uc3m | Universidad Carlos III de Madrid

Grado Universitario en Ingeniería Telemática

2017-2018

Trabajo Fin de Grado

"Análisis comparativo de algoritmos de Deep Learning para la clasificación de textos"

Iván López Pacheco

Tutor

Jesús Cid Sueiro

Leganés, 2018

RESUMEN

Dado el extenso crecimiento que ha experimentado la información en la actualidad, su gestión y análisis se ha convertido en algo esencial para su aplicación en cualquier ámbito de la sociedad, aportando un valor incremental a los datos. Este cambio de mentalidad plantea numerosas cuestiones que requieren técnicas de aprendizaje automático para su resolución. La clasificación automática de textos favorece el desarrollo de un gran número de ellas, atractivo por el que se ha desarrollado todo un sistema que simule el tratamiento de datos textuales, desde la extracción hasta la etapa final, la categorización.

Para llevar a cabo dicho proceso, este proyecto incorpora numerosas técnicas y tecnologías, tanto de procesamiento de lenguaje natural como de construcción de redes neuronales, que han hecho posible su implementación. La creación de una araña web será el punto de partida de este proceso, que continuará con el desarrollo de la fase de preprocesamiento, técnicas de data cleansing para seleccionar la información relevante, que favorecerán el funcionamiento del objetivo final, el clasificador. En esta última etapa se aplicarán técnicas de Deep Learning para implementar las soluciones, a la vez que se establecerá una comparativa entre las prestaciones obtenidas.

Implementar el circuito expuesto necesitará una previa investigación sobre la situación actual de la materia en cuestión, incluyendo los métodos existentes y sus respectivas ventajas y limitaciones, así como los principios teóricos que dan sustento a su desarrollo y las aplicaciones afines a este problema.

En conclusión, se analizará la influencia de las redes neuronales en los problemas de clasificación de textos mediante la alteración de los parámetros que las componen, y su utilidad frente a los algoritmos clásicos de aprendizaje automático.

Palabras clave: Deep Learning; Clasificación de textos; Neurona artificial; Araña web; TensorFlow

AGRADECIMIENTOS

Antes de empezar, me gustaría hacer una mención especial a todas aquellas personas que me han acompañado durante este tiempo.

En primer lugar, agradecer a todos los profesores que me han formado a lo largo de mi vida académica. Todos ellos han creado la base necesaria para que yo pudiera acabar con éxito estos estudios de grado. Agradecimiento especial a mi tutor Jesús, por haberse mostrado a mi disposición cuando he necesitado su ayuda y orientación durante la elaboración de este proyecto, generándome el interés por esta rama de conocimiento.

Agradecer a mis padres el apoyo mostrado durante todo este período, por su comprensión y ayuda, esenciales en estos años, que me ha facilitado el camino para conseguir todos los objetivos que me he propuesto.

Por último, imposible olvidarme de mis compañeros, con los que he pasado incontables horas durante estos cuatro duros años, quienes hemos compartido tantos momentos de agobio y sufrimiento, pero también tantos buenos momentos y experiencias. Porque sus ánimos, apoyo y amistad han sido esenciales para avanzar día tras día. En resumen, sin ellos este camino hubiera sido mucho más difícil.

A todos ellos, gracias.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1. Motivación

La clasificación de textos según determinados criterios es uno de los campos más prometedores en la actualidad. Automatizar los procesos de clasificación reduciría considerablemente los tiempos y los costes de tareas como la gestión de expedientes, valoraciones mediante análisis sentimental, búsqueda de resultados, recomendación de contenidos, o estructuración en taxonomías de las cantidades ingentes de datos de las que se dispone actualmente. La creciente digitalización de los contenidos y la extensión del concepto "Web 3.0", donde la retroalimentación de las publicaciones en la red ha dado lugar a la fluidez del contenido en la nube, necesitan de herramientas de tratamiento de datos que consigan llevar a cabo la automatización de estas funciones.

El tratamiento de datos mediante algoritmos de aprendizaje automático es una disciplina actualmente en extensión con aplicaciones en un gran abanico de sectores como las finanzas, electrónica industrial, marketing o servicios tecnológicos. Una de las disciplinas que permiten encontrar soluciones a necesidades tan versátiles es la lingüística computacional, en la que se centrará este proyecto para la elaboración del análisis de documentos textuales, área en pleno desarrollo en la que se ha considerado interesante profundizar.

El objetivo del sector es conseguir, mediante algoritmos e interacción máquina - usuario, que el sistema sea capaz de entender el lenguaje humano léxica, sintáctica y semánticamente, para lo que es necesario aprender del contexto y de las relaciones a nivel de palabra. Saber interpretar lo que se expresa, y generarlo de acuerdo con normas lingüísticas, estadísticas e informáticas, queda recogido en el concepto "Machine Learning", que, mediante técnicas de modelado del lenguaje natural, su reconocimiento y procesamiento (NLP) permite orientar este proyecto a la clasificación de textos.

En resumen, ser capaz de entender las tendencias del texto adaptándose a la nueva era de la comunicación, estableciendo lógicas y patrones de similitud entre características, tal y como se producirían en la mente humana, presentaría una extensibilidad inmensa. Es en este punto donde entran en juego las técnicas de Deep Learning, concepto innovador en el que se ha basado este proyecto permitiendo crear diferentes niveles de abstracción

de información. Asimismo, dada la reciente creación de la tecnología utilizada para la construcción de algoritmos, existe un número reducido de implementaciones disponibles. Entre ellas, un alto porcentaje corresponde a aplicaciones para corpus de imágenes, por lo que cabe destacar que el diseño de los algoritmos para corpus textuales ha supuesto un reto atractivo de afrontar en este proyecto.

1.2. Objetivos del proyecto

Una vez contextualizado brevemente el ámbito de estudio del trabajo, se plantea llevar a cabo una labor de investigación y desarrollo de un sistema de aprendizaje automático, particularizado para el caso de la clasificación de textos. En él se evaluará la veracidad de la predicción obtenida a partir de diferentes algoritmos de Machine Learning, dedicando especial interés a su evolución en la rama del Deep Learning, entre los que se ofrecerá un análisis comparativo de prestaciones.

Tomando de referencia este objetivo principal, se pretende configurar un entorno completo de tratamiento de datos de principio a fin, participando activamente en todas las fases del proceso: extracción, cleansing, procesado, análisis y visualización de los datos. Para cada una de ellas se estudiarán las herramientas más apropiadas focalizando el interés en su utilidad y posibles limitaciones. De esta forma, la integración de las distintas técnicas, así como adaptación de desarrollos ya existentes, sustentan la parte de investigación del proyecto.

Una vez finalizada su elaboración, se deberá haber conseguido llevar a cabo los siguientes hitos:

- Conocer, contextualizar y profundizar en el estudio del lenguaje natural y sus aplicaciones para su posterior empleo en el desarrollo software.
- Aportar una visión global del concepto “tratamiento del dato” haciendo un recorrido pormenorizado de todas sus fases.
- Realizar una investigación sobre las técnicas existentes utilizadas en algoritmos de aprendizaje automático, que aporte una perspectiva del concepto y, en particular, de su modelado mediante Deep Learning.

- Hacer un uso eficiente de las diferentes tecnologías que se ofertan para llevar a cabo procedimientos de clasificación, implementando los algoritmos más representativos.
- Establecer un análisis comparativo entre los resultados obtenidos a través de las diferentes técnicas.
- Generar un proyecto estructurado que aporte y sea útil para el posterior desarrollo de esta línea de conocimiento.

1.3. Estructura de la memoria

La presente memoria pretende exponer la situación del aprendizaje automático en el mundo actual, centrando la atención en sus aplicaciones en la clasificación de textos, así como servir de guía para el futuro desarrollo de los experimentos realizados. Tras su lectura, se debe haber sido capaz de interiorizar la lógica establecida en los algoritmos y entender con claridad el significado y la utilidad de cada uno de los elementos tratados, a través de los avances en las correspondientes tecnologías.

Se ha decidido organizar una estructura en la que el lector denote un desarrollo incremental de sus conocimientos a lo largo de las distintas fases del procesado de los datos. Siguiendo la misma filosofía, el documento irá enlazando los contenidos de manera que se muestre una evolución desde los aspectos generales hasta los más específicos en cada apartado. Además de esta introducción, las ocho secciones diferenciadas de esta memoria son:

- **Estado de la cuestión**: Contextualizará el marco sociocultural sobre el que se desarrolla este trabajo, comenzando por el análisis genérico de la inteligencia artificial, hasta desembocar en el análisis del estado del arte del problema en cuestión, la clasificación de textos mediante Deep Learning. Asimismo, se incluirán alusiones al contexto histórico que ha marcado la aparición de las técnicas actuales, así como al marco regulatorio por el que se rige la elaboración del proyecto.
- **Extracción y construcción del vector de características:** Contendrá las técnicas y desarrollos realizados en la primera parte del proceso de tratamiento de datos implementado, el cual se corresponde con las etapas de extracción, cleansing y procesamiento. Incluirá constantes alusiones a la justificación teórica que sustenta cada paso de la implementación.

- **Desarrollo e implementación de los clasificadores:** Complementará la sección anterior añadiendo la descripción de la fase de análisis del dato mediante la implementación de clasificadores para la extracción de porcentajes de precisión. Se ha considerado conveniente incluir en esta sección un análisis pormenorizado de los aspectos que determinan las funcionalidades de Deep Learning, en paralelo con su aportación técnica, de gran importancia al ser un pilar fundamental de este trabajo.
- **Evaluación de los resultados obtenidos:** Concluidos los experimentos, se analizarán las prestaciones que es capaz de generar cada uno de los algoritmos tratados, estableciendo procedimientos analíticos para elaborar la comparativa e incluyendo valoraciones personales como criterio de selección.
- **Conclusiones:** Finalmente, se destacarán las conclusiones extraídas tras la implementación del proyecto, el cumplimento de objetivos, así como las posibles líneas futuras para las que podría ser útil el desarrollo de este trabajo.
- **Anexo A – Entorno socio-económico:** Expondrá los principales aspectos del contexto económico que ha tenido que afrontar este proyecto, así como el impacto que generará en la sociedad no visto en anteriores secciones.
- **Anexo B – Resumen de contenidos en inglés:** En este anexo se incluirá el contenido del "abstract" necesario sobre el proyecto, incluyendo un breve extracto de las secciones más relevantes de esta memoria.
- **Bibliografía:** En esta sección final, se utilizará el estándar IEEE para dejar constancia de todas las referencias mencionadas en el documento.

Cabe destacar que el estado de la cuestión sirve de introducción a la situación actual del problema resuelto mediante la exposición de una parte de la investigación realizada. Sin embargo, gran cantidad de la información encontrada en dicha búsqueda es descrita a lo largo de todo el documento según el apartado donde se ha considerado que encajan mejor dichos datos, haciendo siempre alusión a las referencias utilizadas.

Por otra parte, al presentar este documento gran contenido técnico y conceptos relativamente innovadores sin traducción directa al español, se utilizarán términos en su mayoría procedentes del inglés. Se ha procurado utilizar léxico lo más cercano posible a la palabra original, en muchos casos manteniendo su forma en inglés, salvo en aquellas ocasiones en las que la adaptación del anglicismo al español esté considerablemente extendida y favorezca su entendimiento.

2. ESTADO DE LA CUESTIÓN

La descripción del escenario en el que se desarrolla este proyecto será descrita en esta sección, donde se establecerán constantes alusiones a las referencias en las que se fundamenta. Se detallarán los algoritmos ya existentes y las principales tecnologías utilizadas en su implementación, basando la elección en el fundamento teórico que servirá para contextualizar y entender el posterior desarrollo.

Además, actuará como introducción al leitmotiv de este trabajo, el análisis de las técnicas de Deep Learning en la clasificación de textos. Se expondrá la evolución histórica que ha sufrido el sector hasta llegar a las técnicas actuales, para dar paso a la particularización de las mismas en los siguientes apartados. En ellos, se ha optado por aportar una explicación a modo introductorio para captar los conceptos clave necesarios en la posterior implementación de los distintos elementos tratados, sin profundizar en los aspectos matemáticos intrínsecos, pero prestando especial atención a las aplicaciones y consideraciones prácticas de los mismos.

2.1. Impacto de los datos en la sociedad de la información

La revolución digital en la que nos encontramos inmersos ha convertido cada uno de los datos que generamos diariamente en algo esencial para su análisis, lo que aporta un valor añadido a las cantidades ingentes de datos producidas en todo el planeta. La digitalización de la información en prácticamente cualquier acción que llevemos a cabo, como operaciones bancarias, repositorios en la nube o comercio electrónico, ha aportado la capacidad de extraer conclusiones sobre los datos, de manera que el mundo físico se vea reflejado y modelado mediante una representación íntegra en el mundo digital.

Establecer un análisis exhaustivo sobre esta información ha generalizado el concepto de Big Data hasta el punto de considerarse el comienzo de una nueva era tecnológica. Esta corriente ha generado un cambio en la economía global invirtiendo altos presupuestos en las TIC (Tecnologías de Información y Comunicación). Los datos se han convertido en un activo clave a todos los niveles: tecnológico (base de la implementación de los sistemas), económico (modificación en los objetivos de las inversiones, y creación de puestos de trabajo para nuevos profesionales), político (establecimiento de leyes que

recojan la nueva realidad en el sector) y sociocultural (las conclusiones extraídas de los análisis de datos afectan a la sociedad en todos sus aspectos).

El desarrollo del Big Data ha ofrecido soluciones a innumerables retos de la sociedad. A modo de ejemplo cabe destacar aquellos centrados en el ámbito humano y la sostenibilidad (smart-cities); en el marketing, incorporando entre sus máximas el entendimiento del consumidor mediante técnicas de inteligencia artificial; o en la medicina, con los avances en el estudio del genoma humano o la detección previa de pandemias a partir de datos estadísticos. Otro posible caso que destacar es el sistema desarrollado por el MIT en el que los datos de transacciones económicas llevadas a cabo por Internet son almacenados y utilizados para estimar la tasa de inflación con resultados mucho más precisos que los métodos tradicionales [1].

2.2. Introducción a la inteligencia artificial

Conscientes de la sociedad digitalizada en la que nos encontramos y del posicionamiento global de todo lo que ocurre a nuestro alrededor, en los últimos años se ha fomentado la interconexión tecnológica de manera masiva. El fenómeno de análisis de esta cantidad de información pone a nuestro alcance resolver problemas de clasificación, predicción u optimización, siendo una máquina la que infiera los resultados. Este planteamiento ha derivado en el concepto de inteligencia artificial. Son numerosos los estudios donde la tecnología se basa en el componente humano para intentar impregnar de inteligencia a los dispositivos con los que tratamos día a día, punto fundamental en el desarrollo del Internet de las Cosas.

Como se describe en [2]: “El Internet de las cosas es un paradigma computacional reconocido por permitir la conexión entre el mundo físico y virtual al aportar poder a la información rutinaria”. La gestión de esta información y por tanto la explotación de su utilidad sale a la luz mediante algoritmos de aprendizaje automático (Machine Learning).

Desde la robótica, donde los sistemas actúan como humanos, hasta las redes neuronales donde estos piensas como tales, existen numerosas variantes en la lógica de los algoritmos divididas de acuerdo con el proceso de aprendizaje que el algoritmo utiliza, descrito brevemente a continuación:

- **Aprendizaje supervisado:** Consiste en el aprendizaje del sistema cuando se dispone de un conjunto de entrenamiento con datos previamente etiquetados, a partir de los cuales el algoritmo debe modelar su sistema para extrapolar las conclusiones oportunas a un conjunto de datos futuro.
- **Aprendizaje no supervisado:** Técnica que no utiliza datos preentrenados, por lo que la red no dispone de influencia externa para ajustar los pesos de las conexiones entre sus neuronas. La clasificación se lleva a cabo a partir de las propiedades del único conjunto de datos presente. En este caso, mediante asociación de características, o correlaciones entre los datos de entrada, se generan patrones que permitirán su posterior aplicación.
- **Aprendizaje semi-supervisado:** Como es de suponer, utiliza tanto datos previamente etiquetados como no controlados, que suelen configurar el mayor porcentaje. Su objetivo es conseguir la misma precisión que sus variantes, reduciendo el coste humano que genera el etiquetado manual del conjunto de entrenamiento.

Entre los primeros algoritmos desarrollados para su aplicación en el análisis de datos destacaron las redes bayesianas, donde, siguiendo un modelo probabilístico base, se puede llegar a determinar las dependencias condicionales existentes entre los datos mediante un grafo acíclico dirigido. [3]

Otro de los algoritmos a considerar es el aprendizaje a través de árboles de decisión. Se establecen predicciones mediante la construcción de una ruta que adquiere forma a través de las reglas seguidas en cada nivel del árbol. Este procedimiento requiere un modelado experto para establecer decisiones que conformen correctamente la estructura. Además, los patrones cambian continuamente, lo que implica reprogramar las decisiones a menudo para que el algoritmo mantenga su efectividad, por lo que su escalabilidad y generalización estará muy limitada. [4]

Para concluir con esta introducción a los mecanismos existentes, cabe destacar el aprendizaje mediante reglas de asociación, que utiliza los conjuntos de datos para extraer relaciones semánticas entre variables que se puedan utilizar como predicción de futuros corpus. La implementación de cada uno de estos métodos fue un hito para el ámbito de la inteligencia artificial, pero no fue hasta la segunda mitad del siglo XX cuando la aparición de las redes neuronales revolucionó el concepto.

2.3. Evolución a las redes neuronales

Un humano dispone de aproximadamente 10^{11} neuronas que, interconectadas, generan múltiples asociaciones, lo que, procesado y estructurado, llamamos "inteligencia". Esta concepción biológica es la base de los desarrollos de aprendizaje automático y en especial del Deep Learning, las similitudes entre el cerebro humano y su equivalente en el mundo digital. La idea de aunar en un mismo sistema las altas prestaciones de computación de un sistema electrónico, con la capacidad de interpretación de los datos en un contexto determinado que puede aportar la mente humana, fue el motor principal de esta nueva concepción tecnológica.

Desde que Frank Rosenblatt diseñó en 1960 sus experimentos sobre el perceptrón multicapa [5], se desarrolló una corriente de pensamiento en el que la idea de generar una neurona humana de manera artificial era realmente atractiva. Sin embargo, la falta de robustez del algoritmo, probada por Minsky y Papert en [6], hizo que el concepto perdiera fuerza hasta que en 1974 Paul Werbos desarrolló en su tesis el planteamiento sobre redes neuronales: *backpropagation* [7], idea que se asentó en [8] y que será explicada en secciones posteriores de este documento como aspecto de diferenciación y sustento de estas estas. Este hecho generó el resurgir de las redes neuronales, que fueron el tema central de numerosas publicaciones hasta la actualidad, donde siguen apareciendo nuevas técnicas como el *dropout*, que surgió recientemente en 2014 [9] y que será tratado en los experimentos posteriores de este proyecto. Cabe destacar la evolución de las técnicas de aprendizaje automático en los últimos años, ligada al desarrollo de nuevas tecnologías que han permitían implementarlas con mayor simplicidad.

Las neuronas presentan numerosas ventajas semejantes a las de su símil biológico en el sistema nervioso, entre las que destacan [10]:

- **Aprendizaje adaptativo**: Son sistemas dinámicos que, tras un proceso de aprendizaje, autoajustan sus parámetros para adaptarse a las nuevas condiciones de la red mediante el previo entrenamiento con patrones, lo que deriva en una capacidad de autoorganización de las neuronas.

- **Tolerancia a fallos**: La posibilidad de distribuir la información en diversas conexiones, con cierto grado de redundancia, permite tolerar patrones con ruido o incompletos, de manera que, a pesar de que el sistema se ve influenciado, no se produce una caída repentina del mismo.
- **Operación en tiempo real**: Los cómputos neuronales tienen la capacidad de distribuirse de forma paralela, operando en tiempo real y facilitando el proceso de adaptación de los pesos en las conexiones.

El funcionamiento de una red neuronal consiste en la introducción de una serie de entradas en las neuronas que emitirán una salida determinada por tres funciones. Estas serán explicadas a continuación en su versión estándar, a pesar de la existencia de variantes de cada una de ellas en función de la especialización del estudio realizado [11]:

- **Propagación**: Funciona como regulador de las señales emitidas mediante el cómputo del sumatorio ponderado de cada entrada multiplicada por el peso asignado.
- **Activación**: Su objetivo es modificar los resultados procedentes de la función de propagación de cara a obtener valores en un rango determinado. Su función en la neurona es esencial para su funcionamiento. A lo largo del siguiente apartado se prestará atención a su utilidad y posibles variantes.
- **Transferencia o salida**: Acota el valor de salida de cada neurona según la interpretación que se quiera recibir de él, actuando como umbral para la aceptación de los valores finales que serán transmitidos. Es habitual que esta función se omita de manera que la salida tome directamente el valor del propio estado de activación de la neurona ya acotado.

Fig. 2.1. Elementos básicos de una neurona [12]

En la imagen superior, se aprecia cómo la neurona recibe información del exterior a través de entradas (x), cada una de las cuales lleva asignado un peso (w) que representa la intensidad que va a tener dicha entrada dentro de la neurona. Esto hará que algunas de ellas presenten un mayor efecto en el procesamiento de la neurona cuando se realiza la combinación entre ellas. Los pesos serán coeficientes adaptables, cuyo valor influirá en la obtención de resultados [12]. Finalmente, para un caso de función de salida inexistente, la salida (y) vendrá determinada por la expresión:

$$y = f\left(b + \sum_{n} w_n x_n\right) \tag{2.1}$$

siendo $f(z)$ la función de activación y z la función de propagación correspondiente.

Una vez expuestos los componentes matemáticos de la neurona, entra en juego su aplicación. La red neuronal servirá para generar predicciones a partir de los datos de entrada. La función de activación utilizada y el carácter interno de la neurona origina diversas tipologías en el desarrollo de los algoritmos que se describirán en el punto 4 de este documento, al mismo tiempo que se profundizará en su implementación.

2.4. Aplicación del aprendizaje automático a la clasificación de textos

Explicado el concepto y su importancia para la sociedad, es esencial conocer la situación del sector en el ámbito específico de la clasificación de documentos de carácter textual en el que se desarrolla este trabajo. Son numerosas las aplicaciones en las que la inteligencia artificial ha intervenido para esta materia de estudio, entre las que cabe destacar la detección de mensajes spam, motores de búsqueda y recomendación o la traducción automática. Extraer conclusiones sobre un conjunto de textos como si la interpretación estuviera realizada por un humano presenta numerosas ventajas y genera una evolución en el contexto social a gran escala.

El crecimiento de la digitalización de documentos ha convertido la tarea de clasificación en algo esencial para disponer de material organizado. Es por ello por lo que se ha fomentado una innovación en el sector de manera que estas tareas pudieran ser realizadas por ordenador sin necesidad de componente humano, pero con la suficiente precisión para confiar en los resultados. En la actualidad, empresas como Google o

Twitter, que tienen a su disposición enormes cantidades de datos, se han visto obligadas a desarrollar técnicas de aprendizaje automático para su gestión y tratamiento.

Hoy en día, el concepto de web 3.0 ha generado una ingente cantidad de páginas web que se encuentra en crecimiento constante, así como una enorme versatilidad en los temas que estas cubren. La búsqueda efectiva de los temas solicitados por el usuario se complica en estos términos haciendo prácticamente obligatorio un método de clasificación automática para estructurar esta cantidad de información y explotar su máximo potencial. Las técnicas de procesamiento de lenguaje natural (NLP) se han extendido con el objetivo de solventar esta casuística.

Desde la década de los 70, la lingüística computacional se proclama como una de las ramas de mayor interés de la inteligencia artificial. Entre las cuestiones que más investigaciones han recibido destaca el estudio de la traducción automática, donde mediante Deep Learning se pretende conseguir una equivalencia entre lenguajes, a nivel de palabra y de expresión, buscando el contexto acertado en el que se encuentran. En los últimos años, la multinacional Google ha sido pionera en utilizar técnicas de *Word Embeddings*, asociando a cada palabra un vector y creando una estructura multiplano donde cada uno de ellos corresponde a un idioma, de manera que el paralelismo entre plano generaría la traducción correcta.

2.4.1. Algoritmia para la clasificación de textos

Desde técnicas de decisión binomial hasta aquellas que permiten clasificación multietiqueta, existen numerosas investigaciones que intentan averiguar la mejor combinación de factores para alcanzar una precisión óptima en las predicciones. La clasificación de textos mediante métodos de aprendizaje supervisado se lleva tratando desde finales del siglo pasado. En 1999 se desarrolló un sistema para la organización de documentos de patentes de EE.UU. para facilitar su búsqueda, usando como clasificador el algoritmo *k-nearest-neighbor*, dejando entrever la importancia de sus futuras aplicaciones [13].

Analizando estudios publicados desde principios de este siglo, se puede afirmar que el clasificador de textos por excelencia en los experimentos realizados es SVM con sus respectivas adaptaciones [14]. Su utilización está ampliamente extendida en corpus

textuales, para los que también se suelen aplicar Naive-Bayes, que permite obtener resultados fiables, y *Random Forest*; todos ellos algoritmos referentes a la concepción clásica de Machine Learning.

Frente a la clasificación multietiqueta, la modalidad binaria es la que se encuentra más generalizada en las investigaciones. Estos clasificadores necesitan menos esfuerzo computacional al generar salidas booleanas (pertenencia o no a la categoría en cuestión), Cabe destacar la utilización en los estudios de datasets formados por artículos de Reuters y Wikipedia. En [15] se analiza el uso de Naive-Bayes, con su desarrollo matemático intrínseco, ejemplificando el caso con el corpus de Reuters. Años más tarde un nuevo artículo ampliaría el algoritmo para su aplicación antispam [16].

En 2007, se practicó un desarrollo mixto entre clasificación y reglas de asociación (*associative classification*) [17]. Planteado como mejora de la bolsa de palabras, donde la estructura jerárquica de los documentos se pierde en favor de la individualidad de la palabra, mediante este método se establecerían relaciones entre palabras que se pudieran encontrar habitualmente en la misma oración basando el algoritmo en el clasificador Harmony [18]. Esto originaba subestructuras más específicas que permitían delimitar con mayor precisión la actuación del clasificador. Para comprobar su desempeño, se utilizó el corpus de Reuters construyendo una comparativa entre diversos clasificadores. Así, destacan los resultados de SVM en categorías formadas por un gran número de documentos, y Harmony adaptado, en las categorías menos abundantes.

Documentos textuales como aquellos procedentes de Wikipedia han sido estudiados en un elevado número de casos con los mismos algoritmos básicos de Machine Learning que se han comentado hasta ahora. Sin embargo, el afán por sacar el máximo provecho de la información disponible ha buscado evolucionar las técnicas utilizadas de manera que las relaciones existentes entre artículos a través de enlaces influyeran en la decisión. Esto se muestra en [19], donde la anotación manual de información relacional permite conseguir un resultado aproximado de 90.84 ± 4.28% de precisión.

En las numerosas referencias consultadas, se aprecia un constante intento de adaptación de los algoritmos a los corpus analizados, lo que da lugar a heurísticos sobre la fiabilidad de su uso en futuros casos. Por otra parte, destaca como aspecto común a las investigaciones anteriores el previo procesado de los datos mediante técnicas de cleansing

(tokenización, filtrado, stemming...) que permiten el aumento de precisión en los resultados experimentales.

Aunque las redes neuronales se lleven utilizando desde finales del siglo XX, no es hasta estos últimos años cuando la tecnología ha permitido su extensión y difusión. Es por ello que existen estudios de aplicación de redes neuronales básicas pero resulta realmente difícil encontrar desarrollos que presenten análisis de técnicas de Deep Learning en el algoritmo, dado que un gran número de investigaciones relacionadas dejan este aspecto como materia a desarrollar en sus líneas futuras. Con esta premisa, implementaciones en Tensorflow han basado sus experimentos en corpus de imágenes, reduciendo considerablemente la búsqueda si se limita a la clasificación de textos. En este último caso, este proyecto se ha nutrido de publicaciones en repositorios de código abierto como GitHub [20], donde se muestra el proceso de adaptación de corpus textuales para su integración en *deep neural networks* mediante clasificadores previamente implementados. Las reducidas apariciones de esta herramienta se deben al hecho de que su uso favorece la aplicación con recursos visuales. Es por ello por lo que se pueden encontrar numerosos desarrollos con corpus como "iris" o MNIST.

En el artículo [21] de 2015 se establecen comparaciones entre los algoritmos tradicionales como las bolsas de palabras y las recientes técnicas de Deep Learning utilizando redes convolucionales. Una mayor profundidad en estas técnicas se aprecia en la comparativa establecida en 2017 entre algoritmos de aprendizaje automático, donde la categorización del corpus de críticas de IMDB y de mensajes de Twitter sirven para comparar algoritmos de machine learning como Naive-Bayes y de Deep Learning como LSTM, red neuronal multicapa o CNN, obteniendo resultados bastante similares, pero destacando estos últimos [22]. Es interesante precisar que aspectos relacionados con el procesado multietiqueta fueron investigados en 2017 reflejando las innovadoras técnicas utilizadas en [23].

Asimismo, cabe destacar el enfoque actual que está recibiendo el Deep Learning con las técnicas de Word Embeddings (técnica explicada en el siguiente capítulo de este documento), permitiendo abarcar un número creciente de áreas de conocimiento dada la efectividad del método. Es el caso del auge de la traducción automática comentado anteriormente, o el estudio publicado en 2017 de su precisión para el reconocimiento de entidades biomédicas [24].

Tras el análisis y estudio del estado del arte actual, se concluye que los algoritmos de Deep Learning están en pleno desarrollo en los últimos años, por lo que aún queda un amplio terreno por explorar. En un alto porcentaje de investigaciones, estas técnicas son tratadas superficialmente, limitándose a los algoritmos existentes sin apenas matices de innovación ni de profundidad en los innumerables parámetros que pueden intervenir en las redes neuronales profundas. A pesar de ello, el pensamiento general concluye que se obtienen mejores resultados con estas técnicas que con los procedimientos básicos de machine learning como KNN o *decision trees*, destacando los algoritmos de Naive-Bayes y SVM, que, en numerosas ocasiones, proporcionan resultados de elevada precisión.

2.4.2. Casos de estudio

Expuestos algunos de los algoritmos utilizados hasta el momento, es necesario tener en cuenta las posibles aplicaciones por las que la automatización de la clasificación de textos adquiere tanta importancia y puede resultar fundamental. Clasificar documentos no es un simple método de organización, sino una solución a problemas como los descritos a continuación a modo de ejemplo.

Una de las cuestiones que se busca resolver es el análisis sentimental, lo que consiste en el análisis de opiniones determinando la subjetividad que el autor ha expresado en ellas. Esta clasificación puede ser binaria (positiva o negativa) o multietiqueta (gradual), pero en ambos casos, captar la intencionalidad del autor es crucial para su éxito. Por ello, exige una mayor complejidad que la clasificación temática al incorporar el aspecto semántico al análisis. Experimentos para este caso de estudio se han llevado a cabo utilizando como corpus destacados “IMDB reviews”, o colecciones de tweets publicados.

El dataset correspondiente a IMDB contiene 1400 críticas de cine procedentes de dicho portal web, y debe su relevancia a estar compuesto por igual proporción de críticas negativas y positivas, lo cual permite realizar experimentos de clasificación binomial en base a la orientación semántica de cada documento.

Algoritmos de Machine Learning implementados en Python, incluyendo en este caso redes neuronales, articulan el desarrollo del análisis sentimental en Twitter expuesto en 2017 [25]. Teniendo en cuenta las limitaciones presentes, como la restricción del

tamaño del tweet, uso de hashtags que requiere un procesado diferente al lenguaje natural, o la evolución constante de las formas coloquiales de los usuarios, se sometió el dataset a un pre-procesado para reducir las ambigüedades en la matriz de características. Tecnologías de extensión más reciente (Scikit-Learn o NLTK), son utilizadas para construir clasificadores de diversos tipos, destacando, como era de esperar, resultados procedentes de las construcciones de Deep Learning. Algoritmos poco comunes como los "*Ensemble classifiers*", mezclan características de clasificadores básicos como SVM o la "Arquitectura dinámica de redes neuronales artificiales" (sin fijación previa de número de capas ocultas) aportando resultados atractivos. [26], [27].

Se ha de destacar el tratamiento de la clasificación de textos en otros trabajos de fin de grado como [28], donde se aplica a un corpus de tweets con el objetivo de establecer clasificación entre los temas sobre los que se opina en el período de elecciones políticas, mediante clasificador basado en reglas. Finalmente, otra referencia interesante es [29] en la que se aplican los algoritmos al reconocimiento de dígitos manuscritos.

2.4.3. Caso de estudio: Clasificación de páginas web

La clasificación de documentos de lenguaje natural se extiende en este proyecto al caso particular de páginas web en lenguaje HTML, para lo que se analizará si las técnicas de lenguaje natural son también aplicables a este tipo de documentos. Para contextualizar el estado del arte de este caso específico de aplicación y conocer su situación actual, se ha considerado oportuno investigar sobre anteriores experimentos que serán descritos en esta sección .

Tras la alusión a las referencias genéricas vistas anteriormente, cabe destacar la introducción en el campo de clasificación de páginas web realizada en 2007 [30], donde se utilizan técnicas de Machine Learning para clasificar páginas, pero únicamente basándose en la información de la URL, aspecto que se ha considerado extremadamente básico en esta investigación como para desempeñar una función decisiva en la clasificación, dado que un alto número de páginas web presentan un nombre de dominio que no representa su contenido, siendo meros intermediarios en la redirección. En los experimentos realizados, se concluye que SVM y Naive Bayes aportan los mejores resultados para tal fin.

En 2012, una clasificación temática de páginas web se realiza mediante Naive Bayes generando resultados nada despreciables con una precisión media de 89.09 en un diseño multietiqueta con 10 categorías [31]. Ese mismo año, un nuevo estudio sobre clasificación de páginas web fue realizado por un grupo de investigadores de Egipto incorporando en este caso el análisis del contenido de las páginas [32]. En él, se realiza una comparativa de algoritmos para uno de los casos tratados en este proyecto utilizando para su desarrollo SVM, KNN y GIS, todos ellos pertenecientes a técnicas de aprendizaje automático, pero sin incluir alusiones a Deep Learning.

Cabe destacar la dificultad expuesta en [33], utilizando los métodos tradicionales, para obtener una precisión elevada, generando una media de 49.9 mediante regresión logística y mejorando hasta un 87.7 aplicando una variante de SVM para clasificación temática. Finalmente, el tratamiento de redes neuronales se aprecia en 2017 [34], donde se alcanza un 82.6 % de precisión para una categorización temática limitada.

2.5. Mención al marco legal

Todo proyecto que se desarrolle en cualquier ámbito presenta un entorno socioeconómico, cultural, político-legal y tecnológico, al que debe hacer frente tanto para generar los desarrollos como para establecer sus conclusiones. Referente al contexto político-legal, en este apartado se desarrollarán los respectivos criterios que se han tenido en cuenta en su elaboración, así como los matices éticos y morales que influyen en los temas desarrollados.

Las bases de datos utilizadas en esta investigación han sido seleccionadas teniendo en cuenta la importancia del carácter legislativo en la protección de datos y los derechos de autor. En el caso de los artículos de Reuters, se ha verificado que la fuente cedió esos datos públicamente pertenecientes al período entre 20/08/1996 y 19/08/1997 para su uso en experimentos analíticos.

Para el caso de los datos obtenidos a partir del proceso de scraping, ha sido necesario tener varios puntos en cuenta sobre los aspectos legales y éticos con los que está relacionado. Al ser un concepto tan reciente, no existe una legislación que le afecte directamente pero sí que su actividad debe ceñirse a las propiedades de derechos de autor correspondientes, dado que la extracción de contenidos infringiendo el copyright o los

términos de uso, supondría estar actuando de manera fraudulenta [35]. Cabe destacar el hecho de que la Agencia Española de Protección de Datos declaró en 2008 que las páginas web no podrán ser consideradas en ningún caso fuentes accesibles al público a pesar de que su contenido se muestre públicamente. Por ello, se deben evitar fines que vulneren los derechos de propiedad intelectual, conductas no legales como la competencia desleal o incumplimientos de la normativa de protección de datos.

El copyright protege los derechos de explotación de la página según los factores con los que sido registrado, desde únicamente el logotipo, hasta la página como conjunto estructurado de su contenido. El uso de los datos para este proyecto es puramente académico, por lo que simplemente se ha extraído una muestra representativa de cada página, es decir, una parte de su contenido HTML para su posterior análisis.

Con la aprobación de la Ley 19/2013, de 9 de diciembre, de transparencia, acceso a la información pública y buen gobierno, se concede el derecho de todo ciudadano a acceder a la información que esté en posesión de cualquier organismo público siempre y cuando no suponga un perjuicio para la seguridad nacional, propiedad intelectual o la garantía de confidencialidad, entre otros criterios de peso mayor. Con esta base, la obtención de datos relativos a dominios de páginas web se llevó a cabo a través de documentos publicados en "Red.es", entidad pública empresarial adscrita a la Secretaría de Estado de Telecomunicaciones y para la Sociedad de la Información del Ministerio de Energía, Turismo y Agenda Digital, el cual tiene la obligación de publicar a nivel de usuario los dominios públicos existentes hasta ese momento en la base de datos estatal.

Por otra parte, la dimensión alcanzada en la publicación de código *open-source* en repositorios de acceso abierto de Internet ha permitido la utilización legal de librerías implementadas para elaborar parte de los experimentos, de manera que exista una retroalimentación en la innovación tecnológica que permita continuar el desarrollo de la ciencia.

Finalmente, y centrando el foco de atención en la propiedad de los datos utilizados en este documento, a menos que se indique expresamente, todo el contenido gráfico y textual de este trabajo ha sido generado por el autor de este, utilizando técnicas de representación gráfica mediante lenguajes analíticos de programación (Python). En los casos en los que las imágenes hubieran sido tomadas de otras fuentes, estas están indicadas en el pie de dicha figura.

3. EXTRACCIÓN Y CONSTRUCCIÓN DEL VECTOR DE CARACTERÍSTICAS

En este apartado se describirán las decisiones de diseño tomadas a lo largo de la elaboración del proyecto, así como las técnicas de implementación utilizadas. En cada sección se dispone de una descripción teórica ligada a su aplicación práctica, así como los motivos de elección de dicha técnica en el desarrollo.

Dada la utilización de dos corpus diferentes en los experimentos, las distintas secciones podrán estar divididas en la aplicación del procedimiento para el corpus de Reuters, dataset textual básico, y su posterior extensión para los documentos procedentes del proceso de scraping, de mayor complejidad de interpretación para el clasificador.

3.1. Tecnologías aplicadas

La totalidad del proyecto ha sido implementada sobre Python, utilizando la Suite de código abierto Anaconda, dado que presenta numerosas herramientas relacionadas con la ciencia de datos ya integradas. Dentro de las posibilidades que ofrece esta distribución, destaca Spyder, como IDE (Entorno de desarrollo integrado). Sin embargo, se ha considerado preferente el uso de Jupyter Notebook para la elaboración del código, al disponer de una mayor claridad y visualización de los resultados en cada proceso de compilación, proporcionando la capacidad de aunar código, resultados y texto en un mismo documento. A cambio de estas ventajas, la programación en ciertos puntos debe adaptarse a estos mecanismos, forzando a utilizar sentencias especiales.

Como se ha comentado anteriormente, la investigación y aplicación de diferentes tecnologías que puedan afectar al análisis de textos era de principal relevancia en la elaboración del proyecto. Por eso se ha considerado interesante desarrollar una sección en la que se contextualice el uso de estas, así como incluir una breve descripción de su funcionalidad y trascendencia en el sector.

3.1.1. NLTK y Gensim

El desarrollo de la librería NLTK (Natural Language Toolkit) [36] para el procesado de contenido textual se ha erguido como la opción por excelencia. Es la herramienta

utilizada en gran parte de los experimentos citados, destacando en el informe [37] desarrollado por “Temple University” su aplicación para la medida de la similitud entre textos haciendo uso de las técnicas que se desarrollarán también en este proyecto, siendo adaptadas a las necesidades propias de este contexto [38].

Gensim, librería desarrollada como código libre desde 2009, facilita el tratamiento del corpus a nivel léxico y sintáctico [39]. Son numerosas las técnicas que contiene implementadas relacionadas con la creación del vector de características como tf-idf, word2vec o doc2bow. Existe un gran número de experimentos de referencia que utilizan esta tecnología, como es el caso de [40], donde su ejercicio conjunto con NLTK permite la elaboración de software de clasificación mediante *Topic Models*.

3.1.2. Scrapy

En la fase de extracción de datos para el proyecto se buscó un software que permitiera realizar el proceso de scraping de páginas web. Entre la gran diversidad de códigos que podrían llevar a cabo esta labor destacaron dos opciones: BeautifulSoup y Scrapy.

La primera se presenta en forma de librería de código abierto para Python. Fundamenta su funcionamiento en la estructura del árbol DOM, donde sitúa los contenidos web siguiendo un orden que permite la extracción del documento de manera eficaz. La segunda opción, y finalmente seleccionada, se configura en torno a la idea de “arañas web”, funciones independientes que proporcionan soporte para filtrar la extracción por selectores css y expresiones XPath. Además, una reducción en el tiempo de extracción, y la simplicidad que proporciona para una futura extensión del proyecto hacia la implementación de nuevas prestaciones, han sido elementos decisivos en su elección.

3.1.3. Otras herramientas

Trabajar con Python de manera óptima implica hacer uso de librerías externas de código libre para llevar a cabo los desarrollos software con la mayor fluidez y escalabilidad. Entre ellas, la librería *NumPy* ha permitido establecer cálculos con arrays

y matrices de una manera eficiente e intuitiva siendo la base de la computación analítica del proyecto.

Respecto a la fase de visualización, la librería *matplotlib* ha tomado especial relevancia con el fin de representar los resultados obtenidos gráficamente. Por otro lado, en lo relativo al uso de ficheros, la librería *openpyxl* permite extraer y modificar contenido directamente de documentos .xlsx desde el mismo software, sin necesidad de trabajar con ficheros .txt intermediarios de la manera tradicional.

Gestionar los ficheros de entrada y salida ya constituidos para cada algoritmo en el desarrollo del sistema global ha sido labor de Microsoft Excel. Su adaptación a diversos formatos de textos permite un uso eficiente de una misma plataforma para distintos contenidos.

3.2. Fuentes de datos y tipología de clasificación

La primera decisión de relevancia en el proyecto fue la tipología de clasificación sobre la que se iba a llevar a cabo. Para contextualizar el concepto, existen tres tipos principales de algoritmos de clasificación según el carácter del etiquetado:

- **Clasificación multiclase o multinomial:** Su principal característica reside en que cada clase es representada por únicamente un clasificador, por lo que un elemento nunca podrá encontrarse etiquetado en dos categorías. El vector de etiquetas presentará un identificador por categoría, luego estará compuesto de tantos valores diferentes como de etiquetas se disponga. Un caso particular de esta tipología es la clasificación binaria, cuando el número de clases a analizar es 2.
- **Clasificación multietiqueta:** Las etiquetas no tienen por qué ser mutuamente excluyentes. Su diferencia se encuentra en la posibilidad que presenta cada elemento a pertenecer a múltiples clases, por lo que el vector de etiquetas es sustituido por una matriz binaria en la que cada columna hace referencia a una categoría, conteniendo únicamente como valores 1 y 0.
- **Clasificación "multioutput":** Radicalmente diferente a las anteriores opciones. Presenta la posibilidad de etiquetar cada elemento por n criterios simultáneamente, luego el modelado hará uso de identificadores para cada criterio dando lugar a una matriz con valores no binarios de dimensión: Nº elementos × Nº de criterios.

A continuación se plasma un ejemplo de cada tipología de clasificación:

$$\begin{bmatrix} 5 \\ 2 \\ 0 \\ 3 \end{bmatrix} \qquad \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 5 & 1 \\ 3 & 1 & 0 \\ 2 & 0 & 2 \\ 1 & 0 & 3 \end{bmatrix}$$

Clasificación multiclase | Clasificación multietiqueta 6 etiquetas | Clasificación multioutput 3 criterios

Con el fin de delimitar el alcance de este proyecto, conviene destacar que la tipología seleccionada presenta un carácter mutuamente excluyente entre etiquetas, por lo que la tipología multiclase ha sido elegida modelando los datos como se verá en las siguientes secciones.

La segunda decisión básica residió en la elección de los datasets a utilizar, sobre los cuales se iba a sustentar el hilo conductor de todo el desarrollo del mismo. Los modelos de Machine Learning necesitan un conjunto de datos consistente, base de todos los futuros experimentos y extensiones, por lo que hacer una buena elección era algo esencial. A lo largo del proyecto se distingue el uso de dos corpus claramente diferenciados a la hora de llevar a cabo los experimentos. Se consideró necesaria esta distinción al estructurar el proyecto en dos aplicaciones, mismo algoritmo, pero diferentes estructuras léxicas y sintácticas.

Por una parte, se tuvo en cuenta la aplicación directa del clasificador en textos de lenguaje natural basándose en el contenido de los mismos (lo que se realiza mediante el dataset de Reuters). Por otra, se consideró interesante la posibilidad de extender esta idea a estructuras textuales que contuvieran elementos diferentes, pero igualmente posibles de analizar, como pueden ser documentos HTML de páginas web.

Conceptualmente, esto permitió profundizar en las fases iniciales del proceso, extracción y tratamiento del texto, competencias que no aportaba el uso de un corpus ya formado.

3.2.1. Corpus Reuters

Como primera idea se utilizó un corpus ya creado disponible en la librería NLTK para centrar la atención en los resultados del clasificador. Entre las opciones disponibles se ha elegido el corpus de Reuters, que incluye 10.788 documentos de noticias en inglés con longitud variable, ya divididos en dataset de training y test.

La selección de este corpus presentaba como ventajas:

- El número elevado de documentos por el que estaba compuesto.
- La existencia de etiquetas para cada texto que identifican su categoría.
- El acceso al texto en bruto *(raw data*) sin tratamiento previo.

Sin embargo, el hecho de que cada documento pudiera pertenecer a diferentes categorías (multietiqueta) generaba complicaciones en las predicciones futuras sin ser ese el objetivo de este trabajo. Como alternativa, se planteó usar Brown, al considerar la unicidad de cada categoría, pero como contrapartida, pesaba el hecho de disponer de un número excesivamente reducido de muestras, además de un acceso restringido a los datos, dado que el corpus ya se encontraba tokenizado. Otros dataset, como Gutenberg, no contenían datos categorizados o basaban su etiquetado en análisis sentimental. Analizando cuál sería la mejor opción, se optó por los documentos de Reuters, utilización que ha permitido una extracción básica de los recursos del dataset mediante las librerías de Python anteriormente comentadas.

A la hora de implementar este diseño, fue necesario una reestructuración de las categorías. Inicialmente NLTK mantiene los documentos clasificados con 90 etiquetas. Se consideró innecesario tan exhaustiva división dado que cada categoría contenía un número desproporcionado de artículos, de manera que se trató de conglomerar estos conjuntos con la intención de tener el corpus repartido en 8 etiquetas. Cada una contendría artículos relacionados (de etiquetas pertenecientes al mismo campo semántico), que permitieran obtener resultados óptimos y representativos en las predicciones que llevara a cabo el clasificador a desarrollar. Para ello se consideró que las agrupaciones presentaran un mínimo de 500 artículos, quedando las relaciones entre las etiquetas escogidas y las procedentes de Reuters de la siguiente manera:

Material = 'alum', 'copper', 'cotton', 'gold', 'iron-steel', 'lumber', 'naphtha', 'nickel', 'palladium', 'platinum', 'reserves', 'rubber', 'silver', 'strategic-metal', 'tin', 'zinc'

Energy = 'crude', 'fuel', 'gas' 'pet-chem', 'propane'

Commerce = 'dfl', 'dlr', 'dmk', 'gnp', 'trade', 'cpi', 'cpu', 'jet', 'ship'

Nature = 'castor-oil', 'coconut-oil', 'cotton-oil', 'groundnut-oil', 'lin-oil', 'palm-oil', 'rape-oil', 'soy-oil', 'veg-oil', 'hog', 'livestock', 'l-cattle', 'barley', 'cocoa', 'copra-cake', 'coconut', 'coffee', 'corn', 'groundnut', 'meal-feed', 'oat', 'oilseed', 'orange', 'potato', 'rice', 'rye', 'sorghum', 'soybean', 'sugar', 'tea', 'wheat', 'sun-oil'

Economy: ‘'income', 'interest', 'money-supply', 'ipi', 'money-fx'

Acquisitions = 'acq'

Earnings = `earn'

De manera adicional, se configuró una categoría más **(“Others”)** donde se almacenarían los documentos restantes. Tras ello, dada la naturaleza del corpus, lógicamente existían casos en los que un mismo documento se encontraba en distintas categorías. Debido a que el proyecto está orientado a clasificación con documentos categorizados unívocamente, se estableció un criterio de prioridades, según el que los documentos serían asignados a la categoría con prioridad más alta. Mediante condiciones anidadas, aquellos que estuvieran incluidos en esta etiqueta, no volverían a ser etiquetados por posteriores análisis a pesar de pertenecer a más categorías. El orden de prioridad viene determinado de mayor a menor especificidad como queda expuesto en las etiquetas del párrafo anterior.

Tras definir las categorías finales, se guardaron los documentos pertenecientes a cada una de ellas en dos listas. El número de iteraciones realizadas para rellenarlas correspondía con el número de documentos de dicha categoría, accediendo a cada uno de ellos por su identificador en el corpus. En una de ellas fueron almacenados los datos brutos textuales de cada documento, mientras que la restante contenía los identificadores correspondientes de cada documento en el corpus, que serían utilizados posteriormente para la separación train-test.

3.2.2. Corpus de documentos HTML

Como método de extensión del algoritmo planteado, se decidió utilizar un corpus que no estuviera preprocesado de antemano, que permitiera elegir lo que interesa analizar de él y así proporcionar una mayor flexibilidad. De ahí se extraerían resultados de un conjunto de datos sobre el que no se han realizado pruebas anteriormente.

Partiendo de esta premisa, se planteó el problema de clasificación aplicado a páginas web, las cuales utilizan lenguaje HTML, provocando variaciones en los resultados respecto a textos de lenguaje natural. Esto implicaba el estudio de mecanismos de extracción, para lo que fue necesario fijar una fuente de datos de donde poder generar el corpus.

Para conformar el nuevo dataset se planteó observar ejecuciones en un motor de búsqueda. Sin embargo, la proporción de webs de cada categoría estaría sesgada por los resultados obtenidos para dicha categoría, y no reflejarían su proporción real de datos. Además, se dispondría de un dataset donde los ejemplos de cada etiqueta son muy extremos: webs de contenido más ambiguo o difícil de clasificar, incluso por una persona, posiblemente no aparecieran en los resultados de búsqueda y, por tanto, no se incorporarían al dataset final.

La entidad “Red.es” presenta documentos .pdf organizados por meses donde se muestran las altas de dominios .es para ese mes en concreto. Tras un análisis de las prestaciones de dichos documentos, se consideró buena opción utilizarlos para la elaboración de este trabajo. [41]

De ellos, y teniendo en cuenta que el algoritmo de clasificación se basará en predicciones de un segmento del mismo dataset, se debía restringir el número de datos en función del posterior etiquetado. Por ello, se limitó la selección a dominios dados de alta entre 2012 y 2014 en base a dos criterios:

- Dominios de meses más cercanos a la fecha actual han sido descartados al incluir una gran mayoría de dominios vacíos, es decir, dominios que han sido comprados, pero a los que todavía no se les ha dado un uso activo.
- Dominios anteriores a 2012 no presentan estudio estadístico de categorías por parte del Ministerio, por lo que podría darse el caso que las etiquetas utilizadas no fueran

verdaderamente representativas al existir cambios socioculturales en las fechas, que podrían afectar a la investigación.

Con estas limitaciones, y dado que lo que se busca en el proyecto es obtener un porcentaje de precisión en la clasificación, se ha optado por seleccionar una muestra de aproximadamente 10.000 dominios, los cuales han sido extraídos del documento correspondiente a septiembre de 2013. En los próximos apartados se explicarán los procesos de creación de araña web y etiquetado relativos a este corpus.

3.3. Captura y etiquetado de datos web

3.3.1. Implementación de la araña web

La utilización de web scraping se fundamenta en la práctica de recolección de datos a través de técnicas de programación que solicitan al servidor web la información de las páginas, por lo general HTML, para posteriormente almacenar el extracto requerido por el usuario, tal y como se expresa en [42].

Para la creación de este corpus fue necesaria la previa extracción, mediante el paquete de herramientas Scrapy, de los documentos HTML y haber concluido el proceso de etiquetado. Partiendo de este punto, el proceso de creación en Python estuvo basado en el uso de una estructura de ficheros que contuviera los resultados obtenidos. A modo de resumen, se enumeran a continuación los pasos seguidos en el procesado de los datos desde su obtención hasta el etiquetado.

Procedimiento PDF → Etiquetado:

1- Se descargaron los documentos .pdf que contienen la enumeración de dominios .es con los que se va a trabajar.
2- Fue necesaria la conversión a documento .xlsx con el conversor online pdftoexcel.com para la posterior edición del mismo.
3- Como técnica de data cleansing, se necesitó la eliminación de celdas vacías ocasionadas por posibles errores del conversor en el documento generado, así como fijar cada URL con el formato: http://www._NombreDominio_.es para su reconocimiento adecuado a su paso por Scrapy.
4- Las direcciones resultantes fueron llevadas a un documento .txt para su inclusión en la araña web.
5- Durante el proceso de Scrapy, se generó un fichero con los dominios con contenido útil a analizar (htmls no vacíos)
6- Se cargó ese fichero en Excel para su conversión a .csv, formato necesario para su inclusión en el Web Labeler.

Los procedimientos que se siguieron para la construcción completa de la araña web desde su base teórica quedan descritos en los siguientes párrafos.

En el proceso de diseño de la araña se decidió no incluir recurrencia en el algoritmo, es decir, la petición sólo busca obtener el contenido del documento solicitado, sin prestar atención a los enlaces existentes en la propia página, generalmente reconocibles por el elemento "href" del lenguaje HTML. Con esta restricción, el código fue implementado dentro de una clase en Python, que incluyó la sobreescritura de los métodos: `start_requests()`, que proporciona a la araña las urls de las que se debe extraer su contenido, y `parse()`, donde se indican los criterios de extracción.

En primer lugar, el método `start_requests()` dispone del fichero .txt con el listado de urls a analizar, obtenido a partir del .pdf descargado como documento inicial de esta fase del proyecto. A través de las funciones básicas de Python para la persistencia de ficheros, este es abierto en modo lectura, y su información es almacenada línea por línea en una lista que pasa a estar compuesta por un dominio en cada posición. Una vez configurada la estructura en Python, se pasa a solicitar una petición GET, propia del protocolo HTTP, con la finalidad de obtener el documento HTML asociado en línea. Esta

operación es realizada por cualquier navegador al encontrar resultados de búsqueda, y por ello necesita de una librería especial para su uso en Python.

En este punto, las funciones propias de Scrapy son esenciales. Mediante su función `Request()` permite reproducir peticiones que se plantearían en el protocolo original. Esta función se debe realizar n iteraciones, siendo n el número de dominios que compongan el documento y debe incluir una función de callback (necesaria para establecer la recursión en el algoritmo) que en este caso será la utilizada para parsear el contenido, es decir el método `parse()`.

Dada la necesidad de establecer un elemento de asociación entre contenido HTML y su correspondiente etiqueta, se optó por establecer la url para llevar a cabo esta función. Esto implicó tener en cuenta un factor no contemplado anteriormente, la redirección automática de las páginas web. Un gran número de dominios son simples enlaces que redirigen al usuario al contenido presente en otra url, lo que ocasiona un problema para los métodos de extracción, dado que la respuesta a la petición GET reporta la url definitiva que es mostrada en el navegador. Esto ocurre una vez el protocolo HTTP ha realizado los procedimientos oportunos para la gestión de nombres de dominio mediante DNS. Para evitar esta situación, se consideró la opción de enviar un metadato en la petición, que muestre información sobre la propia página web conteniendo el elemento necesitado. Por ello, se incluyó en el metadato "original_request" la url con la que se inicia la consulta para después acceder a este campo en la respuesta.

Una vez enviada la petición, se procede a la definición de su función de callback, la cual proporciona la respuesta con el contenido HTML. Como método de almacenamiento de los documentos posteriormente analizados con el clasificador, se ha optado por la creación de un documento .html por cada respuesta generada. Este documento recibe el nombre de la url asociada a su contenido, que se obtiene mediante el acceso al metadato enviado en la petición, lo que permite mantener la estructura ligada al etiquetado.

El contenido a extraer puede estar filtrado por elementos de estilo CSS, o determinadas secciones del documento, determinadas por caracteres propios del lenguaje HTML. Considerando la finalidad de la extracción, donde cualquier información relativa al documento puede aportar claridad al resultado de la clasificación, se decidió tomar muestras de urls en el navegador y acceder a las herramientas para desarrolladores. Esta

visión global del contenido de las páginas web hizo mantener la idea de que la información relevante podía encontrarse en cualquier sección de la página, con lo que el corpus finalmente se formó con toda la información de cada página en lenguaje HTML.

Con lo anteriormente explicado, la araña ya estaría completamente formada y dispuesta para su funcionamiento. Sin embargo, cabe destacar una última apreciación en su desarrollo: la existencia de enlaces erróneos en el documento inicial, es decir, links vacíos o que redireccionan a páginas no implementadas. Scrapy, en su constitución por defecto, no tiene en cuenta estos dominios al no recibir respuesta de ellos tras la petición. Es el motivo por el que, introduciendo aproximadamente 30.000 urls, la extracción sólo se genere correctamente en aproximadamente un tercio de ellas. Paralelamente al proceso expuesto, y teniendo en cuenta este último detalle, se encomendó al método `parse()` la labor de generar un fichero .txt que recogiera únicamente las urls procesadas. Su uso se verá reflejado en la siguiente sección de este documento correspondiente al proceso de etiquetado.

Finalmente, y sólo por el hecho de desarrollar el software en Jupyter Notebook, se exige la utilización de funciones extra para su correcto funcionamiento, que son `CrawlerRunner`, para ejecución multihebra y `CrawlerProcess` para ejecución secuencial. En este caso, además de las funciones útiles para la construcción de la araña web, fue requisito obligado el uso del objeto `CrawlerProcess` de la librería `scrapy.crawler`. Por tanto, para concluir la sección, este objeto requiere especificar el User Agent a utilizar (navegador compatible que actuará en nombre del usuario en la petición), para lo que se ha elegido Mozilla/4.0, considerado token de sesión por defecto para los sistemas de navegación actual [43]. Arrancado el proceso, el crawling se llevó a cabo con una extensión de aproximadamente seis horas para la generación de contenido del corpus completo.

3.3.2. Etiquetado de documentos procedentes de la araña web

A diferencia del corpus de Reuters, los datos obtenidos a través de la araña no han sido tratados previamente, lo que implica llevar a cabo el proceso de etiquetado. En este punto fue necesario tomar tres decisiones que marcarían el desarrollo del proyecto.

En primer lugar, la multiplicidad de etiquetas, lo que permitiría que un mismo documento pudiera pertenecer a distintas categorías al mismo tiempo. Existen varias propuestas de desarrollos relativos a este tema, [44] en la que se manifiesta el árbol de decisión como mecanismo óptimo. Sin embargo, finalmente, se decidió descartar este diseño al considerar que la complicación generada no era proporcional a la aportación al proyecto en los resultados obtenidos, manteniendo la tipología multiclase como en Reuters.

Por otro lado, qué etiquetas considerar era un tema fundamental. El etiquetado debe tener en cuenta categorías que puedan englobar a todo el corpus. Asimismo, cada subconjunto debe encontrarse relativamente proporcionado en el conjunto de documentos finales para un buen entrenamiento del posterior clasificador (apartado 3.4.2)

La técnica de etiquetado, establecer la categoría a la que pertenecía cada url, podía ser un proceso tedioso si se realizaba manualmente una a una, por lo que se optó por utilizar un etiquetador procedente del repositorio GitHub en abierto [45]. Implementado en Python y ejecutado desde la consola de Anaconda, dispone de una interfaz básica que permite seleccionar la categoría correspondiente de la url analizada en ese momento, quedando registrada en un documento generado a modo de historial. Esto permite una mayor eficiencia en el proceso reduciendo el tiempo empleado en esta fase.

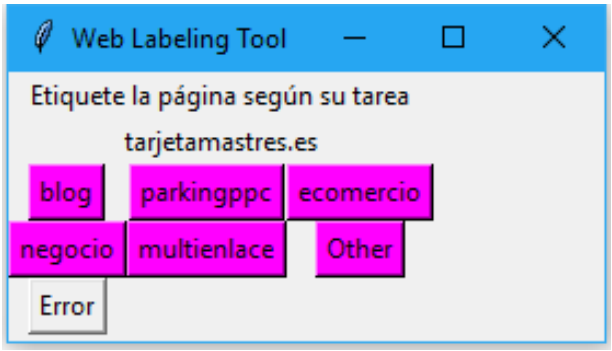


Fig. 3.1. Interfaz del software etiquetador

3.3.3. Procedimiento de uso del Web Labeler

El funcionamiento del software de etiquetado consiste en el análisis de un documento con las urls que el usuario se dispone a etiquetar. Los parámetros que determinan su funcionamiento vienen determinados en un fichero de configuración, que

indica el número de iteraciones a realizar, la estructura de carpetas a seguir y las etiquetas a utilizar, entre otras posibles opciones.

Para el desarrollo de este proyecto, el fichero quedó configurado de manera que existieran seis posibles etiquetas y una adicional para casos de error. Además, se ha tenido en cuenta el caso de etiquetar múltiples veces una misma página, lo cual reduciría la eficiencia del software. Por ello, la probabilidad de reetiquetado se ha fijado a 0 para evitar la existencia de este caso. El resto de las prestaciones pertenecientes a ActiveLearning no se han considerado.

Una vez configurado el Labeler, fue necesaria la estructuración en árbol de las carpetas donde residían los ficheros que forman parte del software. Como se describe en el documento README del mismo proyecto, el documento de análisis debe situarse en una carpeta determinada con formato .csv incluyendo las urls útiles obtenidas tras el proceso de extracción.

En este momento, la ejecución del código terminó con el etiquetado de aproximadamente 1300 páginas. Estas etiquetas quedan almacenadas en un documento historial que permite asociar el nombre de la página web con su correspondiente etiqueta, lo que sirve como unión entre ella y su contenido obtenido por Scrapy.

Para llevar a cabo esta asociación al implementar el código, se optó por generar dos diccionarios, cuya clave fuera la url en ambos casos, y cuyo contenido fuera en un caso la etiqueta y en otro el contenido HTML, lo que facilitaría el acceso al dato solicitado. En la implementación en Python, se encontró productivo encontrar una forma de acceder directamente desde el código al documento en Excel, para lo que se hizo uso de la librería *openpyxl*. Esto permitió, a través de n iteraciones, asignar a la posición del diccionario correspondiente a cada url, el valor de la celda que contuviera su etiqueta.

Este diccionario contenía etiquetas como elementos textuales, difíciles de gestionar en el desarrollo del código e imposibles de tratar en los algoritmos de aprendizaje automático donde se trabaja con matrices numéricas. Por ello se decidió aplicar la técnica *one-hot-encoding* descrita en posteriores apartados.

3.3.4. Criterios para el etiquetado

En la selección de las etiquetas a utilizar se ha tomado como referencia de base la clasificación establecida por el Ministerio para los dominios .es en sus informes estadísticos [41]. Con la finalidad de formalizar los criterios de asignación de las etiquetas seleccionadas a cada página, se detallan sus características a continuación:

- **Blog**: Se definen como páginas de particulares, organizaciones o pequeños negocios, donde los propietarios introducen regularmente entradas, ya sea con contenido textual o multimedia, que admiten la inclusión de comentarios por parte del usuario lector. Por otra parte, no todas las páginas presentan una diferenciación tan definida. Por ello, se ha decidido incluir en esta categoría páginas con características similares, como son todas las creaciones procedentes de WordPress, perfiles de Facebook registrados como webs, así como dominios donde el contenido predominante es textual informativo (como C.V.) que no se presenten como negocio ni encajen en el resto de las categorías.

- **Negocio**: Basa su criterio en páginas de profesionales que oferten sus productos o servicios, así como colegios y otras entidades públicas. Las empresas incluidas no disponen de comercio online, ya sea porque oferten servicios o porque sea una mera reproducción de sus productos disponibles en tienda física. Este tipo de negocios se incorporarán a la siguiente categoría.

- **eCommerce**: Incluye páginas de negocios con acceso a la venta de sus productos mediante comercio electrónico. Considerando que comparten similitudes entre sí, se han incluido también agencias de viajes que dispongan de venta de sus productos.

- **Parking / Pay per click**: Viene determinado por páginas que contengan pago por clic en los anuncios o páginas de estacionamiento, donde el nombre de dominio ha sido adquirido para un futuro uso activo de la página.

Además, visto que en el informe oficial estatal existe una alta cantidad de dominios en la categoría: “Dominios con múltiples páginas”, se ha decidido utilizar dos categorías extra con el siguiente criterio:

- **Multienlace**: Recoge dominios, que no pertenezcan a negocios, donde predominen los enlaces a otras secciones de su propia página. Entre ellos destacarían portales de revistas o periódicos, y páginas de carácter informativo que no dispongan de información útil en la misma página, sino que sea necesario acceder a ella a través de links. También se incluyen páginas que oferten una mera reproducción de contenidos audiovisuales, porfolios de artistas donde exponen sus creaciones o calendarios interactivos. En general, aquellas cuyo HTML a pesar de tener contenido, no tiene suficiente información para analizar, sin acceder a enlaces adicionales.

- **Otros:** Se destinan a esta categoría las páginas restantes sin posibilidad de clasificarse en las anteriores categorías. Se ha intentado que el criterio principal fuera la falta de información en la página. Bajo este criterio se incluyen simples formularios de acceso o webs donde sólo oferte elección de idioma. Al no diseñar la araña para que analice los dominios pertenecientes a los links de las páginas, se considera que en estos casos no va a ser posible detectar la verdadera página principal después del menú inicial. Páginas en idiomas distintos a español e inglés también han sido ubicadas en esta categoría basando la elección en la falta de información para la futura clasificación.

El resto de las etiquetas utilizadas por el Ministerio han sido descartadas por las siguientes razones:

- **'Redirigen a otro indicativo'**: En el diseño de la araña web, se fuerza al algoritmo de Scrapy a que devuelva el dominio inicial a pesar de las múltiples redirecciones que realice el servidor internamente, por lo que en el estudio será imposible detectar cuándo la página elegida está realizando redirección a otro dominio. Esta restricción se debe a que se ha considerado que el clasificador está basado en la determinación del tipo de texto, aplicado a documentos HTML, por lo que esta información no tendría relevancia en el análisis.

- **'Dominios bloqueados'**: Estos dominios no permiten su análisis mediante herramientas de rastreo, por lo que al utilizar Scrapy, cuando se termina la ejecución de la araña web, no son convertidos en documentos .html, y por tanto no forman parte del dataset generado.

- **‘Inmobiliarias’**: El análisis de dominios de webs inmobiliarias comenzó en marzo de 2013, mostrando una proporción de dominios hasta la fecha inferior al 0.1% del total. Teniendo en cuenta la mínima representación de páginas de esta categoría y el hecho de que este estudio utilizará un único mes de referencia, es de suponer que no se podrán encontrar muestras significativas ocasionando problemas en la clasificación. Es por ello que ha sido eliminada del etiquetado. Los resultados relacionados con el sector inmobiliario han sido etiquetados como negocio al considerar que su estructura y contenido será similar.

Cabe destacar que los criterios utilizados en este trabajo para cada una de las categorías pueden ser distintos a los utilizados por el organismo oficial. Por ejemplo “Dominios con múltiples páginas”, es para el ministerio: un dominio que resuelve, que tiene un contenido, pero que no entra en ninguna de las otras categorías, mientras que el alumno lo considera como se describe en los criterios superiores.

Para un etiquetado de mayor precisión, en los dominios no claramente definidos, se ha optado por inspeccionar las herramientas para desarrolladores del navegador que daban acceso al contenido HTML, permitiendo descubrir si realmente el documento en cuestión tenía suficiente contenido para delimitar su pertenencia a una u otra etiqueta.

De manera adicional, se ha añadido una categoría de **“Error”.** Su existencia se fundamenta en los fallos que presenta Scrapy en determinadas páginas al incluir su contenido como válido, cuando, sin embargo, dicho contenido es “Error en el dominio”. De esta manera, incluye documentos que contienen enlaces a páginas no activas actualmente, o que están formados únicamente por comentarios HTML sin visualización en el navegador, entre otras razones.

3.4. Análisis de los corpus

En los algoritmos de clasificación y predicción, los datos brutos (raw data) necesitan experimentar un proceso de adaptación para que sean compatibles con el software de entrenamiento que se desarrolla, y permitir así extraer de ellos la información de mayor valor posible, dicho de otra manera, un mapeo de los documentos para compactar su contenido.

Clasificar textos, como es lógico, tiene su principal sustento en el valor de la palabra, ya sea su significado en sí mismo, como la situación de la palabra en la oración. Es necesario dominar el lenguaje natural para poder entender el procesamiento de textos mediante la lingüística computacional.

Tras la creación de la matriz de etiquetado y el vector de contenidos, se consideró necesario procesarlos a través de distintas técnicas de "data cleansing", implementadas sobre Python. El objetivo es eliminar información que resulte irrelevante, así como conseguir la máxima claridad semántica en la colección de documentos.

Tecnologías como Scikit-Learn ofrecen funciones que reducen la complejidad de este proceso como "CountVectorizer", que, introduciendo el corpus de tokens, lleva a cabo un proceso de limpieza de datos. Sin embargo, en el software implementado se ha decidido hacer uso de la librería NLTK (Natural Language Toolkit), que permite acceder a estas técnicas a bajo nivel, obteniendo un mayor control en las capas más bajas, considerado necesario para la profundización de este proyecto. Las técnicas implementadas se indican a continuación en orden de desarrollo.

3.4.1. One-Hot-Encoding

Con el fin de estructurar las palabras (tokens) almacenadas en los diccionarios, se hizo uso de la técnica: one-hot-encoding.

La efectividad de este método reside en la asociación de valores binarios al conjunto de datos. En este caso, las etiquetas quedarían codificadas con valores 0 y 1, de manera que el documento pertenece a dicha categoría y no a las demás. La implementación de esta técnica se realizó de forma manual sin hacer uso de librerías con las funciones básicas existentes en Python.

En el caso de los documentos web, ante la posibilidad de que existan elementos que no hayan sido etiquetados, el diccionario de contenidos presentará más entradas que etiquetas se hayan proporcionado. Así, se decidió estructurar los datos en dos listas de Python, que, accediendo a los valores de los diccionarios, iban incorporando etiqueta y contenido en listas separadas, pero con posiciones niveladas. Al igual que los diccionarios son beneficiosos en el acceso al dato, las listas permiten una mayor flexibilidad de acceso y modificación, una vez conocida la posición del mismo. A su vez, permiten una

conversión directa a numpy array, estructura de dato utilizada en los algoritmos de aprendizaje automático a los que está enfocado este trabajo.

La técnica de one-hot-encoding requería una previa asociación de las etiquetas textuales a valores numéricos (valores entre 0 y el número total de etiquetas menos 1). Las etiquetas corresponderán ahora a valores entre 0 y 6, posteriormente utilizados como índice de la columna representativa de esa etiqueta en la matriz de datos. La asociación se realizó de la siguiente manera:

'blog' = 0 – 'parkingppc' = 1 – 'ecomercio' = 2 – 'negocio' = 3 –

'multienlace' = 4 – 'otros' = 5 – 'error' = 6

Finalmente, la técnica concluyó su elaboración generando una matriz de ceros que incluía el valor 1 en la posición correspondiente a la etiqueta asociada al documento. Como se ve en el ejemplo inferior, el documento con identificador 2 (fila 2), correspondería a la categoría 'parkingppc' que se sitúa en la columna 1 (Python indexa desde 0):

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}_{Nº\ de\ documentos \times Nº\ de\ etiquetas}$$

A pesar de que la explicación se haya llevado a cabo con referencias al corpus de documentos HTML, el corpus de Reuters recibió un tratamiento equivalente, almacenando en una lista las etiquetas numeradas y estructurándolas posteriormente en una matriz binaria. La única diferencia en este caso recae en la asociación, realizada de la siguiente manera:

'economy' = 0 – 'nature' = 1 – 'material' = 2 – 'energy' = 3 –

'commerce' = 4 – 'earnings' = 5 – 'acquisitions' = 6 – 'others' = 7

3.4.2. Tokenización

El procedimiento de tokenización divide en tokens la estructura textual introducida con el fin de poder analizar unidades semánticas independientes de cada texto por separado y establecer las relaciones necesarias. Técnicamente, se recorrió la lista que

actuaba como repositorio de los documentos, y se aplicó la función `word_tokenize()` del módulo `nltk.tokenize`. El resultado de cada iteración fue almacenado en una estructura matriz, donde cada fila contenía una lista con los tokens correspondientes a cada documento, siendo cada columna de esta el token correspondiente con el que posteriormente se va a trabajar.

3.4.3. Filtrado

En el corpus tokenizado existen numerosos tokens que corresponden a signos de puntuación, o palabras sin contenido semántico útil para analizar. Filtrar los tokens obtenidos forma parte del proceso de homogenización de los datos, en el que, como primer punto, se ha optado por convertir todo a minúscula, de manera que el sistema a desarrollar no cuente con prestación case-sensitive, a la vez que se limita el diccionario únicamente a caracteres alfanuméricos, para lo que se utilizaron las funciones del paquete estándar de Python `lower()` e `isalnum()`.

En un corpus puramente de lenguaje natural como Reuters, el filtrado es conveniente, pero tiene un efecto limitado, sin embargo, en el corpus de páginas web, es de suma importancia su utilización al eliminar caracteres propios del lenguaje HTML que aportarían información errónea al algoritmo clasificador según se plantea en este proyecto, causando una posible distorsión de los resultados.

3.4.4. Stemming y lematización

Hasta ahora, se ha tenido en cuenta el corpus considerando la palabra como ente individual, es ahora cuando pasa a ser caracterizada a nivel morfológico y es necesario considerar el hecho de que una palabra presenta diferentes formas gramaticales. En el caso de un verbo, este está caracterizado por su lexema, mientras que modifica su estructura siendo infinitivo, gerundio, participio, o cualquier conjugación personal. Lo mismo ocurre con las características de número y género o derivación de palabras. Para evitar este conflicto a la hora de analizar el corpus, realizando iteraciones innecesarias, se utilizaron las técnicas expuestas en este apartado, cada una con matices diferentes.

En el caso de la técnica de “Stemming”, se delimita una palabra por su lexema, es decir, la derivación queda literalmente eliminada, dando lugar a un corpus formado

únicamente por la raíz de cada palabra. El inconveniente sale principalmente a la luz en casos como verbos irregulares o palabras derivadas, cuyo prefijo o sufijo afecta al lexema, de manera que el corte de la palabra genera un nuevo elemento sin información relevante, perdiendo parte de su valor. En su implementación se utilizó la función `SnowballStemmer` del módulo `nltk.stem`, que recibe como parámetro los idiomas en los que se desarrolla el corpus.

Por otro lado, se encuentra la técnica de lematización. A cambio de necesitar un mayor uso de los recursos del sistema, presenta un grado superior de veracidad de los resultados al realizar internamente un análisis morfológico para extraer el lexema de cada palabra. A pesar de que por defecto, todas las palabras son consideradas sustantivos, ofrece la posibilidad de mejorar sus prestaciones identificando la categoría gramatical de la palabra a analizar, de manera que se obtendría el lexema correcto a pesar de la irregularidad que pueda presentar. Su implementación se llevó a cabo mediante `WordNetLemmatizer` del módulo `nltk.stem`, con la limitación establecida de que sólo tiene en cuenta palabras en inglés, obligando al desarrollador a crear un nuevo lematizador si se desea aplicar a otros idiomas.

Finalmente, teniendo en cuenta que el tiempo de ejecución del desarrollo software no es limitación en este proyecto, y que la formación del corpus con palabras reales facilitaría el desarrollo de las siguientes etapas, se ha optado por la lematización como técnica de homogenización morfológica al considerar que ofrece mejores prestaciones para el corpus de Reuters. Sin embargo, la imposibilidad de utilizar el español ha derivado en el uso del stemming como técnica para el corpus HTML, al igual que se realiza en [34].

3.4.5. Stopwords

La última técnica utilizada en esta sección se encarga de eliminar del corpus aquellas palabras no representativas para el análisis de textos. Las denominadas stopwords son palabras como artículos, pronombres o preposiciones que aportan un valor vacío al texto sin formar parte de la diferenciación entre categorías. Este listado de palabras puede ser seleccionado manualmente o utilizar el diccionario ya preestablecido de NLTK. Este desarrollo se ha implementado con esta última opción mediante

`stopwords.words()` del módulo `nltk.corpus`, donde el idioma es aportado como parámetro, prestando atención a los resultados para inglés y español en el corpus HTML.

Cabe destacar que el corpus generado tras cada técnica es el dataset utilizado de entrada en la siguiente, de manera que las limitaciones se estructuran de manera anidada formando finalmente un corpus útil para el clasificador. En este punto, se dispone de un dataset formado por los artículos o documentos HTML, en el que cada uno dispone de una fila en la matriz de datos generada, compuesta por una colección de tokens, ya procesados.

3.4.6. Vectorización e implementación del BoW

Tras las técnicas de Data Cleansing, y para favorecer la estructura de los datos de entrada al clasificador, se procede a vectorizar los datos para un mejor tratamiento de estos. Existen multitud de mecanismos para la creación del vector de características, como se denomina al conjunto de elementos de los que se obtiene información en el análisis. Entre estas técnicas destacan:

- **Bag of Words (BoW):** Estructura de almacenamiento donde cada documento es descompuesto y considerado una lista de palabras a modo de diccionario. De esta manera la importancia reside en la frecuencia de las palabras por documento. En el caso de analizar imágenes se denomina BoF (*Bag of Features*), donde se tienen en cuenta los píxeles como característica fundamental, mientras que para el análisis lingüístico se procede a generar BoWs (*Bag of Words*), donde se hace lo propio con tokens escritos.
- **N-gramas:** Toma de referencia el concepto de bolsa de palabras, sólo que en este caso no se consideran entes individuales, si no que junto a cada palabra se selecciona un conjunto de N elementos por los que está acompañada. Se articula como una forma de incorporar sutilmente el contexto de cada token en el análisis.
- **Word Embedding:** Basándose en el método clásico SVM, presenta como objetivo vectorizar los tokens en un plano n-dimensional. La diferencia radica en que en este caso sí se tiene en cuenta la información contextual, organizando una estructura gráfica donde las palabras más similares se encuentren más cercanas entre sí. Esto permite

trabajar con unos datos a nivel semántico y sintáctico. A modo de ejemplo, a continuación se muestra un gráfico generado a partir de los datos del conjunto " Large Text Compression Benchmark" con artículos de Wikipedia, utilizado para la resolución del curso de TensorFlow previo a la implementación de este proyecto [46].

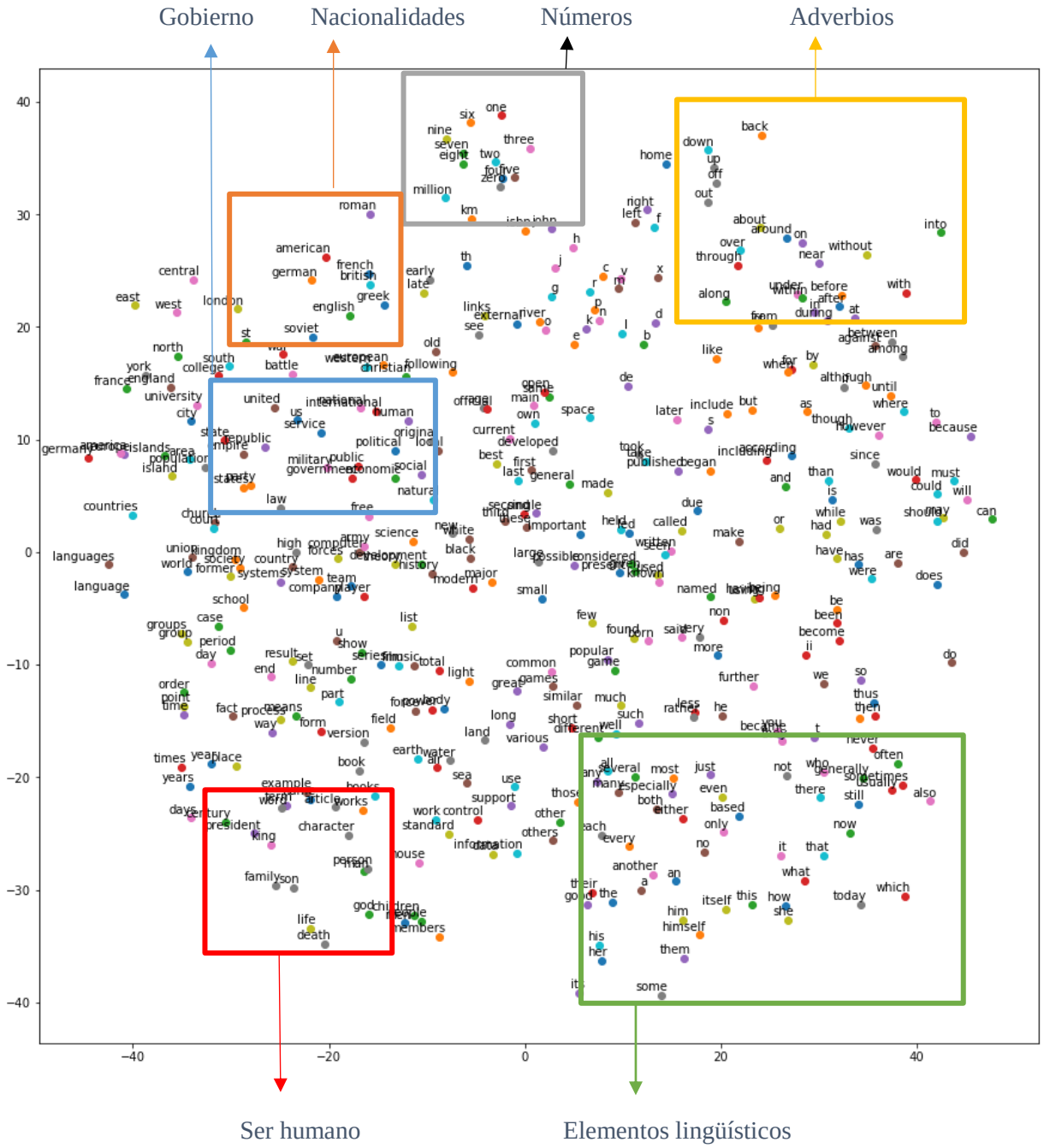


Fig. 3.2. Ejemplo de modelado word embedding generado para un conjunto de artículos de Wikipedia

Una vez expuestas las diversas opciones a tener en cuenta para desarrollos de Deep Learning, en este trabajo se ha optado por evaluar las prestaciones de la primera de ellas (BoW), con todas las variantes que esta puede propiciar en la implementación de los clasificadores.

Basándose en esta técnica, se ha desarrollado un corpus de tuplas, modelo extraído de [47], donde cada una de ellas está formada por: [identificador del token, número de veces que se repite en cada artículo]. Esta metodología consiste en generar bolsas de tokens, donde se encuentren asociados a su aparición en el texto, lo que permite tener una visión global en la estructura del dataset, como se ejemplifica en la imagen siguiente:

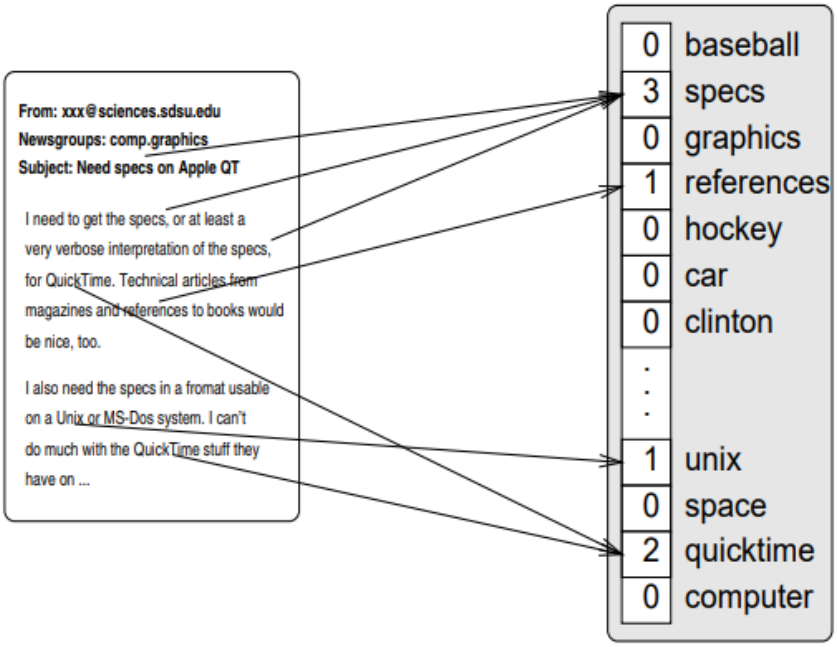


Fig. 3.3. Ejemplo de modelado Bag of Words [14]

El uso de BoWs aporta limitaciones en su estudio originadas por el hecho de ignorar la estructura sintáctica de la oración [48]:

- Expresiones o palabras compuestas que denoten un significado único pierden su concepto al separar las palabras como entes independientes.
- No establece relaciones entre sinónimos como pueden llegar a generar el modelado como word2vec.

- No se tiene en cuenta la multiplicidad de acepciones en palabras polisémicas, lo que indica que no se refleja el hecho de que una palabra adquiera connotaciones distintas según el texto que se analice.

De esta manera, la semántica pasa a ser ignorada en favor de poner el foco de atención a la multiplicidad de cada palabra, su frecuencia en el texto. La decisión de esta técnica estuvo motivada por la importancia asignada al análisis léxico, dejando en segundo lugar las relaciones semánticas entre palabras del texto.

3.5. Evaluación del BoW y sus prestaciones

Ante la necesidad de dar forma a estructuras de datos que permitan formar tuplas de manera eficiente, se optó por implementar la solución mediante herramientas de la librería `gensim.corpora`, que facilita el uso de diccionarios aplicados al diseño de algoritmos de aprendizaje automático para colecciones de texto.

En primer lugar, se transformó el corpus generado tras la eliminación de las stopwords en una representación numérica que proporcionaba un identificador a cada token existente en todas las unidades textuales del dataset, siendo recogidos en un diccionario. La decisión de realizar el desarrollo sobre esta estructura, en detrimento de utilizar una lista básica del lenguaje, residió en la facilidad que aportan estos diccionarios de Python en el acceso al dato evitando automáticamente la repetición de tokens ya existentes. A diferencia de las listas, estos no se organizan por un orden de inserción, si no que disponen de claves (palabras del diccionario) que permiten acceder al contenido (equivalente a la definición en el símil). Esto permite que la búsqueda sea automática a través de estos índices sin necesidad de implementar bucles anidados que recorran la estructura a través de múltiples iteraciones.

En segundo lugar, se procedió a la creación del BoW mediante la función `doc2bow(doc)`, que genera la lista de tuplas anteriormente explicada según recorre cada uno de los documentos a tratar. Cada columna de la matriz resultante pertenece a un documento, y cada fila de la misma es una lista de longitud variable, compuesta por tantas tuplas como tokens existieran en ese documento.

Finalmente, con el objetivo de ilustrar el procedimiento completo, se procede a mostrar un fragmento de ejemplo textual atravesando por las diferentes fases del proceso explicado:

Fragmento tokenizado:

```
['PERGAMON', 'HOLDINGS', 'REDUCES', 'BPCC', 'AND', 'HOLLIS', 'STAKE
S', '&', 'lt', ';', 'Pergamon', 'Holdings', 'Ltd', '>', 'and', 'its
', 'associate', 'companies', 'said', 'that', 'they', 'had', 'sold',
'30', 'mln', 'ordinary', 'shares', 'in', 'the', 'British', 'Printin
g']
```

Fragmento filtrado:

```
['pergamon', 'holdings', 'reduces', 'bpcc', 'and', 'hollis', 'stake
s', 'lt', 'pergamon', 'holdings', 'ltd', 'and', 'its', 'associate',
'companies', 'said', 'that', 'they', 'had', 'sold', '30', 'mln', 'o
rdinary', 'shares', 'in', 'the', 'british', 'printing']
```

Fragmento stemmed:

```
['pergamon', 'hold', 'reduc', 'bpcc', 'and', 'holli', 'stake', 'lt'
, 'pergamon', 'hold', 'ltd', 'and', 'it', 'associ', 'compani', 'sai
d', 'that', 'they', 'had', 'sold', '30', 'mln', 'ordinari', 'share'
, 'in', 'the', 'british', 'print']
```

Fragmento tras lematización:

```
['pergamon', 'holding', 'reduces', 'bpcc', 'and', 'hollis', 'stake'
, 'lt', 'pergamon', 'holding', 'ltd', 'and', 'it', 'associate', 'co
mpany', 'said', 'that', 'they', 'had', 'sold', '30', 'mln', 'ordina
ry', 'share', 'in', 'the', 'british', 'printing']
```

Fragmento con stemmer sin stopwords (homogéneo):

```
['pergamon', 'hold', 'reduc', 'bpcc', 'holli', 'stake', 'lt', 'perg
amon', 'hold', 'ltd', 'associ', 'compani', 'said', 'sold', '30', 'm
ln', 'ordinari', 'share', 'british', 'print']
```

Fragmento BoW:

```
[(3, 1), (22, 1), (131, 3), (132, 1), (151, 2), (175, 1), (196, 1),
(207, 4), (208, 2), (215, 1), (262, 1), (290, 1), (292, 2), (326, 1
), (380, 1), (520, 1), (578, 1), (582, 1), (610, 1), (617, 3)]
```

En el ejemplo se aprecia cómo, mientras con stemming, palabras como ‘associate’ o ‘companies’ quedan recortadas en ‘associ’ o ‘compani’, mediante lematización se mantienen palabras completas donde se omite el plural como en ‘companies’ que pasa a ser ‘company’ o la tercera persona como en ‘shares’, que se convierte en ‘share’. Haber desarrollado los clasificadores utilizando ambas técnicas, según el corpus, aportará mayor diversidad a las conclusiones. Por otra parte, como era de esperar, tras el procesado de stopwords se han eliminado artículos como ‘the’, pronombres como ‘it’ o conjunciones como ‘and’, simples conectores que no aportan información al análisis textual.

La muestra ha sido tomada del dataset de Reuters al resultar más inteligible visualmente por estar compuesto en su totalidad de elementos de lenguaje natural, sin embargo, el corpus de documentos HTML ha sido tratado de la misma forma.

Una vez finalizada la etapa de data cleansing, y tomando este fragmento como ejemplo del corpus completo, se observa que se ha pasado de 30 tokens iniciales a 19, lo que implica una reducción necesaria para utilizar únicamente los tokens de los que se pueda extraer más información. Para concluir, cabe destacar el análisis de la bolsa de palabras de Reuters formada por 26.629 tokens si se utiliza lematización o por 21.622 si se utiliza 'Stemming'. Esto permitió la posterior conversión de cada token a forma de tupla. En el ejemplo, se indica que el token con identificador 207 ("said") aparece 4 veces en el documento en cuestión, mientras que aquel con id 578 ("total") sólo una:

Para fijar conceptos hasta el momento, y visualizar los resultados obtenidos de cada corpus, se consideró generar gráficas que dieran a conocer los avances desarrollados. Concretamente, una aproximación a los tokens más frecuentes por documento (histograma del BoW), así como la tendencia de la repetición de tokens en el corpus global.

Al implementarlo, se necesitó un tratamiento adicional de los datos obtenidos hasta el momento al buscar resultados sobre el corpus completo, y no sobre cada documento. Por ello, mediante iteraciones se rellenó una lista con todas las tuplas del corpus añadidas una detrás de otra, a diferencia de la matriz del desarrollo explicado, donde existía una lista por documento que recogía las tuplas de sus elementos. Recorriendo esta lista única se estableció un numpy array que pudiera almacenar el número de veces totales de cada token en el corpus sumando los segundos elementos de cada tupla con igual identificador (correspondiente al número de repeticiones de dicho token). De esta manera, se permitió la posterior selección de tokens ordenados en función de este indicador mediante la función `np.argsort()`. Finalmente, con funciones de diseño propias de la librería `matplotlib`, se detallan a continuación las gráficas de ambos corpus. Cabe destacar que el histograma teórico suele representarse siendo el eje 'x' el referente a los propios tokens, pero para una visualización más clara se ha considerado útil situarlos en el eje 'y':

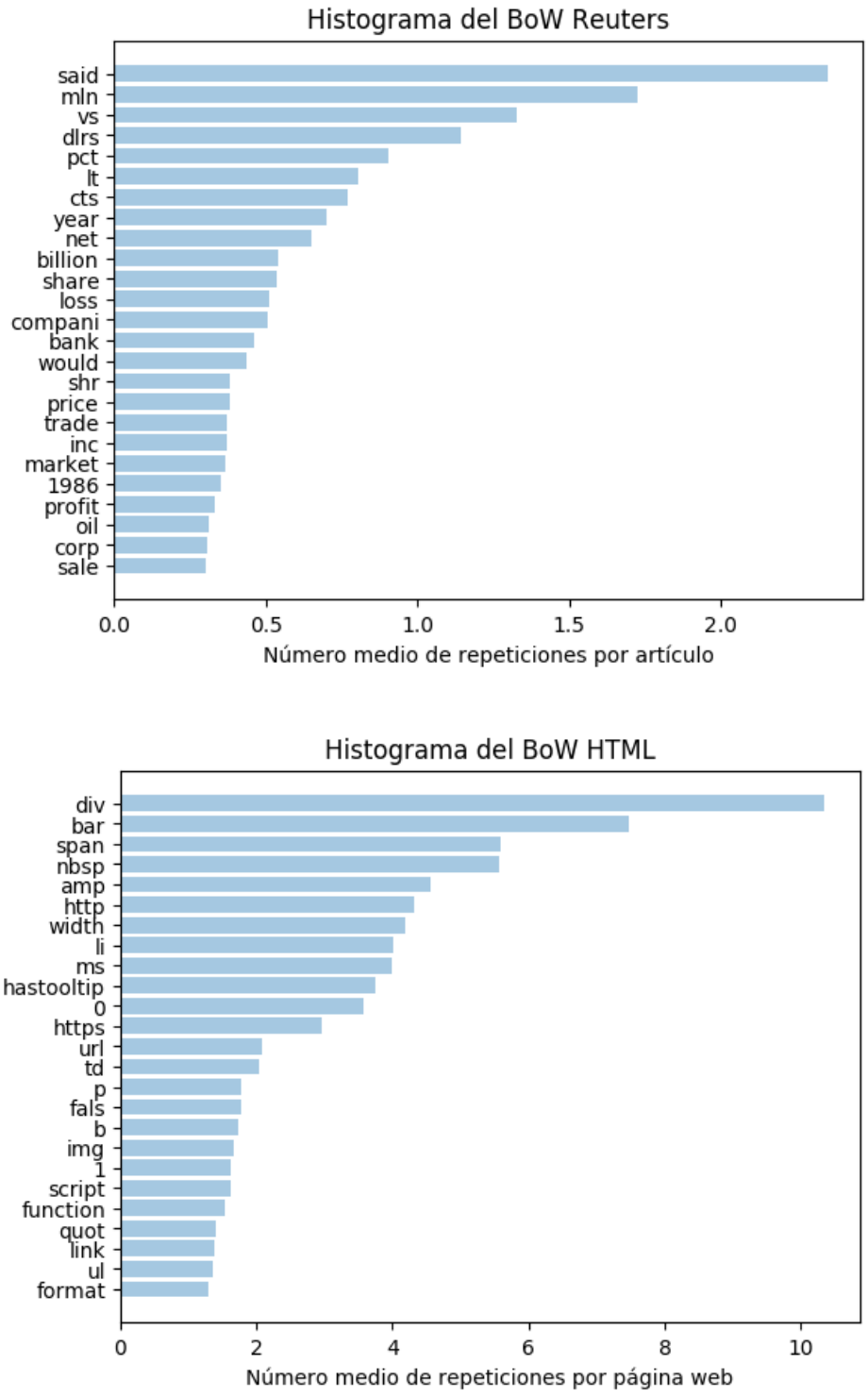


Fig. 3.4. Histograma del BoW para ambos corpus

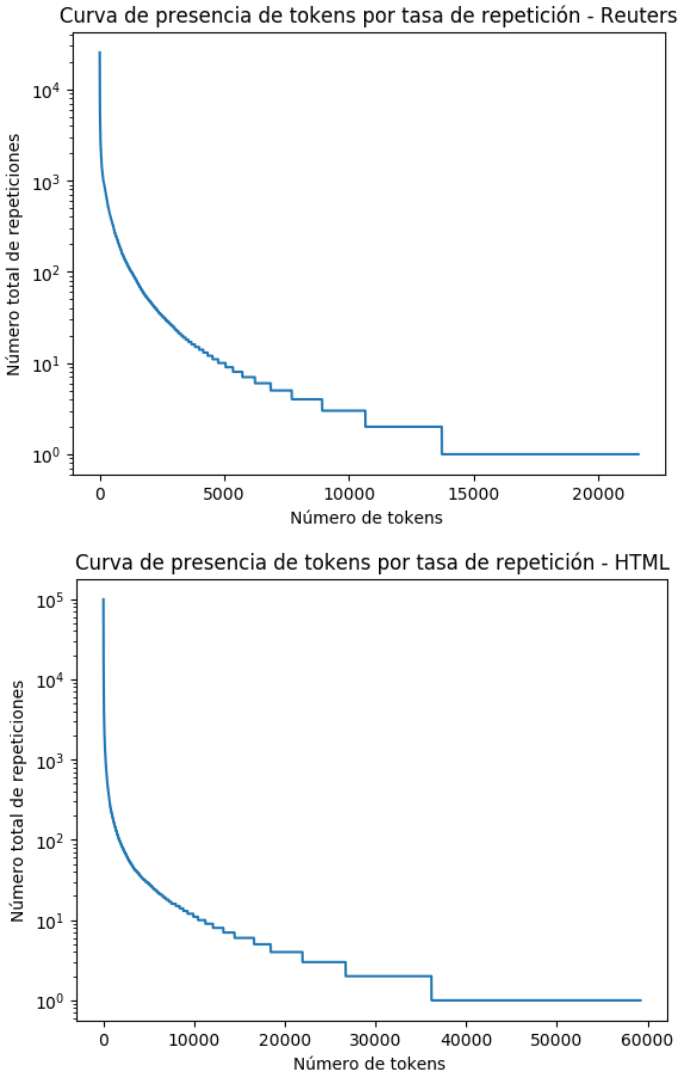


Fig. 3.5. Curvas de presencia de tokens por tasa de repetición para ambos corpus

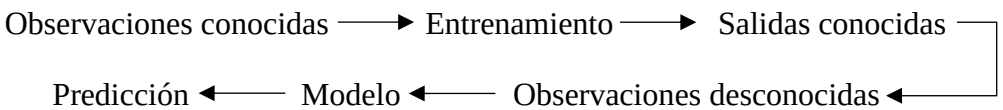
Como se ha comentado anteriormente, el histograma de cada corpus muestra los tokens más repetidos en el conjunto completo del dataset. Por ejemplo, en el corpus de artículos de Reuters destaca la aparición del token “said” que se produce en 25.381 ocasiones, por lo que se estima una media de más de dos veces por artículo.

Por otra parte, el segundo gráfico se interpreta de la siguiente manera: el valor del eje ‘x’ se corresponde con el número de tokens totales mientras que el eje ‘y’ con su porcentaje de aparición en el corpus. Como es lógico, sólo un número reducido de palabras se repite un número elevado de veces, luego a mayor número de palabras, menor será su índice de repetición en el corpus total. Esta tendencia se repite para ambos datasets como verificación de la correcta formación de estos.

4. DESARROLLO E IMPLEMENTACIÓN DE LOS CLASIFICADORES

La investigación de las principales técnicas de Machine Learning queda expuesta en esta sección, donde se ha llevado a cabo un análisis de su comportamiento en su aplicación a la clasificación de textos, así como profundizado en los matices prácticos para su implementación.

En el desarrollo de los clasificadores se ha tomado como referencia el procedimiento seguido en el curso de Deep Learning elaborado por Google [46]. En ese caso se utilizó un dataset formado por imágenes (MNIST) donde los tokens se correspondían con los píxeles por los que estaban compuestas. Partiendo de ello, se ha llevado a cabo una labor de adaptación del dataset para poder configurarlo a nivel textual y poder entrenar el clasificador con datos útiles. De igual manera, los clasificadores han sido construidos bajo la premisa de buscar una mejor eficiencia utilizando técnicas adecuadas para esta aplicación en particular, y por tanto distintas a las utilizadas en dicho curso. En cualquier clasificador, se ha seguido el siguiente esquema:



4.1. Tecnologías aplicadas

4.1.1. Scikit-Learn

El desarrollo del aprendizaje automático dio lugar en 2007 al surgimiento de esta librería, posteriormente liberada como código libre. Su creación provocó una evolución notable en las técnicas del sector al incluir funciones que facilitan la implementación de algoritmos preestablecidos de inteligencia artificial. Su razón de existencia es la amplia versatilidad que ofrece en el modelado de los datos. Sin embargo, las tareas previas de carga, manipulación y tratamiento de estos deben ser gestionadas mediante otras librerías. En [49] se destaca su funcionalidad compartida con NumPy y SciPy, esenciales en la gestión de datos en Python y su tratamiento en arrays y diccionarios.

4.1.2. TensorFlow

TensorFlow es una librería de aprendizaje automático desarrollada por Google y liberada como software de código abierto en 2015 [50]. Basa su funcionamiento en la programación catalogada como "Dataflow Programming", donde el modelado de datos en estructura de grafos es esencial para su tratamiento. Esto permite ejecutar cómputos a un nivel más bajo que otras tecnologías del mismo ámbito, proporcionando cálculos de gradientes automáticamente, su principal atractivo. De esta manera, la esencia de TensorFlow es la flexibilidad que aporta en la construcción de redes neuronales, origen de su utilización en empresas como Airbnb, Snapchat, Ebay, o Dropbox.

La decisión de su uso en la construcción de los clasificadores basa su argumento en la capacidad de despliegue de los mecanismos de Deep Learning en Python. Permite abordar las distintas características que influyen en el aprendizaje profundo a través de procedimientos visuales y de mayor eficiencia que Scikit-Learn, software de mayor extensión hasta el momento en el sector.

Haciendo alusión a lo expuesto en el capítulo 2 ("Estado de la cuestión"), la novedad de esta tecnología hace que implementaciones en Tensorflow no abunden en las investigaciones, siendo aún más limitado el ámbito del análisis textual. Los reducidos experimentos disponibles, en un alto porcentaje, hacen uso de clasificadores desarrollados en librerías ya creadas como *tflearn.models.dnn* lo que reduce la intención de este proyecto de intentar llevar a cabo el análisis del dato desde el principio. Esto ha supuesto una reestructuración del código disponible para análisis de imágenes, por lo que, a continuación, se describe la lógica fundamental que utiliza esta plataforma para entender la posterior construcción e implementación del software indicado.

Los grafos que sustentan los desarrollos en TensorFlow deben incluir el flujo de datos que se va a analizar, es decir, inputs y variables, así como las operaciones que se pretenden aplicar para obtener el resultado en la evaluación final. A través de estos grafos, TensorFlow representa computacionalmente las dependencias entre elementos, lo que evita seguir un orden secuencial como en la programación tradicional. Esto permite realizar operaciones paralelamente, como conexiones entre elementos del programa, que funcionan como "cajas negras" del software que se está desarrollando. Así, una estructura grafo contiene la información necesaria para conectar todos los componentes a analizar,

pero no determina explícitamente cómo deben realizarse estos procesos como se pudiera apreciar en un código fuente clásico.

Tras la creación del grafo, es necesario un objeto sesión que encapsule el entorno de desarrollo en el que se ejecutan las operaciones, estableciendo la conexión entre el cliente y las técnicas de programación distribuida (originarias de C++) realizadas internamente. De esta manera, se inicializan los valores de las variables para recuperar todos los parámetros del grafo, así como obtener los resultados solicitados. En las siguientes secciones se detallará el proceso seguido en la utilización de estas técnicas para la implementación de los clasificadores en cuestión.

4.2. Construcción de los dataset de entrenamiento, test y validación

Un correcto funcionamiento de cualquier software de clasificación, necesita estructurar el corpus utilizado en diversas categorías:

- **Datos de entrenamiento**: Cualquier software de aprendizaje debe tomar consciencia de lo existente para predecir lo que el usuario solicite a partir de lo experimentado. Para que este proceso concluya con éxito, es necesario entrenar al algoritmo con un amplio conjunto de datos que le permitan interpretar el entorno lo más próximo posible a la realidad.
- **Datos de test**: Una vez el software esté entrenado, se necesita de un conjunto de datos que permita poner a prueba la eficiencia de la predicción conseguida mediante el uso del algoritmo elegido. Analiza los parámetros del algoritmo utilizado en su labor por obtener la mejor predicción, y proporciona el error real cometido con el modelo seleccionado
- **Datos de validación**: Este último subconjunto permite seleccionar la mejor opción entre los modelos evaluados basándose en el reconocimiento de los hiperparámetros óptimos para el algoritmo. De su análisis se debe poder extraer la información del momento en el que el entrenamiento del dataset no pueda obtener mejores resultados de precisión sin caer en situación de overfitting.

En los siguientes apartados se indican los procedimientos realizados para establecer dichas divisiones en el caso de cada corpus. Sin embargo, con el objetivo de continuar con un mismo hilo argumental, se ha considerado necesario conocer previamente el tratamiento que recibirá la matriz de entrada de cada tipo de datos, exponiendo la lógica aplicada al conjunto.

Disponer de un vector de tuplas para el procesado de los datos presentaba deficiencias en el rendimiento al incluir información redundante. Para optimizar su funcionamiento, y como parte del proceso de adaptación de los datos al desarrollo a implementar, se diseñó como estructura de entrada a los clasificadores una matriz numérica en Python mediante la librería Numpy. El uso de esta herramienta intermediaria se fundamenta en la obligación de su empleo para el software implementado en TensorFlow.

La adaptación del corpus al formato con el que se va a trabajar se encuentra dentro de las técnicas denominadas "Data Wrangling", expuestas en [51] como proceso necesario para el previo modelado, utilizando Microsoft Excel como herramienta básica. A pesar de que en otras etapas de este proyecto se ha procesado en esta plataforma, como ya ha sido comentado, en este caso particular, se ha hecho de Python para todo lo relativo a esta sección.

Hasta el momento, la estructura existente reflejaba un documento en cada fila, asociando a cada uno la lista con sus tokens correspondientes. Mediante la extracción de los datos de dicha matriz, considerada auxiliar, se decidió generar una definitiva que estuviera *organizada de manera que las filas se correspondieran con cada uno de los tokens del corpus completo, y cada columna hacer lo propio con los documentos que lo forman*. El resultado debía validar que la posición que, por ejemplo, hiciera referencia a las coordenadas (3428, 7) incluiría el número de veces que aparece el token con identificador 3428 en el documento 7. Un ejemplo ilustrativo del resultado sería:

$$\begin{bmatrix} 0 & 0 & \cdots & 0 & 5 & 27 \\ 0 & 33 & \cdots & 0 & 37 & 62 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 57 & 1 & \cdots & 0 & 0 & 0 \\ 3 & 9 & \cdots & 0 & 0 & 83 \end{bmatrix}_{N^{\underline{o}}\text{ tokens totales} \times N^{\underline{o}}\text{ de documentos}}$$

Para llevar a cabo lo expuesto, la construcción de dicha matriz de características era común para ambos corpus. A través de un bucle anidado se fueron añadiendo a la matriz los contenidos como refleja el siguiente pseudocódigo:

```
for i in número de documentos :
  for j in número de tokens del documento i:
    matriz[i][corpus_bow[i][j][0]]= corpus_bow[i][j][1]
```

donde "matriz" es la futura matriz de datos de entrada al clasificador, y "corpus_bow", la matriz de tuplas. Por tanto, corpus_bow[i] corresponde al BoW del documento 'i', que a su vez mediante el índice 'j' accede a cada tupla del documento, que añadiéndole el índice 0 permite obtener el id del token. De igual manera, el índice 1 extrae el número de repeticiones, valor que se utilizará para rellenar finalmente la matriz correctamente indexada. Asimismo, la posición correspondiente al número de documento × id del token será rellenada con este valor.

En ambos casos, ha sido necesario realizar una reestructuración del formato para que las matrices fueran compatibles con TensorFlow, que consistió en la conversión de los dataset de entrenamiento, test y validación en objetos de la librería Numpy con formato float32.

Aplicando lo expuesto al caso que se desarrolla en este trabajo, se han utilizado dos técnicas dependiendo de las prestaciones que ofrece cada dataset. Ambas tienen en común la estructura utilizada en la gestión de datos:

Aplicación al corpus Reuters

En el caso del corpus procedente de artículos de Reuters, la descarga de los mismos ya incluye una separación entre datos de entrenamiento y test estipulada de antemano. Por ello, se decidió hacer uso de ella sin conocer los criterios que se han seguido en la proporción correspondiente a cada subconjunto.

Técnicamente, los documentos del corpus se encuentran titulados como 'test/*xxx*' y 'training/*xxx*' siendo *xxx* el código de referencia del documento. Por ello, para su separación, se implementó una función que, dado el conjunto de títulos proporcionados por NLTK, reconocía aquellos que comenzaban por 'test' o 'training' e insertaba en el array correspondiente cada uno de los resultados obtenidos. De la misma forma, para

asignar a cada título su contenido textual, se tomó de referencia la posición de cada título en la lista global, almacenando los datos en un array de posiciones para train y para test. De esta manera, se establecía una asociación entre la posición en el dataset y el título correspondiente, llevando así intrínseco el valor de su contenido, que se almacenaba en un nuevo y definitivo array como muestra el siguiente pseudocódigo:

```
for i in número de documentos train :
  train_dataset[i] = matriz[train_positions[i]]
```

siendo "matriz" la compuesta por los datos de entrada al clasificador, "train_dataset", el array que almacenará los datos de entrenamiento, y "train_positions[i]", la posición de documento analizado en ese momento ('documento i') en el corpus global. El mismo procedimiento es seguido para los datos de test y validación, previamente obtenido mediante la función `sklearn.model_selection.train_test_split`, proporcionándole un 20 % de los documentos de entrenamiento.

Aplicación al corpus de documentos HTML

En la extensión del algoritmo para los documentos HTML el caso fue diferente. Los datos no estaban preprocesados, sino que se disponía del control sobre ellos desde el nivel más bajo del tratamiento, por lo que se debía dividir el corpus siguiendo otra técnica.

Tras investigar las diferentes opciones, la separación se llevó a cabo utilizando el método `sklearn.model_selection.train_test_split`, al que se le pasa como argumento el dataset completo con sus respectivas etiquetas, así como la proporción de datos que se desea asignar al conjunto de test, y la semilla para fijar la aleatoriedad que presentará la selección. Esto hace alusión a los distintos índices de los documentos que serán escogidos para formar parte de este subconjunto, dado que de no utilizar valores aleatorios se podría producir un estudio sesgado que distorsionaría los resultados finales. Para este trabajo, la proporción de documentos test ha tomado el valor de 0.20, al igual que para los de validación, magnitud estándar para este parámetro que suele fijarse entre 0.15 y 0.3.

4.3. Métodos clásicos de clasificación

Desde el surgimiento de la lingüística computacional, la clasificación de textos ha sido automatizada mediante algoritmos de Machine Learning, actualmente considerados "algoritmos clásicos".

En este proyecto, a modo de contextualizar los futuros resultados, generados mediante redes neuronales, se ha decidido implementar dos de estos clasificadores y analizar sus prestaciones. Su elección se ha basado en el estudio del estado del arte, donde sobre las numerosas posibilidades existentes destacan dos, Naive-Bayes y SVM.

4.3.1. Clasificador Multinomial Naive-Bayes

Naive-Bayes es un algoritmo basado en el principio de máximo a posteriori (MAP) que se caracteriza por asumir la independencia de cada uno de los elementos del vector de características. Centrando el análisis en el modelo multiclase, se utiliza como información para su estudio la frecuencia de los tokens en los documentos. Esto permite calcular la posterior probabilidad condicionada que genera los resultados definitivos en la clasificación [15].

Para su desarrollo, al no hacer uso de redes neuronales, especialidad de Tensorflow, se ha utilizado la librería Scikit-Learn, que dispone de los algoritmos clásicos ya construidos preparados para su adaptación al corpus que se desee analizar. La función utilizada fue `sklearn.naive_bayes.MultinomialNB()`, la cual permite crear el objeto estimador correspondiente al algoritmo de Naive-Bayes para esta tipología. Utiliza internamente la estrategia *OneVsRest*, en la que cada clase es ajustada al modelo y analizada frente al conjunto de todas las demás. Requiere un vector de características de valores discretos, lo que se amolda a la perfección con el uso del BoW, donde los tokens almacenados son asociados a un número identificador.

Con estas condiciones, una vez creado el objeto, se debían ajustar los datos del corpus. La encargada de llevarlo a cabo fue la función `fit()` aplicada a dicho objeto, que recibe como argumentos los datos de entrenamiento y sus respectivas etiquetas. La matriz de datos debía presentar como dimensión: número de documentos × número de tokens, mientras que el vector de etiquetado debía ser el generado previamente a la

construcción de la matriz de etiquetas “one-hot-encoding”. Por tanto, este vector constaba de una lista en la que cada posición se correspondiera a un documento y su valor fuera el número identificador de la etiqueta asociada a ese elemento. La longitud esperada de dicho vector se correspondía, como era de esperar, con el número de documentos. Para su correcto funcionamiento, las etiquetas requerían una transformación a numpy array mediante `np.array(vector_de_etiquetas)`.

Entrenado el algoritmo, la predicción se lleva a cabo con el método `predict()` aplicado al objeto estimador, que debe recibir como argumento la matriz de datos test cuyas etiquetas se desea predecir.

Finalmente, la precisión del algoritmo se puede hallar indiferentemente de dos formas. La primera, haciendo uso de la función `score()` del objeto estimador que, a partir de la matriz de datos de test y su vector de etiquetas, devuelve el valor medio calculado de la precisión. La otra manera se implementaría de un modo más generalizado con la función `sklearn.metrics.accuracy_score()`, aplicable a cualquier otro algoritmo clasificador y que necesitaría el vector de etiquetas de test, así como el valor de las predicciones calculadas para establecer una comparación directa, e internamente calcular la proporción de aquellas que son correctas.

4.3.2. Clasificador SVM (Support Vector Machine)

El algoritmo SVM consiste en la representación de los datos mapeados como puntos en el espacio asociados según la categoría a la que pertenezcan. Como se define matemáticamente en [52], se construyen uno o varios hiper-planos en un espacio n-dimensional que son usados para delimitar las zonas de decisión, maximizando la mínima distancia desde el plano hasta la muestra más cercana para minimizar así el posible error generado.

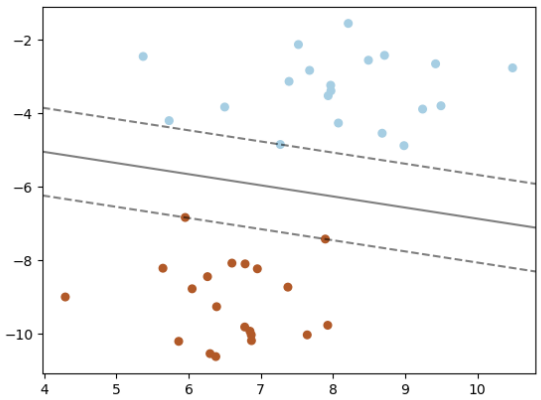


Fig. 4.1. Ejemplo de representación gráfica del clasificador SVM [49]

La implementación se puede llevar a cabo siguiendo una estrategia *OneVsOne*, que clasifica los elementos individualmente respecto a cada una de las clases existentes, o *OneVsRest* como en el caso de Naive-Bayes. Tras analizar ambas opciones, se decidió utilizar la segunda estrategia mediante un SVC lineal. El motivo de la elección ha sido determinado al considerar que *OneVsOne* presenta un alto interés en la demostración teórica pero no proporciona mejores resultados que su variante, la cual necesita de menor coste computacional. Para su implementación se ha utilizado el constructor `sklearn.svm.LinearSVC()` con argumento `multi_class = 'ovr'` para determinar la estrategia *OneVsRest* (utilizado por defecto).

A partir de este punto, el ajuste al modelo, cómputo de la predicción y medida de la precisión, se llevan a cabo siguiendo los mismos procedimientos de implementación en *sklearn* que los anteriormente descritos para el algoritmo Naive-Bayes.

4.4. Clasificación mediante regresión logística

El algoritmo más simple de predicción multiclase mediante Machine Learning es el clasificador basado en regresión logística multinomial. Su diferencia, y ventaja, respecto a la regresión lineal es su distribución en el rango [0,1] para los valores del eje x, a diferencia del rango [-∞,∞], lo cual permite operar con probabilidades correctamente. Para ello se utiliza la función logit como técnica de modelado, de manera que la ecuación resultante computada resolviendo p (**x**) tendría la siguiente forma:

$$\ln \frac{p(x)}{1-p(x)} = \mathrm{b} + \boldsymbol{w}^{\boldsymbol{T}}\mathbf{x} \qquad (4.1)$$

Partiendo de la concepción purista de la técnica, y adaptando su definición al concepto de neurona artificial, se puede considerar este algoritmo como una neurona simple, cuyas entradas son sometidas a una única transformación de carácter lineal que da lugar a una salida. En la expresión, el término "b" es llamado *bias* (sesgo), ya que controla cuán predispuesta está la neurona para determinar si su predicción toma un valor u otro independientemente de los pesos que caracterizan el resto de la expresión. Estos últimos vienen identificados por "w" en la ecuación, que marcará la pendiente de la función resultante. El procedimiento matemático completo se puede encontrar en [53].

A modo de resumen, se ha de exponer que el objetivo de este procedimiento será predecir qué valores deberán tomar los parámetros para poder ofrecer un resultado lo más apropiado posible a la nube de puntos iniciales, factor a tener en cuenta en las predicciones finales. Sin embargo, la intuición del usuario para deducir la función que más encaja con el conjunto de datos es meramente orientativa. La medida cuantitativa que permite extraer estas conclusiones de manera exacta se denomina función de pérdida (*loss function*), y establece un cálculo de los márgenes de error entre los valores originales y la predicción de los mismos.

Aplicando el concepto a la elaboración de software de aprendizaje automático, esta técnica limita las prestaciones del algoritmo que se pueden alcanzar utilizando neuronas artificiales.

4.4.1. Implementación en TensorFlow

Construir el clasificador basado en regresión logística es posible en TensorFlow considerando el diseño de neurona artificial, cuyos datos deben presentar las dimensiones expuestas en la siguiente tabla:

Tabla 4.1. ELEMENTOS DE LA ECUACIÓN DE REGRESÓN

$w^T x + b = y$ (4.2)	
w ⇒ Matriz de pesos	Número de tokens × Número de etiquetas
x ⇒ Matriz de datos de entrada	Número de tokens × Número de documentos correspondiente
b ⇒ Vector de "biases"	Número de etiquetas × 1 (mismo vector para cada etiqueta)
y ⇒ Predicciones	Número de etiquetas × Número de documentos

Para poner en práctica todo lo explicado anteriormente se ha hecho uso de dos objetos de la API de TensorFlow, `tf.Operation` (que actúa como nodo), y `tf.Tensor` (que funciona como límite). Por tanto, una vez creado el objeto grafo, su implementación ha utilizado los constructores `tf.placeholder` para los conjuntos de datos de entrenamiento, validación y test, que serán inicializados en el momento de la ejecución, con formato `tf.float32` pasado como argumento, así como la dimensión requerida. El constructor `tf.Variable` fue utilizado para los pesos y "biases" propios de la ecuación de regresión. Estas funciones crean objetos de operación que producen el cómputo que se está realizando, y a su vez devuelven el tensor que representa su valor. Cabe destacar que la matriz de pesos fue inicializada usando valores aleatorios que seguían una distribución normal truncada con el objetivo de tomar valores próximos a 0 eliminando posibles valores dispersos que no sean representativos del estudio, así como el vector de biases a cero.

Una vez declarados los elementos necesarios para el análisis, comienza el proceso de entrenamiento. Para ello, se aplicó la ecuación teórica de regresión a los tensores obtenidos mediante la función `tf.matmul()` para ejecutar el producto. Concluido este procedimiento, Tensorflow es capaz de obtener computacionalmente la operación de softmax, que fue utilizada como función de salida de la última capa neuronal, determinada por la expresión:

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}} \quad (4.3)$$

Las predicciones obtenidas de la ecuación de regresión (*logits*) presentan valores presentes en un amplio intervalo, por lo que esta función permite normalizar el vector de resultados de manera que se manifiesten en un rango de [0,1], actuando como vectores de probabilidades útiles para los clasificadores, ya que aporta la probabilidad de que cada elemento se corresponda o no con la categoría en particular. Es por ello que su uso está ampliamente extendido como capa final de software de aprendizaje automático basado en redes neuronales, junto a mecanismo de entropía cruzada, técnica que también se ha utilizado en el desarrollo. Su finalidad es analizar el error generado en la ejecución de manera que cuantifica la diferencia entre las distribuciones de probabilidad generadas por el software.

Con el objetivo de poder evaluar las prestaciones del clasificador, era necesario calcular las pérdidas obtenidas en el proceso de entrenamiento como medida gradual de la eficiencia en cada iteración. Su cálculo estuvo basado en la media aritmética de la entropía cruzada a través de todas las iteraciones de entrenamiento. Al ser este conjunto de operaciones muy utilizado en este tipo de algoritmos, TensorFlow ofrece la posibilidad de utilizar una única sentencia anidada para obtener la pérdida:

```
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits
                (labels, logits))
```

Finalmente, se utilizó un optimizador para minimizar el valor de esta pérdida, lo que se llevó a cabo mediante el método de "*gradient descent*" como primera opción, aunque un análisis de distintos optimizadores será desarrollado en la sección de "Evaluación de resultados". El código correspondiente a la técnica mencionada sería:

```
tf.train.GradientDescentOptimizer(learning_rate)
                .minimize(loss)
```

siendo learning rate un valor fijo o una función decreciente, como se expondrá en secciones posteriores.

En este punto, se debían ejecutar las operaciones implementadas. Se creó un objeto `tf.Session` y se inicializaron los valores de las variables mediante la sentencia

`tf.global_variables_initializer().run()` para recuperar todos los parámetros del grafo. A continuación, mediante un bucle con n iteraciones (*steps*), se ejecutó el método *`run(parameters)`* sobre el objeto session. En él actuaban como parámetros los valores que fuera necesarios calcular al llevar explícito un cómputo en el grafo*:* el optimizador, la pérdida minimizada y el array de predicciones de entrenamiento, incluidos todos ellos en una lista.

Para establecer el grado de precisión de los resultados, se decidió calcular el porcentaje de predicciones correctas en función de las totales. Implementar dicha función necesitaba hallar, de cada vector de resultados, la categoría a la que le correspondía la probabilidad más alta (`np.argmax`). De esta manera, se comparó el número de veces en las que dichas predicciones fueran iguales a las etiquetas de entrenamiento, y se dividió entre el número de etiquetas totales para extraer el porcentaje de precisión. La misma técnica fue seguida para los conjuntos de test y validación, donde la función `eval()`, aplicada directamente al tensor, devuelve el numpy array de predicciones, recalculando todo el grafo de dependencias.

4.5. Deep Learning y técnicas asociadas

Siguiendo la analogía con el cerebro humano, las neuronas biológicas presentan un potencial de acción que hace reaccionar a los cambios en el cerebro. Algo similar ocurre con la función de activación en la inteligencia artificial. Como se ha descrito anteriormente, una función lineal presenta grandes limitaciones, por lo que el uso de funciones no lineales se ha extendido obteniendo en un elevado porcentaje de estudios resultados favorables. Para el éxito de uso de estas funciones y la mejora de las consiguientes prestaciones finales conviene utilizar las máximas capacidades de las neuronas artificiales.

Hasta ahora, el concepto de red neuronal se utilizaba como mero elemento de tránsito de la información, pasando por funciones que modificaban el contenido y generaban una salida. Para extender las capacidades de las neuronas surge el término Deep Learning, vertiente que pretende profundizar en las características de la información de las entradas para generar una conclusión más acertada. Es el motivo por el que este proyecto presta especial atención a su diseño.

Esta evolución del concepto de regresión a neurona artificial necesitaba de la incorporación de nuevos elementos a la red, las capas ocultas (*hidden layers*), capas intermedias, que junto al procedimiento de *backpropagation* forman los fundamentos de esta nueva filosofía del análisis de los datos. Esta etapa intermedia puede constar de una o múltiples subcapas que aportarían profundidad al entrenamiento de la red neuronal. El planteamiento para elaborar este mecanismo es aplicar a estas capas ocultas funciones no lineales como función de activación de las neuronas, de manera que permitan una mejor adaptación de las neuronas al conjunto de datos analizado en la fase de entrenamiento.

4.5.1. Hidden layers

La utilidad de las capas ocultas recae en la necesidad de conectar entradas y salidas de manera que el valor de salida sea altamente interdependiente de las entradas, como ocurre en la vida real. Encadenar múltiples transformaciones no lineales a través de capas intermedias permitiría aumentar la capacidad de expresión de la red neuronal. A pesar de ello, estudios de finales del siglo XX demuestran que una red de simplemente 1 capa oculta con función de activación no lineal (como sigmoide o ReLU), y con suficientes neuronas ocultas (*hidden nodes*), debería ser capaz de conformar un *universal function approximator*, es decir, teóricamente debería ser capaz de expresar cualquier mapeo arbitrario de entrada – salida con resultados aceptables [54]. Como era de esperar, un cuarto de siglo más tarde, las tecnologías han conseguido evolucionar estos resultados y, aunque no se desmonta dicha teoría, se ha demostrado que la utilización de más de 1 capa oculta puede mejorar notablemente las prestaciones a cambio de un ajuste más exhaustivo de los parámetros.

Como se ha explicado, la disponibilidad de redes intermedias otorga una mayor complejidad al software, por lo que, como parte de este proceso, es necesario valorar un equilibrio entre simplicidad y efectividad en las funciones de activación a utilizar en las conexiones neuronales. Entre todas las posibles opciones que existen de funciones no lineales destacan dos de ellas: sigmoide y ReLU [55]. Es importante tener en cuenta que estas funciones disponen internamente de un desarrollo lineal como entrada. Considerando la función de regresión logística, de la expresión (4.2), una variable z, la sigmoide modula los datos en un rango entre 0 y 1, siendo descrita por la siguiente expresión:

$$S(z) = \frac{1}{1 + e^{-z}} = \frac{e^{z}}{e^{z} + 1} \tag{4.4}$$

Su análisis deriva en la limitación de la sigmoide para la construcción del perceptrón multicapa dado que el entrenamiento de un número elevado de capas llega a ser complicado al surgir el problema de desaparición de gradiente [56].

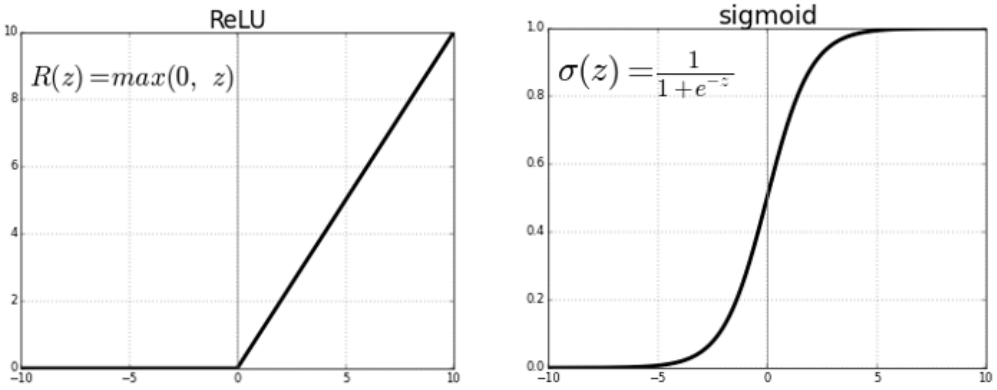


Fig. 4.2. Representación gráfica de funciones ReLU y sigmoide [55]

La otra opción destacada es la ReLU (*Rectified Linear Unit*), que permite el paso intacto de los valores positivos asignando los valores negativos a 0. Su utilización marcó un hito en el estado del arte del aprendizaje supervisado de principios del siglo XXI, desde que su relación simplicidad – efectividad le ha otorgado ser la función de activación por defecto en casos no lineales [57]. Viene determinada por la expresión:

$$f(x) = max(0, x) \tag{4.5}$$

La técnica más común de uso de estas funciones opera con un modelo de dos capas ("2-layer model"), sólo teniendo en cuenta aquellas que disponen de entrada procedente de otras neuronas, como se muestra en la ilustración Fig. 4.3. En ella, las capas estarían distribuidas de la siguiente forma:

- La primera se encarga de procesar los pesos y bias aplicados a los datos de entrada y que pasan a través de la ReLU. Su salida alimenta la entrada de la siguiente capa, pero no obtiene resultados observables, de donde procede el término de "capas ocultas".
- La segunda (y en este caso última), toma los pesos y *biases* de esta capa intermedia y los aplica en la función de salida, que en el caso de los clasificadores suele ser la función softmax para generar probabilidades de la que se hablará más adelante.

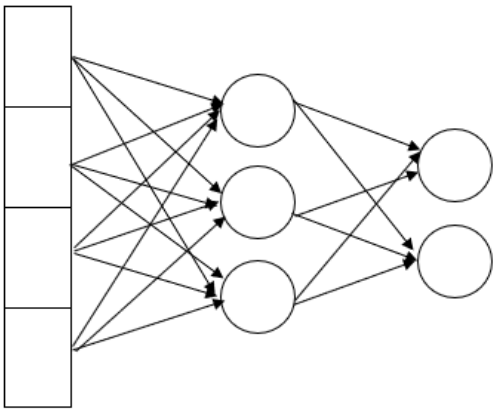


Fig. 4.3. Ejemplo de red neuronal de modelo "2-layers"

En estudios realizados, se asegura que para redes de número reducido de capas los resultados son similares utilizando ambas funciones [57]. Por ello, y aunque en este proyecto no se vayan a realizar experimentos con redes neuronales de más de 3 capas, se ha optado por implementar un algoritmo que se pueda generalizar para líneas futuras de estudio, por lo que en la decisión ha tenido mayor peso el modelo ReLU. En la ilustración inferior se resumen el proceso seguido en el análisis del dato.

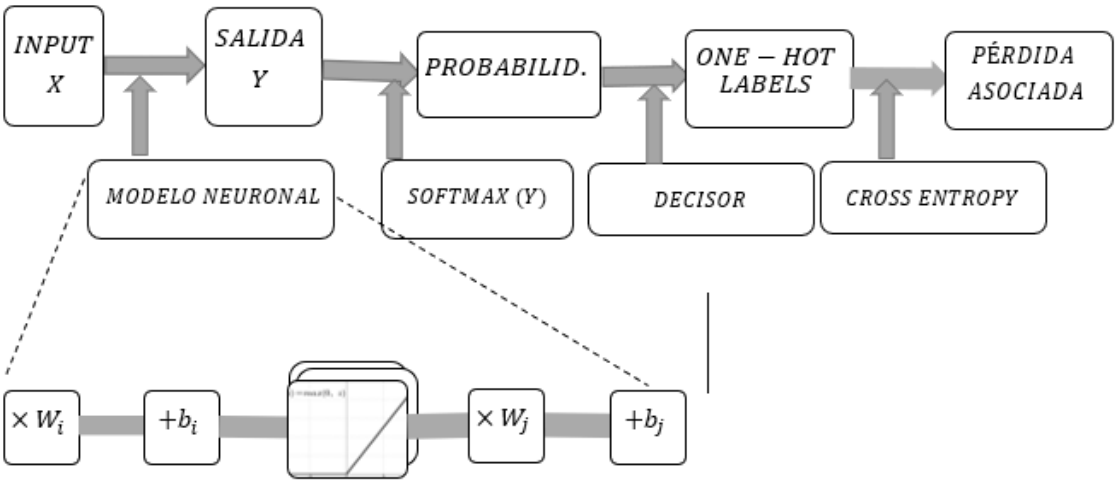


Fig. 4.4. Procesamiento del dato a su paso por el clasificador

En la implementación, cada una de estas capas se configuró mediante un nuevo tensor de pesos cuyas dimensiones se determinaron de la siguiente manera:

- La primera capa estaría compuesta por: Número de tokens × Números de nodos ocultos que se desean incluir en la segunda capa
- Cualquiera de las capas ocultas intermedias presentaría la forma: Número de nodos ocultos que la componen × Número de nodos ocultos de la siguiente capa
- La última *hidden layer* se compondría de: Número de nodos ocultos que la componen × Número de etiquetas utilizado en el algoritmo clasificador, común al número de salidas que se desea obtener.

Hasta este punto, el resto de procedimientos seguían la misma estructura que en el caso de regresión logística: declaración de tensores para train, test y validación, y parámetros de la ecuación. Sin embargo, al cómputo de dicha ecuación hubo que añadirle la función ReLU para que actuara sobre ella, de manera que la expresión quedara:

```
h1 = tf.nn.relu(tf.matmul(tf_train_dataset, W1) + b1)
           logits = tf.matmul(h1, W2) + b2
```

En el caso de existir más de una capa oculta, se aplicó a cada cómputo dicha función, anidando expresiones como, por ejemplo, para dos *hidden layers*:

```
h2 = tf.nn.relu(tf.matmul(h1, W2) + b2)
    logits = tf.matmul(h2, W3) + b3
```

siendo h2 el cómputo utilizado para la segunda capa oculta y tanto W3 como b3 parámetros referentes a la capa de salida.

En el desarrollo de redes con más de una capa oculta se consideró favorable utilizar como función de activación la variante LReLU (Leaky ReLU), cuya eficacia ha sido demostrada en señales acústicas [58]. Analizar si, de igual manera, sus prestaciones harían mejorar este algoritmo, era el objetivo de su uso, siendo analizada en los experimentos de este proyecto. Durante el entrenamiento, neuronas que efectúan la operación ReLU pueden morir si un gradiente elevado actualiza los pesos de manera que la neurona nunca se volviera a activar. Esto sería irreversible dado que los valores negativos se anulan y no permiten la recuperación del sistema. Para evitar esto, LReLU presenta un mínimo valor de gradiente variable para los valores negativos, como se muestra en su expresión:

$$f(x) = \begin{Bmatrix} x & x > 0 \\ \alpha x & otherwise \end{Bmatrix}$$

A pesar de que versiones recientes de Tensorflow implementan automáticamente esta función, para el entorno utilizado ha sido construida en un nuevo método en Python que, haciendo uso de la función ReLU de TensorFlow, genera esta expresión, siendo α el parámetro que marca la inclinación del gradiente para valores negativos [58]:

tf.nn.relu(x) – alpha * tf.nn.relu(-x)

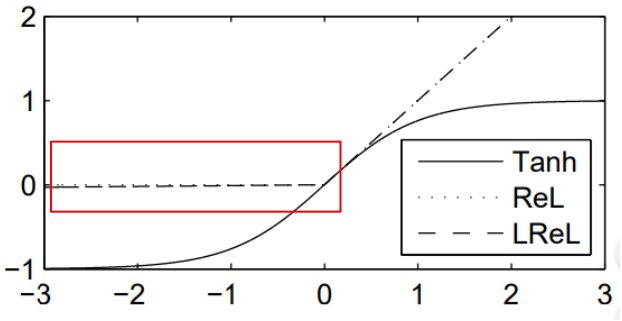


Fig. 4.5. Representación gráfica de la influencia de las redes no lineales [58]

Frente a la mejora de resultados en la precisión utilizando estas técnicas no lineales, se sitúa una dificultad añadida a la función de pérdida. Se pierde el concepto de función cóncava, donde mínimo relativo y absoluto coincidían, para dar lugar al surgimiento de numerosas ondulaciones en su interpretación geométrica, ocasionadas por la no linealidad de la función de activación, que es necesario tratar mediante optimizadores como *gradient descent*.

4.5.2. Optimizadores

Los optimizadores son técnicas matemáticas que buscan reducir la función de coste para obtener la mejor clasificación. El concepto se basa en la búsqueda de direcciones en el espacio que permitan obtener la ruta con la que se alcance el valor mínimo de error. Con ello, se determinaría el valor óptimo de los parámetros partícipes para utilizarlos en el algoritmo de predicción.

Conviene saber que existen ciertos valores en estas técnicas que el usuario debe introducir previo entrenamiento del sistema y cuya elección es decisiva para el resultado final de los experimentos. Parámetros tales como el número de capas ocultas, el tamaño del minibatch o la tasa de aprendizaje utilizada (*learning rate*) son considerados

hiperparámetros (*hyper-parameters*). A continuación, se muestran los optimizadores más frecuentes junto a su implementación en TensorFlow:

- **Gradient Descent (GD):** Surgió como medida de reducción del coste computacional generado si se utilizan todos los datos de entrenamiento para encontrar el mínimo absoluto. Consiste en el análisis del dataset de manera que cada muestra es considerada única actualizando los pesos en cada nueva iteración de cálculo del gradiente. En su ejecución, se utiliza un único *learning rate* que permanece fijo durante todo el entrenamiento (`tf.train.GradientDescentOptimizer`).
- **Adaptive Gradient Algorithm (AdaGrad):** Adapta el *learning rate* al algoritmo ejecutando las principales actualizaciones en función de los datos menos frecuentes, aportando más información al sistema, pero a su vez reduciendo el *learning rate* en cada iteración (`tf.train.AdagradOptimizer`).
- **Root Mean Square Propagation (RMSProp):** El ajuste del *learning rate* se genera según un vector de media de valores variantes, calculado a partir del cuadrado del gradiente para cada componente (`tf.train.RMSPropOptimizer`).
- **Adaptive Moment Estimation (Adam):** Se considera una fusión de los dos anteriores, presentando numerosas ventajas sobre el simple Gradient Descent. Su control sobre un variable learning rate permite establecer steps efectivos que permiten al algoritmo converger sin adaptación manual del hiper-parámetro [59] . La desventaja de este optimizador es una mayor carga computacional efectuada en cada parámetro por step de entrenamiento, al mantener variantes la media y la varianza, calcular el *scaled gradient*, y almacenar de cada parámetro aproximadamente el triple de su tamaño en el modelo (`tf.train.AdamOptimizer`).

En el ajuste del *learning rate* es conveniente aplicar una "tasa de olvido" (*decay rate*) que marque el descenso del parámetro a través de las iteraciones. Esta actualización puede estar basada en distintas funciones como cíclica o exponencial. En esta implementación se ha utilizado esta última, función aplicada con éxito en otros estudios [60]. Para ello se ha hecho uso del método `tf.train.exponential_decay`, cuyos argumentos serán analizados en el apartado de "Evaluación de resultados".

Conviene destacar que los procedimientos y técnicas sobre los que se construyen los anteriormente nombrados se pueden encontrar en [61].

Por otra parte, para reducir el coste de computación se ha generalizado el uso de *minibatches* en el desarrollo del algoritmo, lo que significa utilizar subconjuntos de los datos de entrenamiento que permitan procesar una muestra de estos datos en cada iteración, consiguiendo lo mejor de ambas técnicas, el gradiente mediante mini-batches es más exacto al no generar rutas a partir de una única muestra, y permite la reducción de los datos de entrenamiento.

Además de lo expuesto en esta sección, existen consideraciones especiales a tener en cuenta en cualquier problema de aprendizaje automático que conviene entender para la futura explicación de los resultados experimentales.

4.5.3. Overfitting

Situaciones en las que las predicciones se adecuan extraordinariamente bien a los datos originales de entrenamiento pueden reflejar que el modelo está sobre-optimizado. A pesar de que pueda parecer un resultado ejemplar, la exactitud obtenida se debe a que lo datos aportados no son suficientemente representativos para entrenar al clasificador, de manera que se puede considerar que antes de realizar la predicción, el algoritmo está aprendiendo de su propia salida. Esto ocurre cuando no existe una distribución efectiva entre entrenamiento – test, o el conjunto está demasiado sesgado, lo que se explica en una posible tendencia a una única categoría, asumiendo el clasificador que esa siempre ha de ser la salida (lo más probable).

Durante el entrenamiento, se puede monitorizar la precisión del modelo. Es coherente pensar que cuantas más iteraciones se realicen, mejor será este resultado. Sin embargo, esto ocurrirá únicamente hasta cierto punto, momento en el que este porcentaje detendrá su ascenso y que puede ser detectado por técnicas de validación. Es a partir de este punto cuando comienza a producirse una situación de overfitting.

4.5.4. Cross-Validation

Optimizar un algoritmo con hiperparámetros que hagan que funcione con un porcentaje máximo puede resultar nocivo para el algoritmo al centrarse únicamente en ese corpus. Esto puede generar unos resultados óptimos, pero, sin embargo, una mínima generalización a otros conjuntos de datos, aspecto importante del aprendizaje automático,

dado que, de lo contrario, el entrenamiento no habrá valido para que el software obtenga resultados como pensamiento humano.

Este problema es el origen de la necesidad de los datos de validación, pero más aún de la utilidad de la técnica de validación cruzada. Su objetivo es evitar que se produzca *overfitting* en el algoritmo mediante la división de los datos de entrenamiento en n iteraciones, en vez de una proporción fija como la establecida para el conjunto de test. En cada iteración, por tanto, el conjunto de datos de validación estaría formado por elementos distintos seleccionados según la variante de la técnica elegida. Entre estas opciones, la selección puede consistir en la realización de k iteraciones con el consiguiente reparto de una proporción de 1/k sobre el total en cada iteración, de manera que, al acabar el proceso, todos los datos habrán formado parte del conjunto de validación (K-Fold Cross-Validation). Otra opción es elegir en cada iteración un conjunto de elementos aleatoriamente, de manera que al final del proceso existan datos que hayan quedado sin evaluar y otros que hayan sido evaluados doblemente.

En este proyecto, y dada la dificultad que presenta TensorFlow para implementar la validación cruzada al no disponer de función propia como sklearn, se ha utilizado una repartición de 60 % para datos de entrenamiento y 20 % para test y validación. Como ha sido explicado anteriormente, es el conjunto de test quien determina la efectividad del algoritmo en la precisión final mientras que el conjunto de validación garantiza su escalabilidad y la generalización a otros corpus.

4.5.5. Regularización y Dropout

El proceso de regularización da información de las restricciones que conviene tomar para evitar caer en situación de overfitting, de manera que reduce el número de parámetros libres a utilizar en la implementación del algoritmo. Fundamenta su concepto en la adición de un nuevo término a la expresión de la pérdida, lo que penaliza a los pesos con valores elevados.

Destacan dos técnicas de regularización (L1 y L2) que basan sus diferencias en conceptos matemáticos: L1 se corresponde con la suma exacta de los pesos y L2 con la suma de los pesos al cuadrado. Esta propiedad es origen del carácter único de la

regularización L2, ya que, conceptualmente, su valor se corresponde con el camino más corto entre dos puntos, ofreciendo L1 múltiples soluciones.

En los experimentos realizados por James McCaffrey en 2015 se comparan ambas técnicas de regularización, LI y L2, [62] extrayendo como conclusión una ligera mejora en los resultados de la técnica L1. Sin embargo, como contraprestación, presenta una peor adaptación a los algoritmos de entrenamiento de Machine Learning generando complicaciones en el cómputo del vector gradiente. Por ello, se concluye que la estabilidad adicional proporcionada por el método L2, y su simplicidad en la implementación, es motivo de selección siendo la técnica que se ha decidido utilizar en este desarrollo.

Por tanto, siendo inicialmente la pérdida el valor obtenido por entropía cruzada, su implementación en TensorFlow usando la función `tf.nn.l2_loss()` a través de n iteraciones, se basó en la expresión:

$$Pérdida_{total} \mathrel{+}= Factor\ regulador \times \sum_{i=0}^{n} Pérdida\ (w_i) \qquad (4.6)$$

En los últimos años, se ha profundizado en las técnicas de regularización con el objetivo de reducir el overfitting en los desarrollos. En esta línea de trabajo, en 2014 surgió un nuevo concepto desarrollado en la Universidad de Toronto denominado Dropout [9].

Considerando un estado inicial en el que todas las neuronas artificiales están conectadas entre sí, es lógico pensar que el número de activaciones (conexiones capaces de transmitir información) va a ser un valor elevado. Para obtener un aprendizaje más proporcional, la técnica consiste en desactivar aleatoriamente, para cada iteración, algunas conexiones (*drop out*) que permitan entrenar el clasificador evitando el exceso de dependencia de algunas neuronas, obligando a otra conexión a llevar a cabo dicho entrenamiento. Así, el concepto se sustenta en la eliminación temporal de dichas conexiones de la red un porcentaje aleatorio de veces.

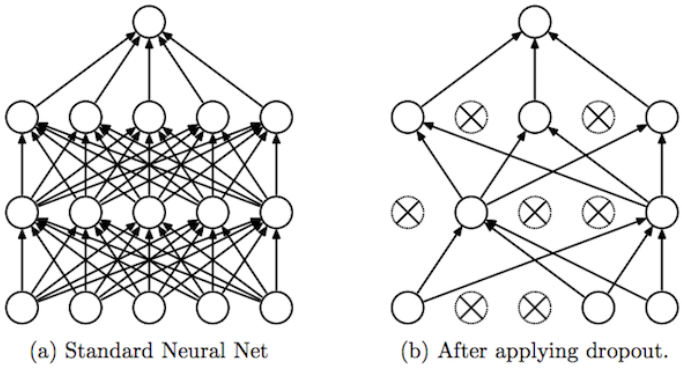


Fig. 4.6. Efecto del dropout en una red neuronal [9]

En el mismo ensayo citado se explica el caso de clasificación utilizando minibatch, el cual será idéntico al algoritmo sin dropout sólo que el conjunto muestreado conformará una red aún más delgada al desactivar conexiones anteriormente necesarias. En la implementación se ha hecho uso de la función `tf.nn.dropout()` aplicada a capa de la red con un coeficiente determinado explicado en los experimentos.

4.5.6. Backpropagation

Para finalizar con las técnicas de aprendizaje automático profundo que se ha considerado necesario tratar en este trabajo para comprender su hilo conductor, es esencial entender el concepto de *backpropagation*.

En el proceso de entrenamiento de las redes actuales se llevan a cabo dos fases. La primera consiste en un proceso de *feedforward* (proceso hacia delante), en la que los datos de entrada y salida conocidos se propagan a través de la red, generando determinadas salidas que actúan como predicciones. En la década de los 70, se publicó la obra Perceptrons [6] basándose en esta premisa, utilizando perceptrones de una capa y asumiendo este procedimiento como única fase del proceso.

Sin embargo, el funcionamiento neuronal no es tan simple. Años más tarde, se añadió una segunda etapa, sustento de las actuales redes neuronales, el concepto de *backpropagation* (proceso hacia atrás). A lo anteriormente explicado, se le añadió la idea de que las salidas obtenidas aportaran información que permitiera la modificación de los pesos y bias de las conexiones recorridas. Estas modificaciones establecerían una

constante adaptación de la red neuronal, desarrollando resultados de mayor precisión que los obtenidos hasta el momento.

Conceptualmente, y para comprender el concepto completo como se ha desarrollado, el proceso se plantea de la siguiente manera:

> La red neuronal genera una salida a partir de un input. Con mayor o menor acierto, esta predicción no será un reflejo perfecto del dato original, por lo que esta variación (error) será información que recibirán las neuronas partícipes del proceso, de manera que queden enteradas de su eficacia en la predicción y puedan aprovechar estos datos para mejorar los resultados en futuras iteraciones.

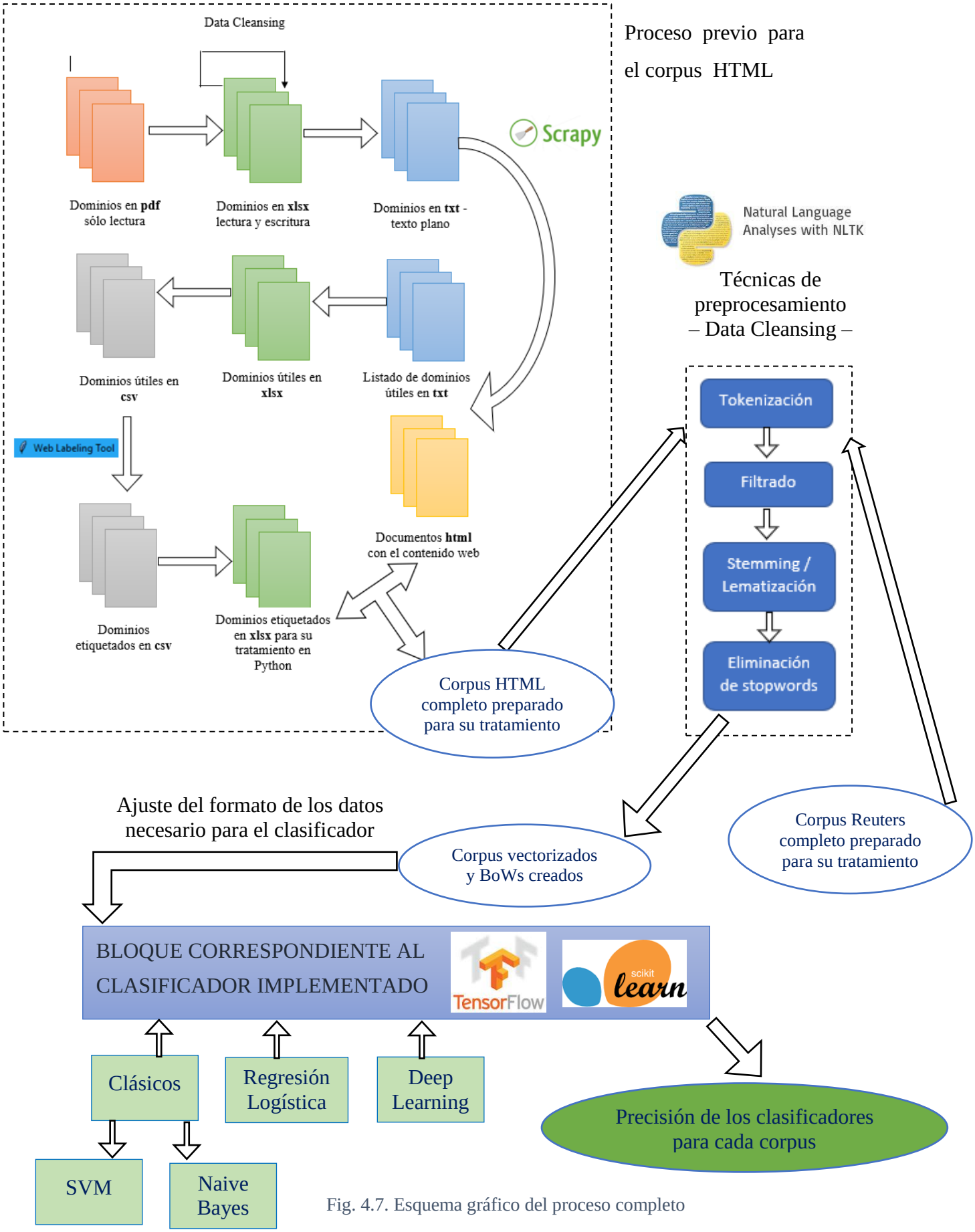


Fig. 4.7. Esquema gráfico del proceso completo

5. EVALUACIÓN DE RESULTADOS

En esta sección se mostrarán los resultados obtenidos por los clasificadores en ambos corpus tras la fase experimental, mientras son mencionados algunos de los análisis ya elaborados por otros autores. A su vez, se establecerá un análisis comparativo de los resultados con los distintos tipos de procedimientos seguidos para su realización, objetivo principal del proyecto. La sección finalizará con la descripción explícita de los métodos que ofrezcan mejores prestaciones al caso práctico de la clasificación de textos.

5.1. Análisis de los experimentos de clasificación

Los modelos de clasificadores desarrollados han sido evaluados en función de su precisión en la predicción, lo que por ende indica el porcentaje de textos analizados que el software ha sido capaz de clasificar correctamente.

Al comienzo del proyecto y como experimento de familiarización con el entorno, se propuso elaborar clasificación binaria como método de comprobación del funcionamiento básico de los algoritmos. Estos datos, lógicamente, permitieron obtener porcentajes de precisión que oscilaban entre aproximadamente el 86% y el 97%, dependiendo del algoritmo, desde la aplicación de SVM al uso de redes neuronales, respectivamente. Indiferentemente de la técnica aplicada, los resultados eran indiscutiblemente elevados dada la mayor simplicidad del decisor binario frente a la clasificación multiclase. Centrando el trabajo en este último caso, en esta sección se describen los resultados obtenidos mediante los experimentos elaborados con los dos dataset que se han venido explicando a lo largo de toda esta memoria.

En el escenario de preprocesado para los experimentos se han llevado a cabo las técnicas de Data Cleansing: tokenización, filtrado, stemming / lematización, y gestión de stopwords, configurando una bolsa de palabras como estructura de almacenamiento de los datos de entrada a los algoritmos clasificadores. Como era de esperar, el proceso de tokenización de los documentos del corpus HTML fue aproximadamente diez veces más costoso al no reconocer únicamente caracteres de lenguaje natural, si no numerosos elementos no alfanuméricos que ralentizan de manera considerable el procesamiento de los documentos.

Para verificar la efectividad de los resultados, se han utilizado los siguientes métodos:

SVM (Support Vector Machine): Algoritmo con kernel lineal muy utilizado en anteriores investigaciones con resultados nada despreciables. Se utiliza el algoritmo ya construido que ofrece sklearn para llevar a cabo los experimentos.

Naive-Bayes: De los algoritmos de Machine Learning que mejores resultados ha sido capaz de obtener. Su ejecución se lleva a cabo mediante el método preconstruido que ofrece sklearn. Se ha considerado necesario comparar las nuevas redes neuronales con este tipo de algoritmos extendidos desde hace un mayor tiempo.

Logistic Regression: Intermediario entre algoritmos clásicos y redes neuronales, se ha implementado mediante TensorFlow con la intención de aportar una mayor perspectiva entre las diferentes técnicas.

Deep Neural Networks: Algoritmo por excelencia en este proyecto implementado en TensorFlow, pero utilizando prestaciones de distintas librerías. Presenta una comparativa propia basada en los diferentes parámetros a utilizar y en su influencia en el aprendizaje y resultados.

En el caso particular de clasificación de páginas web se decidió comprobar la efectividad de los algoritmos con elementos adicionales al lenguaje natural. Además, se plantearon categorías que tenían más peso estructural que temática, lo cual era un factor a tener en cuenta en el análisis de los resultados.

5.2. Análisis de resultados obtenidos

5.2.1. Influencia de los optimizadores en redes neuronales

El empleo de una red neuronal como clasificador abre un sinfín de oportunidades para mejorar los resultados utilizando cualquiera de las numerosas técnicas que han sido explicadas en el apartado 4.5 . Con ello, se buscó demostrar una buena precisión y eficiencia para ambos corpus utilizando Deep Learning.

Como se apreciaba en numerosas referencias antes mencionadas, el método de *GradientDescent* podía aportar unos resultados adecuados de precisión, por lo que se decidió empezar los experimentos partiendo de esta premisa.

En primer lugar, las pruebas se realizaron sobre el corpus de Reuters al contener menos tokens, 21.622 para el entrenamiento frente a los 59.213 que componen el dataset de entrenamiento en lenguaje HTML. Esto permitiría llevar a cabo los experimentos con una mayor agilidad y objetividad, al ser todos ellos de lenguaje natural.

Tras numerosos intentos previos en los que se alteraba el valor de los hiper-parámetros a modo de prueba sobre los datos, obtener igual porcentaje de precisión (35.9 %) era indicador de que se estaba produciendo algún problema en el entrenamiento, por lo que las predicciones generadas no cambiaban y asumían una precisión constante, posible señal de sobreentrenamiento o de mala elección de función de pérdida. Con el fin de entrenar el algoritmo con la menor pérdida posible, y dado que modificar los hiper-parámetros no fue resolutorio, se optó por realizar un análisis de optimizadores, demostrando que, a pesar de que en la gran mayoría de estudios no tienen una suficiente repercusión, su elección puede llegar a ser determinante en los resultados.

La comparativa de optimizadores representada en la siguiente tabla muestra los mejores resultados obtenidos en un límite máximo de 15000 iteraciones, con adecuado porcentaje de validación, fijando un factor de regularización de 0.001 [62] y realizando n iteraciones, siendo n el valor que se ha necesitado para alcanzar la convergencia de resultados. Para no forzar un valor determinado de *learning rate*, se ha utilizado decrecimiento exponencial como método de actualización, comenzando con valor 0.1. Por otra parte, la función de activación seleccionada ha sido LReLU, con un parámetro de variación α de valor 0.001.

En cada optimizador se ha analizado el efecto que genera el número de nodos ocultos, así como el tamaño del minibatch (número de batch $\times$ tamaño del batch) en el caso de una única capa oculta, mostrando el porcentaje de precisión relativo a los datos de entrenamiento, test y validación, así como las pérdidas asociadas al proceso de entrenamiento. Cabe destacar que todo resultado mostrado es el valor medio de la triple repetición de cada experimento para evitar la posible influencia de datos dispersos minoritarios en el análisis.

Tabla 5.1. COMPARATIVA DE RESULTADOS SEGÚN EL OPTIMIZADOR

Optimizador	Nº1	Datos mini-batches	Train	Test	Valid.	Pérdida asociada
GD	**30**	**15 × 500**	86.0 %	81.6 %	79.1 %	56.4
		25 × 300	90.0 %	83.0 %	83.2 %	61.4
		35 × 200	92.3 %	83.4 %	85.3 %	89.3
	15	**15 × 500**	83.4 %	72.5 %	77.6 %	31.6
		25 × 300	84.3 %	75.5 %	75.7 %	43.92
		35 × 200	87.1 %	76.1 %	76.0 %	41.56
	100	**15 × 500**	90.4 %	90.8 %	80.1 %	79.43
		25 × 300	91.3 %	91.0 %	85.4 %	70.12
		35 × 200	91.7 %	92.3 %	83.2 %	74.20
Adam	**30**	**15 × 500**	**97.6 %**	**91.9 %**	**90.1 %**	**0.28**
		25 × 300	96.9 %	92.0 %	86.8 %	0.32
		35 × 200	98.1 %	90.1 %	89.1 %	0.47
	15	**15 × 500**	97.9 %	90.5 %	86.4 %	0.19
		25 × 300	98.4 %	90.5 %	86.9 %	0.23
		35 × 200	98.3 %	90.8 %	88.2 %	0.43
	100	**15 × 500**	98.3 %	85.7 %	83.5 %	0.53
		25 × 300	98.7 %	86.5 %	84.6 %	0.49
		35 × 200	99.1 %	86.4 %	84.8 %	0.46
AdaGrad	**30**	**15 × 500**	73.3 %	70.6 %	75.9 %	71.45
		25 × 300	75.2 %	73.2 %	71.0 %	79.36
		35 × 200	79.1 %	72.5 %	71.5 %	81.63
	15	**15 × 500**	80.5 %	66.5 %	65.5 %	36.03
		25 × 300	86.3 %	70.3 %	77.9 %	36.32
		35 × 200	89.8 %	83.9 %	84.1 %	34.62
	100	**15 × 500**	82.6 %	74.9 %	72.4 %	143.54
		25 × 300	84.7 %	84. 3 %	83.6 %	119.93
		35 × 200	89.2 %	87.2 %	82.4 %	126.51
RMS	**30**	**15 × 500**	**98.0 %**	**93.0 %**	**93.3 %**	**0.15**
		25 × 300	97.6 %	94.0 %	92.1 %	0.27
		35 × 200	98.5 %	92.3 %	85.6 %	0.23

	15	**15 × 500**	93.5 %	87.0 %	87.9 %	0.16
		25 × 300	93.8 %	88.5 %	88.2 %	0.12
		35 × 200	96.2 %	89.1 %	89.5 %	0.13
	100	**15 × 500**	96.1 %	93.1 %	89.4 %	0.89
		25 × 300	97.5 %	92.9 %	90.1 %	0.71
		35 × 200	97.8 %	94.0 %	90.5 %	0.73

Si bien Gradient Descent (GD) es la técnica más extendida por la antigüedad de su uso en algoritmos, no tiene por qué ser la más efectiva. Como se puede observar en la mayoría de los casos, la tabla permite detectar una tendencia ascendente de la pérdida o coste a medida que aumenta el número de nodos ocultos, al igual que ocurre con el tamaño del minibatch. Este último factor presenta una gran relevancia en el coste computacional dado que cuando el tamaño del batch es menor la ejecución se vuelve más ágil y dinámica.

Por otra parte, en algunos casos se detecta una disminución del porcentaje de validación cuando los valores del tamaño del batch y del número de nodos son elevados. Esto puede interpretarse como un sobreentrenamiento de los datos derivando en una baja generalización.

Tras su análisis exhaustivo no cabe duda de que, para las condiciones de este experimento, los mejores resultados se obtienen mediante el uso de RMS y Adam, que proporcionan prestaciones muy por encima del resto con pérdidas ínfimas en comparación con estos (identificados en verde). Además, una única capa oculta con 30 nodos permite obtener precisiones muy fiables. La presencia de 15 nodos proporciona resultados ligeramente más reducidos, pero también suficientemente válidos, a cambio de un coste computacional menor, por lo que sería una correcta elección si el tiempo de ejecución fuera una limitación.

Respecto al número y tamaño del minibatch no se demuestra una influencia decisiva en el conjunto. Se recomienda la combinación de 15 batches de tamaño 500 al reportar porcentajes de precisión elevados y un menor coste computacional.

Antes de continuar, se consideró importante concretar la elección del valor del *learning rate*, dado que en la búsqueda del mínimo de la función de pérdida, tiene un papel esencial este hiperparámetro. Para ello, es necesario partir del concepto en el que. según lo elevado o reducido que sea este, los resultados se verán afectados teóricamente de la siguiente forma, siendo X la curva objetivo y, en orden decreciente, 1 un valor muy alto de learning rate (> 0.5), y 3 un valor muy bajo (< 0.01):

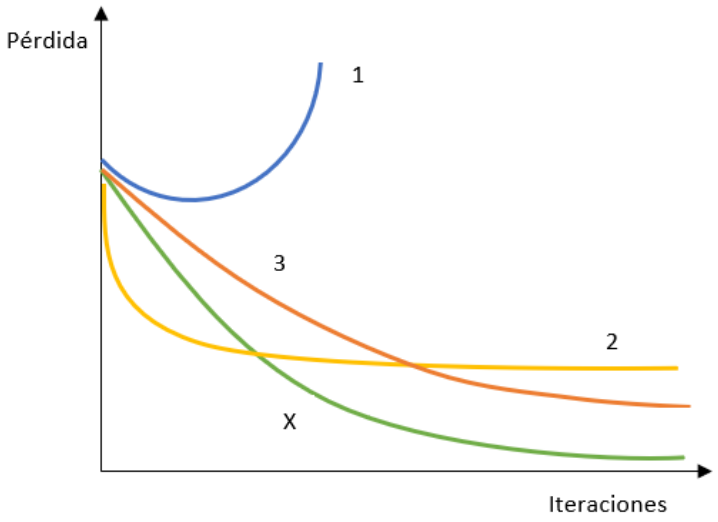


Fig. 5.1. Representación gráfica de las pérdidas para distintos learning rates

De ello se puede concluir que un valor demasiado elevado produciría un aumento prácticamente inmediato de la función de pérdida o una convergencia en valores no del todo precisos (curvas azul y amarilla). De lo contrario, un valor muy bajo exigiría un coste elevado al ralentizar la convergencia. Por tanto, alterar su valor era un aspecto obligado a considerar. Teóricamente, optimizadores como Adam son capaces de actualizar su *learning rate* según aumente el número de iteraciones. Fijando un valor inicial de 0.1, los resultados con las condiciones del mejor caso del análisis previo fueron los siguientes:

Tabla 5.2. PRECISIÓN MEDIANTE ACTUALIZACIÓN AUTOMÁTICA DE LEARNING RATE CON OPTIMIZADOR ADAM

Optimizador	Nº1	Datos mini-batches	Iter.	Train	Test	Valid.	Pérdida asociada
Adam	**30**	**15 × 500**	1000	42.0 %	50.1 %	51.6 %	2174.71
			4000	69.5 %	63.6 %	62.0 %	1834.76
			8000	85.5 %	68.2 %	68.1 %	1399.38
			12000	86.5 %	73.3 %	72.4 %	861.14
			16000	90.5 %	75.3 %	74.9 %	519.02
			20000	94.5 %	76.4 %	75.9 %	388.86
			24000	95.2 %	76.9 %	77.1 %	211.35

Como se puede apreciar, la pérdida inicial es extremadamente alta, y su ritmo de descenso demasiado bajo. Los porcentajes de precisión presentan una tendencia ascendente según aumenta el número de iteraciones, pero tras 24000 de ellas no se ha alcanzado la convergencia, y la precisión para el conjunto de validación es de tan sólo 77.1 %, mientras que para el de test, 76.9 %. La similitud entre los valores de test y validación es un punto a destacar, pero no puede decirse que los resultados obtenidos sean óptimos.

Por consiguiente, y a pesar de que la tendencia es correcta, utilizar un decrecimiento exponencial proporciona porcentajes de precisión elevados en un menor número de iteraciones. De esta manera, se puede rebatir el heurístico extendido en el que se afirma que la actualización propia del *learning rate* para el optimizador da resultados óptimos, pudiendo ser mejorados manualmente con funciones de decrecimiento, como se demuestra en los datos anteriores.

Finalmente, la utilización de la LReLU como función de activación de las neuronas se basa en la realización previa de experimentos con ReLU que proporcionaros peores resultados, al igual que con sigmoide, con unos datos de validación en torno al 8 % y 5 % inferiores, respectivamente. El resto de tablas adicionales de resultados no han sido añadidas al considerar que su información detallada no sería representativa para este

estudio, mientras que la justificación teórica de este suceso viene determinada en el apartado anterior.

Teniendo en cuenta los resultados, y dado que, aunque RMS proporciona los mejores porcentajes, no existe tanta diferencia entre los obtenidos por Adam, se decidió llevar a cabo los sucesivos experimentos haciendo uso de ambos optimizadores. A pesar de ello, cabe destacar que un simple Gradient Descent podría finalmente ser utilizado con prestaciones similares, sin embargo, necesitaría un complicado ajuste *(tuning)* antes de converger de manera óptima.

5.2.2. Elección de los hiper-parámetros para algoritmos de Deep Learning

La variedad de elementos que permite utilizar Deep Learning hace posible desarrollar numerosas investigaciones modificando los parámetros que influyen en la evaluación del resultado. Estos parámetros, denominados *hyper-parameters*, otorgan un carácter decisivo al usuario en su elección, dado que de su valor puede depender la optimización de los resultados. Es en su estudio en lo que ha profundizado esta comparativa.

En numerosas ocasiones, la elevada flexibilidad que proporcionan los valores desde un punto de vista teórico permitiría realizar prácticamente infinitas combinaciones, lo que obliga a limitar el rango de cada parámetro en función de lo que se estima sea la mejor elección. Esta estimación tiene un gran componente humano, que según su criterio debe decidir qué posibles valores puede ser interesante tratar.

Por ello, surgen teorías que intentan exponer razonamientos para la elección de la manera más objetiva posible . Particularizando para el caso del número de nodos ocultos en cada capa, el problema es: si se usan muy pocas neuronas se puede caer en situación de *underfitting*, por el contrario, si intermedian demasiadas, se puede caer en *overfitting*, así como una ralentización innecesaria del proceso. Para delimitar este valor se han planteado pautas como las descritas en [63] a partir de las cuales existen numerosos heurísticos que determinan que una buena elección se conseguiría mediante la siguiente expresión:

$$N_h = \frac{N_S}{\left(\alpha * (N_i + N_o)\right)} \qquad (5.1)$$

siendo N_S el número de muestras del dataset de entrenamiento, N_i el número de neuronas de entrada (características) y $N_o$el número de neuronas de salida, mientras que α correspondería con un valor arbitrario de escalado entre 2 y 10 para no cometer *overfitting*. Se ha visto conveniente citar esta fuente dado que su existencia está suficientemente extendida, sin embargo, no existe unanimidad en la evaluación de su efectividad y, por tanto, se ha considerado necesario establecer un valor adecuado mediante elección a base de prueba – error acorde con el número de nodos de salida (número de categorías).

A continuación, se analizará la influencia del número de nodos ocultos en redes neuronales de 2 y 3 *hidden layers*. En los siguientes experimentos se han utilizado los mismos criterios que en el apartado anterior para analizar la influencia de los restantes hiper-parámetros. Los condicionantes eran: *exponential decay* como función de actualización del learning rate así como un factor de regularización de 3e-3, así como LReLU como función de activación, con un parámetro de variación α de valor 0.0001. Para el corpus de Reuters, las tablas de resultados quedan como se muestra a continuación:

Tabla 5.3. CORPUS REUTERS - MODELO CON 2 CAPAS OCULTAS Y OPTIMIZADOR ADAM

Nº1	**Nº2**	**Datos batches**	**Iter.**	**Train**	**Test**	**Valid.**	**Pérdida asociada**
15	**30**	**40 × 300**	**1000**	57.7 %	47.2 %	40.5 %	109.96
			4000	75.3 %	58.5 %	52.3 %	16.22
			10000	85.3 %	64.7 %	54.9 %	5.19
15	**30**	**20 × 500**	**1000**	72.6 %	59.9 %	56.3 %	104.03
			4000	89.2 %	66.9 %	60.5 %	13.37
			10000	93.3 %	70.7 %	61.8 %	3.82
30	**15**	**40 ×300**	**1000**	56.8 %	52.8 %	50.2 %	80.25
			4000	64.0 %	60.6 %	61.5 %	19.85
			10000	80.0 %	73.1 %	77.2 %	11.63
30	**15**	**20 × 500**	**1000**	52.7 %	45.2 %	46.1 %	56.09
			4000	60.0 %	58.0 %	59.4 %	14.75
			10000	72.8 %	65.2 %	68.3 %	8.66

100	**30**	**40 × 300**	**1000**	90.2 %	77.4 %	70.5 %	465.70
			4000	**96.5 %**	**84.5 %**	**82.9 %**	**230.05**
			10000	92.0 %	82.7 %	75.4 %	393.78

Tabla 5.4. CORPUS REUTERS - MODELO CON 2 CAPAS OCULTAS Y OPTIMIZADOR RMSPROP

Nº1	**Nº2**	**Datos batches**	**Iter.**	**Train**	**Test**	**Valid.**	**Pérdida asociada**
15	**30**	**40 × 300**	**1000**	61.7 %	56.5 %	46.8 %	71.26
			4000	91.7 %	80.4 %	75.7 %	2.31
			10000	98.3 %	87.2 %	86.5 %	0.38
15	**30**	**20 × 500**	**1000**	73.8 %	61.6 %	58.3 %	70.35
			4000	95.2 %	81.9 %	80.5 %	1.98
			10000	97.8 %	91.5 %	91.4 %	0.25
30	**15**	**40 × 300**	**1000**	65.7 %	60.5 %	61.0 %	126.54
			4000	96.0 %	87.2 %	85.6 %	1.41
			10000	96.7 %	91.6 %	90.7 %	0.22
30	**15**	**20 × 500**	**1000**	83.2 %	68.4 %	67.4 %	127.88
			4000	94.0 %	81.4 %	82.4 %	1.18
			10000	**97.8 %**	**93.5 %**	**94.1 %**	**0.13**
30	**100**	**40 × 300**	**1000**	90.6 %	70.7 %	65.5 %	451.09
			4000	92.3 %	82.1 %	80.9 %	10.71
			10000	96.3 %	90.8 %	87.3 %	0.41

En ambos casos, su análisis permite deducir que un mayor número de nodos en la primera capa oculta que en la segunda proporciona, por lo general, mejores resultados a igualdad de dimensiones. En el caso de Adam se ha probado la eficacia del método con un número de neuronas en la primera capa oculta bastante elevado en comparación con la segunda, experimento que da resultados favorables, pero con unas pérdidas muy elevadas. Por el contrario, el experimento inverso se llevó a cabo con RMS, utilizando 100 neuronas en la segunda capa y 30 en la primera, lo que deriva en un ajuste más controlado de pérdidas y porcentajes nada despreciables.

A pesar de ello, de la tabla se puede concluir, por tanto, que la mejor opción para el clasificador consistiría en el optimizador RMSProp con 20 minibatches de 500 elementos cada uno con capas ocultas de 30 y 15 nodos respectivamente.

Una vez se hubo indagado en la importancia de los hiper-parámetros comentados, el mismo estudio se realizó para el corpus de documento HTML, para el que, teóricamente, se esperaba que se repitieran las mismas tendencias que las dispuestas en Reuters, pero con resultados muy inferiores a estos, dada la ausencia de lenguaje natural en todos sus tokens y un menor número de documentos de muestra. Por otra parte, su elevado coste computacional ha obligado a realizar los experimentos partiendo de los resultados exitosos del corpus Reuters, por lo que a continuación se detallan los condicionantes de este análisis y sus resultados:

- Función de activación: LReLU con $\alpha = 0.0001$
- Factor de regularización: 3e-3
- Optimizadores a utilizar: Adam y RMSProp
- *Learning rate* actualizado con decrecimiento exponencial comenzando con valor 0.1

Tabla 5.5. CORPUS HTML - MODELO CON 2 CAPAS OCULTAS Y OPTIMIZADOR ADAM

Nº1	**Nº2**	**Datos batches**	**Iter.**	**Train**	**Test**	**Valid.**	**Pérdida asociada**
30	**50**	**40 × 300**	**1000**	80.3 %	35.9 %	36.9 %	41.84
			4000	87.5 %	41.0 %	38.7 %	9.43
			10000	89.3 %	42.1 %	42.6 %	1.02
30	**50**	**20 × 500**	**1000**	91.3 %	41.9 %	39.4 %	21.9
			4000	96.0 %	46.4 %	44.5 %	1.93
			10000	96.9 %	47.9 %	47.0 %	0.43
50	**30**	**40 × 300**	**1000**	84.3 %	43.1 %	34.5 %	12.94
			4000	94.0 %	51.8 %	39.5 %	3.24
			10000	96.2 %	53.9 %	46.4 %	1.28
50	**30**	**20 × 500**	**1000**	87.0 %	50.1 %	41.4 %	18.33
			4000	95.5 %	56.1 %	46.3 %	1.86
			10000	**97.0 %**	**60.1 %**	**53.5 %**	**0.89**

30	**100**	**40 × 300**	**1000**	93.0 %	54.8 %	46.8 %	31.45
			4000	94.9 %	55.4 %	48.2 %	5.64
			10000	96.3 %	58.9 %	50.2 %	0.78

Tabla 5.6. CORPUS HTML - MODELO CON 2 CAPAS OCULTAS Y OPTIMIZADOR RMSPROP

Nº1	**Nº2**	**Datos batches**	**Iter.**	**Train**	**Test**	**Valid.**	**Pérdida asociada**
30	**50**	**40 × 300**	**1000**	74.6 %	45.3 %	43.8 %	110.32
			4000	81.2 %	45.9 %	43.4 %	6.42
			10000	83.4 %	47.4 %	44.9 %	0.45
30	**50**	**20 × 500**	**1000**	80.2 %	45.7 %	40.1 %	108.82
			4000	85.6 %	49.5 %	41.9 %	1.63
			10000	92.4 %	52.8 %	51.0 %	0.23
50	**30**	**40 × 300**	**1000**	90.1 %	46.2 %	39.8 %	34.92
			4000	94.0 %	47.4 %	47.2 %	3.84
			10000	96.1 %	47.8 %	48.0 %	0.12
50	**30**	**20 × 500**	**1000**	90.7 %	43.8 %	42.5 %	93.64
			4000	94.8 %	51.6 %	55.3 %	9.42
			10000	**98.2 %**	**57.8 %**	**60.2 %**	**0.21**
30	**100**	**40 × 300**	**1000**	91.3 %	41.3 %	40.5 %	73.96
			4000	95.9 %	49.3 %	48.2 %	3.93
			10000	98.3 %	58.4 %	54.2 %	0.95

Para experimentar con los datos HTML, dado que el corpus está compuesto por más tokens, con el fin de equiparar porcentajes, se decidió utilizar capas compuestas por un mayor número de neuronas. En la tabla se puede apreciar que, al igual que los porcentajes de precisión, las pérdidas para este corpus son menores que para Reuters. Uno de los factores que puede propiciar esta situación es la buena predicción que se realiza con los datos de entrenamiento, que manifiesta una diferencia notable respecto a los de validación y test pudiendo derivarse de un efecto de sobreentrenamiento, que esté utilizando características muy sesgadas propias de ese conjunto.

Analizando ambos corpus, la diferencia de resultados es abismal, siendo el mejor porcentaje de precisión para datos de validación en el corpus de HTML un 60.2 %, mientras que en Reuters se llegó a alcanzar un 94.1 %. A pesar de ello, la red que mejor resultado da, en términos generales, es aquella que presenta mayor número de neuronas en la primera capa oculta para 20 minibatches de 500 elementos.

En todos los casos, la presencia de un número elevado de neuronas en una de las capas genera una precisión en los datos de validación inferior a la de los datos de test, lo que deriva en una baja generalización del algoritmo. Es el matiz que hace decantarse por un número más equitativo de neuronas. Cabe destacar que la prioridad en la decisión de la combinación de parámetros la tiene el conjunto de validación, por lo que en este último caso, a pesar de que Adam proporcione un 60.1 % de precisión en test y RMSProp un 57.8 %, se opta por este último al llevar ventaja de aproximadamente 7 puntos porcentuales respecto a Adam en validación.

Para el corpus HTML, con los mismos criterios anteriores, pero sólo para el optimizador RMSProp, el modelo de una capa oculta para este caso responde a:

Tabla 5.7. CORPUS HTML - MODELO CON 1 CAPA OCULTA Y OPTIMIZADOR RMS

Optimizador	Nº1	Datos mini-batches	Train	Test	Valid.	Pérdida asociada
RMS	**30**	**20 × 500**	85.4 %	44.2 %	42.3 %	0.49
		40 × 300	87.2 %	51.1 %	51.6 %	0.34
	15	**20 × 500**	82.3 %	39.5 %	40.5 %	0.17
		40 × 300	84.1 %	42.9 %	43.8 %	0.21
	100	**20 × 500**	92.4 %	45.4 %	42.1 %	0.89
		40 × 300	92.6 %	50.5 %	44.6 %	0.93

Se puede observar que las pérdidas son muy similares en la mayoría de los casos, presentando un valor muy reducido. A pesar de que la precisión de los datos de entrenamiento es alta, no se puede decir lo mismo del resto de subconjuntos que no alcanzan el 52 % de precisión en el mejor de los casos. Entre las opciones computadas,

una red neuronal de una capa oculta con 30 nodos y 40 minibatches de tamaño 300 sería la combinación que mejores prestaciones proporcionaría. Cabe destacar que en este caso la presencia de 2 capas ocultas marca una diferencia importante, de aproximadamente un 9 % superior en la precisión de validación, lo que hace interesante dicho diseño de la red.

Los experimentos realizados dejan claro que el optimizador por excelencia para la clasificación de textos según los corpus que se están utilizando es RMSProp. Por ello, y para no efectuar pruebas innecesarias, a continuación se establecerá una comparativa entre las prestaciones de una red multicapa compuesta por 3 *hidden-layers*, únicamente para el uso de ese optimizador, dado que lo que se pretende ilustrar en este caso es la influencia del número de capas y neuronas que las componen.

Con 3 capas, el número de combinaciones posibles entre neuronas es aún mayor que en el caso anterior. Para evitar que este documento sea una mera reproducción de porcentajes, el número de neuronas utilizado en los siguientes experimentos es orientativo, de manera que se ha intentado cubrir las 3 posibles situaciones de referencia, a pesar de que otras combinaciones pudieran presentar un mejor funcionamiento. Por la misma razón, se han utilizado las dimensiones relativas al minibatch que mejor resultado han dado en los experimentos anteriores. Los resultados se resumen en la siguiente tabla:

Tabla 5.8. CORPUS REUTERS - MODELO CON 3 CAPAS OCULTAS Y OPTIMIZADOR RMSPROP

Nº1	**Nº2**	**Nº3**	**Datos batches**	**Iter.**	**Train**	**Test**	**Valid.**	**Pérdida asociada**
30	**50**	**15**	**20 × 500**	**1000**	72.0 %	63.9 %	55.6 %	139.43
				4000	92.2 %	84.1 %	73.4 %	0.37
				10000	99.6 %	90.5 %	79.3 %	0.18
50	**30**	**15**	**20 × 500**	**1000**	77.4 %	58.0 %	58.5 %	231.52
				4000	93.6 %	77.2 %	70.4 %	1.33
				10000	99.6 %	90.1 %	81.3 %	0.42
15	**30**	**50**	**20 × 500**	**1000**	74.4 %	50.3 %	55.3 %	89.52
				4000	83.3 %	52.1 %	56.6 %	3.38
				10000	92.2 %	65.1 %	59.4 %	2.34

Tabla 5.9. CORPUS HTML - MODELO CON 3 CAPAS OCULTAS Y OPTIMIZADOR RMSPROP

Nº1	Nº2	Nº3	Datos batches	Iter.	Train	Test	Valid.	Pérdida asociada
30	**50**	**15**	**20 × 500**	**1000**	76.4%	51.7 %	53.6 %	342.63
				4000	80.1 %	53.2 %	58.4 %	29.86
				10000	83.7 %	58.3 %	60.1 %	0.82
50	**30**	**15**	**20 × 500**	**1000**	79.5 %	53.4 %	49.5 %	468.92
				4000	87.8 %	58.9 %	52.3 %	35.72
				10000	94.3 %	60.2 %	55.2 %	0.89
15	**30**	**50**	**20 × 500**	**1000**	77.9 %	52.4 %	43.5 %	227.36
				4000	82.2 %	55.3 %	49.2 %	11.45
				10000	90.4 %	56.9 %	54.3 %	3.35

Se observa que el mejor resultado, a pesar de la precisión de este, se consigue con una distribución de mayor a menor número de neuronas en las sucesivas capas utilizadas, con unas pérdidas aceptables. Para el caso del diseño de la red con un número de neuronas incremental según las capas que la componen, surge una estabilización de la pérdida en niveles mejorables, asemejándose a la línea 3 de la ilustración Fig. 5.1. Para el corpus HTML, los resultados generados son bastante similares entre sí, sin mejoras los obtenidos hasta el momento y dejando evidencias del overfitting al que está sometido. Para el de Reuters, conseguir un 81.3 % de precisión para los datos de validación sigue sin mejorar las prestaciones conseguidas.

Una vez extraídos los mejores resultados hasta el momento, y siendo previsible el diseño de la red neuronal óptima, se procedió a analizar si el dropout efectuase alguna mejora sobre los casos analizados, partiendo de la idea de que su efecto optimiza considerablemente los algoritmos con corpus de imágenes sobreentrenados. La probabilidad de eliminación de conexiones que viene definida por el *dropout*, debe ser fijada en el software de clasificación, siendo 0.5 [9] el valor por defecto establecido por los desarrolladores de la técnica para su uso genérico con conjunto de validación.

Tabla 5.10. CORPUS REUTERS - EFECTO DEL DROPOUT

Nº1	Nº2	Nº3	Datos batches	Iter.	Train	Test	Valid.	Pérdida asociada
30	**-**	**-**	**15 × 500**	**1000**	68.2 %	72.0 %	71.7 %	2.54
				4000	92.8 %	89.5 %	90.6 %	0.41
				10000	97.6 %	93.3 %	93.9 %	0.08
30	**15**	**-**	**20 × 500**	**1000**	72.2 %	51.3 %	49.1 %	360.24
				4000	87.9 %	79.5 %	80.4 %	2.03
				10000	98.6 %	93.9 %	94.7 %	0.25
50	**30**	**15**	**20 × 500**	**1000**	69.3 %	54.2 %	59.3 %	194.57
				4000	78.6 %	83.2 %	73.5 %	1.56
				10000	85.6 %	91.6 %	85.1 %	0.23

Tabla 5.11. CORPUS HTML - EFECTO DEL DROPOUT

Nº1	Nº2	Nº3	Datos batches	Iter.	Train	Test	Valid.	Pérdida asociada
30	**-**	**-**	**40 × 300**	**1000**	74.2 %	40.6 %	38.4 %	49.65
				4000	82.5 %	50.3 %	46.9 %	3.54
				10000	90.2 %	54.1 %	55.3 %	0.12
50	**30**	**-**	**20 × 500**	**1000**	78.7 %	39.5 %	40.9 %	165.03
				4000	87.3 %	52.9 %	53.6 %	6.15
				10000	97.9 %	61.5 %	64.8 %	0.18
50	**30**	**15**	**20 × 500**	**1000**	83.5 %	44.4 %	49.5 %	468.92
				4000	89.8 %	58.9 %	52.3 %	35.72
				10000	95.3 %	61.2 %	58.2 %	0.89

Como se puede observar, el efecto del dropout es ligeramente apreciable en el corpus de Reuters, al mejorar los resultados de validación menos de 1 punto porcentual, y produce una mayor mejora en el referente a los documentos HTML, lo cual es posible dados sus bajos resultados en el análisis previo, llegando a incrementarse hasta un 4.5 % los datos obtenidos. Con esta técnica, el aprendizaje inicial es reducido (68.2 % frente a 83.2 %), para luego mejorar considerablemente. A modo de norma general, se puede

afirmar que en la clasificación de textos estudiada tiene un efecto limitado pero válido como técnica que proporciona una mejor precisión en casos en los que el *overfitting* sea elevado.

5.2.3. Comparativa de resultados con algoritmos clásicos

Tras el análisis exhaustivo realizado en la sección anterior, se decidió valorar si las redes neuronales y el desarrollo del Deep Learning era esencial para la clasificación de textos o, por el contrario, los algoritmos clásicos proporcionaban resultados equivalentes. En la comparativa posterior se incluyen los resultados generados de la implementación de los algoritmos definidos en este proyecto, para ambos corpus, seleccionando para el caso de las redes neuronales los mejores resultados para una y dos capas ocultas.

A modo informativo para la posible reproducción de estos experimentos, la regresión logística es llevada a cabo aplicando RMSProp como técnica de optimización aplicando el mismo descenso exponencial que en los casos anteriores para mejorar los resultados.

Corpus Reuters

	SVM	**N-B**	**Log. Reg.**	**DNN 1**	**DNN 2 Dropout**
Entrenamiento	95.6 %	92.4 %	91.8 %	98.0 %	98.6 %
Test	91.3 %	90.7 %	87.9 %	93.0 %	93.9 %
Validación	91.6 %	89.4 %	91.2 %	93.3 %	94.7 %

Corpus HTML

	SVM	**N-B**	**Log. Reg.**	**DNN 1**	**DNN 2 Dropout**
Entrenamiento	94.7 %	68.9 %	94.0 %	87.2 %	97.9 %
Test	52.2 %	45.6 %	58.3 %	51.1%	61.5 %
Validación	57.6 %	48.4 %	56.9 %	51.6 %	64.8 %

Es innegable que el algoritmo SVM proporciona unos resultados considerablemente buenos a una velocidad prácticamente inmediata, por lo que la relación precisión – tiempo de ejecución determina una buena posición para este clasificador. Cabe destacar que el entrenamiento mediante SVM y Naive-Bayes es más eficiente que en las redes neuronales, donde se requiere un mayor coste computacional, proporcionando resultados bastante similares sin necesidad de hacer uso de heurísticos que determinen qué parámetros usar.

A pesar de ello, y siendo técnicamente puristas, los porcentajes de precisión obtenidos mediante redes neuronales artificiales son ligeramente superiores en el caso de Reuters y notan un incremento notable en el caso de las páginas web, dentro de la baja precisión manifestada. A modo resumen, en este último caso, cabe destacar la presencia de tres factores principales que pueden haber afectado a la obtención de resultados de una manera negativa:

- Limitación en el número de muestras procedentes del proceso de scraping.
- Presencia de caracteres de lenguaje no natural (HTML) en las muestras tomadas.
- Categorización con etiquetas basadas en criterios estructurales con menor peso temático, lo que implica una mayor complicación para la decisión por el clasificador.

5.3. Referencia a experimentos de otros autores

La investigación definida en gran parte del estado del arte determinó qué experimentos habían sido previamente realizados y si existen comparativas establecidas entre ellos. Aquellos que hacían uso del corpus de Reuters, respecto a los que se podría establecer una comparación directa con los resultados de este proyecto, o Wikipedia, datasets predominantes en la clasificación de textos, utilizaban clasificación binomial. Esto quiere decir que los resultados disponibles basaban su precisión en pertenecer a una categoría y no a las demás, mientras que, en este caso, la precisión viene determinada por la pertenencia de cada artículo a las categorías con clasificación multietiqueta, y no realizando un análisis individualizado.

Por otra parte, las categorías utilizadas en otros experimentos solían proceder de una selección de las diez consideradas más relevantes, con mayor número de muestras en el corpus. En este caso, englobar subcategorías en una de mayor nivel ha permitido tener

a nuestra disposición un mayor número de elementos para entrenar con cada etiqueta resultante, aportando mayor variedad y, por tanto, dificultad al clasificador, que ha superado sin problemas el clasificado con unos resultados de elevada precisión.

En el campo de la clasificación de páginas web, los experimentos llevados a cabo proporcionan resultados muy diversos dada la difícil unanimidad de los criterios. En [30], para la clasificación por URL, SVM genera un 38 % de precisión, siendo superado en gran medida por lo obtenido en este proyecto. Otra investigación como [34] genera unos resultados utilizando redes neuronales muy superiores a los conseguidos en estos experimentos, alcanzando el 90 % de precisión. Cabe destacar que su extracción estuvo sesgada a una serie de htmls seleccionados y para unas categorías muy específicas.

Por otra parte, en el artículo [33] de 2012, utilizando categorías estrictamente temáticas, se consiguen los siguientes resultados:

Learning method	Accuracy
Binarized L2-SVM	59.4
Binarized L1-SVM	58.9
Multiclass SVM	59.4
Logistic regression	49.9
Perceptron	51.1
Passive–aggressive	58.4

Fig. 5.2. Resultados clasificadores experimento [33]

Tomando esta referencia, a pesar de las diferencias que pudieran existir en etiquetado y contenido de las muestras, podemos apreciar que para los dos métodos comunes a este proyecto, los resultados son similares. La gran disparidad que determina organizar los datos en categorías temáticas o estructurales no se refleja de manera perjudicial en los resultados, llegando incluso a ser mejorados como en el caso de la regresión logística (aproximadamente un 6 % superior), donde en las buenas prestaciones de este proyecto ha podido influir la elección del optimizador. Dada la gran diversidad de criterios existentes para este ámbito del análisis de datos, no se ha considerado relevante proporcionar datos de investigaciones adicionales.

6. CONCLUSIONES

6.1. Conclusiones y valoración del proyecto

En el presente proyecto se han implementado los algoritmos de clasificación de textos propuestos, poniendo el foco de atención en la influencia de los parámetros disponibles para redes neuronales y estableciendo una comparación entre ellos y los algoritmos clásicos. Con esto, se han cubierto satisfactoriamente todos los objetivos que se propusieron al inicio del mismo. La incorporación de procedimientos de machine learning al análisis de textos ha permitido conocer las numerosas técnicas en las que se basa esta disciplina, así como las sólidas bases matemáticas y estadísticas que sustentan su evolución.

Es algo innegable que los cimientos técnicos que ha aportado su desarrollo son altamente útiles en la actualidad. Sin embargo, cabe destacar la gestión del proyecto como pieza importante de la elaboración de este trabajo. No haber ceñido su creación a la implementación, sino también generar la documentación asociada, así como contemplar aspectos relacionados con el marco económico y legal, ha permitido una expansión de las habilidades transversales y de la capacidad de abstracción, contextualizando el planteamiento de la clasificación de textos. Esto ha proporcionado una perspectiva global, de gran importancia para marcar los hitos, tareas y objetivos que deben definirse, con la relevancia correspondiente, así como las posibles mejoras a implementar.

6.2. Principales aportaciones al sector

La investigación llevada a cabo en este trabajo, así como los datos y técnicas aquí recopiladas, forman una base importante para el campo de la clasificación automática de textos.

Recorriendo las fases del procesado de datos, en primer lugar se presentó el concepto de araña web, como método capaz de realizar la extracción de documentos procedentes de páginas web en lenguaje HTML. Se concluye que la extracción mediante la tecnología Scrapy es eficaz para realizar este proceso pudiendo seleccionar los aspectos de interés que influirán en el vector de características. El atractivo de la clasificación de

páginas web en este proyecto reside en la comprobación de si utilizar técnicas de lenguaje natural es útil para textos en los que coexisten otros muchos caracteres propios del lenguaje HTML. Evidentemente, la precisión obtenida para este caso es mejorable, dado que no supera el 61 % en el caso óptimo. Tras la investigación realizada y los resultados experimentales obtenidos, se concluye que utilizar NLP es aplicable a corpus de documentos HTML de manera satisfactoria, a pesar de que estos deben atravesar técnicas de preprocesamiento especiales, como la consideración de stopwords específicas o la eliminación de los tokens que menos aparecen en el corpus, lo que podría evitar que tokens de seguridad o identificadores de acceso a elementos distorsionaran los resultados. Es el caso de tokens alfanuméricos aleatorios generados en la codificación de la página web con formato similar a "e8u8283298ejsj8" que sólo aparecen una única vez en todo el corpus analizado.

En aspectos generales, independientemente del contenido de los documentos, cualquier algoritmo de clasificación recibe un vector de características que identifica los datos de entrada. Estos previamente han tenido que someterse a técnicas de Data Cleansing y Data Wrangling, fases de preprocesado donde se aplican técnicas de tokenización, filtrado, stemming o eliminación de stopwords, con el fin de extrapolar las características más importantes de los documentos a analizar y aplicarles el formato adecuado para su incorporación al clasificador.

Ya en el ámbito de los algoritmos de clasificación propiamente dichos, en ambos corpus, se ha observado que clasificadores supervisados como SVM o Naive-Bayes, métodos clásicos de aprendizaje automático, proporcionan unos resultados de precisión admirables en muchos casos, liderando los algoritmos de esta categoría. Particularmente, para el análisis del corpus de Reuters (lenguaje natural) se demuestra que la precisión obtenida es suficientemente elevada para crear un sistema fiable.

El culmen del estudio se alcanza con el análisis de técnicas de Deep Learning, creando una clara imagen de los parámetros que resulta útil tratar y de qué manera ese tratamiento puede ser favorable.

El estudio de optimizadores diversifica aún más la exploración de prestaciones con los diferentes hiper-parámetros, por lo que su importancia deriva en la elección de un *learning rate* apropiado a la clasificación. Esto ha permitido salir de la zona de confort que proporciona el extendido Gradient Descent en favor de la obtención de mejores

resultados con Adam o RMSProp. Por otra parte, el heurístico generalizado que proponía la actualización automática de la tasa de aprendizaje mediante estos optimizadores, se ha visto rebatido por el adecuado uso del decrecimiento exponencial, técnica que ha permitido aumentar hasta un 4 % la precisión resultante.

Tras ello, la elección de la función de activación entre las no lineales disponibles se debe tener en cuenta siendo Leaky ReLU la que ha permitido obtener mejores prestaciones. Asimismo, la incorporación de técnicas de regularización también es tema de debate. Con los datos obtenidos, se puede afirmar que la regularización L2 genera porcentajes nada despreciables, que se ven sutilmente afectados por la presencia o no de dropout, con mejoras significativas en los casos en los que se sufra sobreentrenamiento.

Esencial en redes neuronales es el análisis de sus capas ocultas, tanto del número de estas como de los nodos que las componen. Respecto a este hecho se ha observado un elevado aumento de complejidad con el número de capas, debido al consiguiente aumento exponencial de las posibles combinaciones. A partir de los resultados obtenidos, para los corpus definidos se ha concluido definitivamente que una capa oculta proporciona resultados suficientemente contundentes y precisos para garantizar la fiabilidad de la clasificación. Asimismo, un sistema con dos capas ocultas llega a mejorar estos resultados de manera sencilla en torno a un 2 % de precisión. Sin embargo, la adición de una tercera capa genera dificultades en el ajuste de los hiper-parámetros, de tal manera que la complicación aportada y el coste computacional que conlleva no compensa en el diseño de la red.

Finalmente, se puede concluir, que para ambos casos, las redes neuronales diseñadas haciendo uso de técnicas de Deep Learning son una buena alternativa para la clasificación de textos, y un ámbito en el que seguir profundizando para aplicaciones futuras.

A modo de recapitulación del procesado completo del dato, y como cierre final de la exposición de este proyecto, se exponen a continuación las diversas etapas en las que se ha basado su implementación y que debe seguir su aplicación a cualquier otro corpus:

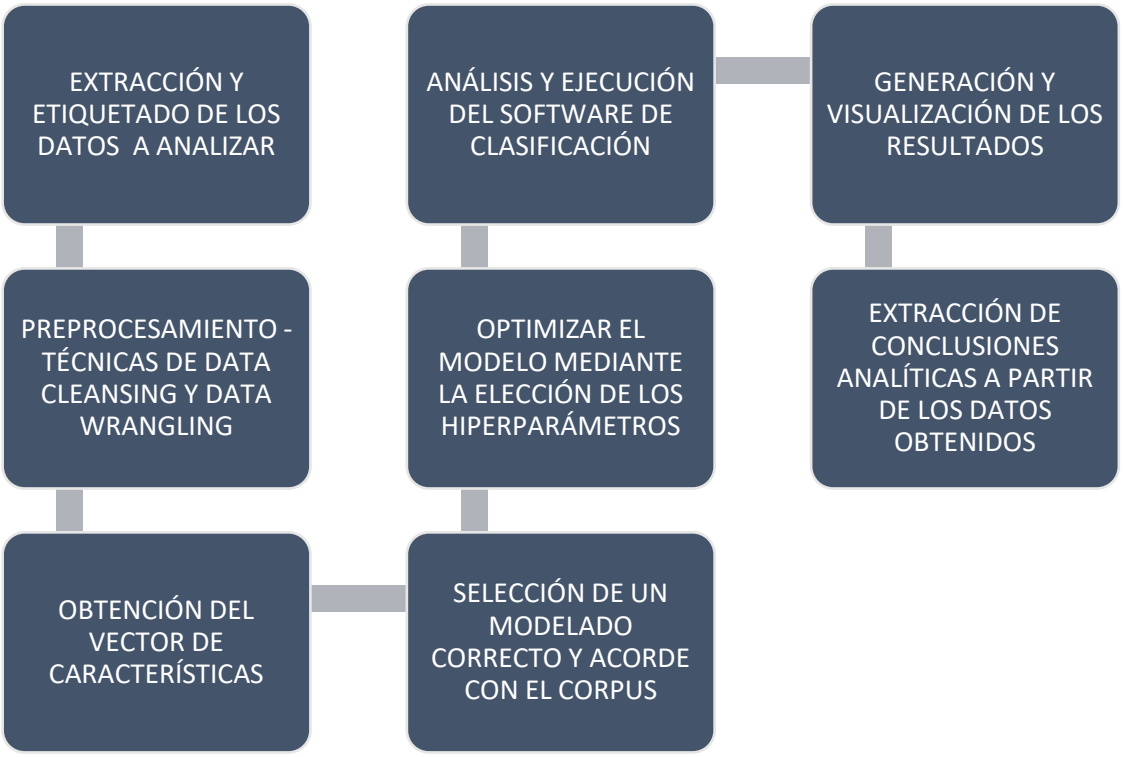


Fig. 6.1. Cuadro resumen de las fases del análisis de datos

6.3. Desarrollo de líneas futuras

La elaboración de cualquier proyecto presenta como limitación el hecho de conseguir abordar estrictamente los objetivos marcados con la intención de profundizar en dichos aspectos. Acotar las líneas de desarrollo favorece la claridad, la cohesión y la consistencia de argumentos para ese determinado estado del arte. Sin embargo, el análisis del problema permite abarcar nuevas técnicas o disciplinas que aportarían valor al trabajo, pero que se ha decidido no tratar al no haber sido aspectos contemplados en un principio.

Dado que el análisis de datos es un tema en plena efervescencia, la creación de nuevos algoritmos aplicables a la categorización o clasificación de textos es un terreno donde aún queda mucho por explorar. Con ayuda de la constante evolución de la tecnología, la implementación presentará todavía más facilidades, lo que permitirá centrar los esfuerzos en el desarrollo teórico y lógico. Entre las futuras líneas de estudio se podrían tratar los siguientes aspectos:

- Actualmente, el resultado consiste en la obtención de porcentajes que maximizan la evaluación de coincidencias, luego una de las principales líneas de trabajo, que actuaría como continuación inmediata de este proyecto, sería la implementación de una interfaz que permitiera la introducción de un documento o página web para su clasificación directa por el algoritmo.
- Incorporar a los clasificadores la tipología de clasificación "multioutput", en la que cada documento pueda pertenecer a más de una etiqueta. Esto aumentaría la complejidad y la consistencia de este proyecto, incorporando utilidades que no han sido alcanzadas en él.
- En el proyecto, se ha abordado la amplia gama de combinaciones que ofrecen las redes neuronales a partir de una bolsa de palabras. Al margen queda el tratamiento de los datos mediante Word Embeddings o LSTM, técnicas innovadoras que podrían proporcionar conclusiones distintas como posible línea de estudio.
- Profundización en los experimentos con sistemas hardware GPU que permitan la ejecución de algoritmos en paralelo y la simulación de capas con un mayor número de nodos ocultos.

Si focalizamos la atención en la clasificación de páginas web, posibles mejoras son:

- Afinar la extracción de contenidos mediante la araña web, delimitando los elementos HTML que sean más concretos y relevantes para la categorización, lo que permitiría recortar tiempos, evitar un preprocesado excesivo y obtener un vector de características más representativo.
- Generar una selección de stopwords y listas de lematizado propias de este lenguaje que permitan formar un corpus de mayor homogeneidad para la entrada al clasificador.

Realizar este proyecto ha requerido tomar decisiones que caracterizan el desarrollo del procesado dada la enorme proporción de combinaciones posibles. Por ello, otras propuestas menores para la obtención de mejores resultados podrían ser la evaluación de nuevas técnicas de preprocesamiento (como la eliminación de los tokens más frecuentes), la investigación de nuevos sistemas de representación de texto que ayuden a su entendimiento por el clasificador o el uso de redes convolucionales con nuevas combinaciones de hiper-parámetros.

ANEXO A – ENTORNO SOCIOECONÓMICO

A.1. Impacto socioeconómico

El grado de innovación que ha supuesto la inteligencia artificial, concretamente en la clasificación de textos, ha generado un cambio en la mentalidad de la sociedad y en la forma de pensar. Ha sido necesaria la consideración de nuevas leyes y de nuevas tendencias, en resumen, un cambio en la forma de actuar que ha llevado a numerosos sectores a hacer uso de estas técnicas para mejorar sus servicios, así como a la creación de nuevas disciplinas anteriormente inexistentes.

El análisis realizado en este proyecto podría tener distintas utilidades si se continúa su desarrollo. Su aplicación estará enfocada a la mejora de sistemas mediante el uso de los algoritmos expuestos.

Entre las aplicaciones más directas, podría resultar valioso como técnica de marketing, ya que, gracias a las respuestas obtenidas en el ámbito de análisis sentimental, una empresa podría ser capaz de detectar el éxito de sus productos o servicios a partir de comentarios publicados en redes sociales u otros portales de Internet. Utilizar redes neuronales con la parametrización adecuada aportará una mayor eficiencia, generando buenos resultados en la clasificación de dichos textos según los criterios establecidos.

Sin cambiar de ámbito, en el terreno empresarial la recomendación de documentos textuales sería otro factor a tener en cuenta, dado que una elevada precisión permitiría un mayor ajuste a los gustos y preferencias de los clientes para un trato más personalizado y ajustado a sus necesidades.

Por otra parte, la detección de spam se alza como otro de los casos por excelencia en este ámbito. Determinar qué correos son perjudiciales para la salud del sistema, y cuáles no, es un serio problema que actualmente puede solventarse mediante técnicas de clasificación automática como KNN, Naive Bayes o SVM, tal y como se describe en los procesos de adaptación del artículo [64]. Tras la investigación de este trabajo, el Deep Learning podría tomar las riendas de la resolución de estas cuestiones en detrimento de los algoritmos clásicos.

Desde un punto de vista sociológico, el estudio de esta rama y la necesidad de diseño e implementación de estos algoritmos generará nuevos puestos de trabajo, nuevas profesiones e incluso nuevos planes de estudio que incluyan estas disciplinas como temario. En resumen, este proyecto supone una profunda investigación de la influencia de factores en técnicas de Deep Learning, de manera que se considere un punto de partida para la creación de algoritmos que permitan afrontar las aplicaciones antes mencionadas, así como solventar nuevas necesidades emergentes.

A.2. Presupuesto del proyecto

Como medio para exponer el contexto económico que afecta al proyecto, se ha considerado efectivo realizar un análisis de los costes efectuados. En él, se incluyen tanto las horas invertidas por quienes forman parte del trabajo como los costes operativos de las herramientas utilizadas.

Se ha definido que la persona que elabora el proyecto presenta altos conocimientos en Python y destrezas que le permitan profundizar en el aprendizaje de software de inteligencia artificial durante su desarrollo. En la labor de abstracción para construir un presupuesto lo más fiel posible a la realidad, se han distinguido los roles que ejerce dicha persona y las horas empleadas en el proyecto por cada una de estas facetas, incluyendo perfiles técnicos como desarrollador y *tester* o perfiles de mayor perspectiva global como analista . Por otra parte, el tutor académico responsable del proyecto se le exige capacidad para la gestión de proyectos, así como un amplio conocimiento en el análisis de textos, siendo considerado jefe de proyecto.

A nivel económico para el cálculo de los costes se asume que el tutor dispone de un perfil de Ingeniero Senior con más de 15 años de experiencia, mientras que la persona encargada de su elaboración es considerada Ingeniero Junior sin experiencia profesional. Por tanto, la retribución económica asociada a las horas de trabajo de las personas que han formado parte del proyecto da lugar a los costes de personal humano, que son reflejados en la siguiente tabla:

Tabla A.1. COSTES DE PERSONAL

Rol	Coste por hora	Número de horas	Coste total
Jefe de proyecto - Tutor	35,00 €	40	1.400,00 €
Analista	25,00 €	150	3.150,00 €
Desarrollador	20,00 €	180	3.420,00 €
Pruebas (*Tester)*	20,00 €	35	595,00 €
Gestión de documentación	15,00 €	180	2.700,00 €
Total		**585**	**11.265,00 €**

Al salario de personal se le deben añadir los recursos utilizados para su elaboración, ya sean software o hardware, llevan asociados costes que ha sido necesario asumir y que se describen a continuación.

En el caso de los recursos software, cabe destacar que los acuerdos otorgados entre la universidad y distintas entidades de ámbito diverso han permitido acceder a estos productos de manera gratuita al utilizarse con fines académicos. En ellos se han incluido los costes generados por la obtención de documentación y fuentes de datos.

Tabla A.2. COSTES ASOCIADOS A RECURSOS SOFTWARE

Concepto	Coste
Distribución Anaconda 3 – Jupyter Notebook	0,00 €
Python: Librerías analíticas de libre acceso	0,00 €
Microsoft Office 2016	Licencia UC3M – 0,00 €
Servicios Drive de almacenamiento en la nube	Licencia UC3M – 0,00 €
Fuentes de extracción de datos	0,00 €
Documentación – Bases de datos de ingeniería y fuentes bibliográficas en papel	Licencia UC3M – 0,00 €
Documentación – Bibliotecas virtuales	0,00 €
Curso online Deep Learning en TensorFlow	0,00 €
Windows 10 Anniversary	0,00 €
Total	**0,00 €**

Por otra parte, en los recursos hardware, el alto consumo al que se ve sometido el ordenador ejecutando las herramientas de análisis de datos ha hecho que, con el objetivo de no ralentizar la obtención de resultados, se utilicen dos ordenadores para trabajar en paralelo. Uno de ellos, únicamente para la ejecución de pruebas y experimentos, mientras el otro se utilizaría para llevar a cabo las tareas oportunas de gestión, documentación, investigación y previa implementación.

En el cálculo de estos costes se ha tenido en cuenta el coste imputable de cada uno de los elementos respecto al período de utilización que ha tenido en la elaboración del proyecto y su período de amortización, es decir, el número de años durante los que se amortizan los bienes de inmovilizado material, valor que suele corresponder con la vida útil. Asumiendo que se ha utilizado una única unidad de cada uno de los recursos y que durante el tiempo efectivo se ha hecho uso de ellos a máximo rendimiento, su cómputo ha seguido la siguiente expresión [65]:

$$Coste\ de\ amortización\ imputable = \frac{coste\ unitario\ \times\ tiempo\ efectivo}{período\ de\ amortización} \qquad (A.1)$$

Tabla A.3. COSTES ASOCIADOS A RECURSOS HARDWARE

Concepto	Coste unitario	Período de amortización	Tiempo efectivo	Coste imputable
Ordenador Portátil HP EliteBook Intel i7 / 1TB	1300,00 €	30 meses	2,33 meses	100,97 €
Ordenador portátil HP Intel i3 / 500 MB	500,00 €	24 meses	8 meses	166,67 €
Ratón inalámbrico Logitech, nano transceptor plug-and-go	25,00 €	15 meses	7,5 meses	12,5 €
Total	**1825,00 €**			**280,14 €**

Para el cómputo total de los costes del proyecto se han tenido en cuenta costes indirectos como electricidad o conexión a Internet, fijando su valor en aproximadamente un 10% de los costes directos, como se define a continuación :

Tabla A.4. RESUMEN COSTES DIRECTOS

Costes directos	
Costes de personal	11.265,00 €
Costes asociados a recursos software	0,00 €
Costes asociados a recursos hardware	280,14 €
Total	**11.545,14 €**

$$Costes\ indirectos = Costes\ directos \times 0.10 = 1.154{,}51\ €$$

Finalmente, se ha decidido no incluir en este estudio los costes derivados por logística, desplazamiento para las reuniones con el tutor o dietas, al considerarse despreciables.

Concluyendo con la visión global de los costes de este proyecto, al coste amortizado le debe ser añadido el IVA correspondiente, no incluido en el cálculo de amortizaciones anterior (21%), luego los costes totales han sido resumidos en la siguiente tabla:

Tabla A.5. RESUMEN COSTES TOTALES

COSTES TOTALES	
Costes directos	11.545,14 €
Costes indirectos	1.154,51 €
Base imponible	**12.699,65 €**
Total con IVA	**15.366,58 €**

ANEXO B – RESUMEN DE CONTENIDOS EN INGLÉS

B.1. Abstract

Given the extensive growth experienced by information today, its management and analysis has become essential to its application in any field of society, providing an incremental value to the data. This change in attitude raises many issues that require automatic learning techniques for its resolution. The automatic classification of texts promotes the development of many of them, attractive which is the reason of having developed an entire system that simulates the processing of textual data, from the extraction to the final stage, the categorization.

To carry out this process, this project includes numerous techniques and technologies, both natural language processing and neural networks construction, which have made possible its implementation. A web spider creation will be the starting point of this process, which will continue with the development of the preprocessing phase, techniques of data cleansing to select relevant information which will facilitate the operation of the final objective, the classifier. In this last stage, Deep Learning techniques will be applied to implement solutions, while a comparison between benefits obtained is established.

Implementing the exposed circuit will need prior research on the current situation of the matter in question, including the existing methods and their respective advantages and limitations, as well as the theoretical principles that give sustenance to their development and applications related to this problem.

In conclusion, it will be analyzed the influence of the neural networks in the problems of text classification through alteration of parameters that compose them, and their usefulness regarding classic machine learning algorithms.

Keywords: Deep Learning; Text Classification; Artificial Neural Network; Web Spider; TensorFlow

B.2. State of art

The digital revolution in which we are immersed has made that each of the data generated daily was something essential for its analysis, which brings an added value to the huge amounts of data produced across the planet. The digitalization of information in practically any action that we carry out, like banking operations, repositories in the cloud or electronic commerce, has provided the capacity to draw conclusions on the data, so that the physical world is seen reflected and modelled through an integral representation in the digital world.

Aware of the digitized society in which we are and the global position of everything that happens around us, in recent years we have promoted the technological interconnection in a massive way. The phenomenon of analysis of this amount of information solve problems of classification, prediction or optimization, being a machine which generate the results. This approach has resulted in the concept of artificial intelligence. There are numerous studies where technology is based on the human component to try to impregnate the devices we deal with every day, a fundamental point in the development of the Internet of Things.

Among the first algorithms developed for its application in the data analysis highlighted the Bayesian networks, where, following a probabilistic model, the conditional dependencies existing between data can be determined by an acyclic graph

Another algorithm to consider is learning through decision trees. Predictions are established building a path that go through the rules followed at each level of the tree. To conclude with this introduction to existing mechanisms, it is worth highlighting the learning through association rules, which uses data sets to extract semantic relationships between variables that can be used as prediction of future corpus . The implementation of each of these methods was a milestone in the field of artificial intelligence, but it was not until the second half of the twentieth century when the emergence of neural networks revolutionized the concept.

A neural network consists on the introduction of a series of entries in neurons that emit an output determined by three functions:

- **Propagation**: Regulator of the signals emitted by the computation of the weight sum.

- **Activation**: Its objective is to modify the results from the Propagation function in order to get values in a given range.
- **Transfer**: Denotes the output value of each neuron according to the interpretation you want to receive from it, acting as a threshold for the acceptance of the final values.

Finally, for a non-existent output function case, the output 'y' will be determined by the expression:

$$y = f\left(b + \sum_{n} w_n x_n\right)$$

Analyzing studies published since the beginning of this century, it can be said that the classifier of texts par excellence in the experiments carried out is SVM with their respective adaptations. Its use is widely extended in textual corpus, for which Naive-Bayes are also usually applied, which allows for reliable results, and *Random Forest*; all of them algorithms referring to the classic conception of Machine Learning.

In the many references consulted, there is a constant attempt to adapt the algorithms to the corpus analyzed, which gives rise to heuristics on the reliability of its use in future cases. On the other hand, a common aspect of the previous research of data is the processing by cleansing techniques (tokenization, filtering, stemming...) that allow the increase in accuracy of the experimental results.

In some articles of 2015, it is established a relationship between traditional algorithms such as word bags and recent Deep Learning techniques using convolutional networks. A greater depth in these techniques is appreciated in the comparative established in 2017 between algorithms of automatic learning, where the categorization of the corpus of IMDB reviews and tweets serve to compare algorithms of machine learning like Naive-Bayes and Deep Learning as LSTM, multilayer neural network or CNN, getting quite similar results, but highlighting these last. It is interesting to point out that related aspects with multilabel processing were investigated in 2017 reflecting the innovative techniques used.

Also noteworthy is the current approach that Deep Learning is receiving with the techniques of Word Embeddings, allowing to cover a growing number of areas of knowledge given the effectiveness of method like text translation.

In addition to the generic references seen previously, the introduction in the field of classification of web pages was made in 2007, where Machine Learning techniques are used to classify pages, but only based on the information of the URL, aspect that has been improved in subsequent studies by deep neural networks like the development of this project.

B.3. Data extraction and creation of features vector

In the classification, it was used Reuters corpus containing a huge number of natural language articles, however, as a method of extension of the developed algorithm, it was decided to use a corpus that was not preprocessed in advance, that allowed to choose what is interested to analyze of it and thus to provide greater flexibility.

For the creation of this corpus it was necessary the previous extraction of documents HTML by means of the package of tools Scrapy and complete the process of labelling. Starting from this point, the process of creation in Python was based on the use of a structure of files that contained the obtained results.

Unlike the Reuters corpus, the data obtained through the spider have not been previously treated, which implies carrying out the labelling process. At this point, it was necessary setting the categories to which each URL belonged, establishing seven and eight, depending on the corpus:

Blog – Business – eCommerce – Parking – Multilink – Others – Error

Material – Energy – Commerce – Nature – Economy – Acquisitions – Earnings

For doing this, it was decided to use a labeler from the GitHub. Implemented in Python and executed from the Anaconda console, it has a basic interface that allows to select the corresponding category of the URL analyzed at that time, being registered in a document generated as a history. This allows greater efficiency in the process by reducing the time spent at this stage.

Once the process of labelling was finished, in order to structure the words (tokens) stored in the dictionaries, we made use of the technique: one-hot-encoding. The effectiveness of this method resides in the association of binary values to the dataset. In this case, the labels would be encoded with values 0 and 1, so that the document belongs

to that category and not to the others. In addition to this, some preprocessing techniques must be carrying out for the correct function of the classifiers. These methods are:

- Tokenization: Procedure that divides into tokens the text structure.
- Filtering: Technique that remove non-alphanumeric characters.
- Stemming and lemmatization: Cleaning methods for extract the root of each word
- Stopwords: Phase for removing words not representative for the analysis of texts.

After Data Cleansing techniques we proceed to vectorize the data for better treatment. There are many mechanisms for the creation of the feature vector, as it is called the set of elements from which information is obtained in the analysis. In this project, it has been chosen to evaluate the benefits of the Bag of Words, with all the variants that this can propitiate in the implementation of the classifiers

Based on this technique, a corpus of tuples has been developed, where each of them consists of: [token identifier, number of times repeated in each document]. For doing this, the generated corpus was transformed in a numerical representation that provided an identifier to each existing token in all the textual units of the dataset, being picked up in a dictionary. Python dictionaries are advantageous in data access and automatically avoid the repetition of existing tokens. This allows search to be automatic through these indexes without the need to implement nested loops that traverse the structure through multiple iterations.

B.4. Solution development and classifiers implementation

Having a tuple vector for data processing generated performance deficiencies by including redundant information. To optimize this operation, and as part of the process of adaptation of the data to the development to be implemented, it was designed like structure of entry to the classifiers a numerical matrix in Python through the Numpy library, organized so that the rows correspond with each of the full corpus tokens, and each column do the same with the documents that form it

By way of contextualizing the future results, generated by neural networks, it has been decided to implement two of these classifiers and analyze their benefits. His choice

has been based on the study of the state of art, where on the many existing possibilities stand out two, Naive-Bayes and SVM.

Naive-Bayes is an algorithm based on the principle of maximum a posteriori (MAP) which is characterized by assuming the independence of each of the elements of the features vector. By focusing the analysis on the multiclass model, the frequency of tokens in documents is used as information for its study.

The SVM algorithm consists of the representation of the mapped data as points in the associated space according to the category to which they belong. One or more hyper-planes are built in an n-dimensional space that are used to delimit the decision areas, maximizing the minimum distance from the plane to the nearest sample to minimize the possible error generated.

The simplest multiclass prediction algorithm using Machine Learning is the classifier based on multinomial logistic regression. Its difference, and advantage, with respect to linear regression is its distribution in the range [0.1] for the values of the x-axis, unlike the range [-∞, ∞], which allows to operate with probabilities correctly. To do this, the logit function is used as a modeling technique, so that the resulting equation computed for solving p (**x**) would have the following form:

$$w^T x + b = y$$

Once this procedure has been completed, Tensorflow is able to obtain computationally the operation of Softmax, which was used as the output function of the last neuronal layer. It was executed with cross-entropy mechanism, technique whose purpose is to analyze the error generated in the execution so that it quantifies the difference between the probability distributions generated by the software. Finally, an optimizer was used to minimize the value of this loss, which was carried out by the method of Gradient Descent as a first choice.

The evolution of the concept of regression to artificial neuron needed to incorporate new elements into the network, the hidden layers, intermediate layers which together with the procedure of backpropagation create the foundations of this new philosophy of data analysis. This stage can include multiple sublayers that would provide depth to the neural network training. The aim to elaborate this mechanism is to apply to these hidden layers

nonlinear functions as function of activation of the neurons, so that they allow a better adaptation of the neurons to the set of data analyzed in the training phase.

In addition to this, there are two regularization techniques (L1 and L2) that base their differences on mathematical concepts: L1 corresponds to the exact sum of the weights and L2 with the sum of the weights squared. Considering an initial state in which all artificial neurons are connected to each other, it is logical to think that the number of activations (connections capable of transmitting information) is going to be a high value. To obtain more proportional learning, the technique consists in randomly disable, for each iteration, some connections, which allows to train the classifier avoiding the excess of dependence of some neurons for fight against overfitting.

To conclude, in this project, given the difficulty that TensorFlow presents to implement cross-validation, not having its own function as Sklearn, it has been decided made the following structure: a 60% distribution has been used for training data, 20% for test and the other 20 % for validation. As explained above, test set determines the effectiveness of the algorithm in the final precision while the validation set guarantees its scalability and its generalization.

B.5. Performance evaluation

The use of a neural network as a classifier opens countless opportunities to improve the results using any of the many techniques that have been explained. With this, we sought to demonstrate good precision and efficiency for both corpora using Deep Learning.

Although Gradient descent (GD) is the most widespread technique for its old use in algorithms, it does not have to be the most effective one. As can be seen in most cases, the results allow to detect an upward trend of loss as the number of hidden nodes increases, and the same happens with batches. This last factor generates a great relevance in the computational cost since when the size of the batch is less the execution becomes more agile and dynamic.

On the other hand, in some cases a decrease in the validation percentage is detected when the values of the batch size and the number of nodes is high. This can be interpreted as overtraining of the data generating a low generalization. Following its exhaustive

analysis, there is no doubt that, for the conditions of this experiment, the best results are obtained through the use of RMS and Adam optimizers, which provide benefits far above the rest with correct percentages compared to these. Although the trend is correct, using exponential degrowth provides high accuracy percentages in a smaller number of iterations. Finally, the use of LReLU as a function of activating neurons is based on the previous realization of experiments with ReLU that provide worse results, as with sigmoid, with validation data were around 8% and 5% lower.

It can be concluded, therefore, that the best option for the classifier would consist of the optimizer RMSProp with 20 minibatches of 500 elements each one with hidden layers of 30 and 15 nodes respectively. The presence of a high number of neurons in one of the layers generates a precision in the validation data lower to the test data. Is the reason that makes choosing a more equitable number of neurons.

The experiments made clear that the optimizer par excellence for the text classification is RMSProp. In addition to this, the experiment with 3 hidden layers generate results that were not worthy to consider.

The probability of connection removing, that is defined by the *Dropout*, must be fixed in the classification software, being 0.5 the default value. As can be seen, the effect of dropout is slightly appreciable in the corpus of Reuters, by improving the validation results less than 1 percentage point, and produces a greater improvement in the reference to HTML documents, which is possible given its low results in the previous analysis, increasing up to 4.5% the data obtained. The final results are shown in the following table:

Corpus Reuters

	SVM	**N-B**	**Log. Reg.**	**DNN 1**	**DNN 2 Dropout**
Training	95.6 %	92.4 %	91.8 %	98.0 %	98.6 %
Test	91.3 %	90.7 %	87.9 %	93.0 %	93.9 %
Validation	91.6 %	89.4 %	91.2 %	93.3 %	94.7 %

Corpus HTML

	SVM	N-B	Log. Reg.	DNN 1	DNN 2 Dropout
Training	94.7 %	68.9 %	94.0 %	87.2 %	97.9 %
Test	52.2 %	45.6 %	58.3 %	51.1%	61.5 %
Validation	57.6 %	48.4 %	56.9 %	51.6 %	64.8 %

In spite of the good results of SVM and Naïve Bayes, if we are technically purists, the percentages of accuracy obtained by artificial neural networks are slightly higher in the case of Reuters and note a noticeable increase in the case of Web pages, within the low precision manifested.

B.6. Conclusions

The research carried out in this work, as well as the data and techniques compiled here, are an important basis for the field of automatic classification of texts.

Through the phases of data processing, first introduced the concept of Web spider, as a method capable of extracting documents from Web pages in HTML language. It is concluded that the extraction by Scrapy technology is effective to carry out this process and can select the aspects of interest that will influence the features vector. The attractiveness of the classification of Web pages in this project resides in the verification of whether to use natural language techniques is useful for texts in which other many characters of the HTML language coexist. After this research, it is concluded that using NLP is applicable to corpus of HTML documents in a satisfactory way, even though these must cross special preprocessing techniques, such as the consideration of specific stopwords.

In general, regardless of the content of the documents, any classification algorithm receives a features vector that identifies the input data. These previously have had to undergo Data Cleansing and Data Wrangling techniques, preprocessing phases where tokenization, filtering, stemming or elimination of stopwords are applied, in order to extrapolate the most important features of the documents to be analyzed and to apply the appropriate format for incorporation into the classifier.

Already within the scope of the classification algorithms, in both corpora, it has been observed that supervised classifiers such as SVM or Naive-Bayes, classic methods of machine learning, provide admirable precision results in many cases, leading the algorithms of this category. In particular, for the analysis of the corpus of Reuters (natural language) it is shown that the precision obtained is enough high to create a reliable system.

The summit of the study is achieved with the analysis of techniques of Deep Learning, creating a clear image of the parameters that is useful to treat and how that treatment can be favorable. The study of optimizers diversifies the exploration of performance with the different hyper-parameters, so its importance derives in the choice of a *Learning Rate* appropriate to the classification. This has allowed to leave the comfort zone that provides the extended Gradient Descent on behalf of obtaining better results with Adam or RMSProp. On the other hand, the generalized heuristic that proposed the automatic updating of the learning rate through these optimizers, has been confronted by the correct use of exponential degrowth, technique that has allowed to increase up to 4% the final precision.

After that, the choice of the function of activation between the available non-linear ones should be considered, being Leaky ReLU which has allowed to obtain better benefits. Furthermore, the incorporation of regularization techniques is also a subject of discussion. With the obtained data, it can be said that the regularization L2 generates correct percentages, which are subtly affected by the presence or not of dropout, with significant improvements in cases in which overtraining is suffered.

Essential in neural networks is the analysis of their hidden layers, the number of these and the nodes that compose them. With regard to this fact has been observed a high increase in complexity with the number of layers, due to the resulting exponential increase of the possible combinations. From the obtained results, it has been definitively concluded that a hidden layer provides sufficiently precise results to guarantee the reliability of the classification. Also, a system with two hidden layers will improve these results in a simple way around 2% accuracy. However, the addition of a third layer generates difficulties in the adjustment of the hyper-parameters, in such a way that the complication contributed and the computational cost that it implies does not compensate the design of the network.

Finally, it can be concluded that, for both cases, the neural networks designed using Deep Learning techniques are a good alternative for text classification, an area in which there are so much to continue researching for future applications.

BIBLIOGRAFÍA

[1] A. Monleón-Getino, «El impacto del Big-data en la Sociedad de la Información. Significado y utilidad,» *Historia y Comunicación Social*, vol. 20, nº 2, pp. 427-445, 2015 [En línea]. Disponible en: revistas.ucm.es/index.php/HICS/article/download/51392/47672 .

[2] J. Batalla, G. Mastorakis, C. Mavromoustakis and E. Pallis, "Technologies and Challenges in Internet of Everything Environments," in *Beyond the Internet of Things*, Springer, 2017.

[3] I. Ben-Gal, «Bayesian Networks,» *Encyclopedia of Statistics in Quality & Reliability*, 2007 [En línea]. Disponible en: http://www.eng.tau.ac.il/~bengal/BN.pdf .

[4] S. B. Kotsiantis, «Decision trees: a recent overview,» *Artificial Intelligence Review*, vol. 39, nº 4, pp. 261-283, 2013 [En línea]. Disponible en: https://link.springer.com/article/10.1007/s10462-011-9272-4 .

[5] F. Rosenblatt, "Perceptron Simulation Experiments," in *Proceedings of the IRE*, vol. 48, IEEE, 1960, pp. 301-309.

[6] M. Minsky y S. Papert, Perceptrons: an introduction to computational geometry, 1969.

[7] P. Werbos, «Beyond regression : new tools for prediction and analysis in the behavioral,» 1974. [En línea]. Disponible en: https://www.researchgate.net/publication/35657389_Beyond_regression_new_tools_for_prediction_and_analysis_in_the_behavioral_sciences.

[8] D. E. Rumelhart, G. E. Hinton y R. J. Williams, «Learning representations by back-propagating errors,» *Nature*, pp. 533-536, 323.

[9] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever y R. Salakhutdinov, «Dropout: A Simple Way to Prevent Neural Networks from Overfitting,» *Journal of Machine Learning Research 15*, 2014.

[10] Universidad Tecnológica Nacional - Facultad Regional de Rosario, «Redes Neuronales: Conceptos Básicos y Aplicaciones,» 2001. [En línea]. Disponible en: https://www.frro.utn.edu.ar/repositorio/catedras/quimica/5_anio/orientadora1/monografias/matich-redesneuronales.pdf.

[11] «Redes neuronales,» [En línea]. Disponible en: http://www.redes-neuronales.com.es/. [Último acceso: 2018].

[12] «Advanced Tech Computing Group UTPL,» [En línea]. Disponible en: https://advancedtech.wordpress.com/2007/08/31/elementos-basicos-de-una-red-neuronal-artificial/. [Último acceso: 2018].

[13] L. S. Larkey, «A Patent Search and Classification System,» de *Proceedings of the fourth ACM conference on Digital libraries*, Berkeley, 1999 [En línea]. Disponible en: https://dl.acm.org/citation.cfm?id=313304 .

[14] T. Joachims, «Transductive Inference for Text Classification using SVM,» de *Proceedings of the Sixteenth International Conference on Machine Learning*, 1999 [En línea]. Disponible en: https://www.cs.cornell.edu/people/tj/publications/joachims_99c.pdf.

[15] A. McCallum y K. Nigam, «A Comparison of Event Models for Naive Bayes Text Classification,» de *Learning for Text Categorization: Pa*pers from the 1998 AAAI Workshop, 1998 [En línea]. Disponible en: www.cs.cmu.edu/~knigam/papers/multinomial-aaaiws98.pdf .

[16] K.-M. Schneider, «A Comparison of Event Models for Naive-Bayes Anti-Spam E-Mail Filtering,» de *Proceedings of the tenth* conference on European chapter of the Association for Computational Linguistics , Budapest, 2003 [En línea]. Disponible en: www.aclweb.org/anthology/E03-1059 .

[17] B. Li, N. Sugandh, E. V. Garcia y A. Ram, «Adapting Associative Classification to Text Categorization,» de *ACM symposium on Document engineering*, Winnipeg, 2007, doi: 10.1145/1284420.1284470 .

[18] J. Wang y G. Karypis, «HARMONY: Efficiently Mining the Best Rules for Classification,» de *SIAM International Conference on Data Mining*, 2005, doi: 10.1137/1.9781611972757.19 .

[19] V. Nastase y M. Strube, «Transforming Wikipedia into a large scale multilingual concept network,» Artificial Intelligence, nº 194, pp. 62-85, 2012, doi: 10.1016/j.artint.2012.06.008.

[20] A. Pai, «Tensorflow Text Classification – Python Deep Learning,» Source Dexter, 12 Octubre 2017. [En línea]. Disponible en: https://sourcedexter.com/tensorflow-text-classification-python/.

[21] X. Zhang, J. Zhao y Y. LeCun, «Character-level Convolutional Networks for Text Classification,» de *NIPS Proceedings*, New York, 2015, doi: arXiv:1509.01626.

[22] M. Amajd, «Text Classification with Deep Neural Networks,» de *Fifth International Conference of Actual Problemas of System and Software Engineering*, Moscow, 2017 [En línea]. Disponible en: https://amaaz.github.io/Paper.pdf .

[23] J. Liu, W.-C. Chang, Y. Wu and Y. Yang, "Deep Learning for Extreme Multi-label Text Classification," in *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, Tokyo, 2017, doi: 10.1145/3077136.3080834 .

[24] M. H. e. al., «Deep learning with word embeddings improves biomedical named entity recognition,» *Bioinformatics*, vol. 33, nº 14, pp. i37-i48, 2017, doi: 10.1093/bioinformatics/btx228 .

[25] B. Gupta y et al., «Study of Twitter Sentiment Analysis using Machine Learning Algorithms on Python,» *International Journal of Computer Applications*, vol. 165, nº 9, pp. 29-34, 2017, doi: 10.5120/ijca2017914022.

[26] A. Abdullah, R. Veltkamp y M. A. Wiering, «An Ensemble of Deep Support Vector Machines for Image Classification,» de *International Conference of Soft Computing and Pattern Recognition*, 2009, doi: 10.1109/SoCPaR.2009.67.

[27] V. Kocian y E. Volná, «Ensembles of Neural-Networks-Based Classifiers, University of Ostrava,» 2012, doi: 10.1142/S0129065712003079

[28] C. G. Rubio, «Clasificación automática de texto para el seguimiento de campañas electorales en redes sociales,» Trabajo de Fin de Grado, Univ. Carlos III Madrid, 2015.

[29] N. C. Gil, «Sistema basado en redes neuronales para el reconocimiento de dígitos manuscritos,» Trabajo de Fin de Grado, Univ. Carlos III Madrid, 2012 .

[30] M. I. Devi, R. Rajaram y K. Selvakuberan, «Machine Learning Techniques for Automated Web Page Classification using URL Features,» de *International Conference on Computational Intelligence and Multimedia Applications*, 2007, doi: 10.1109/ICCIMA.2007.342..

[31] B. P. Ajay S. Patil, «Automated Classification of Web Sites using Naive Bayesian Algorithm,» *Proceedings of the International MultiConference of Engineers and*

Computer Scientists, vol. 1, 2012, [En línea]. Disponible en: http://www.iaeng.org/publication/IMECS2012/IMECS2012_pp519-523.pdf

[32] Scientific Research Group in Egypt, «Machine Learning Algorithms in Web Page Classification,» *International Journal of Computer Science & Information Technology*, vol. 4, nº 5, 2012, doi: 10.5121/ijcsit.2012.4508. 93.

[33] A. B. e. al., «Machine Learning for Automatic Classification of Web Service Interface Descriptions,» 2012, doi: 10.1007/978-3-642-34781-8_17.

[34] A. A. AbdulHussien, «Comparison of Machine Learning Algorithms to Classify Web Pages,» *International Journal of Advanced Computer Science and Applications,* vol. 8, nº 11, 2017, doi: 10.1080/23742917.2017.1321891.

[35] V. Krotov and L. Silva, "Legality and Ethics of Web Scraping," in *Twenty-fourth Americas Conference on Information Systems,* New Orleans, 2018 [En línea]. Dispomible en: https://www.researchgate.net/publication/324907302_Legality_and_Ethics_of_Web_Scraping

[36] «NLTK Documentation,» 2017. [En línea]. Disponible en: https://www.nltk.org/.

[37] L. Huang, «Measuring Similarity Between Texts in Python,» [En línea]. Disponible en: https://sites.temple.edu/tudsc/2017/03/30/measuring-similarity-between-texts-in-python/. [Accedido: 30-Marzo-2017]

[38] E. Loper y S. Bird, «NLTK: the Natural Language Toolkit,» de *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing,* Philadelphia, 2002, doi: 10.3115/1118108.1118117.

[39] «Gensim - Topic modelling for humans,». [En línea]. Disponible en: https://radimrehurek.com/gensim/tutorial.html.

[40] S. Li, «Topic Modelling in Python with NLTK and Gensim,» [En línea]. Disponible en: https://towardsdatascience.com/topic-modelling-in-python-with-nltk-and-gensim-4ef03213cd21.

[41] Ministerio de energía, turismo y agenda digital, «Red.es,» [En línea]. Disponible en: http://www.dominios.es/dominios/es/todo-lo-que-necesitas-saber/estadisticas.

[42] R. Mitchell, Web Scraping with Python, O'Reilly, 2015.

[43] «Internet Explorer User Agents,» [En línea]. Disponible en: https://developers.whatismybrowser.com/useragents/explore/software_name/internet-explorer/.

[44] C. V. Blockeel, J. Struyf, L. S. Hendrik y S. Džeroski, «Decision trees for hierarchical multi-label classification».

[45] UC3M - Departamento Teoría de la Señal y Comunicaciones, «GitHub - Label Factory,». [En línea]. Disponible en: https://github.com/Orieus/labelFactory . [Accedido 20-12-2017]

[46] Udacity , «Deep Learning by Google,». [En línea]. Available: https://classroom.udacity.com/courses/ud730. [Accedido 08-2017]

[47] S. Skansi, «Mathematical and Computational Prerequisites,» de *Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence*, Springer, 2018, pp. 17-25.

[48] P. Wang, C. Domeniconi y J. Hu, «Cross-domain Text Classification using Wikipedia,» *IEEE Intelligent Informatics Bulletin,* vol. 9, nº 1, pp. 5-17, 2008 [En línea]. Disponible en: https://pdfs.semanticscholar.org/9a06/232b7268a606ceac4c406ce8e2154c44f78d.pdf .

[49] E. Bressert, SciPy and Numpy: An overview for developers, O'Reilly, 2012.

[50] «Tensorflow,» 2017. [En línea]. Disponible en: https://www.tensorflow.org/programmers_guide/.

[51] S. Kandel, A. Paepcke, J. Hellerstein y J. Heer, «Wrangler: Interactive Visual Specification of Data Transformation Scripts,» de *ACM Human Factors in Computing Systems*, 2011, doi: 10.1007/s007780100057.

[52] Scikit-Learn, «SVM algorithm,» [En línea]. Disponible en: http://scikit-learn.org/stable/modules/svm.html.

[53] F. A. Llaugel y A. I. Fernández, «Evaluación del uso de modelos de regresión logística para el diagnóstico de instituciones financieras,» *Ciencia y Sociedad,* vol. XXXVI, nº 4, pp. 590-627, 2011, doi: 10.22206/cys.2011.v36i4.pp590-627.

[54] K. Hornik, «Multilayer Feedforward Networks are Universal Approximators,» *Neural Networks,* vol. 2, pp. 359-366, 1989, doi: 10.1016/0893-6080(89)90020-8.

[55] TowardsDataScience, «Activation Functions Neural Networks,» [En línea]. Disponible en: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[56] S. Hochreiter, Y. Bengio, P. Frasconi y J. Schmidhuber, «Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies,» [En línea]. Disponible en: http://www.bioinf.jku.at/publications/older/ch7.pdf.

[57] P. Ramachandran, B. Zoph y Q. V. Le, «Searching for Activation Functions,» 2017. [En línea]. Disponible en: https://arxiv.org/pdf/1710.05941.pdf.

[58] A. L. Maas, A. Y. Hannun y A. Y. Ng, «Rectifier Nonlinearities Improve Neural Network Acoustic Models,» Stanford University, 2014, [En línea]. Disponible en: https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf

[59] D. P. Kingma y J. L. Ba, «Adam: A Method For Stochastic Optimization,» de *International Conference for Learning Representations*, San Diego, 2015, doi: arXiv:1412.6980 .

[60] L. N. Smith, «Cyclical Learning Rates for Training Neural Networks,» de *IEEE Winter Conference on Applications of Computer Vision (WACV)*, Santa Rosa, CA, USA, 2017, doi: 10.1109/WACV.2017.58.

[61] S. Ruder, «An overview of gradient descent optimization algorithms,» 2016. [En línea]. Disponible en: https://arxiv.org/abs/1609.04747.

[62] J. McCaffrey, "Test Run - L1 and L2 Regularization for Machine Learning," Microsoft, 2015. [En línea]. Disponible en: https://msdn.microsoft.com/en-us/magazine/dn904675.aspx.

[63] J. Heaton, Introduction to Neural Networks for Java, Heaton Research, 2008.

[64] N. Sharma, G. Kaur y A. Verma, «Survey on Text Classification (Spam) Using Machine Learning,» *International Journal of Computer Science and Information Technologies*, vol. 4, pp. 5098-5102, 2014, [En línea]. Disponible en: https://pdfs.semanticscholar.org/1747/730fa3b5f2d7904ff10884a644e756203122.pdf

[65] C. M. Rodriguez y A. R. Nicolau, Contabilidad de dirección para la toma de decisiones, Profit Editorial, 2014.