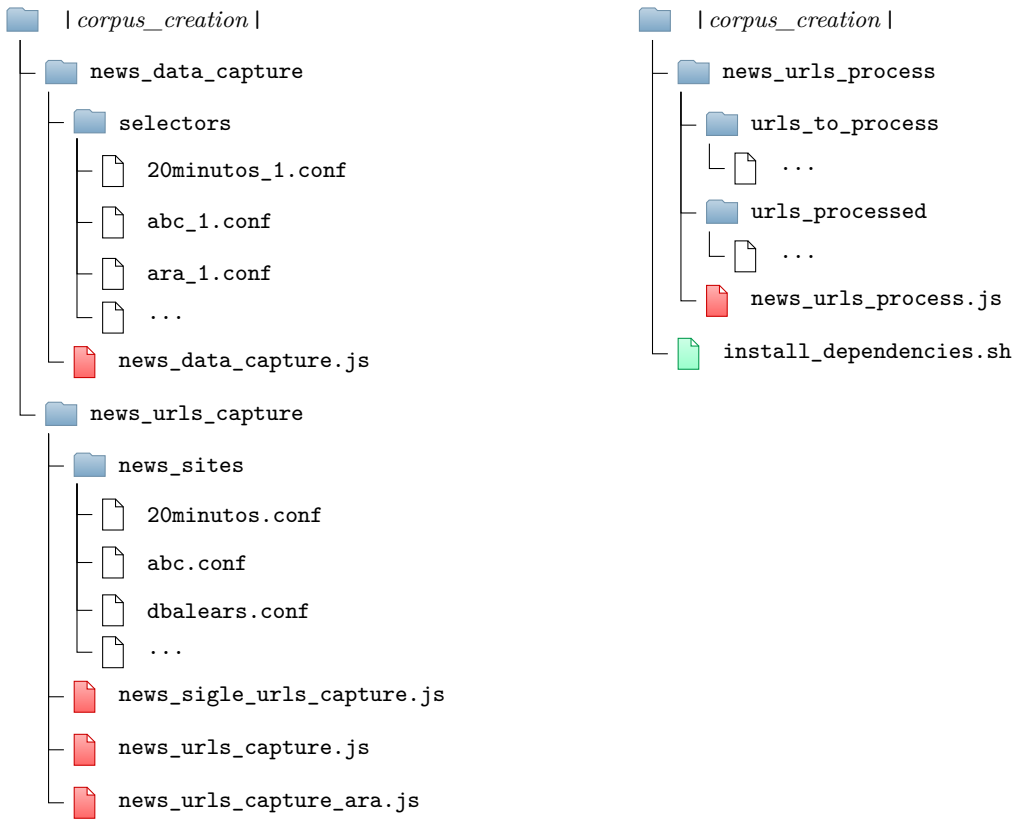

Figura 2.2: Creación de *corpus*: Estructura de directorios

En la *Figura 2.2* se muestra la estructura de directorios y módulos comentados anteriormente. Cada directorio incluye los ficheros necesarios para el correcto funcionamiento del módulo.

La primera carpeta que nos encontramos es `news_data_capture`, la cual referencia al módulo *URLs a JSON*; esta carpeta contiene la implementación así como las configuraciones de extracción disponibles, almacenadas dentro de la carpeta `selectors` –se han llamado *selectors* debido a que se utilizan selectores CSS para su definición–. Por lo general, vamos a encontrar una configuración por diario, aunque podemos encontrar más de una configuración para un mismo diario debido a que a lo largo del tiempo, la estructura de las noticias puede variar.

El segundo directorio que nos encontramos es `news_urls_capture`, la cual hace hace referencia al módulo *Recogida de URLs* que se encarga de recoger todos los enlaces de un listado de noticias de una fuente periodística. Al igual que sucedía con el primer módulo, dentro de esta carpeta encontramos la implementación, así como un directorio que contiene

la configuración para cada periódico, llamado `news_sites`. En este caso sí que existe una sola configuración por cada uno de los periódicos de los que deseamos recoger noticias para el *corpus*.

En tercer lugar, encontramos el directorio `news_urls_process` que engloba el código relacionado con la implementación del módulo encargado de procesar las URLs de las noticias y guardarlas en la base de datos. En este directorio también se encuentran las carpetas `urls_to_process` y `urls_processed`; la primera almacena los ficheros de noticias a procesar y la segunda los archivos procesados junto con otro fichero de resultados.

Tras finalizar con la descripción de la estructura de directorios relacionada con la creación del *corpus*, se pasará a explicar como se ha organizado la base de datos.

Debido a que utilizamos una base de datos NoSQL orientada a documentos –*MongoDB*–, se hablará de colecciones de documentos y la estructura del documento, que son los términos básicos utilizados dentro del motor de bases de datos elegido. Para cada *corpus* se ha creado una base de datos distinta, para la cual se tendrá una colección por cada diario del que hayamos capturado las noticias; cada colección contendrá una serie de documentos, que son cada una de las noticias capturadas. Por último, se dispone de dos colecciones necesarias para la recolección de errores y las noticias procesadas para las que no se han econtrado contenido; las primeras se guardarán en `errors` y las segundas en `nocontent`, cada documento de estas colecciones contendrá la *url* y la configuración –el selector– utilizada para procesar ese enlace.

Para las colecciones de cada uno de los periódicos, todos siguen la misma estructura, que es la siguiente:

Código 2.4: Estructura del documento de una noticia

```
{
    _id: ...,
    content: ...,
    url: ...,
    title: ...,
    keywords: ...,
    description: ...,
    publi_date: ...,
    category: ...,
    author: ...,
    twitter_title: ...,
    twitter_descrip: ...,
    twitter_image: ...,
    language: ...,
    config: ...
}
```

- **`_id`:** Identificador único usado por MongoDB.
- **`content`:** Campo donde se almacena el cuerpo de la noticia.
- **`url`:** Enlace de la noticia.
- **`title`:** El titular de la noticia.

- **keywords:** Palabras clave asignadas por el periódico.
- **description:** Resumen de la noticia.
- **publi_date:** Fecha de publicación de la noticia.
- **category:** Categoría/categorías normalizadas.
- **category_ori:** Categoría/categorías asignadas por el periódico.
- **author:** Autor de la noticia.
- **twitter_title:** Título que se muestra en Twitter al compartir la noticia.
- **twitter_descrip:** Descripción mostrada en el *tweet* al compartir.
- **twitter_image:** Enlace de la imagen que se mostrará en el *tweet*.
- **language:** Idioma de la noticia.
- **config:** Selector usado para extraer todos los campos nombrados anteriormente. Este campo se usa para comprobar y corregir posibles errores en los selectores.

2.4.3 Tecnología utilizada

2.4.3.1 MongoDB

Para el almacenamiento del *corpus* se ha optado por utilizar el motor de base de datos NoSQL *MongoDB*. Este motor se ajusta completamente a las necesidades de este estudio, debido a que proporciona una buena escalabilidad y la flexibilidad dentro de la estructura de cada documento. Una base de datos NoSQL es aquella que no está basada en un sistema relacional, tal y como puede ser la base de datos de *Oracle*; esto implica que no hay un sistema de restricciones que comprueba la consistencia de los datos. En nuestro caso, cada noticia es un documento autónomo y que no depende del resto de noticias procedentes de otras fuentes o dentro de la misma fuente periodística. *MongoDB* guarda los documentos en un formato BSON –binary JSON–, lo cual también ayuda a interactuar más fácilmente con la base de datos. A continuación, pasaremos a citar aquellas características que nos han parecido más destacables:

- ***Alta disponibilidad.*** A través de replicación y recuperación automática.
- ***Escalabilidad horizontal.*** Permite la partición de la base de datos de manera transparente y en consecuencia aumenta el rendimiento del procesado de peticiones.
- ***Flexibilidad.*** No tiene esquemas rígidos de datos.
- ***Auto balanceo de carga.*** El balanceador de carga distribuye automáticamente los datos entre los diferentes servidores de manera uniforme.
- ***Replicación nativa.*** Los servidores se sincronizan automáticamente.

- ***Automatic failover.*** Elección automática de un nuevo servidor primario cuando el anterior ha caído.
- ***Opensource.*** Código abierto.

2.4.3.2 Node.js

Para la creación de los módulos del sistema de captura de noticias, se ha utilizado *Node.js* –entorno de ejecución *opensource* multiplataforma que ejecuta código escrito en *JavaScript* fuera de los navegadores– el cual está orientado a eventos asíncronos. Se ha seleccionado este entorno de ejecución por las siguientes razones/características:

- ***Opensource.*** Código abierto.
- ***Fácil implementación.*** Como *JavaScript* es un lenguaje de *scripting* podemos escribir código de manera muy rápida e intuitiva.
- ***JSON.*** Trabaja de manera nativa y transparente con JSON.
- ***Módulos.*** Existen gran cantidad de librerías que agilizan la implementación de muchas funcionalidades.
- ***Comunidad activa.*** Debido al auge que tiene *Node.js*, aparecen y se actualizan diariamente una gran cantidad de módulos para este entorno.
- ***Rendimiento.*** Gracias al motor de *JavaScript* V8 implementado por *Google* y usado por *Node.js* nos garantiza un alto rendimiento en la ejecución del código.

2.5 Desarrollo de la Solución

En esta sección se explicará qué librerías se han utilizado para la implementación de cada uno de los módulos; también se detallará la estructura de cada uno de los ficheros de configuración utilizados. Por último, se darán detalles de los problemas más importantes que han surgido durante la implementación y cómo se han solucionado.

2.5.1 Contexto previo

Inicialmente se buscó información sobre librerías que ya hubiesen implementado la extracción de datos de un documento *HTML* a *JSON*; en este contexto se encontró la librería *Readability* –actualmente renombrada a *Mercury Toools*– que se nombraba en el *paper* en el cual se creó el *corpus Summari.es* [8]. Inicialmente, el uso de la librería nos pareció una muy buena solución, ya que, con unos simples ficheros de configuración, la librería extraía los datos deseados; no obstante, encontrábamos que la librería era demasiado genérica y esto dificultaba la modificación del código e incluía funcionalidades que nosotros no requeríamos.

Finalmente se usó el concepto básico de la librería –el uso de selectores CSS para la extracción de datos– y se optó por implementar solo aquellos elementos necesarios para nuestra

solución. La implementación de todos los módulos se hizo con el lenguaje de *JavaScript* y el entorno de ejecución *Node.js*, debido a que es un entorno muy utilizado en el contexto de servicios web.

2.5.2 Recogida de URLs

Este módulo, como ya se ha comentado anteriormente, es el encargado de extraer ficheros que contienen una lista de enlaces que apuntan a noticias. Para ello hará uso de configuraciones específicas para cada periódico.

2.5.2.1 Librerías utilizadas

- ***request.*** Esta librería implementa métodos para realizar peticiones *HTTP*. Se utiliza para realizar las peticiones a la web de la cual se quiere extraer el listado de noticias –enlaces–.
- ***cheerio.*** Esta librería es la encargada de transformar un fichero HTML en un DOM, de modo que podremos utilizar los selectores CSS para realizar consultas al DOM y extraer el listado de enlaces a noticias.
- ***fs.*** Librería que implementa todos los métodos necesarios para el uso de ficheros.
- ***date-utils.*** Esta librería implementa métodos para trabajar con fechas; incrementar fecha, decrementar fecha, *parsear* cadenas correspondientes a fechas, etc.
- ***crypto.*** Se usa para hacer un *hash* de una cadena. Así, se pueden utilizar para generar los nombres de los ficheros de salida, de modo que no existe colisión entre nombres de ficheros. Además, mantenemos una longitud fija para el nombre de fichero, de modo que no hay problemas de longitud máxima.

2.5.2.2 Implementación

La principal funcionalidad de este módulo es la siguiente: dada una cierta configuración y una fecha de inicio y una de fin–si son necesarias–, el módulo debe ser capaz de extraer todos aquellos enlaces de noticias que esa *URL* contenga y escribirlos en un fichero de salida. Los enlaces pueden ser de dos tipos: enlaces apuntando a una hemeroteca o enlaces que apuntan directamente al periódico con listado de noticias agrupadas sin ningún tipo de filtro por fecha.

2.5.2.2.1 Periódico con hemeroteca

Decimos que un diario tiene hemeroteca cuando es posible obtener un listado de noticias para una fecha en concreto. En este caso, el módulo necesitará un rango de fechas para los cuales queremos extraer las noticias. Veremos en la configuración que la fecha es utilizada en el enlace a la hemeroteca mediante el comodín `{fecha}`. Este comodín es reemplazado para cada día en concreto dentro del rango de fechas indicado.

A continuación mostraremos un fichero de configuración para el diario *Diario de Mallorca*:

Código 2.5: Recogida de URLs: Ejemplo de configuración de periódico con hemeroteca

```
{
 "site" : "https://www.diariodemallorca.es",
 "url_list" : ["https://www.diariodemallorca.es/{fecha}"],
 "default_conf" : "diaridemallorca_1",
 "default_selector_urls" : "div.noticia a",
 "collection" : "diaridemallorca",
 "remove" : [
    "span.epigrafe",
    "div.autor_comentarios",
    "li.relacionada",
    "div.foto_principal",
    "div.apoyos",
    "div.imagen"

  ]
}
```

Podemos observar que la configuración –al igual que el resto de configuraciones de los otros módulos– tiene un formato de tipo *JSON*; esto nos permite leer estas configuraciones de manera muy senzilla, ya que *JavaScript* tiene integrado de manera nativa el *parseo* y utilización de este tipo de objetos. Dicho esto, pasaremos a explicar qué utilidad tiene cada parámetro:

- ***site.*** Indica cual es el enlace al periódico.
- ***url_list.*** Es un listado de enlaces de los que se van a extraer las noticias. Como el periódico tienen hemeroteca, este listado normalmente solo contendrá un enlace.
- ***default_conf.*** Esta es la configuración básica que se usará para la extracción de los datos de cada noticia. Esta configuración es la que será utilizada por el módulo `URL a JSON`.
- ***collection.*** Es la colección en la cual se insertarán las noticias después de extraer los datos. Esto será utilizado por el módulo `Procesador de URLs` para conocer donde guardar las noticias de este periódico.
- ***default_selector_urls.*** Este selector CSS es el que se utilizará para realizar las consultas al DOM para que nos devuelva el listado de enlaces a noticias.
- ***remove.*** Existen ciertos enlaces que no son noticias pero el selector de enlaces nos los va a devolver. Es por ello, que mediante estos selectores, se eliminan del DOM aquellos enlaces que no estamos interesados en recuperar.

2.5.2.2.2 Periódicos sin hemeroteca

Para poder capturar noticias de periódicos sin hemeroteca, se ha observado que todos aquellos que carecían de ella tenían algo en común: disponían de enlaces a listados de noticias agrupadas por los conceptos más generales –política, economía, sociedad, etc–. Es por ello que se decidió asumir que si un periódico no dispone de hemeroteca, al menos dispone de una serie de enlaces que nos proporcionan listados de noticias.

Otro elemento que introduciremos en esta sección, pero que es aplicable a los periódicos con hemeroteca, es la paginación. Un listado de noticias está paginado cuando el listado completo de noticias no se muestra en un mismo *HTML*, si no que se muestran particionados en varias páginas que debemos recorrer para obtener el listado completo de noticias.

En este caso mostraremos la configuración para el periódico *El Independiente*, el cual carece de hemeroteca:

Código 2.6: Recogida de URLs: Ejemplo de configuración de periódico sin hemeroteca

```
{
"site" : "https://www.elindependiente.com",
 "url_list" : [
    "https://www.elindependiente.com/politica/",
    "https://www.elindependiente.com/espana/",
    "https://www.elindependiente.com/economia/",
    "https://www.elindependiente.com/futuro/",
    "https://www.elindependiente.com/sociedad/"
 ],
 "default_conf" : "elindependiente_1",
 "default_selector_urls" : "div.article-body h3.entry-title a",
 "collection" : "elindependiente",
 "num_max_pages" : 1000,
 "next_page" : "link[rel=\"next\"]"
}
```

Lo primero que podemos observar es que en *url_list* existe más de un enlace y que ninguno de ellos tiene el comodín de `{fecha}`, lo cual nos indica que este periódico no tiene hemeroteca. De cada uno de los enlaces se recogerán todas las *URLs* a noticias que contengan.

Pasaremos a describir la configuración del diario *El Independiente*. Para evitar repetirnos, solo describiremos aquellos parámetros que no han sido descritos en el caso de periódicos con hemeroteca.

- **num_max_pages.** Sirve para limitar el número de páginas máximo que deseamos recorrer. Este parámetro se usa en el caso de los periódicos sin hemeroteca, dado que los listados que se muestran son de todas las noticias de más recientes a más antiguas, por lo que deseamos limitar la cantidad de noticias a recoger.
- **next_page.** Este selector CSS lo utilizamos para acceder al DOM y nos devuelve –si

existe– el enlace a la siguiente página.

2.5.2.2.3 Caso especial: diario *Ara*

Si bien el diseño genérico funciona perfectamente en la mayoría de periódicos, hemos encontrado una fuente periodística que contenía un detalle especial que cambiaba parcialmente el planteamiento. Es el caso del diario *Ara*.

Este periódico no contiene hemeroteca y se listan las noticias por categorías. Hasta este punto, todo parece dentro del ámbito de la implementación realizada; no obstante, existía un pequeño detalle que lo diferenciaba del resto. Para evitar recargar toda la página HTML completa, los desarrolladores de este portal decidieron realizar peticiones asíncronas al servidor y modificar el DOM cuando se pasaba de página. Estas peticiones al servidor devolvían un *JSON* que contenía solo la parte del listado de noticias correspondiente a la página en cuestión.

Por todo ello, y dado que solo era un caso aislado, se decidió optar por una solución *ad-hoc* que recogiese el listado de noticias de esta fuente en concreto.

2.5.2.2.4 Formato del fichero de salida

Los ficheros de salida de este módulo serán los ficheros de entradas del módulo `Procesador de URLs`. Dado que ya se ha descrito anteriormente, solo no centraremos en el formato.

El fichero de salida consiste en un fichero en donde los campos van separados por tabulaciones. Los campos son los siguientes:

- ***URL.*** Enlace de la noticia
- ***Configuraciones.*** Configuraciones posibles, separadas por coma, para la extracción de los datos de la noticia.
- ***Colección.*** Colección en la que será guardada la noticia.

Para asignar el nombre del fichero de salida, se optó por hacer un *hash* de tipo *MD5* de la *URL* de la cual se extraen las noticias, concatenado con la colección en la que se insertarán estas. De este modo podemos saber qué ficheros pertenecen a un cierto periódico.

2.5.3 URL a JSON

Este es el módulo encargado de la extracción de los datos de una noticia dada una configuración. Es decir, se obtienen los datos deseados de un enlace a una noticia y estos se empaquetan dentro de un objeto *JSON*.

2.5.3.1 Librerías utilizadas

- ***express.*** Librería que facilita la creación de aplicaciones y servicios web.

- ***charset-parser.*** Librería que nos permite extraer de un HTML que codificación se ha utilizado.
- ***iconv-lite.*** Libería para decodificar un buffer a un cierto *charset*. La página HTML se decodifica mediante el *charset* indicado por la página o, en su ausencia, a *ISO-8859-15*.
- ***cheerio, fs, request y date-utils.*** Explicados ya en el módulo de `Recogida de URLs`.

2.5.3.2 Implementación

El módulo está diseñado como un servicio web que recibe dos parámetros: el enlace de la noticia (`url`) y la configuración que deseamos utilizar para extraer los datos de la noticia (`config`). El servicio devolverá un objeto *JSON* en caso de poder extraer correctamente los datos o, en caso de error, una respuesta con el estado de *HTTP* diferente a `200` con información sobre el error.

2.5.3.2.1 Fichero de configuración

En este apartado explicaremos la configuración para la extracción de los campos de una noticia. En el *Código 2.7* podemos ver un ejemplo sacado de la configuración para el diario *20 Minutos*

Código 2.7: URLs a JSON: Ejemplo de configuración para la extracción de los datos de una noticia

```
{
  "content" : {
    "selector" : "div.gtm-article-text p",
    "list" : true,
    "list-join" : " \n "
  },
  "title" : "title",
  "keywords" : "meta[name=\"keywords\"]",
  "description" : {
    "selector" : "meta[name=\"description\"]",
    "attr" : "content"
  },
  "publi_date" : {
    "selector" : "meta[property=\"article:published_time\"]",
    "format" : "yyyy-mm-dd"
  },
  "category" : "meta[property=\"article:section\"]",
  "author" : {
    "selector" : "section.article-titles span.author strong a",
    "attr" : ""
  },
  "url" : "meta[property=\"og:url\"]",
  "twitter_title" : "meta[name=\"twitter:title\"]",
  "twitter_descrip" : "meta[name=\"twitter:description\"]",
  "twitter_image" : "meta[name= \"twitter:image\"]",
  "language" : "castellano"
}
```

Como se puede observar, el formato *JSON* tiene prácticamente la misma estructura que en el *Código 2.4*. Esto no es casualidad, ya que en la configuración cada uno de los parámetros está pensado para extraer un campo del *JSON* de la noticia; por ello, no nos detendremos a

explicar de nuevo a que hace referencia cada campo y pasaremos a explicar los campos que puede contener un parámetro de extracción. Un parámetro de extracción es cualquiera de los parámetros en la figura 2.7, salvo `language`.

A continuación, indicamos aquellos parámetros que tienen elementos destacables. No obstante, hay que indicar que todos los parámetros de extracción tienen exactamente los mismos campos de configuración, menos `publi_date` que es un caso especial.

Código 2.8: URLs a JSON: Parámetros de extracción con campos destacables

```
{
    "content" : {
      "selector" : "div.gtm-article-text p",
      "list" : true,
      "list-join" : " \n "
    },
    ...
    "description" : {
      "selector" : "meta[name=\"description\"]",
      "attr" : "content"
    },
    "publi_date" : {
      "selector" : "meta[property=\"article:published_time\"]",
      "format" : "yyyy-mm-dd",
      "separator" : '/',
      "months" : []
    },
    ...
    "author" : {
      "selector" : "section.article-titles span.author strong a",
      "attr" : ""
    },
    "url" : "meta[property=\"og:url\"]",
    ...
}
```

Antes de comenzar a explicar los campos de configuración de un parámetro de extracción, debemos indicar que todo parámetro de extracción se puede definir solo con una cadena de texto que es el selector CSS para ese campo, tal y como ocurren en `url`. Este tipo de definición, asignará unos valores por defecto dependiendo del parámetro de extracción que sea; esto es debido a que cada campo "normalmente" está situado en el HTML de un modo en concreto. De la misma manera, cuando un cierto campo de la configuración del parámetro de extracción no está especificado, también tiene un valor por defecto. Esto nos permite ahorrar líneas repetitivas dentro de la configuración y dentro del código, tratando todos los parámetros de extracción por igual. Dicho esto, los campos generales son los siguientes:

- ***selector.*** Es el selector CSS para buscar la información en el DOM.
- ***attr.*** Indica qué atributo del *item* o *items* devueltos por el selector es el que contiene el dato deseado. La cadena vacía indica que recogemos el texto contenido dentro de una marca HTML; de no ser vacia la cadena, el texto está dentro de un atributo

  indicado. Normalmente el valor será "" o "*content*".

- ***list.*** Indica –*true* o *false*– si el selector devuelve más de un elemento. En caso de no indicarse o estar a falso y devolver más de un elemento, el servicio muestra un *warning*.
- ***list-join*:** Si es una lista, especifica qué cadena vamos a utilizar para unir un selector que devuelva más de un *item*.

Así mismo, a parte de los campos indicados anteriormente, para la fecha de publicación existen unos campos específicos para la conversión de la fecha indicada en el diario a una fecha normalizada:

- ***format.*** Es el formato que se utilizará para *parsear* la fecha. Este formato siempre va separado por -. En el caso del ejemplo en el *Código 2.8*, primero tenemos el año, después el mes con un número de uno o dos dígitos y finalmente, el día indicado en dígitos. Otra opción para el mes es `month` que indica que es el nombre del mes en vez del número.
- ***separator.*** Indica qué separador se está usando en la fecha de la página del periódico.
- ***months.*** En caso de elegirse `month` en el formato del mes, este *array* tendría doce elementos con los nombres del mes –ya sean los nombres completos o abreviaturas para cualquier idioma–.

2.5.3.2.2 Control de errores

El servicio, si está funcionando correctamente, devuelve un *JSON* cuando la *request* sobre la *URL* se ha procesado correctamente o un código de estado distinto de `200` cuando ha habido algún error –en la petición o interno–. En general, se hace uso de los códigos *HTTP*, no obstante, se han añadido dos códigos más para indicar si no se ha encontrado la configuración indicada (`521`) o si ha habido un error en el *parseo* de la configuración (`522`).

2.5.4 Procesador de URLs

Este último módulo se encarga de procesar los ficheros generados por `Recogida de URLs`, mediante el uso del servicio habilitado por el segundo módulo, el de `URL a JSON`. Finalmente, también se encarga de realizar las peticiones de inserción en la base de datos.

Una característica que tiene este módulo es que puede arrancarse sin que haya ficheros para procesar y los irá procesando conforme se vaya añadiendo a la carpeta `urls_to_process`. Por tanto, podemos decir que se trata de un servicio que recibe sus peticiones mediante ficheros y devuelve la salida mediante la generación de dos ficheros en `urls_processed`.

2.5.4.1 Librerías utilizadas

- ***file-state-monitor.*** Librería utilizada para monitorizar el estado de una carpeta.
- ***mongodb.*** Utilizada para realizar peticiones a la base de datos.
- ***fs***: Usada para leer y escribir los ficheros de entrada y salida del proceso.
- ***request.*** La necesitamos para realizar las peticiones al servicio del módulo `URL a JSON`.

2.5.4.2 Implementación

Como ya hemos comentado en la introducción, el módulo se encarga de monitorizar la carpeta `urls_to_process`, en la cual se añaden ficheros de salida del módulo de extracción de enlaces de noticias; es por ello, que el formato de los ficheros escritos por el módulo de `Recogida de URLs` es el mismo que el de los ficheros de entrada del presente módulo.

Este módulo abre un fichero de la carpeta monitorizada y va procesando cada uno de los enlaces de las noticias. Recordemos que, en el fichero tabular, a parte del enlace, se incluía un campo con las configuraciones separadas por comas y la colección a la que pertenece la noticia. El campo de la colección está claro que es necesario para indicar dónde se debe insertar en la *BDA* la noticia una vez obtenido su *JSON*. Por otro lado, debemos aclarar el porqué de la necesidad de más de una configuración.

Durante el desarrollo, nos dimos cuenta que había un cierto tipo de noticias puntuales que seguían una estructura diferente, pero no había forma de detectar cuales eran, es por ello que se optó por una solución de tipo *fallback*. Consideramos que una noticia no tiene contenido si el cuerpo de la noticia está vacío. Por ello, el proceso funciona de la siguiente manera: si recibe un *JSON* de una noticia sin contenido, en caso de haber más de una configuración, prueba con la siguiente configuración hasta que no le queden o encuentre una configuración con la que el contenido no sea vacío; en caso de seguir vacío el contenido, esa noticia se añadirá a la colección `no_content` y si lo tiene, se añadirá a la colección que se haya indicado. Por último, en caso de haber error al procesar la noticia, esta se insertará en la tabla `errors`.

Para controlar los duplicados, se optó por comprobar antes de comenzar a procesar si la *URL* estaba insertada en la colección a la que pertenece la noticia, en `no_content` o en `errors`, de modo que si estaba en alguna de ellas, el enlace no se volvía a procesar.

2.5.5 Limpieza y homogeneización de los *corpus*

El proceso de extracción de datos se realizó mediante selectores CSS lo más restrictivos posibles. No obstante, había ciertos textos que algunos periódicos insertaban dentro del mismo cuerpo de la noticia de manera automática. Para eliminar este contenido, se implementaron algunos *scripts* simples que eliminaban el texto repetido a lo largo de las noticias; esto sucedió con las noticias de *Nació Digital* y *El País*.

Por otro lado, decidimos homogeneizar las categorías, de modo que tuviésemos unas categorías comunes para todas las noticias de los distintos periódicos. Para ello, primero se hizo un listado de todas las categorías encontradas en los diferentes periódicos y después, se estableció una correspondencia –en caso de haberla– a una de las clases que se habían definido para la unificación o se asignó la categoría `otra` en caso de no haberla. Finalmente, para no perder las categorías originales, se creó un nuevo campo en los documentos de las noticias llamando `category_ori`, donde se guardó la clasificación original.

Las 23 categorías definidas, son: `ciencia`, `cultura`, `deportes`, `economía`, `educación`, `entrevistas`, `eventos`, `gastronomía`, `internacional`, `moda`, `motor`, `nacional`, `negocio`, `opinión`, `otra`, `política`, `salud`, `social`, `sorteos`, `sociedad`, `sucesos`, `tecnología`, `televisión`. Donde `otra` es una categoría comodín que se usa para indicar que no se ha podido encontrar correspondencia con el resto de categorías definidas.

2.6 Resultados y Análisis

En esta sección presentamos la distribución de las noticias en cada uno de los dos *corpus*, tanto el de castellano como el de catalán, obtenidos usando la solución descrita en las secciones anteriores.

En la *Tabla 2.1* y en la *Figura 2.3* se muestra como se distribuyen las noticias entre los distintos periódicos en el *corpus* de castellano.

Fuente	**Noticias**
20 Minutos	302 486
ABC	126 317
Diario de Mallorca	12 247
El Confidencial	38 098
El Diario	9341
El Español	49 920
El Independiente	25 811
El País	85 271
Expansión	39 728
La Ser	15 355
Las Provincias	6912
Levante	15 645
Público	13 039
Última Hora	32 131
Total	772 301

Tabla 2.1: *Corpus* en castellano: número de noticias por fuente periodística

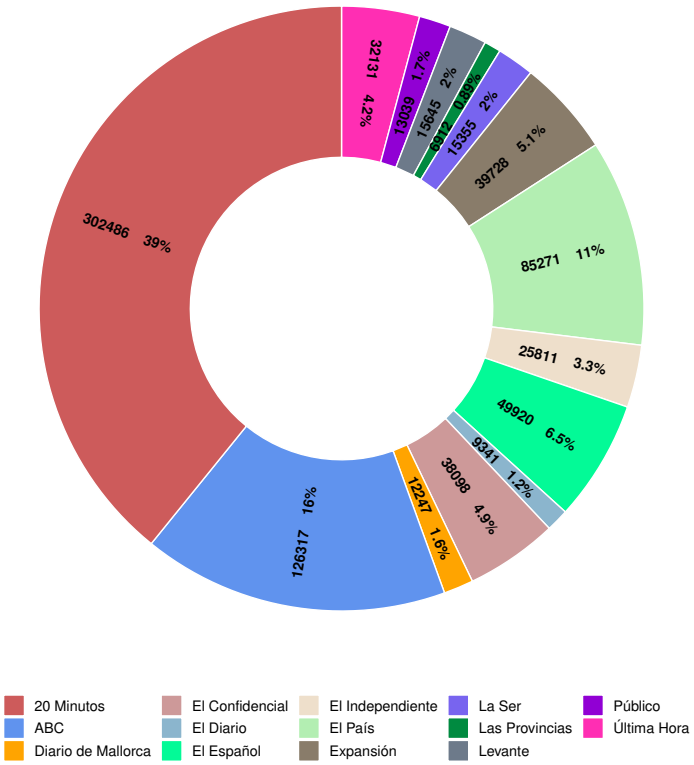

Figura 2.3: *Corpus* en castellano: distribución de noticias por fuente periodística

Como se puede observar, se han obtenido noticias de 14 fuentes distintas para el *corpus* de castellano. De estas fuentes digitales se han extraído un total de 772 301 noticias.

Para analizar la distribución que hemos obtenido utilizaremos la *Figura 2.3*, en la cual podemos ver que la fuente de la que más se han obtenido noticias es *20 Minutos* con 302 486,

un 39% de las noticias totales del *corpus*. Seguidamente, tenemos el diario *ABC* del que se han extraído 126 317 (16%), que es menos de la mitad de las noticias extraídas de la fuente principal. En tercer lugar tenemos *El País*, él cual incluye 85 721 (11%) noticias. Sumando el porcentaje de estas tres fuentes, obtenemos un 66% del total de las noticias. Del resto, tenemos 7 fuentes con entre el 6.5% y el 2%.

Después de analizar la distribución del *corpus* en castellano, pasaremos a hacer lo mismo con el de catalán –ver *Tabla 2.2* y *Figura 2.4*–.

Fuente	**Noticias**
Ara	64 024
DBalears	1567
Diari de Girona	13 104
Diari la Veu	39 673
El Punt Avuí	6956
El Temps	4659
Nació Digital	56 240
Regio 7	11 806
Vilaweb	35 817
Total	233 846

Tabla 2.2: *Corpus* en catalán: número de noticias por fuente periodística

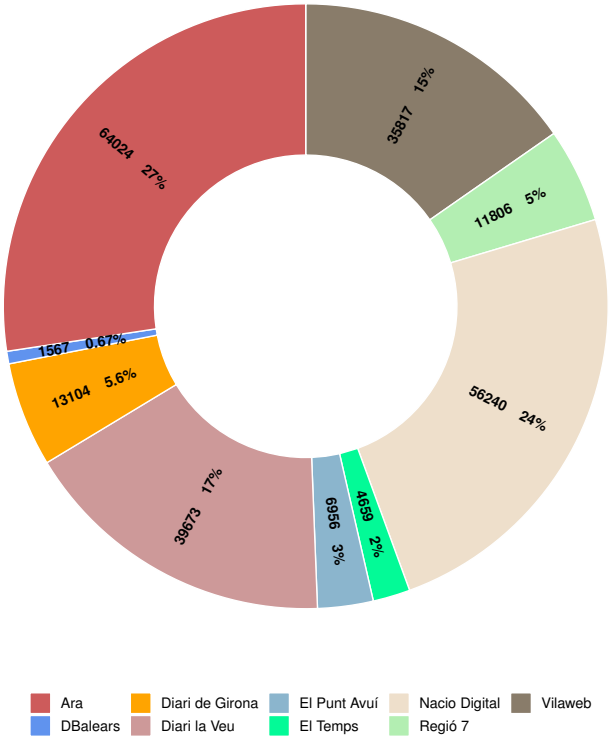

Figura 2.4: *Corpus* en catalán: distribución de noticias por fuente periodística

Para el idioma de catalán se han capturado noticias de 9 periódicos distintos, dado que existen menos periódicos que publican noticias en este idioma. En total se han capturado 233 846 noticias.

A diferencia del castellano, en este caso disponemos de un *corpus* en el que existen cuatro fuentes "dominantes". En concreto, podemos observar que el diario *Ara* contiene el 27% de las noticias totales (64 024 noticias), seguido de *Nacio Digital* con un 24% que son 56 240 noticias; después de estas dos fuentes, está *Diari la Veu* con 39 673 (17%) y *Vilaweb* con 35 817 (15%). Con estas cuatro fuentes, formamos el 88% del total del *corpus* de catalán. El 12% se dividen entre el resto de fuentes sin grandes diferencias, salvo *DBalears* que contiene solo el 0.68% de las noticias totales.

2.7 Aspectos legales

Dado que las noticias obtenidas de los diferentes periódicos pueden contener limitaciones de distribución del contenido, se ha decidido no distribuir el *corpus* generado, ya que podría acarrear problemas legales. No obstante, se liberará el código utilizado para la creación de los *corpus*, lo cual ayudará a todo aquel que lo desee, a poder replicar los datos utilizados para este trabajo. El código está disponible en el enlace de *GitHub* `https://github.com/vicfer-tfg/corpus-creation` bajo licencia GNU General Public License v3.0 (GPLv3).

3 Categorización de noticias

3.1 Estado del Arte

En esta sección, enumeraremos algunos trabajos que tratan aspectos de la clasificación de texto. En este caso no se diferencia entre idiomas, ya que las bases y las técnicas usadas pueden utilizarse para cualquier tipo de texto, idioma, temática, etc. Los trabajos serán enumerados en orden decreciente de año de publicación.

- ***Classification of News Dataset* (2018) [13].** Este trabajo analiza distintos métodos de clasificación supervisados, así como el uso de redes neuronales para clasificar una noticia en base a su título y su entradilla.
- ***Clasificación automática de documentos. Un caso práctico* (2017) [14].** El autor analiza distintos tipos de análisis léxico –sin filtros en el contenido, quitando *stopwords*, con *stemming*, bigramas y trigramas– y como estos afectan a la eficiencia de un clasificador de tipo *Naive Bayes*.
- ***Online news classification using Deep Learning Technique* (2016)[4].** A partir de una red neuronal, los autores clasifican las noticias en cuatro categorías distintas. Miden la efectividad del clasificador variando el número de iteraciones de entrenamiento.
- ***Assessing and Implementing Automated News Classification* (2015)[3].** Utilizan tres clasificadores –*Naive Bayes*, SVM y regresión *softmax*– y distintas técnicas de extracción de características de texto.
- ***Desarrollo de un sistema inteligente para la clasificación de documentos ya digitalizados aplicando redes neuronales supervisadas* (2015)[5].** Los autores detallan el proceso completo de diseño y desarrollo de una red neuronal para la clasificación de documentos, así como los resultados obtenidos.
- ***The Automatic Classification of Thai news by Similarity Method* (2013)[15].** Los autores, utilizan la similitud coseno para catalogar noticias en tailandés en 2 grupos distintos.
- ***Text Classification Using Support Vector Machine with Mixture of Kernel* (2012)[16].** Se detalla la creación de una mixtura de kernels y su uso y beneficios en la clasificación de un conjunto de documentos reducido mediante un clasificador SVM.

En este listado se han nombrado aquellos trabajos que, por el modelo de clasificación o los métodos de extracción usados, nos han parecido más relevantes en relación con el trabajo que se desea realizar en este proyecto. No obstante, existe un amplio y extenso abanico de trabajos relacionados con la clasificación de texto. Estos y otros trabajos, han sido de gran

ayuda para recoger las principales ideas y conclusiones que otros autores han obtenido sobre el tema que estamos tratando.

3.2 Conceptos previos

Para la categorización de noticias, es necesario hacer uso de los algoritmos de aprendizaje automático con el fin de estimar los clasificadores que utilizaremos para la categorización, así como las técnicas de representación vectorial del texto que lo define numéricamente. En esta sección, al igual que en el Capítulo 2, introduciremos las ideas básicas que fundamentan los métodos utilizados en la clasificación de noticias.

3.2.1 Representación del texto

Un algoritmo de aprendizaje automático, trabaja mediante cálculos numéricos. Es por ello que, para poder utilizar las muestras de texto, necesitamos encontrar métodos que transformen el texto en un vector numérico de características que lo defina. En este apartado veremos algunos métodos de representación de texto.

3.2.1.1 *Bag of Words*

La traducción literal de *Bag Of Words (BOW)* es *bolsa de palabras*. La idea principal es que tenemos una lista de palabras, que representan el conjunto de características que definen un texto; de modo, que cada posición del vector de características hará referencia a una palabra en concreto. La posición se determina a partir del orden de aparición de cada palabra a lo largo de los textos de la colección.

Existen dos posibilidades para decidir que valor tomará cada una de las posiciones del vector de características:

- ***Representación binaria:*** Si una palabra aparece en el texto, la posición correspondiente a esa palabra tendrá valor 1, en caso contrario tendrá valor 0.

  Por ejemplo, si tenemos la frase "*cuando cuenta cuentos cuenta cuantos cuentos cuenta*" y caracterizamos un texto por las palabras `cuando`, `cuenta`, `cuentos`, `noche`; se obtendría el siguiente vector como resultado:

`cuando`	`cuenta`	`cuentos`	`noche`
1	1	1	0

- ***Representación aditiva:*** El valor de cada posición del vector, dependerá del número de veces que aparezca en el texto la palabra a la que hace referencia esa posición.

  Siguiendo con el ejemplo de la representación anterior, en este caso obtendríamos el siguiente resultado con la *representación aditiva*:

`cuando`	`cuenta`	`cuentos`	`noche`
1	3	2	0

Podemos observar que la *representación binaria* es una particularización de la *representación aditiva*, en donde el valor que puede adquirir cada elemento del vector es como máximo 1. Es por ello, que la representación de Bag Of Words (BOW) mediante el método *binario* nos restringe la información suministrada en el vector de características, pero nos ayuda a reducir la memoria necesaria para la representación del texto; por ello, si este método es suficiente para nuestro problema, siempre deberemos elegirlo antes que la *representación aditiva*.

3.2.1.2 *Term Frequency-Inverse Document Frequency*

Term Frequency-Inverse Document Frequency (TF-IDF) técnicamente no es un método de representación vectorial del texto o extracción de características, sino un método de pesado de las palabras características o términos. No obstante, hemos preferido explicarlo en este apartado por simplificación y debido a que más adelante, en los resultados, se separará TF-IDF como si de un método de extracción de características se tratase.

El propósito de TF-IDF es pesar la relevancia que tiene cada uno de los términos en un documento teniendo en cuenta la aparición de ese término dentro del conjunto de documentos. Con esto, podemos realzar aquellos términos que son más importantes dentro de ese conjunto de documento. Si un término aparece muchas veces en un conjunto reducido de documentos, será más importante que otro que aparezca en la mayoría de documentos.

Definimos TF-IDF como el producto de la frecuencia de aparición de un término en un documento (TF) por la frecuencia inversa de documentos para ese término (IDF). TF_i se calcula mediante el método aditivo de BOW explicado en el apartado anterior para cada uno de los elementos del vector de características. Por otro lado, IDF_i se obtiene mediante la siguiente ecuación [17]:

$$IDF_i = log\frac{N}{n_i}$$

Donde N es el número de documentos de la colección y n es el número de documentos de la colección que tienen ese término. De modo que cuantos más documentos tengan un término, menor será la importancia de esa palabra en la colección. Por supuesto, cada término elegido para el vector de características, debe estar como mínimo en un documento.

3.2.1.3 *Hashing*

Podemos definir la técnica de *Hashing* como [18]:

> Una aproximación de una función aleatoria $h : \alpha \to \beta$ desde un universo o conjunto de elemento U de tipo α a un conjunto de elementos de tipo β. En la mayoría de los casos, el conjunto de origen es significativamente más grande que el conjunto de destino, por lo que la función no solo corta y mezcla, sino que también produce distorsiones. De hecho, el conjunto de origen puede tener un tamaño infinito, como todas las cadenas de caracteres, mientras que el conjunto de destino siempre tiene un tamaño finito. Además, el conjunto de origen puede constar de elementos complejos, como el conjunto de todos los grafos dirigidos, mientras que

> el de destino suele ser el conjunto de números enteros en un rango fijo. Es por ello, que las funciones *hash* son funciones de muchos a uno.

En el caso de la extracción de características [19], lo que se hace es aplicar la función *hash* sobre cada una de las palabras y transformarlas en índices del vector de características y contabilizar su aparición, tal y como haríamos en un BOW de tipo aditivo.

Las ventajas de usar esta técnica, son las siguientes:

- Se requiere menos memoria para colecciones de gran tamaño, ya que no se debe almacenar el diccionario.
- Puede usarse de manera incremental, ya que no necesita guardar ningún tipo de estado.

Y sus desventajas son las que siguen:

- Se pierde la relación entre índice y palabra, ya que un índice puede pertenecer a varias palabras.
- Dependiendo del tamaño del vocabulario en comparación con el tamaño del vector de características, puede haber colisiones. Una colisión es cuando a más de una palabra le corresponde el mismo índice.
- No se puede aplicar la ponderación por IDF.

3.2.1.4 *Embeddings*

El proceso de generar *embeddings* de palabras, según *Yang Li* y *Tao Yang* [20], tiene como propósito:

> El objetivo de los *embeddings* de palabras es mapear las palabras sin etiquetar a un espacio de baja dimensionalidad y valor continuo, con el fin de capturar la información semántica interna y sintáctica.

Los *embedings* se extraen a partir de redes neuronales con un número reducido de capas y el entrenamiento de estas con *corpus* de texto de grandes dimensiones. Con esto conseguimos que cada una de las palabras aprendida tenga su representación vectorial con la que se pueden realizar operaciones, además, esta representación consigue capturar elementos semánticos y sintácticos relacionados con la palabra. Como ejemplo, los autores de *Word2Vec* exponen lo siguiente en su página web [21]: si realizamos la operación ***vector('king') - vector('man') + vector('woman')*** el vector más cercano es ***vector('queen')***. Como podemos observar, el resultado de la trasformación vectorial mediante técnicas de *embedding* da como resultado algo más que una mera conversión.

Después de explicar el concepto de *embeddings*, no debemos olvidar que estamos intentando representar una noticia como un vector numérico de dimensión constante y los *embeddings* de palabras, convierten las palabras en vectores. El siguiente paso es usar esos vectores para codificar los textos de algún modo, para ello existen distintas técnicas de combinación de palabras, de las cuales citaremos algunas de ellas brevemente:

- **Suma.** La frase o texto la representaremos mediante la suma de los vectores de todas las palabras que forman ese texto. Esta técnica es adecuada cuando el sentido de la frase o texto viene dado por la conjunción semántica de todas las palabras que lo forman; de no ser así, el vector resultante podría contener mucho ruido que reduce la representación final. Es útil sobre todo para texto de longitud corta como *tweets*.
- **Centroide.** Es idéntico a la suma, pero normalizamos el vector resultante respecto al número de palabras que conforman el texto o frase.
- **Palabra próxima al centroide.** Después de haber calculado el centroide, buscamos la palabra que esté a menor distancia de él. Podríamos decir que el texto viene determinado por la semántica de la palabra de referencia encontrada.

3.2.2 Métodos de Aprendizaje Automático y Clasificación

Después de haber explicado los métodos de representación del texto mediante vectores de características que son usados a lo largo de este trabajo, pasaremos a citar aquellos métodos de aprendizaje automáticos que son utilizados para el estudio de clasificación de noticias.

3.2.2.1 Multinomial (*Naive Bayes*)

El método de clasificación *Naive Bayes* parte de la premisa de que las características utilizadas en el proceso de clasificación son condicionalmente independientes entre sí. De aquí podemos deducir el porqué de ingenuo (*naive*) del método, ya que raramente esta suposición va a ser verdadera; no obstante se puede demostrar [22] que a pesar de que las estimaciones de probabilidad de este método son de baja calidad, el resultado obtenido para su uso en clasificación es aceptable. En conclusión, si bien *Naive Bayes* sobreestima la probabilidad a posteriori de que un dato pertenezca a una clase en concreto, esta estimación suele ser suficiente para ser utilizada en el proceso de clasificación.

Para encontrar la clase a la que pertenece un texto, debemos encontrar la clase que maximiza la probabilidad a posterior de pertenencia. Es decir, para conocer la clase a la que pertenece el i-ésimo documento con K características o *tokens* por documento, debemos utilizar la siguiente fórmula:

$$\widehat{c}_i = \arg\max_{c \in C}(P(c|d_i)) = \arg\max_{c \in C}\left(P(c)\prod_{1 \leq k \leq K} P(t_k|c)\right)$$

El modelo de *Multinomial Naive Bayes* es un caso particular utilizado para la clasificación de textos. Este modelo esta descrito en *Manning et al* (2008) [22]. En él se calcula la probabilidad condicional de que una palabra o termino aparezca en una clase mediante la frecuencia de aparición de ese término en los documentos pertenecientes a la clase en cuestión. Es decir:

$$\widehat{P}(t|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}}$$

Dónde T_{ct} es el número de veces que aparece un término t en el conjunto de documentos pertenecientes a la clase c. El divisor es la suma del total de apariciones del conjunto total de términos a lo largo de todos los documentos pertenecientes a la clase c.

Los modelos basados en *Naive Bayes* son interesantes debido a que, computacionalmente hablando, tiene un consumo menor de recursos que otros métodos como pueden ser los SVM y, en algunos casos, conseguir una tasa de predicción similar [23]. Es por ello que se ha incluido este tipo de modelo de aprendizaje entre los considerados para este trabajo.

3.2.2.2 Regresión Logística

Antes de comenzar esta sección es necesario destacar que: dado que es un modelo estadístico complejo, se ha decidido realizar un resumen de [24] en donde se destaquen los puntos clave que permiten obtener este modelo de clasificación.

El modelo de Regresión Logística (RLOG), es un modelo estadístico de regresión basado en los mismos fundamentos que el modelo de Regresión Lineal (RL). Mientras que el modelo RL se usa para aproximar una variable continua a partir de un conjunto de variables independientes –continuas o discretas–, en el caso del modelo RLOG se obtiene el valor de una variable discreta (clase) a partir de una serie de variables independientes que, al igual que el modelo RL, pueden ser continuas o discretas.

Se partirá del diseño de un clasificador binario, por tanto, el modelo RLOG pretende modelar la probabilidad condicional $P(Y = 1|X = x)$ en función de x. Cualquier parámetro desconocido que pudiese haber dentro de la función de probabilidad debe de estimarse mediante máxima verosimilitud.

Para disponer de una función lineal acotada entre 0 y 1, se hará uso de la transformación *logística* o *logit*, $\log \frac{p}{1-p}$, de modo que se obtiene esta función lineal en función de x.

Formalmente, el modelo de regresión logística se extrae a partir de la siguiente función:

$$\log \frac{p(x)}{1 - p(x)} = \beta_0 + x \cdot \beta$$

Y, al resolver para p, nos da como resultado:

$$p(x; b, w) = \frac{e^{\beta_0 + x \cdot \beta}}{1 + e^{\beta_0 + x \cdot \beta}} = \frac{1}{1 + e^{-(\beta_0 + x \cdot \beta)}}$$

Como se puede observar, la probabilidad de "*x pertenezca a la clase 1*", se estima en función de x, unos parámetros: los pesos (w ó β) y el umbral (b ó β_0). Estos parámetros se calcularán mediante máxima verosimilitud.

Para la obtención de los parámetros que maximicen la verosimilitud $L(\beta_0, \beta)$, se hará a partir de muestras de entrenamiento convenientemente etiquetadas. Para cada muestra de entrenamiento x_i y su etiqueta (clase) y_i, calculamos la probabilidad p en caso de que la

muestra pertenece a la clase 1 ($y_i = 1$) y $1 - p$ en caso que la muestra pertenece a la clase 0 ($y_0 = 0$). La función de verosimilitud se define como sigue:

$$L(\beta_0, \beta) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i)^{1-y_i})$$

Dado que queremos maximizar, podemos usar la *log-verosimilitud* para transformar los productos en sumas:

$$\ell(\beta_0, \beta) = \sum_{i=1}^{n} - \log 1 + \mathrm{e}^{\beta_0 \cdot + x_i \cdot \beta} + \sum_{i=1}^{n} y_i(\beta_0 + x_i \cdot \beta)$$

Por último, una forma de estimar la máxima verosimilitud es derivar en función de los parámetros y resolver la derivada cuando da cero. La función que deberemos resolver por aproximación numérica –dado que normalmente no se va a poder resolver de manera exacta– y nos dará los parámetros que buscamos, es la siguiente:

$$\frac{\partial \ell}{\partial \beta_j} = \sum_{i=1} n(y_i - p(x_i; \beta_0, \beta)) \cdot x_{ij}$$

Como se ha comentado, el modelo RLOG propuesto hasta ahora era un clasificador binario. Para tratar con más de dos clases, en vez de disponer de unos parámetros únicos (β_0, β), se debe tener parámetros para cada clase $(\beta_0^{(c)}, \beta^{(c)})$. Para predecir la probabilidad condicional sería

$$P(Y = c \mid \vec{X} = x) = \frac{\mathrm{e}^{\beta_0^{(c)} + x \cdot \beta^{(c)}}}{\sum_{c' \in C} \beta_0^{(c')} + x \cdot \beta^{(c')}}$$

Al igual que para dos clases, se deberían realizar los mismos pasos para el cálculo de los parámetros que maximicen la verosimilitud.

3.2.2.3 Máquinas de Vector Soporte

Para explicar que son las Máquina de Vector Soporte (SVM) citaremos a [25]:

> Las máquinas de vectores soporte (SVM, del inglés *Support Vector Machines*) tienen su origen en los trabajos sobre la teoría del aprendizaje estadístico y fueron introducidas en los años 90 por Vapnik y sus colaboradores.
>
> Las SVMs pertenecen a la categoría de los clasificadores lineales, puesto que inducen separadores lineales o hiperplanos, ya sea en el espacio original de entrada, si estos son separables o cuasi-separables, o en un espacio transformado (espacio de características), si los ejemplos no son separables linealmente en el espacio original. La búsqueda del hiperplano de separación en estos espacios transformados, normalmente de muy alta dimensión, se hará de forma implícita utilizando las denominadas funciones *kernel*.

> Mientras la mayoría de los métodos de aprendizaje se centran en minimizar los errores cometidos por el modelo generado a partir de los ejemplos de entrenamiento (error empírico), el sesgo inductivo asociado a las SVMs radica en la minimización del denominado riesgo estructural. La idea es seleccionar un hiperplano de separación que equidista de los ejemplos más cercanos de cada clase para, de esta forma, conseguir lo que se denomina un margen máximo a cada lado del hiperplano. Además, a la hora de definir el hiperplano, sólo se consideran los ejemplos de entrenamiento de cada clase que caen justo en la frontera de dichos márgenes. Estos ejemplos reciben el nombre de vectores soporte. Desde un punto de vista práctico, el hiperplano separador de margen máximo ha demostrado tener una buena capacidad de generalización, evitando en gran medida el problema del sobreajuste a los ejemplos de entrenamiento.

Las SVM se basan en funciones discriminantes lineales de la forma $\phi : \mathbb{R}^d \to \mathbb{R} : \phi(x; \Theta) = \sum_{i=1}^{d} \theta_i x_i + \theta_0$ donde $\Theta = (\theta, \theta_0) : \theta \in \mathbb{R}^d$ es el vector de pesos y $\theta_0 \in \mathbb{R}$ es el umbral de separación. Al ser un clasificador lineal solo puede separar las muestras en dos clases, esto se conoce como *clasificador binario*. El clasificador decide a cuál de las dos clases pertenece la muestra en base al signo del valor resultante de la función de discriminación para esa muestra.

Como ya se habrá percatado el lector, pueden existir múltiples hiperplanos que separen las muestras correctamente, de modo que las muestras de una clase queden en un lado y las de la otra clase en el otro –*Figura 3.1*, ilustración izquierda–. No obstante, deseamos aquel hiperplano, el cual maximice la distancia con respecto a ambas clases. Es por ello, que finalmente, solo podrá existir un hiperplano solución; los parámetros que definan dicho hiperplano, serán los parámetros resultantes del entrenamiento de la SVM –*Figura 3.1*, ilustración derecha–.

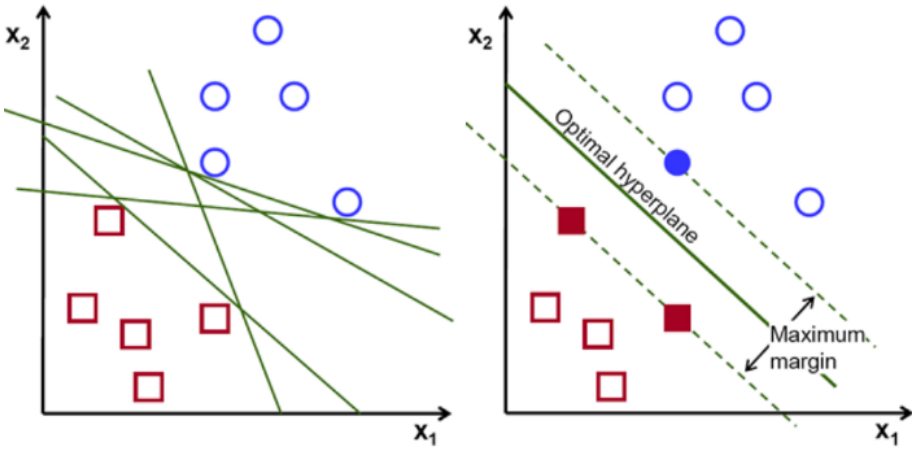

Figura 3.1: SVM: Posibles hiperplanos (izquierda) e hiperplano óptimo (derecha).

Después de haber planteado el caso ideal, el que existe un hiperplano que cumple todo lo citado anteriormente, ¿qué hacer cuando las muestras no son linealmente separables? Tenemos varias situaciones/posibilidades:

- Si las muestras no son linealmente separables debido a que existen muestras concretas que se entremezclan debido a anomalías, pero el conjunto de muestras de cada clase está bien definido y los dos son linealmente separables, entonces podemos *jugar* con el umbral mínimo y asumir esa pequeña tasa de error de clasificación para las muestras de entrenamiento que están mal clasificadas. Esto nos da lugar a un clasificador SVM con margenes blandos.
- Por otro lado, puede ser que realmente no sean linealmente separables, entonces por mucho que queramos modificar el umbral para tolerar muestras mal clasificadas, la precisión del clasificador resultante será nula. En ese caso, podemos hacer uso de los *kernels* que modelan el producto escalar en un espacio alternativo, sin la necesidad de transformar las muestras a ese espacio alternativo; dando como resultado la posibilidad de trabajar en nuevos espacios dimensionales en los cuales las muestras si son linealmente separables.

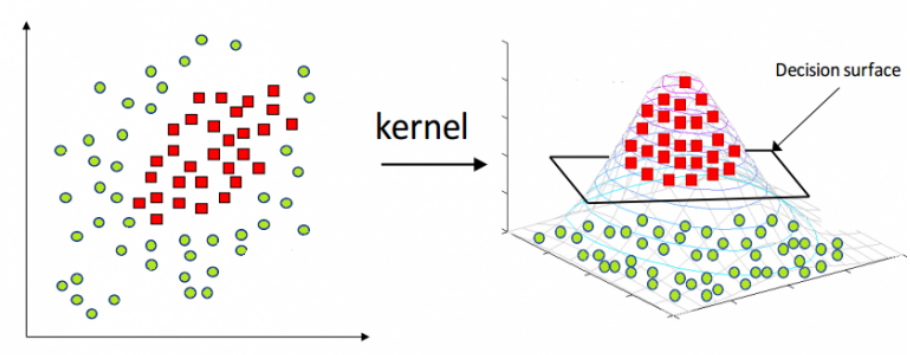

Figura 3.2: SVM: Distribución de muestras linealmente no separables en dos dimensiones (izquierda) y estas mismas muestras en un espacio tridimensional en el que si son separables (derecha)

Hasta este punto solo hemos hablado de la posibilidad de realizar clasificaciones binarias, no obstante existen métodos para poder utilizar las SVM con colecciones de muestras con más de dos clases. Pasaremos a comentar los más extendidos:

- ***One vs One (OVO).*** Se tiene tantos clasificadores de dos clases como combinaciones de pares de clases haya –sin importar el orden; 1vs2 es lo mismo que 2vs1–. A partir del resultado de cada clasificador binario se pueden usar métodos de votación, árboles de decisión, etc.; para realizar la tarea de clasificación.
- ***One vs the Rest (OVR).*** Se tiene un clasificador por cada clase, en dónde se ha entrenado el mismo como si solo estuviesen las muestras de esa clase y, el resto de muestras, perteneciese a otra clase única. También se siguen los métodos de combinación de clasificadores para realizar la clasificación.

3.2.2.4 Redes Neuronales

El modelo de redes neuronales es uno de los paradigmas más extendidos hoy en día en el campo de la computación y la inteligencia artificial. A grandes rasgos, definimos una *red*

neuronal como [26]:

> Una red neuronal es un conjunto interconectado de elementos simples de procesamiento, *unidades* o *nodos*, cuya funcionalidad se basa vagamente en la neurona animal. La capacidad de procesamiento de la red reside en la interconexión entre *nodos* y los pesos de cada conexión; estos pesos se obtienen mediante un proceso de adaptación o aprendizaje de un conjunto de muestras entrenamiento.

Las redes neuronales, están inspiradas en la estructura biológica que forman las neuronas de manera natura en la mente animal. Estas redes neuronales artificiales usadas como modelo de aprendizaje automático, son una simplificación de las naturales y adaptadas a las necesidades prácticas dentro del campo de la computación.

Una red neuronal es un grafo direccional, formada por nodos y conexiones (aristas) direccionales. Los nodos se agrupan por capas, dónde los nodos de una misma capa no pueden conectarse entre ellos. Cada arista tiene asignado un peso específico que irá variando a lo largo del proceso de entrenamiento y cada nodo adquirirá un valor concreto que dependerá del valor recogido de cada una de las conexiones entrantes. El valor de cada una de las conexiones entrantes depende del valor del nodo de origen, el peso asignado a la conexión y una función de umbral o activación; esta función de activación es común para todos los nodos de una misma capa e impide que el valor de la conexión saliente influya en el cálculo del valor del nodo destino si el valor del nodo origen no sobrepasa un cierto umbral.

Como ya se ha comentado anteriormente, las redes neuronales están formadas por capas de nodos, las cuales las podemos separar en tres tipos:

- ***Capa de entrada.*** Solo existe una capa de este tipo en una red neuronal (la primera capa). Los nodos de esta capa no reciben conexiones provenientes de otras capas, por lo que el valor de los nodos de esta capa se calcula a partir de los datos de entrada. El número de nodos en la entrada es igual al número de características que tienen los datos con los que va a trabajar la red neuronal.
- ***Capa de salida.*** Al igual que la capa de entrada, solo existe una capa de salida. Es la última capa de la red neuronal. Esta capa no tiene conexiones de salida a otros nodos. En el caso de clasificación, la capa de salida tiene el mismo número de nodos que clases o categorías existan.
- ***Capa/s oculta/s.*** El número de capas de este tipo y el número de nodos que contenga estas, será dependiente del diseño de la red neuronal. Los nodos de estas capas pueden tener conexiones entrantes de nodos de la capa de entrada o de otras capas ocultas; así mismo, los nodos de la capa oculta pueden tener conexiones salientes a otras a nodos de otras capas ocultas o a los de la capa de salida.

Aclarar, que la descripción dada para cada una de las capas es la más general y simple posible. Existen redes neuronales más elaboradas en las que los nodos de la capa de entrada pueden recibir conexiones entrantes provenientes de otros nodos –normalmente la capa de salida– y los nodos de la capa de salida pueden tener conexiones salientes a otras capas

–incluida la capa de entrada–.

El proceso de aprendizaje de la red neuronal es automático e incremental. Se realiza mediante la variación de los pesos de cada una de las conexiones existentes mediante el algoritmo de *backpropagation* –retropropagación del error–. El error obtenido en la capa de salida se irá propagando hacía *atrás* hasta la capa de entrada e irá influyendo en la variación de los pesos de cada una de las conexiones. Con este proceso, la red neuronal irá minimizando el error obtenido por el sistema.

En la *Figura 3.3* se puede observar una red neuronal con una capa de entrada para datos con una dimensión de cuatro características, seguida de una capa oculta con cinco nodos y por último una capa de salida de tres nodos; en caso de clasificación, este modelo estaría clasificando en tres clases y cada nodo representaría la probabilidad de que un dato observado perteneciese a la clase 1, 2 ó 3 y eligiendo la clase que maximizase la probabilidad de pertenencia.

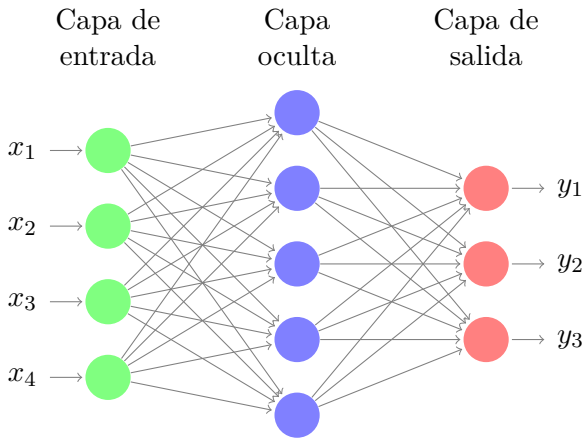

Figura 3.3: Red neural con una capa oculta

3.2.3 Validación

Durante el desarrollo de un sistema de clasificación una de las etapas más importantes, es la validación del sistema. En este punto realizaremos una estimación la fiabilidad de nuestro sistema y si las decisiones tomadas durante la etapa del diseño de este son correctas o no.

Definimos la esperanza estadística de error de la función f dado un conjunto de muestras pertenecientes a X como:

$$p = E[error(f)] = \sum_{x \in X} error(f(x)) \cdot P(x)$$

Por lo general, no es posible calcular la esperanza de error exacta, debido a que se desconoce $P(x)$ o solo disponemos de una aproximación. No obstante, podemos aproximar p utilizando las muestras de entrenamiento $p' = \frac{N_e}{N}$ donde N_e es el número de errores obtenidos al clasificar

las muestras y N el número de muestras. Si N es muy grande, p' sigue una distribución normal $p' \sim \mathcal{N}(p, \frac{p(1-p)}{N})$, con lo que, a mayor número de muestras utilizadas para test, mejor será la aproximación de p' respecto a p. Por ello, cuantas más muestras de test utilicemos, mejor. No obstante, la mayoría de veces disponemos de un número de muestras limitado, que además, también hemos de usar para entrenar el sistema. Es por ello que existen distintas técnicas de particionado para maximizar la aproximación de p' en algunos casos:

- **Resubstitución:** Todos los datos se utilizan tanto para entrenamiento como para test. Este método tiene el inconveniente de que es extremadamente optimista, debido a que en la etapa de entrenamiento ya se está intentando minimizar el número de errores y estamos validando el sistema con los mismos datos con los que se entrenó. La formula en este caso sería $p'_e = \frac{N_e}{N}$.
- **Partición:** Las muestras se dividen en un conjunto de entrenamiento y otro de test, donde el tamaño de entrenamiento suele ser siempre mucho más grande que el de test. La esperanza de error en este caso la aproximaríamos mediante la formula $p'_e = \frac{N'_e}{N'}$ donde N' es número de muestras disponibles en el conjunto de test y N'_e el número de errores obtenidos al clasificarlas.
- **Validación cruzada en B bloques:** Se realizan B particiones distintas del conjunto de muestras, dando lugar a B pares de conjuntos *entrenamiento-test*. El inconveniente que tiene este método es que el coste computacional aumenta linealmente con el número de particiones realizadas, no obstante, obtenemos una mejor aproximación de la esperanza de error. La expresión para el cálculo de la esperanza de error empírico es $p'_e = \frac{1}{N}\sum_{i=1}^{B} N_e^i$, donde N_e^i es el número de errores obtenidos en la clasificación del conjunto de test de la i-ésima partición.
- **Exclusión individual:** Es lo mismo que la validación cruzada en B bloques asignando $B = N - 1$. Es decir, en cada partición excluimos una muestra distinta. Este tipo de partición sería interesante cuando disponemos de muy pocas muestras y no deseamos desaprovecharlas.

Existen métricas para evaluar los clasificadores, estas métricas se basan en tomar como referencia una cierta clase c y observar aquellos documentos que han sido clasificados en la clase en cuestión (*positivos*) y los que, según el clasificador, no pertenecen a dicha clase (*negativos*). Los elementos que han sido clasificados correctamente, son considerados *verdaderos* y los incorrectos *falsos*. En la *Tabla 3.1*, se muestra un resumen con todas las posibles combinaciones:

	Positivos	**Negativos**
Clasificados en *c*	positivos verdaderos (`tp`)	positivos falsos (`fp`)
No clasificados en *c*	negativos falsos (`fn`)	negativos verdaderos (`tn`)

Tabla 3.1: Medidas usadas para el cálculo de las métricas

De los conceptos citados anteriormente se pueden extraer tres métricas:

- ***Precision.*** Es el porcentaje de documentos bien clasificados en la clase c respecto al total de documentos clasificados en dicha clase. Que se traduce en la siguiente fórmula.

$$precision = \frac{tp}{tp + fp}$$

- ***Recall.*** Es el porcentaje de documentos bien clasificados en la clase c respecto al total de documentos existentes para dicha clase. Expresamos esta idea mediante la fórmula.

$$recall = \frac{tp}{tp + fn}$$

- $\boldsymbol{F_1}$**:** Es la media ponderada entre la precisión y la cobertura.

$$F_1 = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

Tal como se ha comentado anteriormente, las métricas citadas anteriormente son indicadores relacionados con una sola clase. En el caso de querer evaluar un modelo en su totalidad, debemos utilizar las mismas métricas en su forma *micro* o *macro*. En este trabajo utilizaremos la *macro* ya que este formato de métricas da el mismo peso a todas las clases, mientras que las *micro* darían más importancia a aquellas clases que tienen más presencia dentro del *corpus*[27].

- ***Macro-Precision.*** Se calcula mediante la siguiente fórmula.

$$macro\_precision = \frac{\frac{\sum_{i=1}^{|C|} tp_i}{\sum_{i=1}^{|C|} tp_i + fp_i}}{|C|}$$

- ***Macro-Recall*:** Se obtiene mediante la fórmula.

$$macro\_precision = \frac{\frac{\sum_{i=1}^{|C|} tp_i}{\sum_{i=1}^{|C|} tp_i + fn_i}}{|C|}$$

- $\boldsymbol{Macro - F_1}$**:** Calculada utilizando la fórmula.

$$F_1 = \frac{\frac{\sum_{i=1}^{|C|} 2 \cdot tp_i}{\sum_{i=1}^{|C|} tp_i + fn_i + fp_i}}{|C|}$$

Donde tp_i, fn_i, fp_i son respectivamente, el número de verdaderos positivos, falsos negativos y falsos positivos para la i-ésima clase.

3.3 Diseño de la Solución

En esta sección se expondrá la arquitectura que se ha diseñado para el estudio de clasificación de noticias. Así mismo, mostraremos la estructura de directorios que se ha seguido para organizar el código. Por último, expondremos las tecnologías utilizadas para este apartado del trabajo y las razones que nos han llevado a decidirnos por ellas.

3.3.1 Arquitectura del Sistema

Para el estudio sobre la clasificación de noticias, se han diferenciado tres tareas o módulos. Nuestra primera necesidad es la de extraer las noticias almacenadas en la base de datos a ficheros tabulares con la información necesaria para nuestro trabajo. Nuestra segunda tarea es la de entrenar y guardar en ficheros los modelos resultantes. Por último, queremos evaluar el funcionamiento de cada uno de los modelos entrenados con distintas técnicas de extracción de características y distinto número de características.

Después de haber introducido de manera general las tareas que hemos identificado, describimos a continuación con más detalle sus particularidades:

- ***BDA a Fichero.*** Dado que tenemos las noticias en la base de datos, nos interesa almacenarlas en ficheros por los siguientes motivos. Por una parte, necesitamos separar el conjunto de noticias en tres subconjuntos disjuntos: *entrenamiento (train)*, *desarrollo (dev)* y *testing (test)*; el de entrenamiento lo usaremos para el aprendizaje del modelo –ya que todos nuestros modelos están basados en aprendizaje supervisado–, el de desarrollo lo utilizaremos para identificar las configuraciones que mejor funcionan y, por último, tenemos el conjunto de test que nos sirve para valorar el rendimiento de cada modelo con sus distintas combinaciones –número de características y método de extracción de estas–. Por otro lado, debemos hacer un proceso de *tokenización* consistente en eliminar del contenido de las noticias los signos de puntuación, saltos de linea, ... y separar todas las palabras del contenido noticia de manera homogénea –en nuestro caso el carácter de espacio–. Por último, en este proceso también se eliminan las *stopwords* [1] del contenido de la noticia.

- ***Entrenar Modelo.*** En esta tarea deseamos variar entre los cinco métodos de aprendizaje automático que se han elegido [2] y entrenarlos usando los ficheros de *train*, variando el método de extracción de características [3] y la cantidad de estas. Después de haber entrenado el modelo, debemos guardarlo en un fichero para tener acceso a él posteriormente.

- ***Testear Modelo.*** La última tarea consiste en probar cada una de las combinaciones de los modelos entrenados y obtener las prestaciones del modelo mediante las medidas de *precision*, *recall* y *F-1*.

[1] Son aquellas palabras que, generalmente, no contribuyen al significado de las frases debido a que se usan con demasiada frecuencia en una lengua. Por ejemplo, conjunciones, artículos, etc.
[2] Multinomial NB, Regresión Logística, SVM Lineal y Perceptrón Multicapa
[3] BOW Booleano, BOW Acumulativo, TF-IDF, *Hashing* y Combinación de *embeddings* (Suma y Centroide)

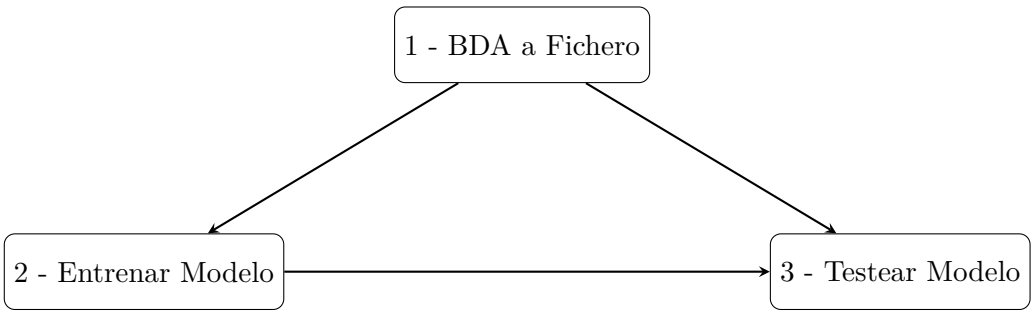

Figura 3.4: Arquitectura para la generación y test de modelos de categorización

En la *Figura 3.4* podemos ver el flujo de dependencias y orden en que se realizan las tareas. Primero generamos los ficheros tabulares, seguidamente se entrena un modelo mediante el fichero tabular de entrenamiento y por último, *testeamos* el modelo entrenado mediante el fichero de *test*.

3.3.2 Diseño Detallado

En esta sección detallaremos la estructura de directorios y ficheros que hemos seguido para el desarrollo del código usado para el estudio de la categorización de noticias. Iremos detallando cada una de las carpetas de la *Figura 3.5*.

data. Contiene los ficheros tabulares de los dos *corpus* –castellano y catalán–. Para cada idioma dispondremos de los conjuntos *train, dev, test*. Por otro lado, también se guardan los modelos de *embeddings* entrenados para los dos idiomas.

feature_extraction. Esta carpeta se llama así para seguir la estructura conceptual de la librería de *Scikit-Learn*. El fichero `text.py` contiene la implementación de la extracción de características para *embeddings*, tanto la suma como el centroide.

model_testing. Esta es la carpeta que contiene todos los ficheros relacionados con la tarea 3 de *testeo*. Primero, tenemos los métodos de *testeo* para todos los modelos implementados usando solamente la librería *Scikit-Learn*, que son ficheros `test_model`. Seguidamente disponemos del *testeo* para las implementaciones con *embeddings* (`embd`). Por último, los ficheros `mlp`, que sirven para probar el modelo perceptrón multicapa.

model_training. Son los encargados de implementar la tarea 2 para cada uno de los modelos de aprendizaje automático.

models. Aquí guardamos las distintas configuraciones –número de características, método de extracción de características y modelo de aprendizaje– que ya han sido entrenadas. Agrupamos los `.model` por número de características usadas y tendremos cuatro ficheros por cada modelo de aprendizaje automático, menos el perceptrón multicapa que lo guardaremos en otra carpeta distinta debido a que la implementación es completamente diferente.

models_mlp. Esta es la carpeta donde se guardan las configuraciones del modelo de perceptrón multicapa. Por cada método de extracción se guardan en `-wi` los parámetros que obtienen mejor resultado para el conjunto de datos de desarrollo. A parte, tenemos el fichero `mlp.model` que representa el objeto de la red neuronal, que sólo es dependiente del número de características. Por último, tenemos por cada extractor de características, un `.model`, que es el objeto que se ha entrenado con el conjunto de *train* y que nos servirá para extraer las características de las noticias que se quieran clasificar.

scripts. Ficheros utilitarios para generar/entrenar todas las combinaciones de modelos deseadas.

utils. En esta carpeta están aquellos ficheros que no van ligados con el entrenamiento o test propiamente dicho. `bda_to_file.py` implementa la primera tarea comentada. `categories.py` contiene la lista de categorías así como métodos auxiliares relacionados con éstas. `io.py` contiene los métodos de carga de ficheros tabulares y carga/guardado de objetos de *Python*. `models.py` contiene métodos útiles para la paralelización del código que más adelante comentaremos.

Figura 3.5: Clasificación de Noticias: Estructura de directorios

3.3.3 Tecnología utilizada

En este apartado expondremos los motivos que nos han llevado a utilizar *Python* para toda la implementación del código relativo a la clasificación de noticias; así mismo, también hemos utilizado *R* como lenguaje auxiliar para la generación de la gráficas que se muestran en el presente trabajo.

3.3.3.1 Python

Python es un lenguaje imperativo de alto nivel de propósito general. Tiene la característica de ser multiparadigma, lo cual permite usarlo como un lenguaje procedural, como un lenguaje orientado a objetos o, en menor medida, como uno funcional; además, es de tipado dinámico, multiplataforma e interpretado –si bien, se puede compilar–. A parte de las características intrínsecas del lenguaje, hemos de destacar las siguientes:

- ***Opensource.*** Es un lenguaje de código abierto
- ***Rápido y legible.*** Gracias a la riqueza de su sintaxis y tipado dinámico, permite realizar cierto tipo de tareas con muy pocas líneas de código. Así mismo, debido a que *Python* distingue los bloques mediante la indentación, hace que el código permanezca legible y bien estructurado.
- ***Portable.*** Puede ejecutarse en cualquiera de los principales sistemas operativos.
- ***Muy extendido.*** Es un lenguaje muy usado entre la comunidad de investigadores y personas dedicadas al *Machine Learning*. Esto nos ha permitido encontrar mucha documentación útil para este trabajo.
- ***Librerías.*** Gracias a la gran recepción del lenguaje, la comunidad está muy activa y existe un gran abanico de librerías y utilidades disponibles.
- ***Multiparadigma.*** Dado que el lenguaje es multiparadigma, nos permite realizar *scripts* simples, realizar programación estructurada o incluso utilizar programación orientada a objetos cuando se requiere; esto nos aporta una gran flexibilidad durante el desarrollo.

3.3.3.2 R

R es un lenguaje multiparadigma, comúnmente utilizado para usos estadísticos, análisis de datos y otros campos en los que la estadística está muy presente; esto es debido a que este lenguaje contiene una gran cantidad de funcionalidades pensadas para este tipo de tareas. Dado que en este tipo de tareas se suele realizar visualización de datos, también dispone de librerías con una gran variedad de tipos de gráficas. Nuevamente, las características que deseamos destacar, son:

- ***Opensource.*** Es un lenguaje de código abierto
- ***Funciones estadísticas.*** Dispone de una gran variedad de herramientas estadísticas que nos ayudan a analizar más fácilmente los datos.

- ***Visualización de datos.*** Facilidad para la generación de gráficas de muchos tipos, lo cual ayuda a agilizar el proceso de análisis de datos.
- ***Comunidad.*** El uso amplio dentro del análisis de datos, permite encontrar mucha información sobre este tema implementado para el lenguaje *R*.
- ***Portable.*** Está disponible para cualquiera de los principales sistemas operativos.
- ***Librerías.*** La comunidad es muy activa y existe una gran variedad de librerías que permiten agilizar el desarrollo del código.

3.4 Desarrollo de la Solución

Después de explicar la arquitectura que hemos elegido para el estudio relacionado con la clasificación de noticias, pasaremos a citar qué librerías se han utilizado para cada una de las tareas, así como explicar los detalles más destacables en el desarrollo del proyecto. Por último, se debe hacer hincapié en que todas las tareas que se expliquen, han sido desarrolladas con el lenguaje *Python*, tal y como se comentó en la sección de diseño de la solución.

3.4.1 BDA a Fichero

En esta tarea se realizarán las siguientes acciones. La primera y más evidente, extraer las noticias de cada una de las dos colecciones –castellano y catalán– y guardarlas en ficheros tabulares con los datos necesarios de cada noticia que son requeridos para el presente trabajo; estos datos son: *el contenido*, *el periódico* y *la categoría*. La segunda, es separar el conjunto de noticias en tres subconjuntos disjuntos: *train* (90%), *dev* (5.5%), *test* (4.5%). Por último, el cuerpo de la noticia se debe *tokenizar*, sin signos de puntuación, saltos de linea, ...; así como de las palabras consideradas *stopword* dentro del idioma de las noticias.

3.4.1.1 Librearías utilizadas

Para el proceso de extracción de noticias de la base de datos a ficheros, se han utilizado las siguientes librerías.

- ***html.*** Se ha utilizado para convertir posibles caracteres especiales –por ejemplo. >, >, >– a sus cadenas correspondientes en formato *Unicode*.
- ***pymongo.*** Utilizada para realizar la conexión a la base de datos de *MongoDB*.
- ***nltk.word_tokenizer.*** *Tokenizador* empleado para generar el *array* de palabras del contenido.
- ***pandas.*** Se usa para generar los ficheros tabulares con las noticias.
- ***sklearn.utils.*** Se utiliza la función *shuffle* para particionar el conjunto de noticias.
- ***re.*** Empleado para eliminación de caracteres indeseados utilizando expresiones regulares.
- ***stop_words.*** Librería que nos permite obtener un *array* con el listado de palabras consideras *stopword* para el castellano o el catalán –entre otros idiomas–.

3.4.1.2 Implementación

El primer paso seguido para la extracción de las noticias en la base de datos a ficheros fue la de normalizar el texto del contenido. En el *Código 3.1* se pueden observar los dos pasos seguidos. Primero se pasó el texto a minúsculas, cambiamos los códigos HTML que pudiesen haber en el cuerpo de la noticia a cadenas *Unicode* y se eliminaron todos los caracteres indeseados. Después de todo esto, se realizó una *tokenización* y se volvió a unir el *array* generado. Se aprovechó para excluir todas aquellas noticias que tuviesen la categoría `otra` o más de una categoría.

Código 3.1: BDA a Fichero: Limipieza del contenido de la noticia

```
# Normalizamos el texto
  content = re.sub('[^0-9a-záéíóúàèòäëïöüçñ \-]', ' ',html.unescape(content.lower()))
# Tokenizamos y volvemos a unir, para que haya un solo espacio de separación entre cada palabra
  content = " ".join([w for w in word_tokenize(content) if w != '-'])
```

Puesto que se deseaba preservar el ratio de noticias por diario en cada uno de los ficheros tabulares –entrenamiento, desarrollo y test– se fueron separando las noticias por diario mientras se iban limpiando, de modo que al finalizar se sabía la proporción de noticias por diario y se pudieron generar los ficheros manteniendo la proporción correcta: un 90% de cada periódico para `train`, un 5.5% para `dev` y el restante 4.5% para `test`; mezclando cada subconjunto mediante la función *sklearn.utils.shuffle* antes de pasarlo a fichero.

Por último, recalcar que los campos que se guardaron en el fichero tabular son el de *contenido*, *categoría* y *periódico*; este último se extrae de la colección a la que pertenecía la noticia en la base de datos.

> Como tal vez se ha dado cuenta el lector, las particiones se han hecho para conseguir la misma proporción de noticias en cada una de las particiones y no se ha tenido en cuenta el número de noticias por categoría para un mismo periódico. Esta hubiese sido la partición correcta para el trabajo actual, no obstante, no se pudo realizar otra partición por el motivo que se citará a continuación. Debido a que los modelos de *embeddings* son computacionalmente costosos de entrenar, se decidió hacer esta tarea en común para este trabajo y el de Fernando Alcina [1]. La tarea entrenamiento se realizó antes de comenzar el trabajo de clasificación propiamente dicho. La tarea de entrenamiento de los modelos requería de las particiones de *train*, *dev* y *test*; las cuales se hicieron manteniendo la proporción de noticias por periódico en cada partición. Ya que en este trabajo se utilizan dichos modelos de *embeddings*, surgió la restricción de continuar con las particiones inicialmente realizadas.

3.4.2 Entrenar Modelo

La segunda tarea que contiene la arquitectura propuesta es la de entrenar cada uno de los modelos usando el fichero de `train` de cada uno de los *corpus* y variando el método de extracción de características y el número de estas.

3.4.2.1 Librearías utilizadas

- ***sklearn.*** Esta es la librería más usada en esta parte del trabajo, es por ello que detallaremos un poco más que partes de ella se han usado y para qué.

  - *feature_extraction.text*: este fichero contiene métodos de extracción de características que normalmente se usan para texto. En concreto, se ha usado las clases `CountVectorizer` para BOW booleano y acumulativo, `TfidfVectorizer` y `HashingVectorizer`.

  - *naive_bayes*: de aquí se ha usado el modelo de aprendizaje `MultinomialNB`.

  - *linear_model*: este fichero contiene la implementación del modelo de regresión logística, `LogisticRegression`.

  - *svm*: de este paquete se ha usado `LinearSVC`.

  - *pipeline*: contiene la clase `Pipeline`, usada para compactar el proceso de extracción de características y un cierto modelo de predicción, de modo que se tienen los dos pasos –extracción y clasificación– en un solo objeto y esto facilita su almacenamiento.

- ***multiprocessing.*** Aquí se encuentra la clase `Pool` la cual ha ayudado a paralelizar el proceso de entrenamiento.

- ***numpy.*** Principalmente se ha usado para realizar operaciones sobre los *arrays*, tales como filtrado, mezclado, etc.

- ***scipy.sparse.*** Se ha utilizado el método `issparse` para comprobar si la variable era una matriz dispersa y en caso de no serlo –y ser un *array*– convertirla mediante `csr_matrix`.

- ***keras.*** Con esta librería se ha implementado el perceptrón multicapa.

- ***gensim.*** Se ha usado para poder cargar los ficheros de *embeddings*.

3.4.2.2 Implementación

Debido a que esta tarea es en la que más detalles técnicos se presentan, se separará la parte de implementación en varias subsecciones. En la *General* se explicará el proceso general de implementación de una *Pipeline* de clasificación. Seguidamente en la de *Paralelización* se concretará como se ha paliado el problema del coste computacional que supone el entrenamiento de los modelos de aprendizaje con todas las variaciones que se han decidido incluir. Por último, detallaremos brevemente la implementación del perceptrón multicapa.

Cabe destacar que, para no aumentar el número de variables para este estudio, se tomó la decisión de utilizar los parámetros que viniesen por defecto en la librería *sklearn* v0.21.2 [28] tanto para la extracción de características como para los modelos de aprendizaje. El único parámetro que se varió fue el número de máximo de características y, si procedía, el número máximo de iteraciones –1000 para el presente trabajo–.

3.4.2.2.1 General

En el aprendizaje y clasificación de textos, es necesario realizar dos tareas básicas. La primera es convertir cada documento en un *array* que caracterice numéricamente ese texto y si

esto se realiza para un conjunto de documentos dados, obtenemos una matriz; seguidamente, se usa la matriz generada para entrenar el modelo de aprendizaje deseado o, en caso de ya estar entrenado, para predecir la clase de los documentos proporcionados.

Estos dos pasos se pueden encapsular en un objeto de la clase `Pipeline`. Para ello, se crean instancias del método de extracción y el modelo de aprendizaje deseado y se añaden a la *pipeline*. Es importante resaltar que el primer objeto de una tubería ha de ser un extractor o *vectorizador* del texto y el último debe ser un objeto clasificador. Cuando se llama al método `fit`, este objeto tubería realiza los siguientes pasos:

1. `fit_transform`. Utilizando el objeto de extracción de características, llama a su método `fit` –con el *array* de textos pasados por parámetro– para que decida qué características se van a tener en cuenta cuando *vectorizamos* numéricamente un texto; seguidamente, el *array* de textos es transformado, utilizando para ello el método `transform`, que devolverá una matriz de <*número documentos*>x<*número de características*>.

2. `fit`. Utilizando el último objeto de la tubería –que es el clasificador–, se realiza el proceso de aprendizaje del clasificador, recibiendo como parámetro la matriz de características.

Para la predicción o clasificación de documentos, al llamarse al método `predict` de la tubería, esta hace los siguientes pasos:

1. `transform`. En este caso se llama al objeto de extracción de características, el cual solo realiza la transformación de los documentos.

2. `predict`. Mediante el último objeto de la tubería, que es el clasificador, se realiza la clasificación de los documentos en base a la matriz de características.

Como se puede ver, al encapsular todos los pasos en una *pipeline*, es mucho más fácil realizar el proceso de entrenamiento y clasificación de textos, ya que con un solo objeto se realizan todos los pasos. Por tanto, al guardar en fichero el objeto de tipo *Pipeline*, estamos guardando una configuración completa, la cual solo deberemos volver a cargar usando ese mismo fichero para clasificar uno o varios textos.

3.4.2.2.2 General: Paralelización

Uno de los puntos más críticos de esta parte del trabajo ha sido la necesidad de entrenar muchas combinaciones de <*número características*> + <*método de extracción*> + <*modelo de aprendizaje*>, ya que el estudio se basa justamente en la influencia de cada uno de estos tres parámetros en la eficiencia del clasificador. Es por ello, que una de las preocupaciones que apareció fue acelerar el proceso de ”generación” de cada una de las *pipelines* entrenadas.

Python, en este aspecto, tiene herramientas muy variadas y útiles. Una de ellas es el *pool* de procesos, el cual gestiona una lista de llamadas a métodos estáticos que se van procesando conforme la anteriores terminan. Este *pool* cuenta con un limitador de procesos que pueden ejecutarse de manera paralela.

Para generalizar estas llamadas, se creó un método estático por cada método de extracción posible; cada uno de estos métodos estáticos recibía como parámetro el número de características máximo, el tipo de modelo de aprendizaje que se quería usar y el número máximo de iteraciones si procedía. Finalmente, se creo un fichero de tipo *script* de *Python* para cada uno de los modelos de aprendizaje, el cual creaba una *pipeline* entrenada por cada posible combinación deseada.

3.4.2.2.3 Perceptrón Multicapa

Para la implementación del perceptrón multicapa, se optó por el uso de la librería *keras*, la cual permite paralelizar el proceso de entrenamiento, utilizando para ello las tarjetas gráficas.

El *Código 3.2* muestra el método implementado para la creación del modelo:

Código 3.2: Entenar Modelo: Creación Perceptrón Multicapa

```
def build_mlp(input_dim, n_layers, n_hidden,
                  n_classes, activation,
                  dropout_input, dropout,
                  verbose = True):

    assert n_layers >= 1 and len(n_hidden) == n_layers

    inp = Input(shape=(input_dim,))
    d_inp = Dropout(dropout_input)(inp)

    h = Dense(n_hidden[0])(d_inp)
    h = BatchNormalization()(h)
    h = Activation(activation)(h)
    h = Dropout(dropout)(h)

    for i in range(1, n_layers):
        h = Dense(n_hidden[i])(h)
        h = BatchNormalization()(h)
        h = Activation(activation)(h)
        h = Dropout(dropout)(h)

    o = Dense(n_classes, activation="softmax")(h)
    mlp = Model(inputs = inp, outputs = o)
    mlp.compile(optimizer="adam", loss="categorical_crossentropy", \
        metrics=["accuracy"])

    if verbose: print(mlp.summary())
    return mlp
```

Los parámetros utilizados para la construcción de nuestro modelo, son los siguientes:

- `input_dim`. El número de características deseado, según qué modelo estemos entrenando.
- `n_layers`. Es el número de capas ocultas. Se utilizan 2 capas ocultas.
- `n_hidden`. Es un array con el número de nodos para cada capa oculta. El modelo utilizado cuenta con 128 nodos por capa oculta.
- `n_classes`. Este trabajo cuenta con 23 clases.
- `activation`. Función de activación para las capas ocultas. En este trabajo será la *relu*.

- `dropout_input`. Probabilidad de que una característica de la entrada se le asigne un valor 0; se puso la probabilidad al 0.3. Ayuda a evitar el sobreentrenamiento.
- `dropout`. Lo mismo que `dropout_input` pero para las capas ocultas. Tiene asignada la misma probabilidad.

Para el entrenamiento se utilizó un *batch_size* de 512 documentos por *batch* y un total de 200 *epoch*. No obstante, dado que cada *epoch* aumentó su coste de forma lineal con el número de características –llegando a más de 6 minutos por *epoch*– se decidió añadir un parado temprano cuando hubiesen pasado más de 10 épocas sin mejorar el valor de acierto respecto al subconjunto de referencia de desarrollo. Así mismo, también se guardaron solo aquellos parámetros del modelo que maximizaban el acierto respecto al conjunto de desarrollo. Se observa que hablamos del conjunto de desarrollo y no del de entrenamiento; esto es importante ya que, si se guardase aquellos que mejoran la tasa de acierto para el conjunto de entrenamiento, se estaría sobreentrenando el modelo.

3.4.3 *Testear* Modelo

En esta sección se presentará la implementación de la tarea de evaluación de los modelos sobre la predicción del conjunto de *test*. En general, lo único que se debe hacer para esta tarea, es cargar un cierto modelo que está guardado en un fichero generado por la tarea anterior y probarlo con un conjunto de test, de modo que se pueda calcular la tasa de acierto.

3.4.3.1 Librearías utilizadas

Debido a que el proceso de aprendizaje y el de testeo tienen muchas partes en común, se utilizaron todas la librerías descritas en el apartado de entrenamiento, menos la librería `multiprocessing`. Además, se utilizó `sklearn.metrics` para calcular las medidas de evaluación de *precision*, *recall* y F_1.

3.4.3.2 Implementación

Para la implementación del testeo de un modelo –con todas sus variaciones– se decidió guardar todas las combinaciones en los modelos de manera ordenada, de modo que se pudiesen recorrer de una forma automática. Como disponíamos de estas tres variables *número características*, *método de extracción*, *modelo de aprendizaje*, se optó por dejar como variable el número de características y fijar las otras dos a valores concretos, de modo que esto ayudara a la creación de ficheros estructurados para obtener las gráficas de evolución.

La tarea de testeo se dividió en dos subtareas. La primera la de calcular una de las cuatro métricas –*accuracy*, *Macro-Precision*, *Macro-Recall* o Macro F_1–dada una cierta configuración del clasificador, variando su número de características. La segunda, calcular la precisión, el *recall* o macro F_1para una cierta configuración para todas las categorías. Por ello se implementó un *script* para cada una de estas subtareas, en vez de uno que contuviese las dos.

3.5 Experimentación

En esta sección se explicará qué distribución de las categorías se ha obtenido en cada uno de los dos *corpus* creados –castellano y catalán–. Así mismo, también se detallarán aquellos criterios seguidos para aceptar o no una noticia como válida para el presente trabajo de clasificación.

3.5.1 Estadísticas sobre categorías

Después de la generación de los dos *corpus* y la homogeneización de las categorías detallado en la sección 4 de este capítulo, debemos presentar las estadísticas que nos ayuden a comprender la distribución de las noticias respecto a las distintas categorías establecidas previamente.

Puesto que vamos a tratar con un clasificador que asigna una sola clase/categoría a cada una de las noticias, se agrupan las noticias con más de una categoría en un grupo llamado *"Más de una categoría"*. Así mismo, las categorías que tienen menos de un 0.56% de representación, no tienen indicado el número de noticias que la conforman.

Primero se mostrará la distribución obtenida para el *corpus* de castellano –*Figura 3.6*–.

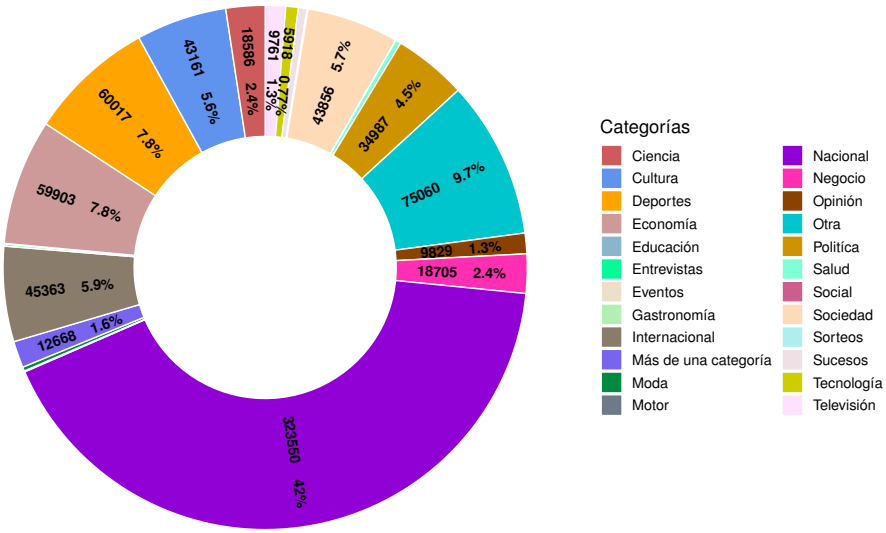

Figura 3.6: Distribución de noticias por categoría para el corpus de castellano

Como se puede observar en la *Figura 3.6*, la mayoría de noticias han quedado clasificadas en la categoría de `nacional`, en concreto un 42% del total. Seguidamente, las categorías que más noticias contienen son las de `economía` y `deportes` con un 7.8% del total. Con un porcentaje alrededor del 5%, nos encontramos con –citadas de mayor a menor porcentaje– `motor` (5.9%), `sociedad` (5.7%), `cultura` (5.6%) y `politica` (4.5%). Por último, el resto de categorías con más de un 0.55% del total de noticias del *corpus* de castellano, son: *ciencia* (2.4%), `negocio` (2.4%), `televisión` (1.3%), `opinión` (1.3%), `tecnología` (0.77%).

Por otro lado, podemos observar como el 9.7% de las noticias han sido incluidas en la categoría de `otra`, por lo que a estas noticias no se les ha podido encontrar ninguna categoría de las preestablecidas. Así mismo, un 1.6% del total tienen más de una categoría.

Después de haber comentado los porcentajes para el *corpus* de castellano, pasaremos a realizar lo mismo en el caso del de catalán:

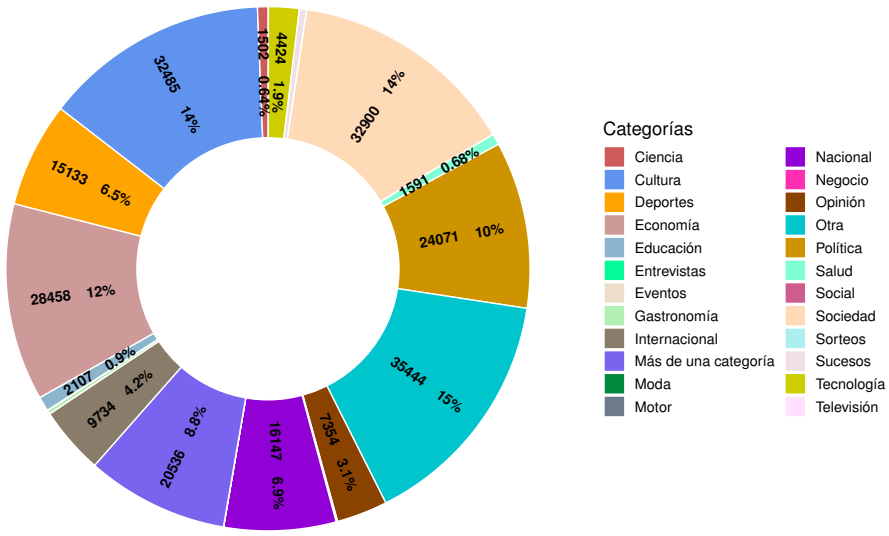

Figura 3.7: Distribución de noticias por categoría para el corpus de catalán

Viendo la *Figura 3.7*, se puede observar que a diferencia del *corpus* de castellano, no existe ninguna categoría que contenga un alto porcentaje de noticias tal y como ocurría con la categoría de `nacional` en el primer *corpus* analizado. Las categorías principales en este caso son `sociedad` (14%), `cultura` (14%), `economía` (12%) y `política` (10%). Seguidamente, encon-

tramos las categorías que contienen más de 3% y menos de un 10% de las noticias, que son las siguientes: `nacional` (6.9%), `deportes` (6.5%), `internacional` (4.2%) y `opinión` (3.1%). Finalmente, las categorías con al menos un 0.56% son: `tecnología` (1.9%), `educación` (0.9%), `salud` (0.68%) y `ciencia` (0.64%).

A diferencia del *corpus* en castellano, en este caso –porcentualmente– nos encontramos con un número mayor de noticias para las que no se ha encontrado ninguna categoría de las predefinidas y muchas más noticias que contienen más de una categoría. En concreto existe un 15% de las noticias que no entran dentro de ninguna de las categorías preestablecidas y un 8.8% de los documentos del *corpus* de catalán que contienen más de una clase.

3.5.2 Estadísticas de noticias utilizables

Después de exponer las estadísticas sobre la distribución de las noticias en el conjunto de categorías, se pueden sacar ciertas conclusiones que pueden ayudar a la simplificación en el trabajo de clasificación.

La primera simplificación está relacionada con las noticias multicategoría que se han obtenido. Dado, que el porcentaje de noticias con más de una categoría es relativamente bajo –1.9% en castellano y un 8.8% en catalán– hemos considerado que estas noticias no participaran en el estudio realizado en este trabajo.

La segunda, simplificación va relacionada con las noticias que carecen de una categoría *homogenea* –las categorías definidas para este trabajo–. La primera idea que se ha valorado es la de incluir la clase/categoría `otra` en el proceso de aprendizaje y categoriación. No obstante, el autor de este trabajo considera que esto dificultaría el proceso de encontrar las características que definen cada categoría y que ayudan al modelo a clasificar, ya que dentro de la categoría `otra` estarían todas las noticias para las cuales, a partir de las categorías originales, no se ha conseguido dar una categoría exacta; esto provoca que en esta categoría heterogénea puedan encontrarse noticias de todo tipo, lo cual confundiría al modelo a la hora de aprender y clasificar.

En resumen, por todo lo citado anteriormente, para este trabajo, se ha decidido excluir aquellas noticias que tengan más de una categoría o que no tengan categoría en aras de concretar y mejorar la fuente de entrenamiento para los modelos de clasificación que la van a utilizar.

Teniendo en cuenta que los *corpus* de este trabajo han sido creados a partir de varios periódicos o fuentes periodísticas, debemos detenernos a analizar el impacto que tiene en cada una de las fuentes la decisión de excluir las noticias que no cumplen los requisitos. Esto nos dará una visión correcta y concreta de los *corpus* que van a ser utilizados para la tarea de clasificación propuesta en este trabajo.

Para valorar el impacto de nuestra decisión de exclusión de las noticias que no cumplan las características citadas anteriormente, primero mostramos, mediante una tabla, como se distribuyen las noticias de cada periódico entre las que no tienen categoría, las que tienen más de una y aquellas que solo tienen una categoría y por tanto, las consideramos como validas.

Así mismo, para facilitar la visualización y análisis, también mostraremos una gráfica que plasma los porcentajes de noticias válidas por cada periódico.

Periódico	**Noticias**	**Sin categoría**	**Multicategoría**	**Categoría única**
20 Minutos	302 486	1123 (00.23%)	477 (00.16%)	300 886 (99.48%)
ABC	126 317	829 (00.65%)	12 (00.01%)	125 476 (99.34%)
Diario de Mallorca	12 247	1484 (12.12%)	92 (00.75%)	10 671 (87.13%)
El Confidencial	38 098	9857 (25.87%)	830 (02.18%)	27 411 (71.95%)
El Diario	9341	503 (05.38%)	6 (00.06%)	8832 (94.55%)
El Español	49 920	18 360 (36.78%)	598 (01.20%)	30 962 (62.02%)
El Independiente	25 811	367 (01.42%)	1048 (04.06%)	24 396 (94.52%)
El País	85 271	16 897 (19.82%)	946 (01.11%)	67 428 (79.07%)
Expansión	39 728	2393 (06.02%)	6773 (17.05%)	30 562 (76.93%)
La Ser	15 355	4618 (30.07%)	127 (00.83%)	10 610 (69.10%)
Las Provincias	6912	652 (09.43%)	899 (13.01%)	5361 (77.56%)
Levante	15 645	899 (05.68%)	608 (03.89%)	14 138 (90.37%)
Público	13 039	254 (01.95%)	2 (00.02%)	12 783 (98.04%)
Última Hora	32 131	16 824 (52.36%)	250 (00.78%)	15 057 (46.86%)
Total	772 301	75 060 (09.72%)	12 668 (01.64%)	684 573 (88.64%)

Tabla 3.2: Distribución de noticias según el número de categorías, para el *corpus* en castellano

Para las noticias sin categoría en el *corpus* de castellano, en la *Tabla 3.2*, podemos observar que existe un periódico que contiene un alto porcentaje de noticias sin categoría, en concreto es *Última Hora* con un 52.36%, muy por encima del resto de fuentes. Seguidamente tenemos cuatro fuentes que contienen un porcentaje alto de noticias sin categoría; en concreto: *El Español* (36.78%), *La Ser* (30.07%), *El Confidencial* (25.87%) y *El País* (19.82%). Por último, destacar las fuentes con menos exclusiones –a nivel porcentual– son: *Público* (1.95%), *El Independiente* (1.42%), *ABC* (0.65%) y *20 Minutos* (0.23%).

En cuanto a las noticias con más de una categoría, podemos observar que es una característica que se cumple con mucha menos frecuencia que la de no tener categoría. Concretamente, las fuentes periodísticas con más noticias con multicategoría, son: *Expansión* (17.05%) y *Las Provincias* (13.01%). Por otro lado, los periódicos que menos noticias multicategoría contienen, son: *El Diario* (0.06%), *Público* (0.02%) y *ABC* (0.01%).

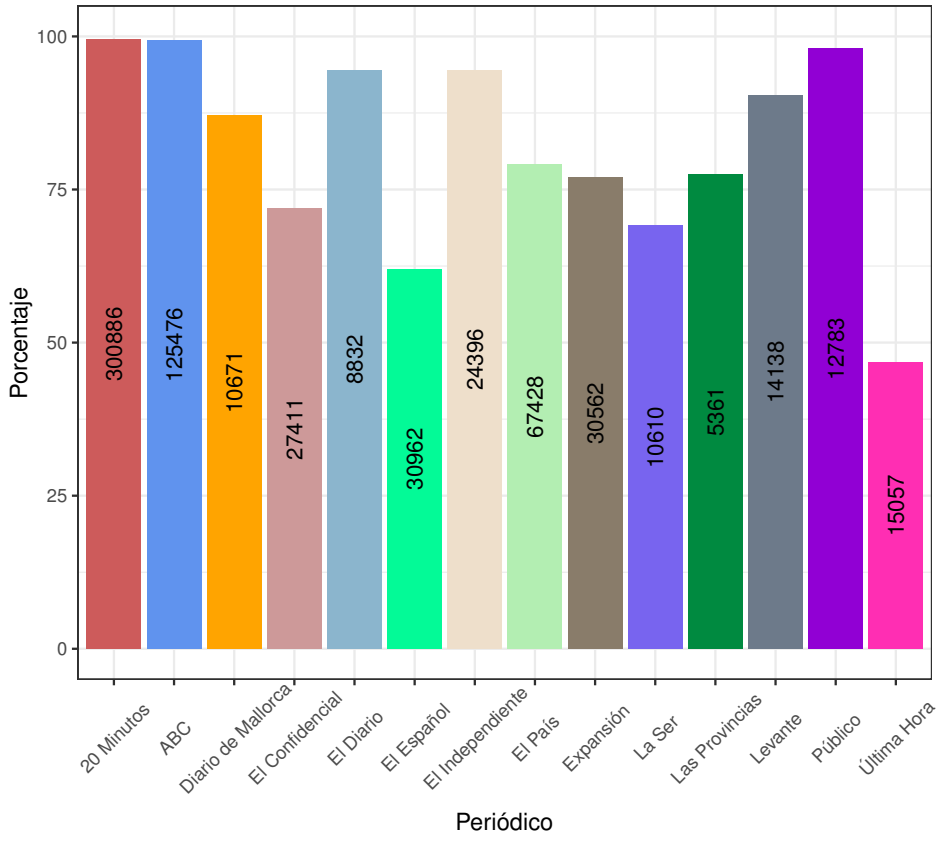

Figura 3.8: Distribución de noticias válidas por periódicos en castellano

Para comentar la distribución de noticias válidas haremos uso de la *Figura 3.8*, en la cual se aprecia gráficamente las diferencias de porcentaje. Claramente, se puede percibir que las fuentes que han sido más mermadas por nuestras restricciones son: *Última Hora* –con menos del 50% de las noticias originales–, seguido de *El Español* que se ha reducido en casi un 40% y *La Ser* con menos del 70% de las noticias originales. Los periódicos que menos se han visto afectados –con menos de un 10% de noticias excluidas– son: *Levante*, *El Confidencial*, *El Español*, *Público*, *ABC*, *20 Minutos*.

Después de haber analizado como han afectado las restricciones sobre el *corpus* de castellano, pasaremos a realizar el mismo análisis para el de catalán. Para ello, primero observaremos la *Tabla 3.2* en la que encontramos la distribución de noticias para cada periódico y después analizaremos la *Figura 3.9*, para tener una visión general de como ha mermado el porcentaje de noticias según la fuente periodística.

Periódico	**Noticias**	**Sin categoría**	**Multicategoría**	**Categoría única**
Ara	64 024	6929 (10.82%)	329 (00.51%)	56 766 (88.64%)
DBalears	1567	367 (23.42%)	0 (00.00%)	1200 (76.58%)
Diari de Girona	13 104	1051 (08.02%)	927 (07.07%)	11 126 (84.91%)
Diari la Veu	39 673	5373 (13.54%)	0 (00.00%)	34 300 (86.46%)
El Punt Avuí	6956	20 (00.29%)	0 (00.00%)	6936 (99.71%)
El Temps	4659	1097 (23.55%)	448 (09.62%)	3114 (66.84%)
Nacio Digital	56 240	8188 (14.56%)	17 693 (31.46%)	30 359 (53.98%)
Regió 7	11 806	5330 (45.15%)	72 (00.61%)	6404 (52.24%)
Vilaweb	35 817	7089 (19.79%)	1067 (02.98%)	27 661 (77.23%)
Total	233 846	35 444 (15.16%)	20 536 (08.78%)	177 866 (76.06%)

Tabla 3.3: Distribución de noticias según el número de categorías, para el *corpus* en catalán

En la *Tabla 3.3* notamos que el periódico que contiene más noticias sin categoría es *Regió 7* (45.15%). Además, tenemos tres fuentes alrededor del 20%: *El Temps* (23.55%), *DBalears* (23.42%) y *Vilaweb* (19.79%). Por último, solo existen dos periódicos que contienen menos del 10% de noticias sin categoria: *Diari de Girona* (8.02%) y *El Punt Avuí* (0.29%).

Por otro lado, en el apartado de noticias con más de una categoría, vemos que existe una fuente que contiene una gran cantidad de noticias multicategoría y que destaca por encima del resto; esta es *Nació Digital*, la cual presenta un 31.46% de noticias multicategoría respecto al su total. Además, si hacemos el cálculo, vemos que el 86.16% de las noticias multicategoría del *corpus* de catalán provienen de este periódico. Es por ello, que el resto de fuentes periodísticas presentan un porcentaje muy bajo –menor del 10%– o, incluso, algunas no contienen ninguna noticia multicategoría, cosa que no pasaba en el *corpus* de castellano.

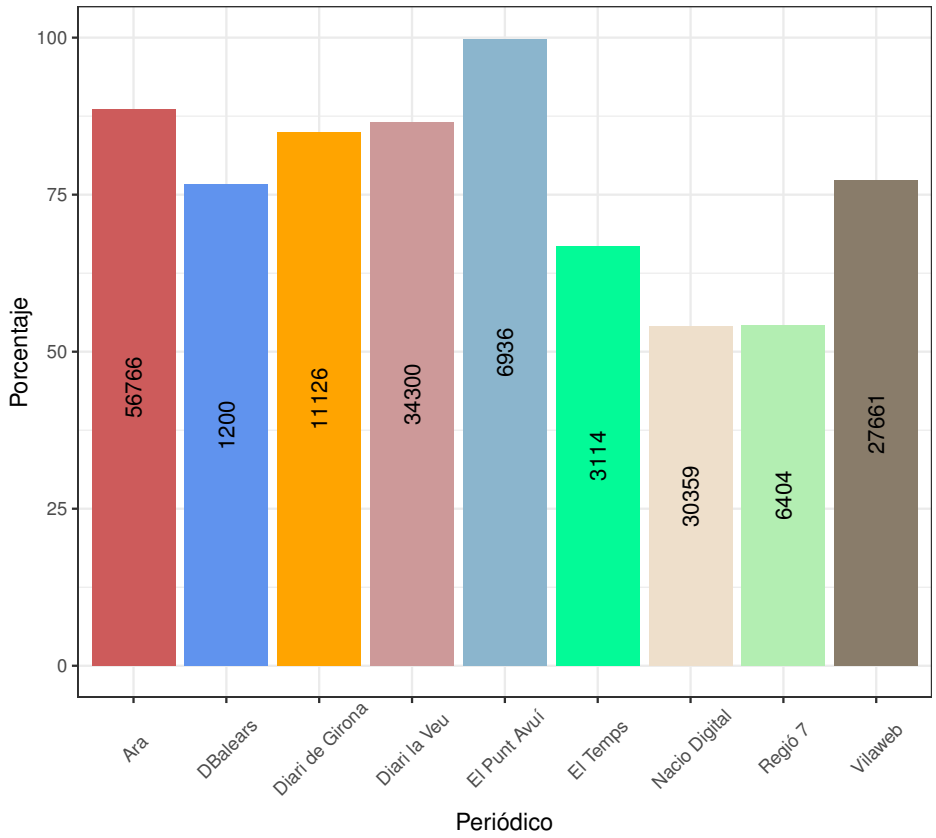

Figura 3.9: Distribución de noticias válidas por periódicos en catalán

Después de haber comentado la distribución de las noticias para cada uno de los periódicos del *corpus* de catalán, pasaremos a analizar el porcentaje de noticias válidas para cada fuente mediante la *Figura 3.9*. En general, podemos percibir que las restricciones aplicadas a los *corpus* han afectado en mayor medida al *corpus* en catalán que a su homónimo en castellano. Dos de las nueve fuentes han reducido su tamaño en casi un 50%, éstas son: *Nació Digital* y *Regió 7*. Después está *El Temps*, el cual no contiene más del 70% de las noticias válidas. Así mismo el resto de periódicos no han reducido su tamaño en más del 25%, destacando *El Punt Avuí*, el cual solo contenía un 0.29% de noticias que no cumplían las restricciones preestablecidas.

3.6 Resultados y Análisis

En esta sección se presentarán los resultados, tanto de la fase de desarrollo como la de test. Así mismo se analizarán aspectos que pueda presentar cada uno de los modelos mediante la ayuda de gráficas y tablas. Se persiguen tres metas en esta sección, que son las siguientes:

- Estudiar la evolución de las métricas *accuracy* y macro F_1 en relación con distintos modelos de extracción de características y, además, como influye el número de características en combinación un cierto modelo de aprendizaje y un método de extracción dado. También se seleccionarán de cada modelo las configuraciones que maximicen cada una de las métrica y minimicen el número de muestras, lo cual se verá reflejado en las gráficas mediante puntos de referencia para cada método de extracción.
- Con la elección de las mejores configuraciones, de cada modelo, que maximicen las dos métricas citadas. Se medirán y estudiará la eficiencia de dichas configuraciones respecto al conjunto de test del *corpus* para el que se haya entrenado el modelo. Esta eficiencia se medirá en base a cuatro métricas: *accuracy*, macro *precision*, macro *recall* y macro F_1.
- Profundizar en aquellas configuraciones que presenten mejor puntuación de *accuracy* y macro F_1 en el conjunto de test para analizar la distribución que sigue el *precision* y *recall* a nivel de clase, y así ver las diferencias entre configuraciones.

Las métricas citadas dan la siguiente información:

- ***Accuracy.*** Esta métrica indica como de eficiente es un clasificador catalogando un conjunto de noticias. Se compara el número de aciertos frente al total de noticias en el conjunto.
- ***Precision* y *Macro Precision.*** Indica como de eficiente es un modelo en identificar noticias de una categoría o, en el caso de la macro, un conjunto de categorías.
- ***Recall* y *Macro Recall.*** Esta métrica es un indicativo de como de eficiente es el clasificador en devolver el total de noticias de una clase o, en el caso de la macro, de un conjunto de clases.
- F_1 **y *Macro* F_1.** Sirve para medir como de balanceado está un clasificador respecto a las dos métricas anteriores, *precision* y *recall* (o las macro, si calculamos la macro F_1).

Como se puede observar, las dos métricas que se han deseado maximizar, en algunos casos, puede implicar objetivos diametralmente opuestos, y la configuración resultante tendrá comportamientos distintos con la clasificación de una u otra clase. Un clasificador con una buena macro F_1, tendrá una eficiencia similar en el mayor número de clases posibles. Por otro lado, un clasificador con una alta tasa de acierto, puede dejar ciertas clases minoritarias de lado, para conseguir mejorar su acierto en las clases mayoritarias.

Debido a las restricciones impuestas por los modelos de *embeddings*, comentadas en la *3.4.1.2*. Existen ciertas categorías que no contienen muestras para el conjunto de desarrollo o

de test. Para obtener una métrica consistente entre los dos *corpus* y las particiones de desarrollo y test, se decidió no tener en cuenta aquellas categorías en las cuales no hubiese algún conjunto de test o desarrollo en uno de los dos *corpus* que no tuviese muestras. Las categorías que han sido omitidas para el apartado de resultados, son las siguientes: `entrevistas`, `eventos`, `moda`, `motor`, `social`, `sorteos`, `televisión`. En total se han omitido 698 noticias en la partición de desarrollo del *corpus* de castellano y 579 en la de test. Por su parte, en el *corpus* de catalán se han omitido 5 noticias en el conjunto desarrollo y ninguna en el de test.

Por último comentar que todas configuraciones se han entrenado para el siguiente número de características: 10, 50, 100, 500, 1000, 5000, 10 000, 25 000, 50 000, 75 000, 100 000, 125 000, 150 000, 175 000, 200 000, 225 000, 250 000, 275 000, 300 000. Si bien, para el perceptrón multicapa solo se ha llegado hasta las 100 000 características, debido al incremento del coste temporal ligado al número de características.

3.6.1 Multinomial NB

A continuación, se pasará a comentar el modelo multinomial *naive bayes*. Este paradigma presenta un método de aprendizaje que no requiere de un alto tiempo de entrenamiento, debido a que no precisa de iteraciones como otros paradigmas que se citarán.

Una limitación que tiene este modelo, es que precisa de valores numéricos positivos –incluyendo el cero–, al menos en la implementación de *sklearn*; esto hace que los métodos de extracción de características de *Hashing* y *Embeddings* no puedan ser usados.

3.6.1.1 Ajuste de parámetros

En esta sección se elegirán la configuraciones del modelo *multinomial naive bayes* que mejores resultados presenta para la tasa de acierto – *accuracy*– y para la puntuación macro F_1 . Para medir estas dos metricas se utilizará la partición de desarrollo. Así mismo es necesario recordar que los parámetros a determinar son el número de características y el modelo de extracción de características.

3.6.1.1.1 *Corpus* de castellano

En la *Figura 3.10* se observa la evolución de la tasa de acierto del modelo *multinomial naive bayes* conforme se va aumentando el número de características para la partición de desarrollo del *corpus* de castellano. Cada línea representa la tasa de acierto obtenida para cada uno de los tres métodos de extracción de características utilizados junto con el modelo actual.

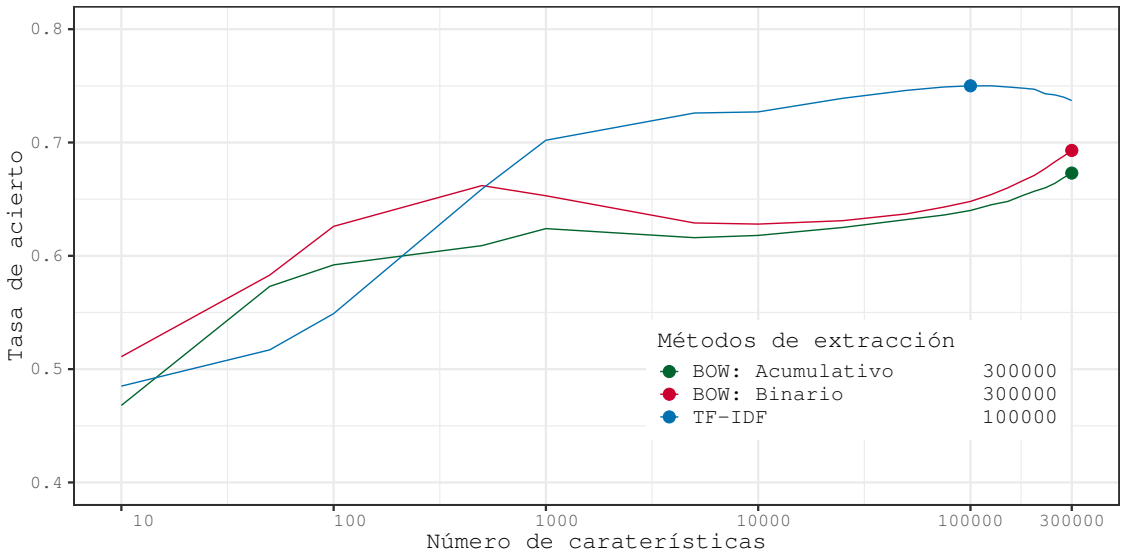

Figura 3.10: Multinomial NB: Tasa de acierto respecto al número de características para la partición de desarrollo del *corpus* de castellano

Se observa, para la *Figura 3.10*, que el modelo no ha conseguido sobrepasar el 0.75 de tasa de acierto para ninguno de los métodos de extracción. Así mismo, los métodos BOW

producen una caída notable del *accuracy* al llegar las 500 características, incrementándose de nuevo a partir de las 35 000 hasta las 300 000 que es el máximo de características para este trabajo. No obstante, los dos métodos *BOW* producen una tasa de acierto significativamente menor que la obtenida con *TF-IDF*. Por último, al utilizar el método *TF-IDF* se aprecia como el modelo aumenta su *accuracy* rápidamente hasta que se llega a las 1000 características, a partir de las cuales la tasa de acierto sigue aumentando más lentamente hasta llegar a las 100 000 características en donde se encuentra el *accuracy* máximo, para después descender gradualmente hasta el número máximo de características.

Para **maximizar la tasa de acierto** mediante el modelo *multinomial naive bayes*, se tomará como método de extracción el ***TF-IDF*** y una dimensionalidad de **100 000** para el vector características. Esta configuración produce una tasa de acierto de 0.750±0.005 en la partición de desarrollo.

Se pasará a la elección de los parámetros que maximicen la puntuación macro F_1 mediante el análisis de la *Figura 3.11*.

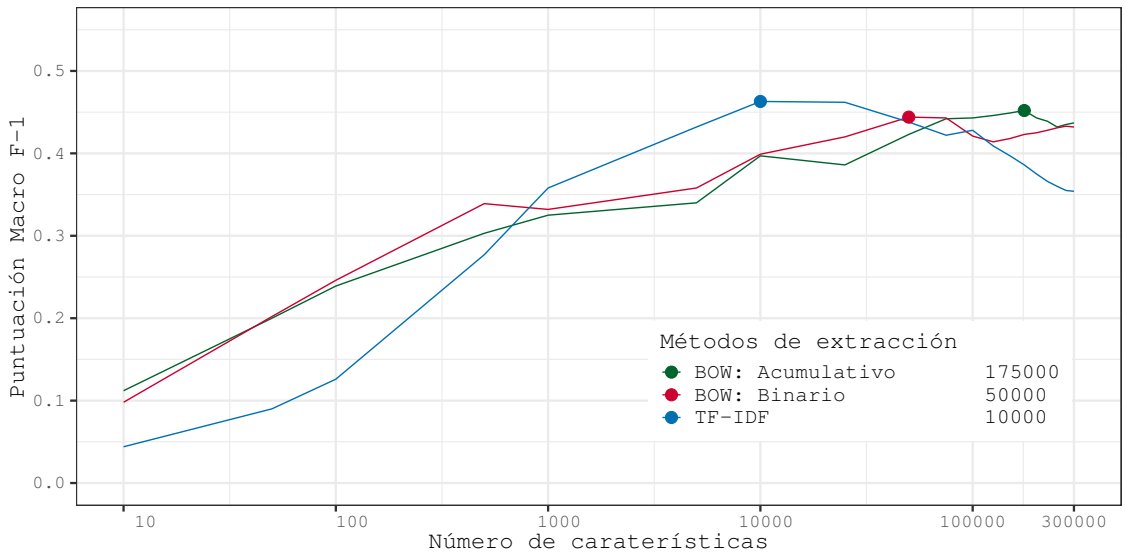

Figura 3.11: Multinomial NB: Puntuación macro F_1 respecto al número de características para la partición de desarrollo del *corpus* de castellano

Mediante la *Figura 3.11*, se puede observar que el modelo se sitúa por debajo de 0.5 en la métrica actual. Por un lado, la curva de evolución de la macro F_1 al utilizar el método TF-IDF es similar a la que se muestra en la tasa de acierto para este mismo método; si bien, la caída de la puntuación es a las 10 000 características, lo cual indica que el aumento tasa de acierto hasta las 100 000 se realiza mediante la focalización del modelo en las categorías mayoritarias en detrimento de las minoritarias. Por otro lado, se observa que al utilizar los métodos BOW, la puntuación macro F_1 va aumentando paulatinamente y no presentan una bajada constante tal y como si que ocurre con TF-IDF; lo cual indica que, al utilizar los métodos *BOW*, el modelo no se centra en las categorías mayoritarias.

Los parámetros elegidos para **maximizar la puntuación macro $\mathbf{F}_1$** para el *corpus* de castellano son: el método de extracción ***TF-IDF*** y una dimensionalidad del vector de características de **10 000**; lo cual proporciona una puntuación macro F_1 de 0.463±0.005 en la partición de desarrollo.

3.6.1.1.2 *Corpus* de catalán

Se procederá a analizar la *Figura 3.12*, la cual presenta la tasa de acierto en función del número de características para la partición de desarrollo.

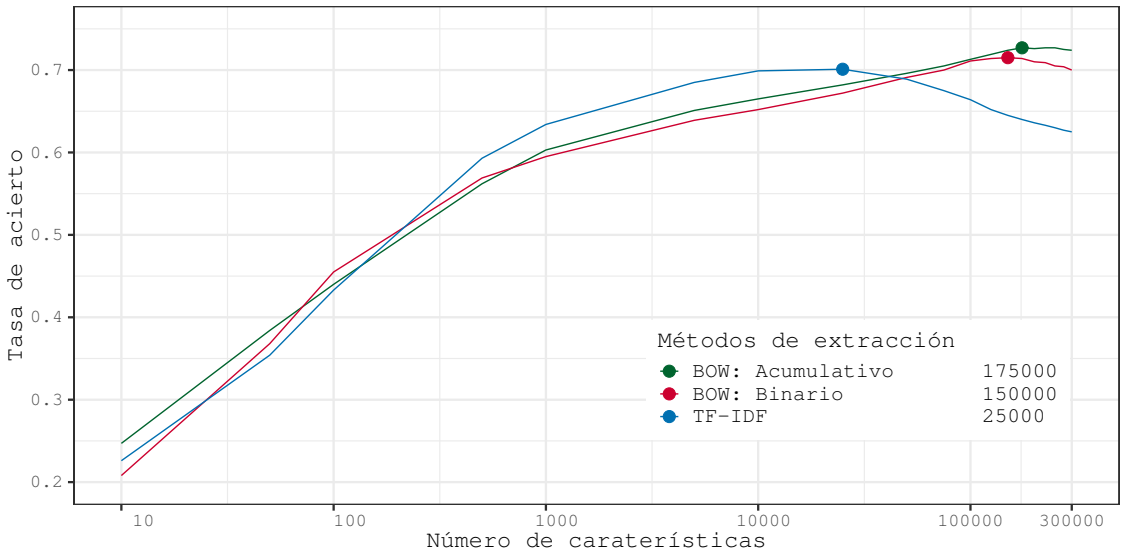

Figura 3.12: Multinomial NB: Tasa de acierto respecto al número de características para la partición de desarrollo del *corpus* de catalán

Nuevamente, al igual que en castellano, se observa un decremento de la tasa de acierto al utilizar el método de extracción de características *TF-IDF* a partir de una cierta dimensionalidad del vector de características. No obstante, al utilizar los métodos *BOW* esto no sucede hasta prácticamente el máximo de características marcado, en donde se observa un ligero descenso en el *accuracy*. Otra diferencia, respecto a lo observado en *Figura 3.10*, es que al utilizar los métodos *BOW*, se produce una mejor tasa de acierto que al utilizar el *TF-IDF*. Con el conjunto de muestra de desarrollo del *corpus* de catalán, se observa que el model *multinomial naive bayes* consigue sobrepasar el 0.7 de *accuracy*, sin llegar al 0.75.

Para **maximizar el *accuracy***, el modelo multinomial para catalán, tendrá los siguientes parámetros: el método de extracción ***BOW Acumulativo*** y **175 000** como dimensionalidad del vector de características. Esta configuración proporciona una tasa de acierto del 0.727±0.009 en la partición de desarrollo.

Después de analizar la evolución de la tasa de acierto, se procederá a analizar la de la

puntuación macro F_1 mediante la *Figura 3.13*.

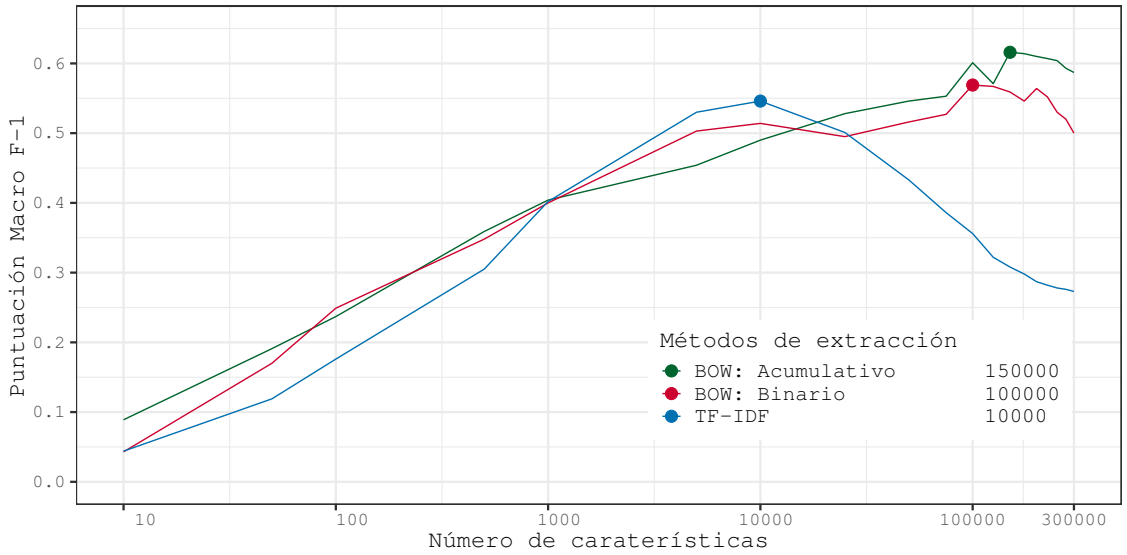

Figura 3.13: Multinomial NB: Puntuación macro F_1 respecto al número de características para la partición de desarrollo del *corpus* de catalán

A diferencia de los resultados obtenidos en castellano para macro F_1, en la *Figura 3.13* podemos observar que los tres métodos de extracción reducen la métrica obtenida por el modelo a partir de un cierto número de características. No obstante, la más notable es la curva producida por el método *TF-IDF*, donde se pasa de una puntuación cercana al 0.5 con 10 000 características y se termina con una puntuación menor a 0.25 al llegar a las 300 000. El método *BOW Binario* también produce ciertas inconsistencias en la evolución de la métrica, como se puede observar de las 10 000 a las 100 000. Por último, vemos que el acumulativo es el método que ayuda al modelo multinomial a aumentar su eficiencia para la puntuación macro F_1, quedandose muy cerca del 0.55 en la métrica.

Por todo lo citado anteriormente, los parámetros que se utilizarán para **maximiazar la puntuación macro F_1** para el *corpus* de catalán, son: ***BOW Acumulativo*** como método de extracción y una dimensionalidad para el vector de características de **150 000**. Esta configuración consigue una puntuación macro F_1 de 0.542±0.010.

3.6.1.2 Evaluación

3.6.1.2.1 *Corpus* de castellano

En la *Tabla 3.4* se muestran los parámetros elegidos en la sección anterior para maximizar la tasa de acierto y la puntuación macro F_1.

	Método de Extracción	**Dimensionalidad**
Accuracy	TF-IDF	100 000
Macro F_1	TF-IDF	10 000

Tabla 3.4: Multinomial NB: Parámetros de configuración elegidos para el *corpus* de castellano

La *Tabla 3.5* muestra los resultados obtenidos para las dos configuraciones con la partición de test, la que maximiza el *accuracy* y la que maximiza la puntuación macro F_1.

	Accuracy	**Macro *Precision***	**Macro *Recall***	**Macro F_1**
Accuracy	0.751±0.005	0.569±0.005	0.389±0.006	0.403±0.006
Macro F_1	0.727±0.005	0.459±0.006	0.486±0.006	0.455±0.006

Tabla 3.5: Multinomial NB: Resultados obtenidos para la partición de test del *corpus* de castellano

Como se puede observar, la configuración que maximiza la tasa de acierto, tiene mejor *accuracy* y *precision*, sin embargo, tiene un *recall* por debajo del 0.4. Lo cual indica que el modelo se está centrando en las categorías mayoritarias en detrimento de las minoritarias, esto hace aumentarla tasa de acierto y la precisión; dado que al identificar con más eficiencia las categorías mayoritarias, se produce un porcentaje más alto de acierto y en conjunto el clasificador es más preciso. No obstante, el *recall* decae debido a que no se identifican bien las clases minoritarias.

Por otro lado, la configuración que maximiza la macro F_1, reduce su tasa de acierto en un 0.12 y su precisión en un 0.13, sin embargo, aumenta en un 0.18 el *recall*. El aumento significativo del *recall* indica que las categorías minoritarias están contempladas por el modelo. Vemos que es una configuración que tiene unas métricas más balanceadas que la utilizada para maximizar la tasa de acierto.

3.6.1.2.2 *Corpus* de catalán

La *Tabla 3.6* muestra los parámetros de configuración elegidos para el modelo *multinomial naive bayes* con el *corpus* de catalán.

	Método de Extracción	**Dimensionalidad**
Accuracy	BOW Acumulativo	175 000
Macro F_1	BOW Acumulativo	150 000

Tabla 3.6: Multinomial NB: Parámetros de configuración elegidos para el *corpus* de catalán

Se pasará a analizar la *Tabla 3.7*, que contiene los resultados de las dos configuraciones para la partición de test.

	Accuracy	**Macro *Precision***	**Macro *Recall***	**Macro F_1**
Accuracy	0.729±0.010	0.592±0.010	0.640±0.010	0.608±0.010
Macro F_1	0.728±0.010	0.589±0.010	0.665±0.010	0.613±0.010

Tabla 3.7: Multinomial NB: Resultados obtenidos para la partición de test del *corpus* de catalán

Los resultados mostrados en la *Tabla 3.7* son completamente diferentes a los obtenidos por el modelo para castellano. Las dos configuraciones tienen el mismo método de extracción y solo difieren en la dimensionalidad, donde la configuración que maximiza la tasa de acierto tiene una dimensionalidad mayor a la seleccionada para maximizar la puntuación macro F_1. Por otro lado, la configuración que maximiza la puntuación macro F_1 tan solo obtiene un *accuracy* un 0.001 menor y una precisión 0.003 menor; no obstante, se gana un *recall* 0.025. Esto lleva a concluir que la segunda configuración es más eficiente que la primera en términos generales.

3.6.2 Regresión Logística

Tal como se comentó en la *Subsubsección 3.2.2.2*, para realizar el cálculo de los parámetros del modelo, se lleva a cabo la búsqueda de la solución a una serie de ecuaciones mediante aproximación numérica; esta aproximación se realiza de manera iterativa y terminará más o menos rápido, dependiendo de la interdependencia de las variables.

En la implementación de la librería *sklearn*, al crear el modelo, se indica el número de iteraciones máximo, el cual limita el número de ciclos que se realizan para buscar la solución aproximada a los parámetros del modelo. La implementación de la regresión lineal en *sklean* contiene varios métodos de aproximación numérica, entre ellas Stochastic Average Gradient Accelerated (SAGA) [29] que es la usada para este trabajo. Uno de los beneficios que conlleva el uso de este método de resolución, es que es independiente del número de términos en el sumatorio, por lo que su coste temporal no aumenta con el número de términos/características. Con ello se consigue que el modelo de regresión logística logre reducir su coste temporal de entrenamiento cuando el número de características se incrementa, dado que, al aumentar el número de características, se aumenta la probabilidad que haya más variables independientes. Esta es una gran diferencia respecto de otros modelos de aprendizaje como SVM o las redes neuronales, donde el proceso de aprendizaje está ligado al número de características y el coste temporal se incrementa cuando la dimensionalidad aumenta.

3.6.2.1 Ajuste de parámetros

3.6.2.1.1 *Corpus* de castellano

La *Figura 3.14* muestra la evolución del *accuracy* respecto al número de características, la cual pasaremos a analizar.

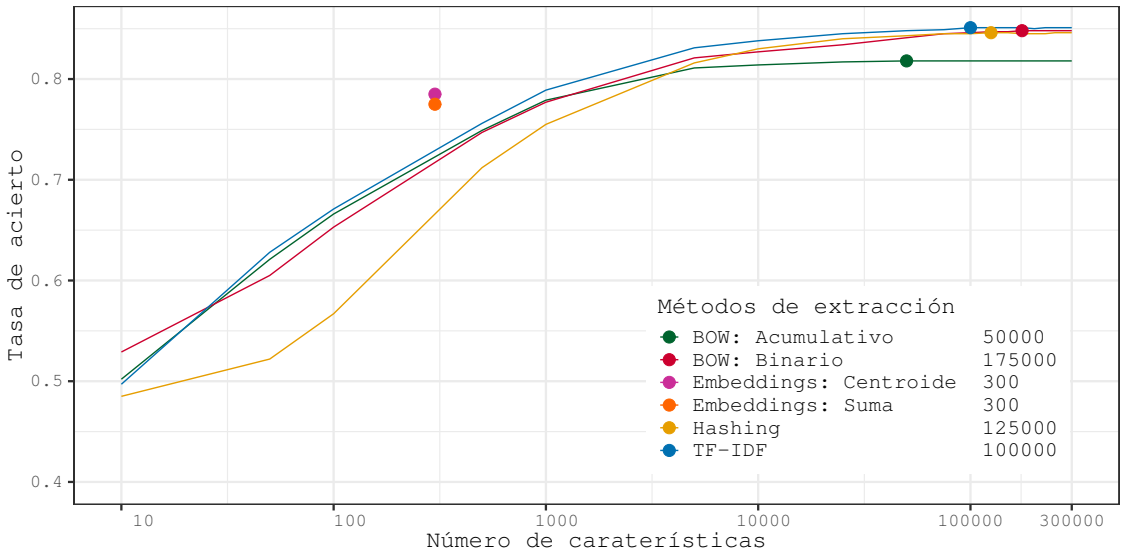

Figura 3.14: Regresión Logística: Tasa de acierto respecto al número de características para la partición de desarrollo del *corpus* de castellano

En la *Figura 3.14* se observa como tres de los seis métodos de extracción de características, permiten al modelo de aprendizaje actual llegar a una tasa de acierto máxima muy similar –alrededor de 0.85–, estos son: *BOW Binario*, *TF-IDF* y *Hashing*. Así mismo, al analizar la evolución del *accuracy* al usar método *BOW Acumulativo*, se ve como al comienzo presenta una pendiente muy similar a la de *TF-IDF*; no obstante, a partir de las 5000 el *accuracy* comienza a convergir, quedándose lejos del máximo obtenido con los otros tres métodos. Por último, destacar como los métodos de *embeddings* permiten al modelo conseguir una tasa de acierto mucho más alta que con el resto de métodos y con tan solo 300 características.

Los parámetros de configuración seleccionados para **maximizar el *accuracy*** en el *corpus* de castellano, son: ***TF-IDF*** como método de extracción de características y una dimensionalidad del vector de características de **100 000**. Con la configuración citada, para la partición de desarrollo, se obtiene una tasa de acierto de 0.851±0.004.

Después del análisis y ajuste de parámetros para la maximización del *accuracy*, se pasará a hacer lo propio para la puntuación macro F_1, mediante el análisis de la *Figura 3.15*.

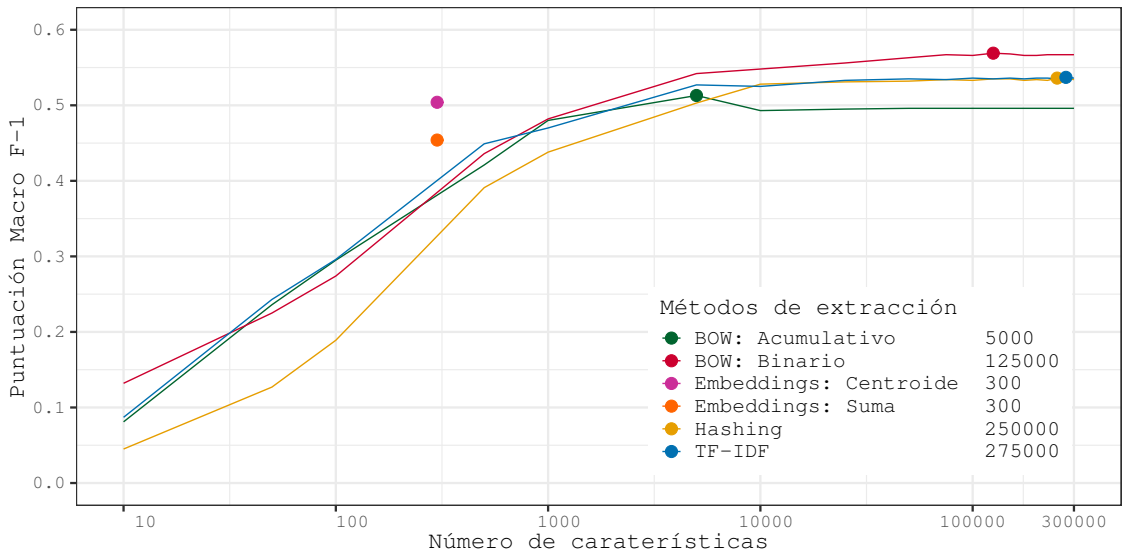

Figura 3.15: Regresión Logística: Puntuación macro F_1 respecto al número de características para la partición de desarrollo del *corpus* de castellano

Para la puntuación macro F_1, se observa en la *Figura 3.15*, que el máximo obtenido es un valor superior a 0.55 con el método de extracción *BOW Binario* y el modelo de regresión lineal. Ninguno de los otros métodos consigue acercarse a este valor. Destacar que con los métodos *TF-IDF* y *Hashing* se consigue una puntuación muy similar –cercana a 0.55–. La utilización del *BOW Acumulativo*, nuevamente, produce la peor puntuación de los cuatro métodos con un tamaño de características variable. Al igual que para el *accuracy*, la puntuación que se obtiene con el uso de los métodos con *embeddings*, para 300 características, es superior a los cuatro métodos para los que se varía el número de características, destacando el método *centroide* por encima del de *suma*. Esto es de esperar, ya que en los cuatro métodos princi-

pales las características representan palabras y por tanto, estamos limitando la información de cada noticia, mientras que con los *embeddings* cada palabra es representada por un vector de dimensión 300 y después se están mezclando todas las palabras del del texto en un vector de dicha dimensionalidad.

La configuración elegida con el fin de **maximizar la puntuación macro $\mathbf{F}_1$** para el castellano, es la que sigue: se usará el método de extracción ***BOW Binario*** y **125 000** características. La puntuación obtenida con esta configuración, para la partición de desarrollo, es de 0.569±0.005.

3.6.2.1.2 *Corpus* de catalán

Se comenzará analizando la *Figura 3.14*, que muestra la evolución de la tasa de acierto del modelo de regresión logística para el *corpus* de catalán.

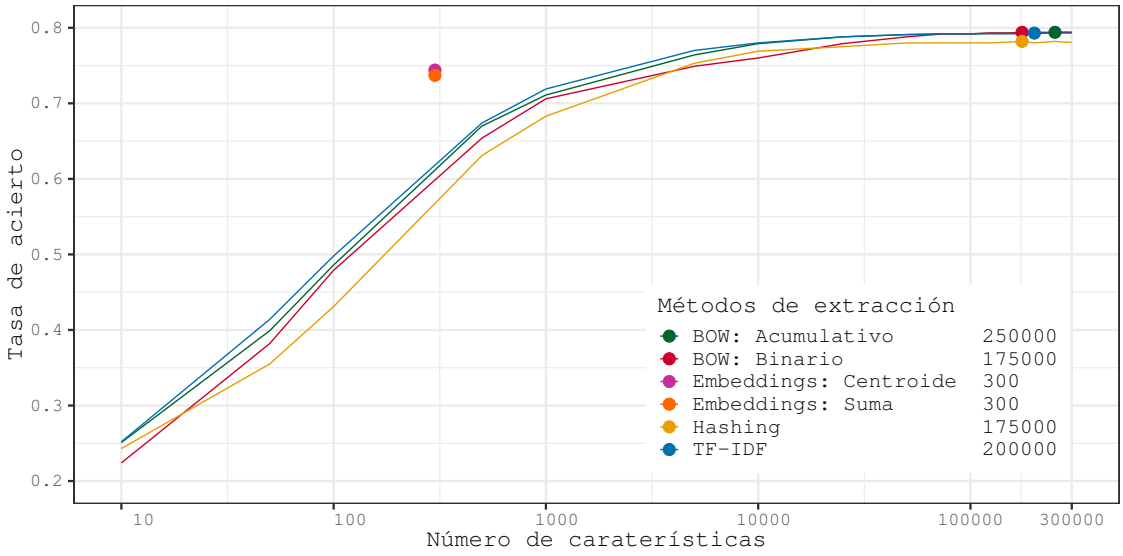

Figura 3.16: Regresión Logística: Tasa de acierto respecto al número de características para la partición de desarrollo del *corpus* de catalán

La *Figura 3.16* muestra una evolución de la tasa de acierto con una curva de crecimiento prácticamente idéntica cuando se usan los métodos de extracción *TF-IDF* y *BOW Acumulativo*. Por otro lado, al usar el método *BOW Binario*, se mantiene con una tasa de acierto menor hasta aproximarse a las 100 000 características, en donde las dos curvas citadas anteriormente y estas, convergen en un *accuracy* próximo 0.8. Además, el uso del método de *Hashing* muesta una tasa de acierto menor que con los otros tres métodos mencionados, a lo largo de prácticamente toda la gráfica.

Para **maximizar la tasa de acierto** se eligen los siguientes parámetros de configuración: el método de extracción ***BOW Binario*** y una dimensionalidad del vector de características de **175 000**. La tasa de acierto obtenida para el conjunto de desarrollo del *corpus* de catalán

es de 0.794±0.009.

Se pasará a analizar la puntuación macro F_1, mediante la *Figura 3.17*.

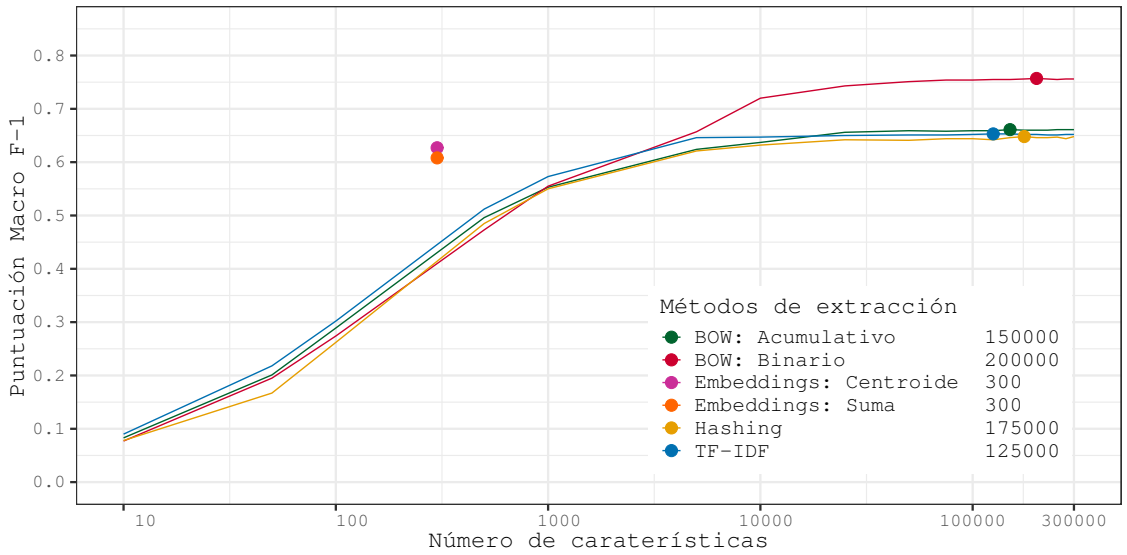

Figura 3.17: Regresión Logística: Puntuación macro F_1 respecto al número de características para la partición de desarrollo del *corpus* de catalán

Se observa, en la *Figura 3.17*, que la distribución de las curvas de la puntuación macro F_1 para los cuatro métodos en los que se puede variar el número de características, es similar a la obtenida para el castellano. Por un lado, el uso del *BOW Binario* maximiza la puntuación macro F_1 –aproximadamente 0.75– con una amplia diferencia respecto al uso del resto de métodos de extracción. Por otro lado, los métodos *TF-IDF*, *Hashing* y *BOW Acumulativo* convergen en una puntuación muy similar, aproximadamente 0.1 por debajo del mejor. Respecto al uso de los métodos de mezclado de *embeddings*, se observa que con solo una dimensionalidad de 300 se aproximan a la puntuación máxima obtenida por los tres métodos nombrados anteriormente.

Para el catalán, los parámetros de configuración que **maximizan la puntuación macro F_1** son: como método de extracción ***BOW* Binario** y **200 000** características. Para la partición de desarrollo, se obtiene una puntuación de 0.757±0.009.

3.6.2.2 Evaluación

Los parámetros de configuración elegidos para el modelo de regresión logística son los mostrados en la *Tabla 3.8*.

3.6.2.2.1 *Corpus* de castellano

	Método de Extracción	**Dimensionalidad**
Accuracy	TF-IDF	100 000
Macro F_1	BOW Binario	125 000

Tabla 3.8: Regresión Logística: Parámetros de configuración elegidos para el *corpus* de castellano

Se analizarán los resultados mostrados en la *Tabla 3.9* para la partición de test del *corpus* de castellano.

	Accuracy	Macro ***Precision***	Macro ***Recall***	Macro F_1
Accuracy	0.848±0.003	0.598±0.005	0.525±0.005	0.554±0.005
Macro F_1	0.844±0.004	0.606±0.005	0.542±0.005	0.564±0.005

Tabla 3.9: Regresión Logística: Resultados obtenidos para la partición de test del *corpus* de castellano

Se observa en la *Tabla 3.9*, que utilizando la configuración para maximizar la tasa de acierto, se obtiene un *accuracy* ligeramente mejor que con la otra configuración. Por otro lado, la configuración que maximiza la puntuación macro F_1, obtiene mejores valores en el resto de métricas; esto indica que dicha configuración rinde mejor que la primera en términos generales.

3.6.2.2.2 *Corpus* de catalán

Los parámetros de configuración elegidos para el modelo de regresión logística en el *corpus* de catalán son los que se presentan en la *Tabla 3.10*.

	Método de Extracción	**Dimensionalidad**
Accuracy	BOW Binario	175 000
Macro F_1	BOW Binario	200 000

Tabla 3.10: Regresión Logística: Parámetros de configuración elegidos para el *corpus* de catalán

Los resultados obtenidos con la partición de test y la configuración elegida, se muestran en la *Tabla 3.11*.

	Accuracy	Macro ***Precision***	Macro ***Recall***	Macro F_1
Accuracy	0.796±0.009	0.804±0.009	0.759±0.009	0.778±0.009
Macro F_1	0.796±0.009	0.804±0.009	0.758±0.009	0.778±0.009

Tabla 3.11: Regresión Logística: Resultados obtenidos para la partición de test del *corpus* de catalán

En la *Tabla 3.11* se puede observar que el aumento de dimensionalidad no ha producido ningún beneficio en ninguna de las métricas con el conjunto de test. Es más, la configuración que maximiza la macro F_1 con la partición de test pierde un 0.001 de *recall*, lo cual es insignificante; pero es una perdida, no una ganancia.

3.6.3 SVM Lineal

La implementación del modelo de *SVM Lineal* que se utilizará, tal y como se había comentado anteriormente, es la de *sklearn* que a su vez utiliza la librería *liblinear*, implementa una SVM con *kernel* lineal con un enfoque de clasificación "uno contra todos".

3.6.3.1 Ajuste de parámetros

3.6.3.1.1 *Corpus* de castellano

Se comenzará analizando la *Figura 3.18*, que muestra la evolución del *accuracy* para el modelo SVM con kernel lineal y distintos métodos de extracción de características.

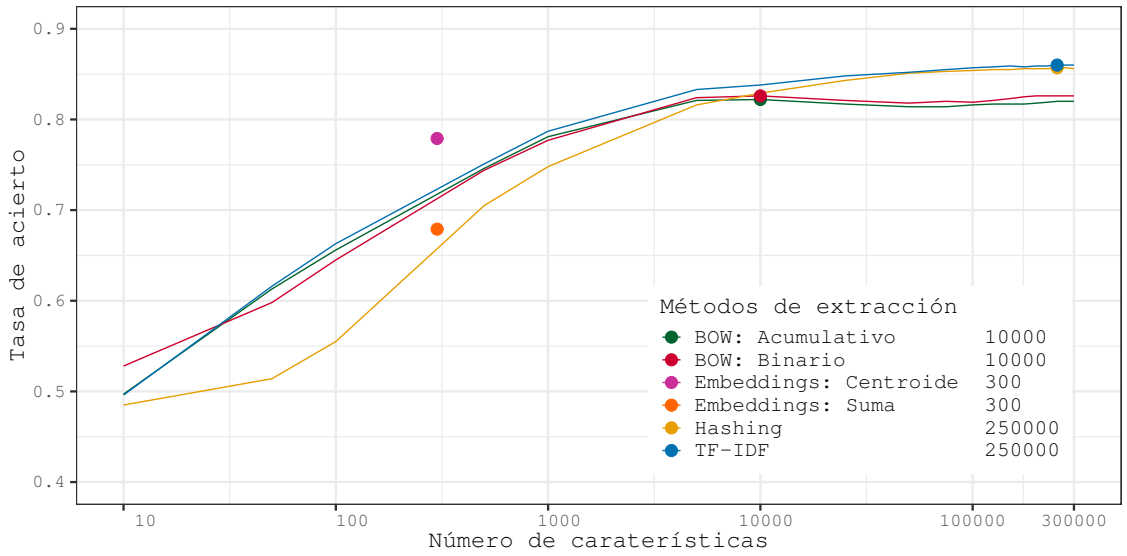

Figura 3.18: SVM Lineal: Tasa de acierto respecto al número de características para la partición de desarrollo del *corpus* de castellano

En la *Figura 3.18* se observa como dos de los cuatro métodos principales, al ser usados junto con el modelo de aprendizaje actual, convergen a una tasa de acierto prácticamente idéntica –ligeramente por encima de 0.85–, estos métodos son: *TF-IDF* y *Hashing*. La diferencia que se aprecia entre ambos, es el que con el primero se muestra una evolución de la tasa de acierto más constante hasta llegar a las 5000 características, mientras que el segundo evoluciona de manera más lenta hasta acercarse a las 100 características, punto en el cual incrementa significativamente la pendiente de la curva hasta llegar a las 35 000 características, donde el *accuracy* obtenido está muy próximo al de *TF-IDF*. Por otro lado, usando los métodos *BOW* comienzan con una pendiente idéntica a *TF-IDF*, pero pasadas las 1000 características, el *accuracy* conseguido converge incluso baja ligeramente. Por último, se aprecia que el método mezcla de *embeddings* suma produce una tasa de acierto inferior a la conseguida con el uso del método centroide.

Para **maximizar la tasa de acierto** para el castellano, se elegirán los siguientes parámetros de configuración: el método de extracción ***TF-IDF*** y una dimensionalidad de **250 000** para el vector de características. Con esta configuración se ha conseguido, con la partición de desarrollo, una tasa de acierto de 0.860±0.004.

Al igual que para la tasa de acierto, realizaremos el análisis de la *Figura 3.19*, que presenta la evolución de la puntuación macro F_1.

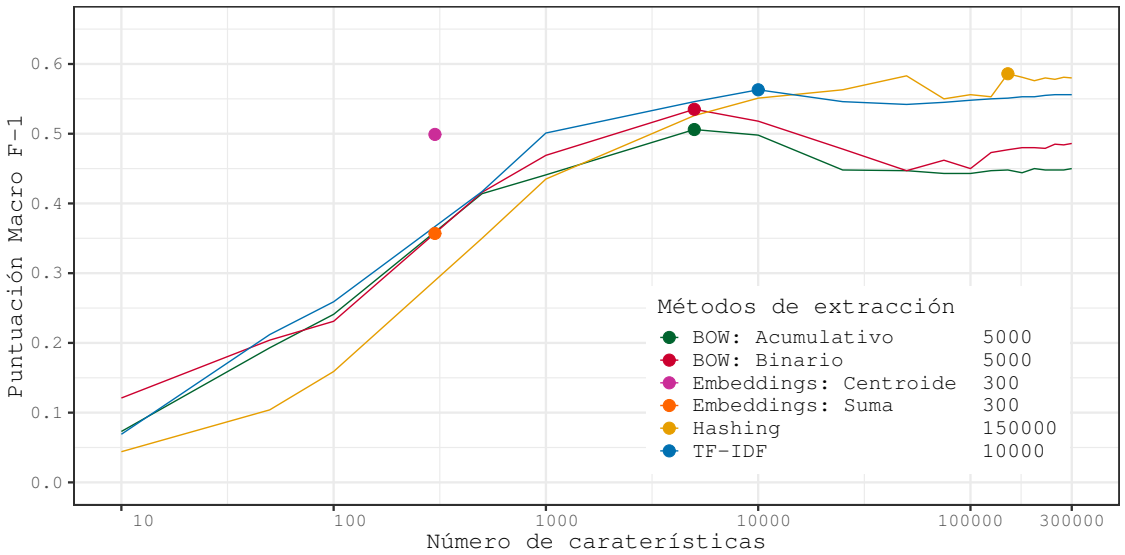

Figura 3.19: SVM Lineal: Puntuación macro F_1 respecto al número de características para la partición de desarrollo del *corpus* de castellano

La *Figura 3.19* muestra una distribución muy parecida conceptualmente a la analizada en el *accuracy*. Por un lado se sitúan en la parte superior las curvas relacionadas con los métodos *TF-IDF* y *Hashing*, más abajo se sitúan las de los métodos BOW y las puntuaciones de los *embeddings* se sitúan con la misma distribución. No obstante, se puede apreciar que las curvas no son suaves, si no que están llenas de picos. Esto puede relacionarse con la evolución de la tasa de acierto, ya que se puede observar como los cuatro métodos principales alcanzan su cota más alta con un número de características más alto de lo que lo hace para la puntuación macro F_1; por lo que hace sospechar que el modelo, llegado un punto, comienza a centrar sus esfuerzos en las categorías mayoritarias –se aprecia en las pendientes de bajada después de la cota máxima–.

Para **maximizar la puntuación macro F_1**, en este caso se elige la siguiente configuración: el método de extracción de características ***Hashing*** y **150 000** características. Al clasificar la partición de desarrollo da como resultado una puntuación macro F_1 de 0.586±0.005.

3.6.3.1.2 *Corpus* de catalán

Se comenzará analizando la tasa de acierto para catalán, mediante la *Figura 3.20*.

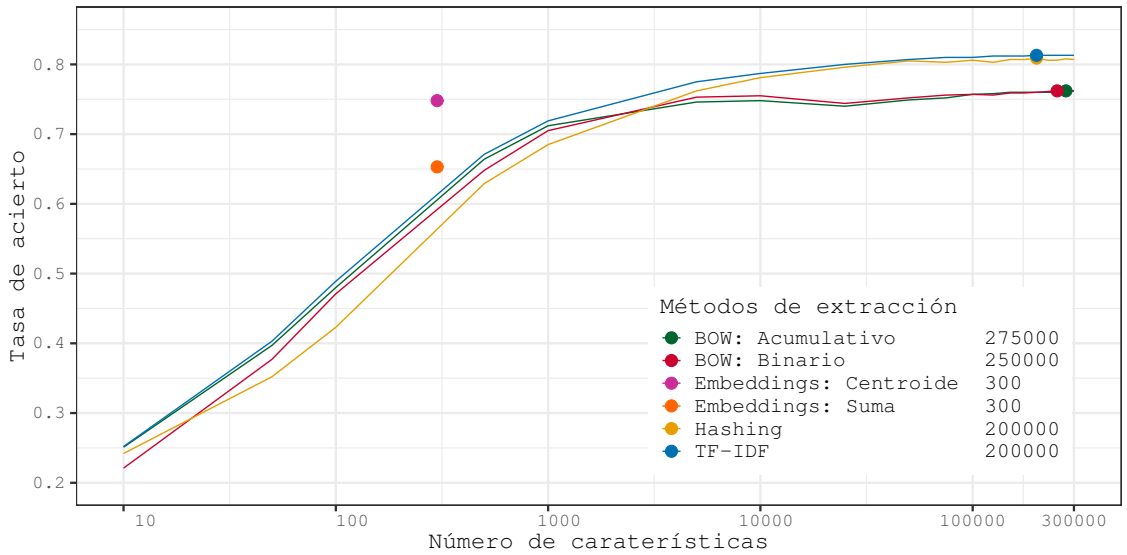

Figura 3.20: SVM Lineal: Tasa de acierto respecto al número de características para la partición de desarrollo del *corpus* de catalán

Se puede observar en la *Figura 3.20*, que la evolución de la tasa de acierto es muy similar a la de castellano para cualquiera de los cuatro métodos de extracción para los que se varía el número de características. Para el catalán, el *accuracy* máximo obtenido está alrededor de 0.8. Este máximo ha sido alcanzado al usar el método *TF-IDF* o el de *Hashing*. El uso de los métodos *BOW*, produce una tasa de acierto menor a los dos citados anteriormente, tal y como ya sucedía en castellano. Por último, destacar que el método de mezclado de los *embeddings* influye significativamente en la tasa de acierto. El método de *centroide* produce un *accuracy* un 0.1 superior al de *suma* y, además, con el método de *centroide* se obtiene una tasa de acierto similar a la de los BOW con una dimensionalidad dos ordenes de magnitud mayor.

Para **maximizar el *accuracy*** se eligen los siguientes parámetros de configuración: como método de extracción de características se utiliza ***TF-IDF*** con un vector de dimensionalidad **225 000**. Esta configuración, para la partición de desarrollo, resulta en una tasa de acierto de 0.813±0.008.

Finalizado el análisis de la evolución de la tasa de acierto, se pasará a hacer lo propio para la *Figura 3.21*, donde se muestra la evolución de la puntuación macro F_1 para el catalán.

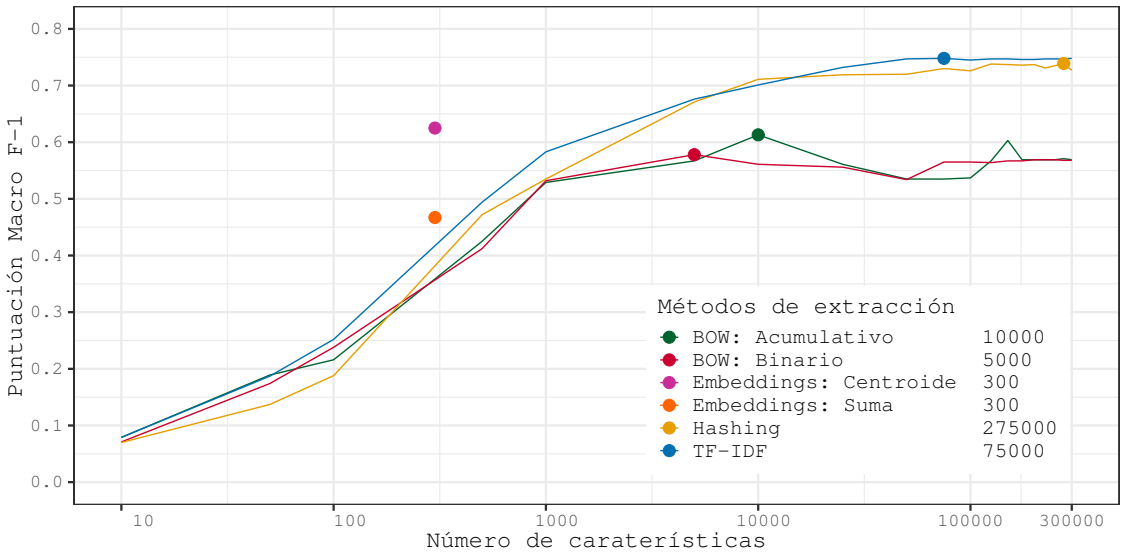

Figura 3.21: SVM Lineal: Puntuación macro F_1 respecto al número de características para la partición de desarrollo del *corpus* de catalán

En la *Figura 3.21* se percibe como el comportamiento es similar al de la tasa de acierto. Al utilizar los métodos *TF-IDF* o *Hashing* se llega a una puntuación cercana 0.75. También se puede destacar que, a diferencia de castellano, estos métodos consiguen que el modelo de aprendizaje aumente su rendimiento con el aumento de características, lo cual indica que gracias a la dimensionalidad y la información que aportan, el modelo consigue identificar mejor las características que permiten generar una mejor clasificación. Por ejemplo, se observa como los métodos *BOW* a partir de las 1000 características dejan de ayudar al modelo de la misma forma. Por último, en este *corpus*, el uso del método de mezclado *embeddings* por *centroide*, ayudan al modelo a obtener una puntuación mejor que la conseguida con los métodos *BOW*.

Los parámetros de configuración que **maximizan la puntuación macro F_1** son: extracción mediante ***TF-IDF*** y **75 000** características. Esta configuración obtiene una puntuación de 0.748±0.010 al clasificar la partición de desarrollo.

3.6.3.2 Evaluación

3.6.3.2.1 *Corpus* de castellano

Los parámetros de configuración seleccionados para maximizar cada una de las dos métricas deseadas, son los que se muestran en la *Tabla 3.12*.

	Método de Extracción	**Dimensionalidad**
Accuracy	TF-IDF	250 000
Macro F_1	*Hashing*	150 000

Tabla 3.12: SVM Lineal: Parámetros de configuración elegidos para el *corpus* de castellano

Se analizarán los resultados mostrados en la *Tabla 3.13* para la partición de test del *corpus* de castellano.

	Accuracy	**Macro *Precision***	**Macro *Recall***	**Macro F_1**
Accuracy	0.858±0.004	0.668±0.005	0.559±0.006	0.589±0.006
Macro F_1	0.854±0.004	0.627±0.005	0.537±0.006	0.565±0.006

Tabla 3.13: SVM Lineal: Resultados obtenidos para la partición de test del *corpus* de castellano

En la *Tabla 3.13* se observa que, utilizando la configuración para maximizar la tasa de acierto, obtiene mejor puntuación en la tasa de acierto y además también en la macro F_1. Esto nos indica que el uso del método de extracción de *Hashing* aporta menos estabilidad al modelo de predicción que el de TF-IDF.

3.6.3.2.2 *Corpus* de catalán

En la *Tabla 3.14* se muestran los parámetros de configuración seleccionados para catalán. Destacar, el hecho que se utiliza el mismo método de extracción con una diferencia de dimensionalidad muy alta.

	Método de Extracción	**Dimensionalidad**
Accuracy	TF-IDF	200 000
Macro F_1	TF-IDF	75 000

Tabla 3.14: SVM Lineal: Parámetros de configuración elegidos para el *corpus* de catalán

Los resultados obtenidos con la partición de test y la configuración elegida, se muestran en la *Tabla 3.15*.

	Accuracy	**Macro *Precision***	**Macro *Recall***	**Macro F_1**
Accuracy	0.814±0.009	0.753±0.010	0.686±0.010	0.710±0.010
Macro F_1	0.810±0.009	0.797±0.010	0.726±0.010	0.750±0.010

Tabla 3.15: SVM Lineal: Resultados obtenidos para la partición de test del *corpus* de catalán

En la *Tabla 3.15* se puede aprecia que el aumento de dimensionalidad no ha producido ningún beneficio en ninguna de las métricas con el conjunto de test. La configuración para maximizar la puntuación macro F_1 tiene un rendimiento general mejor que la encargada de maximizar la tasa de acierto.

3.6.4 Perceptrón Multicapa

De todos los modelos, las redes neuronales son las que más tiempo computacional requieren por iteración; si bien, el proceso de aprendizaje también es el más paralelizable y –gracias a las tarjetas gráficas– se puede reducir notablemente el tiempo de aprendizaje. No obstante, tiene un tiempo de aprendizaje mucho mayor que los modelos expuestos hasta ahora.

Existe una diferencia en las gráficas respecto al resto de modelos de aprendizaje, debido al coste computacional de entrenar la red neuronal, solo se ha entrenado hasta un máximo 100 000 características en vez de hasta 300 000 características.

Por último, tal y como se detalló anteriormente en la *Sección 3.4.2.2.3*, el modelo de perceptrón multicapa ha sido entrenado para guardar aquellos parámetros internos del modelo que maximizan la tasa de acierto respecto a la partición de desarrollo. Puesto que el coste temporal de entrenar las redes neuronales es muy alto, se ha decidido analizar la puntuación macro F_1 para el subconjunto de parámetros internos antes mencionado y no volver a realizar un entrenamiento específico de la red neuronal en el que hallase el conjunto de parámetros internos que maximizasen la puntuación macro F_1.

3.6.4.1 Ajuste de parámetros

3.6.4.1.1 *Corpus* de castellano

La *Figura 3.22* muestra la evolución de la tasa de acierto en función del número de características para el perceptrón multicapa para castellano.

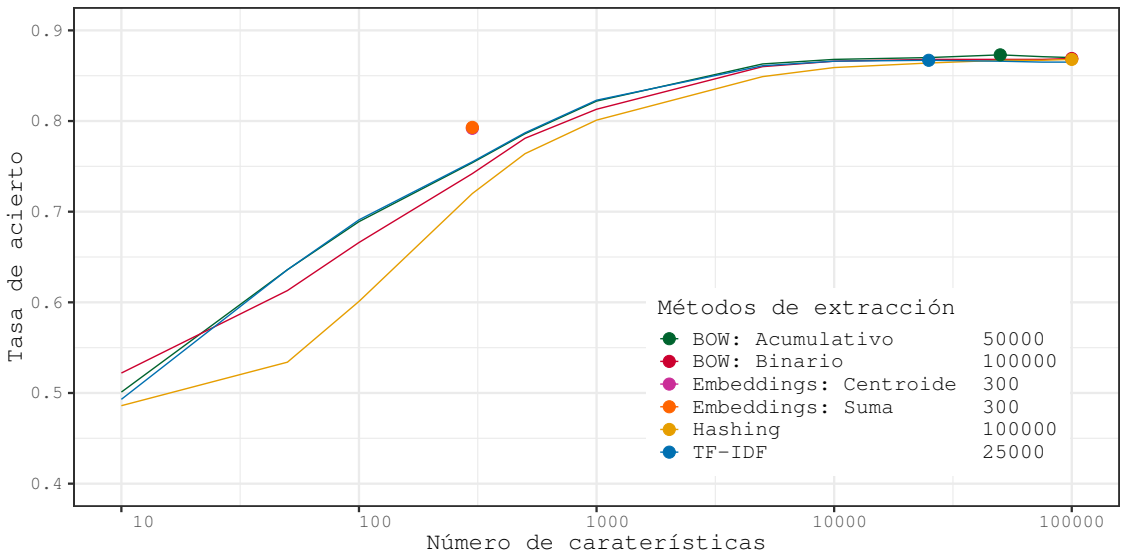

Figura 3.22: Perceptrón Multicapa: Tasa de acierto respecto al número de características para la partición de desarrollo del *corpus* de castellano

Las curvas mostradas en la *Figura 3.22*, a diferencia de otros modelos de aprendizaje vistos,

terminan convergiendo a una tasa de acierto muy similar, independientemente de cual de los cuatro métodos de extracción de características haya sido usado. Se observa también que el uso del método de *Hashing* sigue produciendo la misma forma de evolucionar que en otros modelos en la parte inicial de la gráfica. La tasa de acierto máxima obtenida por el modelo, está cerca de 0.875, que es ligeramente más alta que la obtenida para los modelos de *regresión logística* y *SVM lineal*. Por último, se puede apreciar como el *accuracy* con los métodos de *embeddings* es prácticamente idéntica –el punto naranja solapa al rojo–.

Para **maximizar la tasa de acierto** se eligen los siguientes parámetros de configuración: ***BOW Acumulativo*** como método de extracción de características y con una dimensionalidad del vector de características de **50 000**. Esta configuración, con la clasificación de la partición de desarrollo, resulta en 0.873±0.003.

El análisis de la evolución de la puntuación macro F_1 para el modelo actual, se realizará utilizando la *Figura 3.23*.

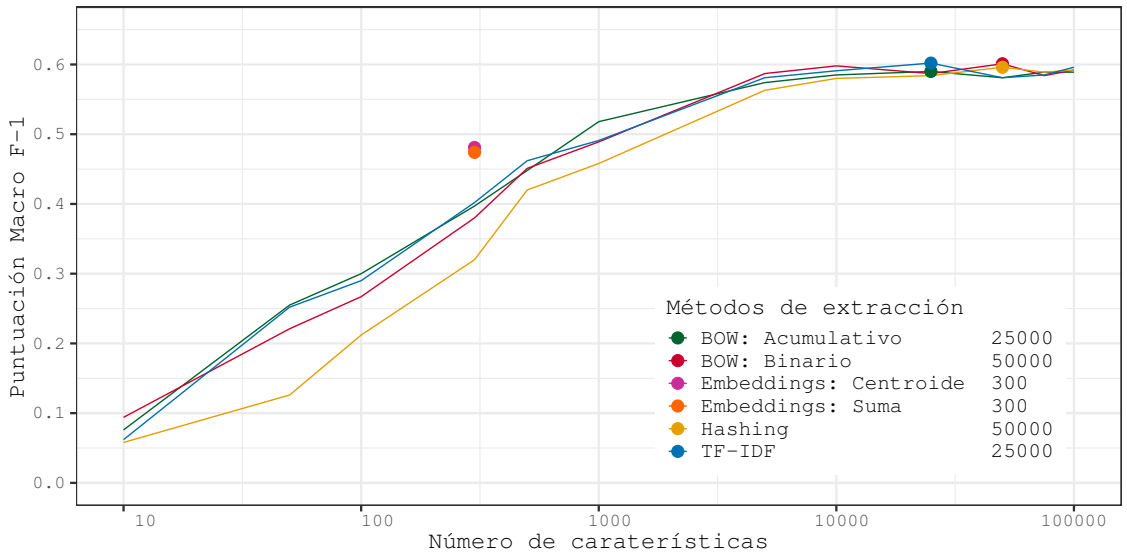

Figura 3.23: Perceptrón Multicapa: Puntuación macro F_1 respecto al número de características para la partición de desarrollo del *corpus* de castellano

Al igual que para la tasa de acierto, en la *Figura 3.23*, se puede observar como las curvas generadas al utilizar uno de los cuatro métodos de extracción para los cuales se varía el número de características, converge a puntuaciones muy cercanas entre ellas; si bien, no están tan próximas como en la tasa de acierto. Para el *corpus* de castellano, se ha llegado a una puntuación muy cercana al 0.6, algo que no se había conseguido con los otros modelos. Con los métodos de *embeddings* vemos que las puntuaciones ya tienen una ligera disparidad entre ellas.

La configuración elegida para **maximizar la macro F_1** es la siguiente: el método de extracción ***TF-IDF*** y **50 000** características. Esta configuración elegida, con la partición de desarrollo, da una puntuación macro F_1 de 0.601±0.005.

3.6.4.1.2 *Corpus* de catalán

A partir de la *Figura 3.24*, se analizará la evolución de la tasa de acierto del modelo perceptrón multicapa para el *corpus* de catalán.

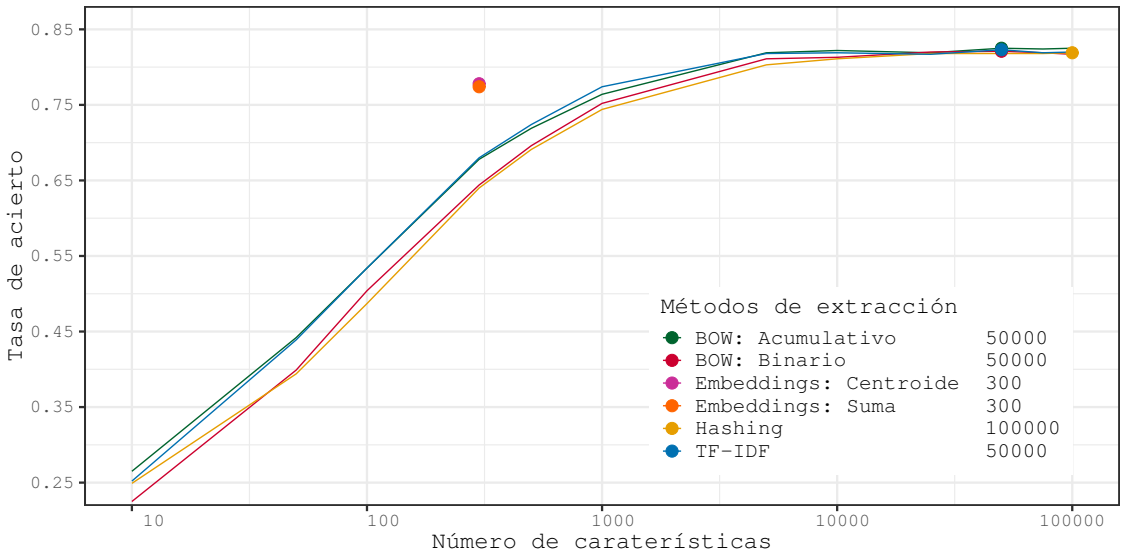

Figura 3.24: Perceptrón Multicapa: Tasa de acierto respecto al número de características para la partición de desarrollo del *corpus* de catalán

Las conclusiones para la *Figura 3.24*, son idénticas a las extraídas para el *corpus* de castellano en la *3.6.4.1.1*, debido a que el comportamiento del modelo es muy similar para castellano y catalán. Cabe mencionar, que la tasa de acierto máxima para este *corpus* está alrededor del 0.825.

Los parámetros de configuración elegidos para **maximizar el *accuracy***, son los siguientes: se escoge el modelo de extracción de características ***BOW Acumulatio*** y **50 000** características. Con la configuración citada, para la clasificación de la partición de desarrollo, se obtiene una tasa de acierto de 0.825±0.008.

Mediante la *Figura 3.25*, se realizará un análisis de la evolución de la puntuación macro F_1 obtenida por este modelo para el *corpus* de catalán.

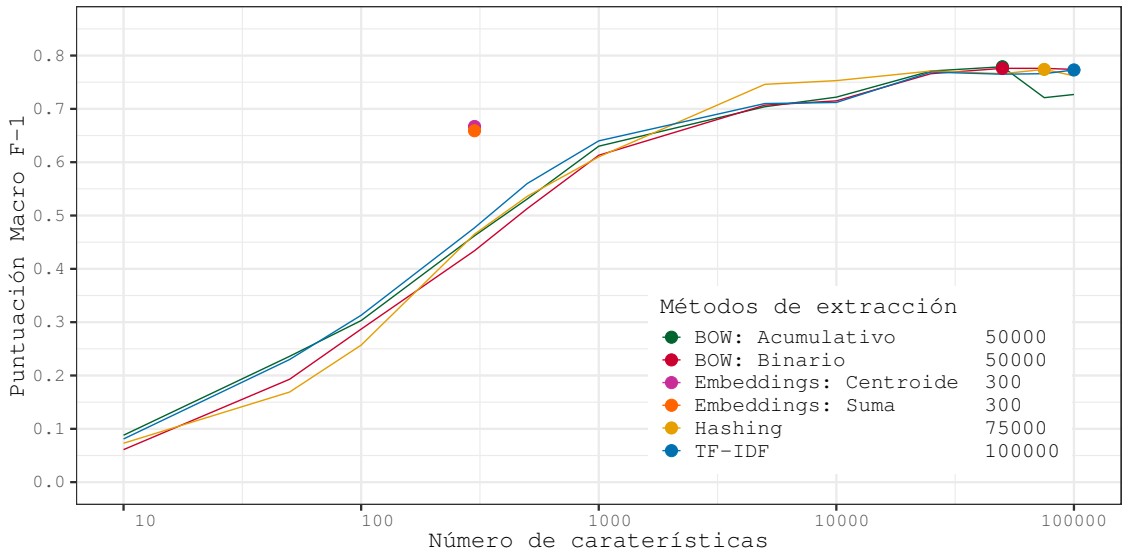

Figura 3.25: Perceptrón Multicapa: Puntuación macro F_1 respecto al número de características para la partición de desarrollo del *corpus* de catalán

En el caso del catalán, a diferencia del castellano, se observa en la *Figura 3.25* que se comportan similar a como lo hacen en la tasa de acierto, convergiendo los cuatro métodos a una puntuación macro F_1 similar, alrededor del 0.775. Nuevamente, esta puntuación es ligeramente superior a la obtenida por el resto de modelos anteriormente analizados. Los métodos basados en *embeddings* obtienen una puntuación muy similar, al igual que pasaba con la tasa de acierto.

Los parámetros de configuración que se utilizarán para **maximizar la puntuación macro F_1** son los que siguen: el método de extracción ***BOW Acumulativo*** y **50 000** características. Esta configuración obtiene, con la partición de desarrollo, una puntuación de 0.779±0.008.

3.6.4.2 Evaluación

3.6.4.2.1 *Corpus* de castellano

Los parámetros de configuración elegidos para el *corpus* de castellano, se muestran en la *Tabla 3.16*.

	Método de Extracción	**Dimensionalidad**
Accuracy	TF-IDF	50 000
Macro F_1	BOW Binario	50 000

Tabla 3.16: Perceptrón Multicapa: Parámetros de configuración elegidos para el *corpus* de castellano

A partir de la *Tabla 3.17*, se pasará a analizar los resultados obtenidos al clasificar la partición de test para castellano.

	Accuracy	**Macro *Precision***	**Macro *Recall***	**Macro F_1**
Accuracy	0.869±0.004	0.677±0.005	0.566±0.004	0.597±0.005
Macro F_1	0.866±0.004	0.682±0.005	0.581±0.006	0.618±0.005

Tabla 3.17: Perceptrón Multicapa: Resultados obtenidos para la partición de test del *corpus* de castellano

Los resultados presentados en la *Tabla 3.17*, la configuración para maximizar la tasa de acierto consigue una precisión de 0.003 superior respecto a la configuración para maximizar la puntuación macro F_1, mientras que la segunda obtiene una puntuación macro F_1 casi un 0.02 superior. Es por ello, que la segunda configuración es, en términos generales, mucho más eficiente que la elegida para maximizar el *accuracy*.

3.6.4.2.2 *Corpus* de catalán

Para el perceptrón multicapa, tal y como se observa en la *Tabla 3.18*, la configuración que maximiza el *accuracy* y coincide con la que maximiza la puntuación macro F_1.

	Método de Extracción	**Dimensionalidad**
Accuracy	TF-IDF	50 000
Macro F_1	BOW Acumulativo	50 000

Tabla 3.18: Perceptrón Multicapa: Parámetros de configuración elegidos para el *corpus* de catalán

En la *Tabla 3.19* se presentan los resultados para la partición de test con las configuraciones citadas en la *Tabla 3.18*.

	Accuracy	**Macro *Precision***	**Macro *Recall***	**Macro F_1**
Accuracy	0.820±0.008	0.808±0.009	0.762±0.009	0.781±0.009
Macro F_1	0.821±0.008	0.813±0.009	0.759±0.009	0.780±0.009

Tabla 3.19: Perceptrón Multicapa: Resultados obtenidos para la partición de test del *corpus* de catalán

Se observa en la *Tabla 3.19*, que las puntuaciones de las métricas están por encima o cercanas al 0.8, lo cual nos indica que, para catalán, el modelo tiene un buen rendimiento.

3.6.5 Resumen de las configuraciones seleccionadas y resultados

A continuación, se presenta un resumen de las ocho configuraciones seleccionadas –dos por modelo– para cada uno de los *corpus*, así como las tablas de resultados para la partición de desarrollo y test. Se ha decidido presentar los resultados para ambas particiones para poder comparar y analizar los resultados, y ver si presentan valores similares.

Para las tablas de resultados, debido a que estos tienen diferencias significativas, se ha decidido eliminar el intervalo de confianza, en aras de facilitar la lectura de la tabla. Además, se puede observar que en las métricas de *accuracy* y *macro* F_1 se muestra un superíndice, el cual indica que orden –de mayor a menor– ocupa la configuración en la métrica en cuestión.

3.6.5.1 *Corpus* de castellano

En la *Tabla 3.20* se muestran todas las configuraciones elegidas para el *corpus* de castellano.

Modelo	Maximizar	Método de Extracción	Dimensionalidad
Multinomial NB	*Accuracy*	TF-IDF	100 000
	M-F_1	TF-IDF	10 000
Regresión Logística	*Accuracy*	TF-IDF	100 000
	M-F_1	BOW Binario	125 000
SVM Lineal	*Accuracy*	TF-IDF	250 000
	M-F_1	*Hashing*	150 000
Perceptrón Multicapa	*Accuracy*	BOW Acumulativo	50 000
	M-F_1	BOW Binario	50 000

Tabla 3.20: Configuraciones seleccionadas para cada modelo para el *corpus* de castellano

En la *Tabla 3.20* se puede apreciar que casi todas las configuraciones coinciden en el orden de magnitud del número de características, salvo el perceptrón multicapa y el caso de maximización de macro F_1 para el modelo *Multinomial NB*.

Como ya se pudo observar en las gráficas cuando se realizaba el ajuste de parámetros para el modelo *Multinomial NB*, este modelo pierde eficiencia llegado a un número de características. Cuando se usaba el método de extracción TF-IDF este fenómeno se acrecentaba más que con los métodos BOW. Esto lleva a la conclusión que el modelo multinomial funciona mejor cuando se le aportan las características clave y que un exceso de características puede llevar al modelo a empeorar su rendimiento.

La *Tabla 3.21* presenta los resultados obtenidos para la partición de desarrollo.

Modelo Aprendizaje	**Max**	***Accuracy***	***M-Precision***	***M-Recall***	**M-F$_1$**
Multinomial NB	*Accu*	0.750^{7}	0.614	0.414	0.428^{8}
	M-F$_1$	0.727^{8}	0.467	0.486	0.463^{7}
Regresión Logística	*Accu*	0.851^{5}	0.577	0.508	0.536^{6}
	M-F$_1$	0.847^{6}	0.607	0.546	0.569^{4}
SVM Lineal	*Accu*	0.860^{3}	0.602	0.528	0.556^{5}
	M-F$_1$	0.855^{4}	0.632	0.557	0.586^{2}
Perceptrón Multicapa	*Accu*	0.873^{1}	0.641	0.554	0.581^{3}
	M-F$_1$	0.868^{2}	0.647	0.569	0.601^{1}

Tabla 3.21: Resultados obtenidos para las configuraciones seleccionadas utilizando la partición de desarrollo del *corpus* de castellano

En la *Tabla 3.22* se muestran los resultados obtenidos para la partición de test.

Modelo Aprendizaje	**Max**	***Accuracy***	***M-Precision***	***M-Recall***	**M-F$_1$**
Multinomial NB	*Accu*	0.751^{7}	0.569	0.389	0.403^{8}
	M-F$_1$	0.727^{8}	0.459	0.486	0.455^{7}
Regresión Logística	*Accu*	0.848^{5}	0.598	0.525	0.554^{6}
	M-F$_1$	0.844^{6}	0.606	0.542	0.564^{5}
SVM Lineal	*Accu*	0.858^{3}	0.668	0.559	0.589^{3}
	M-F$_1$	0.854^{4}	0.627	0.537	0.565^{4}
Perceptrón Multicapa	*Accu*	0.869^{1}	0.677	0.566	0.597^{2}
	M-F$_1$	0.866^{2}	0.682	0.581	0.618^{1}

Tabla 3.22: Resultados obtenidos para las configuraciones seleccionadas utilizando la partición de test del *corpus* de castellano

Por lo general, al comparar *Tabla 3.21* y *Tabla 3.22*, se observa que para las dos métricas –*accuracy* y macro F$_1$–, el orden de eficiencia se mantiene tanto para desarrollo, como para test. El perceptrón multicapa, seguido del modelo SVM lineal, después el modelo de regresión logística y por último el modelo multinomial *naive bayes*. La eficiencia de los tres primeros modelos –independientemente de la configuración elegida– dista mucho de lo que puede conseguir el modelo multinomial.

3.6.5.2 *Corpus* de catalán

La *Tabla 3.23* se muestran las configuraciones selecionadas para el *corpus* de catalán.

Modelo	Maximizar	Método de Extracción	Dimensionalidad
Multinomial NB	*Accuracy*	BOW Acumulativo	175 000
	M-F_1	BOW Acumulativo	150 000
Regresión Logística	*Accuracy*	BOW Binario	175 000
	M-F_1	BOW Binario	200 000
SVM Lineal	*Accuracy*	TF-IDF	200 000
	M-F_1	TF-IDF	75 000
Perceptrón Multicapa	*Accuracy*	TF-IDF	50 000
	M-F_1	BOW Acumulativo	50 000

Tabla 3.23: Configuraciones seleccionadas para cada modelo para el *corpus* de catalán

En la *Tabla 3.23*, se observa que el número de características medio de las configuraciones es más alto de lo que se había visto en el *corpus* de castellano. Esto indica que la cantidad de información que se puede recoger de cada categoría para distinguir correctamente las noticias que contiene es mayor. Esto se puede percibir en el modelo multinomial, donde la cantidad de características que puede utilizar para aumentar la puntuación macro F_1 es significativamente mayor que para el anterior *corpus*.

En la *Tabla 3.24* se presentan los resultados obtenidos para la partición de desarrollo.

Modelo Aprendizaje	Max	***Accuracy***	***M-Precision***	***M-Recall***	**M-F_1**
Multinomial NB	*Accu*	0.727^{7}	0.599	0.650	0.614^{8}
	M-F_1	0.724^{8}	0.591	0.668	0.616^{7}
Regresión Logística	*Accu*	0.794^{5}	0.775	0.743	0.756^{4}
	M-F_1	0.794^{6}	0.776	0.745	0.757^{3}
SVM Lineal	*Accu*	0.813^{3}	0.798	0.720	0.746^{6}
	M-F_1	0.810^{4}	0.800	0.722	0.748^{5}
Perceptrón Multicapa	*Accu*	0.823^{1}	0.794	0.752	0.765^{2}
	M-F_1	0.825^{2}	0.833	0.751	0.779^{1}

Tabla 3.24: Resultados obtenidos para las configuraciones seleccionadas utilizando la partición de desarrollo del *corpus* de castellano

La *Tabla 3.24* muestra los resultados obtenidos para la partición de test.

Modelo Aprendizaje	**Max**	***Accuracy***	***M-Precision***	***M-Recall***	**M-F$_1$**
Multinomial NB	*Accu*	0.729^{7}	0.592	0.640	0.608^{8}
	M-F$_1$	0.728^{8}	0.589	0.665	0.613^{7}
Regresión Logística	*Accu*	0.796^{5}	0.804	0.759	0.778^{4}
	M-F$_1$	0.796^{6}	0.804	0.758	0.778^{3}
SVM Lineal	*Accu*	0.814^{3}	0.753	0.686	0.710^{6}
	M-F$_1$	0.810^{4}	0.797	0.726	0.750^{5}
Perceptrón Multicapa	*Accu*	0.820^{2}	0.808	0.762	0.781^{1}
	M-F$_1$	0.821^{1}	0.813	0.759	0.780^{2}

Tabla 3.25: Resultados obtenidos para las configuraciones seleccionadas utilizando la partición de test del *corpus* de catalán

En general se aprecia que las puntuaciones obtenidas por las configuraciones para el *corpus* de catalán son significativamente mayores que las de castellano. Esto sucede con todas las métricas menos el *accuracy* que es la más débil de todas las métricas. El hecho de que las configuraciones utilizadas obtengan puntuaciones de la macro F$_1$ alrededor de 0.78 a diferencia del máximo de castellano (0.618) es un indicativo claro de que las noticias de catalán son más fáciles de clasificar que las de castellano.

Nuevamente, el orden obtenido por las configuraciones en *accuracy* y macro F$_1$, se mantiene entre desarrollo y test, salvo el caso curioso del perceptrón multicapa, donde la configuración que maximiza la tasa de acierto obtiene una macro F$_1$ ligeramente superior a la elegida para maximizar dicha métrica y viceversa, la que maximiza la macro F$_1$ obtiene mejor tasa de acierto que la primera. Otra diferencia importante, es que las configuraciones de regresión logística obtienen mejor puntuación macro F$_1$ que las del modelo de SVM lineal. Para este *corpus*, la eficiencia conseguida por la regresión logística para la macro F$_1$, es prácticamente idéntica a la del perceptrón multicapa; algo realmente significativo si comparamos el coste computacional de entrenar uno y otro modelo.

3.6.6 Análisis detallado de las configuraciones

Se han decidido analizar solo las mejores configuraciones de cada *corpus* . Es decir, aquellas que para la partición de test han obtenido mejor puntuación en la métrica de *accuracy* o en la de macro F_1. Para el análisis de estas configuraciones, se mostrará una tabla con las métricas *precision* y *recall* para cada categoría.

3.6.6.1 *Corpus* de castellano

Categoría	*Precision*	*Recall*
Ciencia	0.875	0.815
Cultura	0.883	0.884
Deportes	0.956	0.956
Economía	0.769	0.831
Educación	1.000	0.125
Gastronomía	0.837	0.632
Internacional	0.823	0.849
Nacional	0.929	0.955
Negocio	0.661	0.562
Opinión	0.887	0.782
Política	0.760	0.636
Salud	0.619	0.320
Sociedad	0.673	0.630
Sucesos	0.769	0.595
Tecnología	0.740	0.615

Tabla 3.26: *Precision* y *Recall* por categoría para el mejor en *Accuracy*. *Corpus* castellano

Categoría	*Precision*	*Recall*
Ciencia	0.867	0.813
Cultura	0.886	0.881
Deportes	0.944	0.958
Economía	0.757	0.828
Educación	1.000	0.375
Gastronomía	0.864	0.667
Internacional	0.813	0.855
Nacional	0.925	0.955
Negocio	0.658	0.525
Opinión	0.923	0.782
Política	0.749	0.634
Salud	0.653	0.385
Sociedad	0.696	0.607
Sucesos	0.742	0.638
Tecnología	0.794	0.558

Tabla 3.27: *Precision* y *Recall* por categoría para el mejor en M-F_1. *Corpus* castellano

A la izquierda se presentan los resultados para la configuración de perceptrón *Accuracy* (*Tabla 3.26*) y a la derecha los de la configuración perceptrón *M-F_1* (*Tabla 3.26*). En este caso, la configuración pensada para maximizar la tasa de acierto ha sido la que se esperaba y lo mismo para la macro F_1.

En general, al tratarse del mismo modelo de aprendizaje, no se ven demasiadas diferencias entre las tablas. No obstante, si que hay sutiles diferencias que se deben remarcar. Por lo general en la *Tabla 3.26* aparecen mejores métricas que superan sutilmente a las de la *Tabla 3.27*, como por ejemplo en precisión: `ciencia`, `deportes`, `internacional`, `negocio`,... Y en *recall*, también aparecen algunas categorías en las que el modelo que maximiza el *accuracy* supera al que maximiza la macro F_1en la tasa de acierto, como por ejemplo: `ciencia`, `cultura`, `economía`,... No obstante, no aparecen clases en las que el que maximiza la tasa de acierto supere notablemente al que maximiza macro F_1, pero si al contrario. El más claro es la métrica de *recall* en `educación` o la *precision* de `opinión`.

El hecho de que la configuración que maximiza la macro F_1 tenga unas métricas más estables y altas, reafirma lo que busca la métrica macro F_1; una configuración que obtenga las mejores puntuaciones posibles para el mayor número de categorías. El hecho que *educacion* tenga una puntuación notablemente más baja, al igual que *opinion*, es algo que la métrica macro F_1 penaliza, tal y como se puede ver en la *Tabla 3.22*.

3.6.6.2 *Corpus* de catalán

Categoría	*Precision*	*Recall*
Ciencia	0.600	0.621
Cultura	0.865	0.915
Deportes	0.939	0.956
Economía	0.872	0.885
Educación	0.771	0.660
Gastronomía	0.844	0.818
Internacional	0.866	0.896
Nacional	0.805	0.629
Negocio	0.909	0.769
Opinión	0.855	0.765
Política	0.752	0.821
Salud	0.726	0.697
Sociedad	0.733	0.748
Sucesos	0.800	0.400
Tecnología	0.856	0.803

Tabla 3.28: *Precision* y *Recall* por categoría para el mejor en *Accuracy*. *Corpus* catalán

Categoría	*Precision*	*Recall*
Ciencia	0.745	0.603
Cultura	0.872	0.904
Deportes	0.948	0.963
Economía	0.868	0.874
Educación	0.699	0.742
Gastronomía	0.900	0.818
Internacional	0.871	0.910
Nacional	0.772	0.664
Negocio	0.909	0.769
Opinión	0.881	0.728
Política	0.770	0.795
Salud	0.743	0.724
Sociedad	0.723	0.766
Sucesos	0.724	0.420
Tecnología	0.849	0.758

Tabla 3.29: *Precision* y *Recall* por categoría para el mejor en M-F_1. *Corpus* catalán

A la izquierda se presentan los resultados para la configuración de perceptrón *M-F_1* (*Tabla 3.28*) y a la derecha los de la configuración *Accuracy* (*Tabla 3.29*).

Nuevamente se aprecia que existen métricas en la *Tabla 3.29* que tienen un valor notablemente más alto que el homónimo a la *Tabla 3.28*. Por ejemplo, *ciencia* (*precision* o educación (*recall*). No obstante, en este caso, también existen diferencias al contrario, como es la precisión obtenida en *educación* o *sucesos*.

En general se observan dos tablas que contienen muy pocos desvalanceos –salvo *sucesos* por ejemplo–. Lo cual indica que, a priori, deberían tener una buena macro F_1 ambos. Al comprobarlo en la *Tabla 3.25*, se percibe que las dos configuraciones obtienen puntuaciones muy similares. Por lo que no hay diferencias destacables como para hablar de una configuración más estable que la otra.

Una de las posibles explicaciones por las que los resultados han sido muy similares es que, tal y como se ha comentado en la *Subsección 3.6.4*, se ha sacado la macro F_1 de las mejores *epochs* que maximizaban la tasa de acierto. Probablemente, si se hubiesen buscado las mejores *epochs* que maximizaban la macro F_1, entre todas las *epochs*, los resultados hubiesen tenido más diferencias entre ellos.

4 Conclusiones Finales y Trabajo Futuro

4.1 Conclusiones

En el presente trabajo se ha elaborado un sistema capaz de recolectar noticias de distintas fuentes periodísticas, con el fin de generar dos *corpus*; uno proveniente de periódicos con noticias en castellano y otro con periódicos que publican en catalán. Estos *corpus* han servido para realizar el posterior estudio de clasificación descrito en esta memoria, así como un estudio sobre resúmenes automáticos desarrollado por Fernando Alcina [1], junto al cual se realizó el desarrollo del sistema de captura de noticias. Para la captura de las distintas fuentes, se trabajó con un código único para todas ellas, donde solo se debían crear las configuraciones necesarias para capturar las noticias de cada una de las fuentes. Así mismo, para agilizar el proceso de captura, se replicaron los distintos módulos con los que cuenta el sistema en diversos ordenadores y se ejecutaron de manera concurrente, de modo que se paralelizó de manera efectiva el proceso de captura. Estos procesos se ejecutaron bajo distintos sistemas operativos, así como en distintas arquitecturas de procesador. Además, se realizó un análisis de la distribución por periódicos para cada *corpus*, donde se pudo observar aquellas fuentes periodísticas que predominaban en cada uno de los *corpus* y las que tenían menos presencia. Por último, dado que se utilizaron librerías y herramientas de software libre, se ha podido liberar el código que permite replicar los *corpus* generados para el presente trabajo, ya que la liberación de los *corpus* como tal podría acarrear problemas legales relacionados con el uso y autoría de los textos.

Para el estudio de categorización de noticias, se han elegido cuatro modelos de aprendizaje automático: *Multinomial NB*, *Regresión Logística*, *SVM Lineal* y *Perceptrón Multicapa*. Junto a estos paradigmas, se han utilizado distintos métodos de extracción de características de documentos de texto. Mediante la combinación de los modelos y los métodos de extracción, se ha variado el número de características, se ha hecho un estudio para las puntuaciones de *accuracy* y macro F_1. En él, se ha podido visualizar y analizar la relación que tiene el número de características en la mejora de la métrica. Se ha observado que, dependiendo del modelo de aprendizaje y el método de extracción, un número de características mayor puede no solo no mejorar los resultados, si no empeorarlos significativamente.

Para cada modelo de aprendizaje, se han buscado dos configuraciones –método de extracción y número de características–, una que maximizase la tasa de acierto y otra que maximizase la puntuación macro F_1; además dichas configuraciones debían minimizar el número de características, con el fin de reducir –en la medida de lo posible– el coste temporal y/o espacial durante el proceso de entrenamiento y clasificación. Al observar las configuraciones elegidas para cada *corpus*, se ha visto que en términos generales, el número de características es similar dentro de un mismo *corpus* y no tiene relación con el tamaño de este. En el caso de

los *corpus* de este estudio, el número de características para las configuraciones de castellano, en promedio, ha sido menor al de las configuraciones para catalán.

También se han observado diferencias entre las configuraciones que maximizaban la tasa de acierto y las que hacían lo propio para macro F_1. Las primeras tendían a reducir su eficiencia de clasificación para clases minoritarias de forma que aumentaban considerablemente el número de noticias que conseguían clasificar correctamente debido a que se centraban más en las clases mayoritarias. Por otro lado, las configuraciones que se elegían por maximizar la puntuación macro F_1, mostraban una eficiencia más balanceada entre todas las categorías, penalizando aquellas configuraciones que obtenían un rendimiento bajo en alguna/s de las clases. Es por ello que ninguna de las configuraciones elegidas ha coincidido en que maximizase el *accuracy* y a su vez lo hiciese también para macro F_1, ya que cuando la tasa de acierto se maximizaba, la macro F_1 bajaba en mayor o menor medida y viceversa.

En general, se ha percibido que, para los *corpus* usados para este trabajo, los modelos de aprendizaje *Regresión Logística*, *SVM Lineal* y *Perceptrón Multicapa* han obtenido unas puntuaciones muy próximas, tanto en *accuracy*, como en macro F_1. No obstante, el modelo basado en redes neuronales ha tenido un rendimiento ligeramente mejor que los otros en los dos *corpus*. Así mismo, el *SVM Lineal* ha obtenido mejores puntuaciones que el modelo de *Regresión Logística* en el *corpus* de castellano; no obstante, en el *corpus* de catalán, el modelo de *Regresión Logística* ha tenido un rendimiento ligeramente mejor que el de *SVM Lineal*. Esto lleva a dos conclusiones. Por un lado, el modelo basado en redes neuronales es más estable en sus puntuaciones que los otros dos modelos, independientemente del contexto. Por otro lado, la estabilidad del modelo de redes neuronales, conlleva un coste computacional mayor que los otros dos modelos de aprendizaje, que en algunos casos esta diferencia no es lo suficientemente amplia como para justificar el uso de este tipo de modelos en detrimento de otro –como son en este caso *Regresión Logística* y *SVM Lineal*–. Por ello, se llega a la conclusión que no se debe descartar un modelo, a priori "peor", sin tener en cuenta el contexto, ya que se podría acarrear un coste computacional y/o temporal innecesario para la tarea que se quiera cumplir.

Por otro lado, se ha apreciado que el uso de los métodos de mezclado de *embeddings* ha aumentado significativamente el rendimiento de los clasificadores en comparación con el resto de métodos de extracción de características para una dimensionalidad con el mismo orden de magnitud. Sin embargo, se debe tener en cuenta que el *overhead* temporal añadido por el mezclado de *embeddings* es superior al del resto de métodos de extracción de características, para el mismo número de características.

Por último, se han observado diferencias entre las puntuaciones obtenidas en el *corpus* de castellano y en el de catalán. En el *corpus* de castellano, los modelos han conseguido una tasa de acierto ligeramente más alta que en catalán. Sin embargo, la puntuación macro F_1 se ha visto afectada de manera significativa en el *corpus* de castellano, donde no ha podido pasar de un promedio 0.6; mientras que en el *corpus* de catalán, se ha obtenido un promedio de 0.75 para esta métrica. Esta diferencia, puede ser debida a que en castellano existe la categoría *nacional* con más del 40% de las noticias, esto crea una mayor probabilidad de que haya una mayor diversidad de textos dentro de ella, lo cual hace que los modelos no sean capaces

de diferenciar claramente las noticias dentro de este conjunto sin penalizar la eficiencia de clasificación para otras categorías. Sin embargo, si vemos la distribución del *corpus* de catalán, observamos que las noticias están más repartidas entre las distintas categorías y no hay una diferencia como la que ocurre en castellano.

4.2 Trabajo Futuro

En esta sección se presentan investigaciones complementarias derivadas del trabajo realizado para este estudio.

Profundizar en las causas de las diferencias de puntuación en las métricas. Si bien en este trabajo se ha lanzado una hipótesis de las razones que han llevado a los modelos entrenado en castellano a obtener un puntuación macro F_1 inferior a la obtenida en catalán, esto no se ha confirmado con un estudio.

Analizar el impacto de ciertos parámetros de los modelos en las métricas. En este trabajo, se han presentado dos parámetros de configuración externos a los modelos, como son el modelo de extracción y el número de características. Con esta información presente, se podría extender el estudio a parámetros de configuración intrínsecos a cada modelo y analizar el impacto que tienen en el rendimiento para la clasificación de noticias de los *corpus* utilizados.

Mejorar el uso de *embeddings* en las tareas de clasificación de noticias. A lo largo de este trabajo, se ha observado como el uso de *embeddings* en la clasificación ha proporcionado, en algunos casos, unas métricas iguales o superiores a las que proporcionaban algunos de los otros métodos de extracción de características con una dimensionalidad muy superior; no obstante, el coste temporal añadido al proceso de vectorizado era significativo. Por ello, sería interesante buscar otros métodos de mezclado o investigar como mejorar los utilizados en el presente trabajo, de modo que se pudiese tener el ahorro en la coste espacial y a su vez, que la penalización en el coste temporal no fuese tan notable.

Bibliografía

[1] F. Alcina Sanchis, "Creación de un corpus de artículos de prensa y generación automática de resumentes," Universitat Politècnica de València, 2019.

[2] C. Goller, J. Löning, T. Will, and W. Wolff, "Automatic document classification: A thorough evaluation of varios methods," *Proceedings des 7. Internationalen Symposiums für Informationswissenschaft (ISI 2000)*, vol. 08, no. 10, pp. 145–162, 2000.

[3] F. Romero and Z. Koochak, "Assessing and implementing automated news classification," Standford University, 2015. [Online]. Available: http://cs229.stanford.edu/proj2015/306_report.pdf

[4] S. Kaur and N. Kaur Khiva, "Online news classification using deep learning technique," *International Research Journal of Engineering and Technology (IRJET)*, vol. 03, no. 10, pp. 588–563, 2016.

[5] D. Jimenez, H. Paz-Arias, and A. Larco-A, "Desarrollo de un sistema inteligente para la clasificación de documentos ya digitalizados aplicando redes neuronales supervisadas," *Revista Tecnológica ESPOL*, vol. 28, no. 1, pp. 8–23, 2015.

[6] "Document understanding conference." [Online]. Available: https://www-nlpir.nist.gov/projects/duc/data.html

[7] K. Moritz Hermann, T. Kočiský, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom, "Teaching machines to read and comprehend," in *Advances in Neural Information Processing Systems (NIPS)*, 2015. [Online]. Available: http://arxiv.org/abs/1506.03340

[8] M. Grusky, M. Naaman, and Y. Artzi, "Newsroom: A dataset of 1.3 million summaries with diverse extractive strategies," *CoRR*, vol. abs/1804.11283, 2018. [Online]. Available: http://arxiv.org/abs/1804.11283

[9] "Proyecto Aracne." [Online]. Available: https://www.fundeu.es/aracne/index.html

[10] D. Raggett, "Getting started with html," 2005. [Online]. Available: https://www.w3.org/MarkUp/Guide/

[11] W3C, "HTML & CSS." [Online]. Available: https://www.w3.org/standards/webdesign/htmlcss

[12] ——, "What is the document object model?" [Online]. Available: https://www.w3.org/TR/DOM-Level-2-Core/introduction.html

[13] O. Fuks, "Classification of news dataset," Standford University, 2018. [Online]. Available: http://cs229.stanford.edu/proj2018/report/183.pdf

[14] C. Figuerola, "Clasificación automática de documentos. Un caso práctico," Universidad de Salamanca, 2017.

[15] W. Kunnu and N. Kaewrattanapat, "The automatic classification of thai news by similarity method," 11 2013.

[16] L. Wei, B. Wei, and B. Wang, "Text classification using support vector machine with mixture of kernel," *A Journal of Sofware Engineering and Applications*, pp. 55–58, 2012.

[17] S. L. Y. Lam and D. Lun Lee, "Feature reduction for neural network based text categorization," in *Proceedings. 6th International Conference on Advanced Systems for Advanced Applications*, 1999, pp. 195–202.

[18] Carnegeie Mellon University, "Hashing." [Online]. Available: http://www.cs.cmu.edu/afs/cs/academic/class/15210-s14/www/lectures/hash.pdf

[19] Scikit-Learn, "Hashingvectorizer," 2019. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.HashingVectorizer.html#sklearn.feature_extraction.text.HashingVectorizer

[20] Y. Li and T. Yang, *Word Embedding for Understanding Natural Language: A Survey*, 05 2017, vol. 26.

[21] Google, "word2vec," 2013. [Online]. Available: https://code.google.com/archive/p/word2vec/

[22] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[23] J. Huang, J. Lu, and C. Ling, "Comparing naive bayes, decision trees, and svm with auc and accuracy," 12 2003, pp. 553– 556.

[24] C. Shalizi, "Advanced Data Analysisc. Chapter 12 - Logistic Regression," Carnegie Mellon University, 2017. [Online]. Available: http://www.stat.cmu.edu/~cshalizi/uADA/12/lectures/ch12.pdf

[25] E. J. Carmona Suárez, "Tutorial sobre máquinas de vectores soporte (svm)," 2014. [Online]. Available: http://www.ia.uned.es/~ejcarmona/publicaciones/%5B2013-Carmona%5D%20SVM.pdf

[26] K. Gurney, *An Introduction to Neural Networks*. UCL Press, 1997.

[27] F. Pla and L. Hurtado Oliver, "Language identification of multilingual posts from twitter: a case study," *Knowledge and Information Systems*, vol. 51, 09 2016.

[28] Scikit-Learn, "Documentation of scikit-learn 0.21.2," 2019. [Online]. Available: https://scikit-learn.org/0.21/documentation.html

[29] A. Defazio, F. Bach, and S. Lacoste-Julien, "Saga: A fast incremental gradient method with support for non-strongly convex composite objectives," *Advances in Neural Information Processing Systems*, vol. 2, 07 2014.

[30] J. Caicedo-Espinoza, G. Parra-Chico, and X. Ochoa-Chehab, "Marco de trabajo para indexación, clasificación y recopilación automática de documentos digitales," Escuela Superior Politécnica del Litoral, 2009. [Online]. Available: http://www.dspace.espol.edu.ec/handle/123456789/1088

[31] E. Pérez-Perdomo, A. Rivas Méndez, M. I. Castellanos Domínguez, and V. Juan Góngora Zaldívar, "Biblioteca digital con técnicas de clasificación automática de documentos," 04 2017.

[32] M. A. Pérez Abelleira and C. A. Cardoso, "Minería de texto para la categorización automática de documentos," Universidad de Católica de Salta, 2010. [Online]. Available: https://www.ucasal.edu.ar/htm/ingenieria/cuadernos/archivos/5-p11-alicia-articulo-cuadernos-formateado.pdf

[33] K. P. Bhoyar, N. S. Hiwase, S. K. Ankurkar, P. P. Bhagat, and D. Wardha, "A review paper on automatic text and image classification for news paper," *International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE)*, vol. 03, no. 02, pp. 191–194, 2014.

[34] M. K. Dalal and M. A. Zaveri, "Automatic text classification: A technical review," *International Journal of Compute Applications (0975 - 8887)*, vol. 28, no. 28, pp. 37–40, August 2011.

[35] M. De Giusti, G. Villarreal, A. Sobrado, and A. Vosou, "Recuperación y clasificación automática de información, resultados actuales y perspectivas futuras," 01 2009.

[36] Scikit-Learn, "Working with text data," 2019. [Online]. Available: https://scikit-learn.org/0.21/tutorial/text_analytics/working_with_text_data.html

[37] N. J. Nilsson, "Introduction to machine learning," 1998. [Online]. Available: http://ai.stanford.edu/~nilsson/MLBOOK.pdf

[38] C. Williams, "Support vector machines," 2008. [Online]. Available: http://www.inf.ed.ac.uk/teaching/courses/iaml/docs/svm.pdf

[39] D. Graf and C. Cieri, "Gigaword," 2003. [Online]. Available: https://catalog.ldc.upenn.edu/LDC2003T05

[40] E. Sandhaus, "New York Times corpus," 2008. [Online]. Available: https://catalog.ldc.upenn.edu/LDC2008T19

[41] D. Graff and G. Gallegos, "Spanish news text." [Online]. Available: https://catalog.ldc.upenn.edu/LDC95T9

[42] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *Proceedings of Workshop at ICLR*, vol. 2013, 01 2013.

[43] G. Forman and E. Kirshenbaum, "Extremely fast text feature extraction for classification and indexing," 01 2008, pp. 1221–1230.

[44] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009.

[45] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[46] A. Cucchiara, "Applied logistic regression," *Technometrics*, vol. 34, pp. 358–359, 03 2012.

[47] J. D. Rennie, L. Shih, J. Teevan, and D. R. Karger, "Tackling the poor assumptions of naive bayes text classifiers," in *Proceedings of the 20th international conference on machine learning (ICML-03)*, 2003, pp. 616–623.

Siglas

API Application Programming Interface.

BOW Bag Of Words.

CSS Cascading Style Sheets.

DOM Document Object Model.

DUC Document Understanding Conference.

GPLv3 GNU General Public License v3.0.

HTML HyperText Markup Language.

LDC Linguistic Data Consortium.

OVO One vs One.

OVR One vs the Rest.

RL Regresión Lineal.

RLOG Regresión Logística.

SAGA Stochastic Average Gradient Accelerated.

SVM Máquina de Vector Soporte.

TF-IDF Term Frequency-Inverse Document Frequency.

W3C World Wide Web Consortium.