Universidad Internacional de La Rioja
Escuela Superior de Ingeniería y Tecnología

Máster Universitario en Inteligencia Artificial

Comparativa de modelos de aprendizaje profundo para la detección de odio en castellano en medios de información social.

Trabajo fin de estudio presentado por:	Carlos Simón Gallego
Tipo de trabajo:	Comparativa de soluciones
Director/a:	Almudena Ruiz-Iniesta
Fecha:	08/02/2023

Resumen

Con este trabajo tratamos de determinar la viabilidad que existe en la detección automática de expresiones de odio en castellano mediante la aplicación de Deep Learning (DL) sobre el dataset del proyecto Hatemedia[1]. Para ello realizamos una comparativa de soluciones para determinar qué modelo de DL ofrece mejor rendimiento para esta tarea. Se han realizado las mismas pruebas con diferentes versiones del dataset; una versión con todos los registros y otras versiones reducidas para intentar solventar los problemas derivados del desbalanceo de clases. Las pruebas realizadas para los datasets balanceados exploran distintas casuísticas en base a criterios como la longitud de los textos o el uso de textos pertenecientes a un mismo medio, con el fin de entender si estas variables tienen importancia en el rendimiento de los modelos. Tras el trabajo comparativo, encontramos que el dataset original resulta inservible debido al problema del desbalanceo de clases, ocasionando que todos los modelos acaben prediciendo únicamente la clase dominante, obteniendo un 98% de accuracy pero un 0% de recall para la clase minoritaria. Si nos centramos en las pruebas con los datasets balanceados, el modelo BETO (versión cased) es el que mejor rendimiento ofrece, superando los resultados obtenidos por otros modelos del estado del arte entrenados con diferentes datasets. Finalizamos exponiendo todas las dificultades encontradas y ofreciendo alternativas de mejora para trabajos futuros.

El presente trabajo ha sido realizado dentro del proyecto: “Taxonomía, presencia e intensidad de las expresiones de odio en entornos digitales vinculados a los medios informativos profesionales españoles – Hatemedia”. Proyecto PID2020-114584GB-I00, financiado por la Agencia Estatal de Investigación - Ministerio de Ciencia e Innovación.

Palabras Clave: Discurso de odio, Aprendizaje profundo, Aprendizaje por transferencia, BETO, Procesamiento de lenguaje natural, Clasificación de texto

[1] https://www.hatemedia.es/

Abstract

With this work we try to determine the feasibility of the automatic detection of hate speech in Spanish by applying Deep Learning (DL) on the dataset of the Hatemedia project. For this purpose, we carried out a comparison of solutions to determine which DL model offers the best performance for this task. The same tests have been carried out with different versions of the dataset; one version with all the records and other reduced versions to try to solve the problems derived from class imbalance. The tests carried out for the balanced datasets explore different cases based on criteria such as the length of the texts or the use of texts belonging to the same medium, in order to understand whether these variables are important in the performance of the models. After the comparative work, we find that the original dataset is useless due to the class imbalance problem, which makes all the models end up predicting only the dominant class, obtaining 98% accuracy but 0% recall for the minority class. If we focus on the tests with the balanced datasets, BETO model (cased version) is the one that offers the best performance, outperforming the results obtained by other state-of-the-art models trained with different datasets. We conclude by exposing all the difficulties encountered and offering improvement alternatives for future work.

This work has been carried out as part of the project: "Taxonomy, presence and intensity of hate speech in digital environments linked to Spanish professional media - Hatemedia". Project PID2020-114584GB-I00, funded by the State Research Agency - Ministry of Science and Innovation.

Keywords: Hate speech, Deep learning, Transfer learning, BETO, Natural language processing, Text classification

Índice de contenidos

Índice de figuras

Índice de tablas

Introducción

Hoy en día, el auge de las redes sociales y medios informativos online genera una enorme cantidad de información y proliferación de contenidos (desinformativos o no) que en muchas ocasiones ponen en entredicho la tolerancia, civismo y respeto a determinados colectivos. Además, el anonimato y la interactividad propias de la web facilitan el aumento y la permanencia de los comentarios opresivos (Frenda et al., 2018).

En este contexto, la detección automática del discurso de odio o *Hate Speech* (HS, de sus siglas en inglés) juega un papel importante. Sin embargo, nos encontramos ante el problema de que no existe una definición única para el discurso de odio, lo que complica en gran medida la labor de crear algoritmos que detecten el odio automáticamente y con precisión en un texto. En los últimos años, se han introducido varias definiciones ad hoc por parte del sector legal, académico y por las mismas redes sociales. Sin embargo, la elaboración de una definición precisa del discurso del odio es una tarea difícil dada su naturaleza subjetiva. (Papcunová et al., 2021). Al final, un texto escrito en internet podrá ser considerado discurso de odio en función de varios elementos que van más allá de las simples palabras que lo componen, como pueden ser las características del propio emisor, su intención, el contexto en el que se realiza, la cultura del país, etc.

Otra dificultad a tener en cuenta es que el mensaje de odio a veces se confunde con el término "lenguaje ofensivo". Por este motivo, es importante remarcar la diferencia entre ambos conceptos. Un texto es ofensivo si contiene alguna forma de lenguaje no aceptable. En esta categoría pueden incluirse los insultos, las amenazas o las expresiones malsonantes (Plaza-del-Arco et al., 2021).

Por último, no podemos olvidar la complejidad intrínseca al propio lenguaje y sus peculiaridades: la ironía, el humor, el doble sentido, el odio implícito, metáforas... Incluso podemos encontrar textos absolutamente inocuos que utilizan términos malsonantes y comúnmente utilizados en lenguaje ofensivo, siendo este un caso muy común de falso positivo en muchos clasificadores de texto (especialmente los basados en lexicón). Por si esto fuera

poco, el castellano presenta un alto grado de complejidad morfológica que requiere normalmente de tareas de preprocesamiento adicional para lograr aumentar el rendimiento de los modelos.

Por todos estos motivos, la detección automática de odio online presenta un reto de grandes dimensiones que la comunidad científica se esfuerza en solucionar.

1.1. Motivación

Actualmente existe una fuerte motivación para estudiar la detección automática del discurso del odio debido a la abrumadora difusión de información online, impulsado por las nuevas tecnologías y formas de relacionarnos en internet. La detección automática del discurso del odio mediante algoritmos de inteligencia artificial (IA) éticos y fiables va a ser una tarea crucial para proteger los derechos fundamentales de las personas, especialmente importante ante escenarios tan radicales como los que nos toca vivir hoy en día, con guerras en curso y una sociedad extremadamente polarizada. Dentro de este escenario, internet se convierte en una potencial herramienta para distorsionan la realidad, atacar a personas e incluso deshumanizar a ciertos colectivos. Por poner un ejemplo, los estudios han demostrado un aumento de la incitación al odio contra China en las redes sociales, especialmente los contenidos racistas y abusivos que acusan a las personas de causar el brote de COVID-19[2].

El discurso de odio generalizado tiene importantes implicaciones sociales por motivos obvios. Sin embargo, este puede tener otras consecuencias mucho menos obvias, como que puede ser precursor de delitos más graves cometidos en nuestra sociedad. De hecho, algunos estudios afirman que existe una correlación entre el número de violaciones y el número de mensajes misóginos por estado dentro de los Estados Unidos (Filippo et al., 2015). En un marco del discurso de odio más amplio, tenemos varios estudios que plantean la hipótesis de una correlación entre el incremento de los mensajes de odio emitidos en internet y los crímenes de odio cometidos en determinados lugares y contextos específicos (Müller &

[2] Twitter Sees 900% Increase in Hate Speech towards China Due to Coronavirus, 2020

Schwarz, 2021), (Lingiardi et al., 2020), (Alkomah & Ma, 2022). Estos estudios consideran fundamental estudiar este tipo de mensajes de discurso de odio online con el fin de tomar acciones preventivas y contrarrestar sus posibles efectos negativos. En el trabajo de Ligiardi et al. (2020) se insta a realizar una investigación futura que trate de verificar si los picos de tuits intolerantes hacia un grupo objetivo tienden a coincidir con acontecimientos sociopolíticos relacionados.

1.2. Planteamiento del problema

Dada la enorme cantidad de contenidos generados por los usuarios en redes sociales, no es adecuado confiar únicamente en la supervisión humana para combatir el discurso de odio en internet. Las plataformas sociales a gran escala están invirtiendo actualmente importantes recursos para detectar y clasificar automáticamente los contenidos de odio.

A pesar de los numerosos estudios en este campo, el discurso del odio sigue siendo un reto desafiante. El estado del arte informa de que tanto las personas como los modelos de aprendizaje automático tienen dificultades para detectar el discurso de odio debido a la complejidad y variedad de las categorías de odio. Además, las definiciones teóricas existentes del discurso del odio no están suficientemente elaboradas, por lo que actualmente no se dispone de una definición totalmente precisa en la que poder basarnos a la hora de crear datasets etiquetados y algoritmos automáticos.

Este estudio pretende contribuir a la detección automática del discurso de odio en español. Para ello, hacemos uso del corpus etiquetado por el equipo del proyecto Hatemedia[3] y comparamos varias técnicas de clasificación basadas en modelos de aprendizaje profundo.

1.3. Estructura de la memoria

La estructura de la memoria está organizada de la siguiente manera:

En la Sección 2 se hará un análisis del contexto y el estado del arte reflejando la importancia del campo de estudio. Para ello, repasaremos en primer lugar los talleres y eventos más relevantes de los últimos años enfocados a tratar el problema de la detección de expresiones de odio en textos, así como los datasets y sistemas basados en inteligencia artificial más conocidos que se utilizan para intentar abordar este complejo problema.

[3] Proyecto PID2020-114584GB-I00, financiado por la Agencia Estatal de Investigación - Ministerio de Ciencia e Innovación

Los objetivos generales y específicos son descritos con más detalle en la Sección 3, donde también se detallarán los pasos necesarios para la consecución de estos.

En la Sección 4 se describe el procedimiento que se va a seguir para llevar a cabo la comparativa. Esto comprende desde la descripción de las versiones del dataset que se van a utilizar, hasta los modelos seleccionados y las métricas de evaluación utilizadas.

En la Sección 5 pasaremos a describir el desarrollo del trabajo, mostrando los resultados obtenidos, para continuar en la Sección 6 con una discusión sobre la relevancia de los resultados, identificando las conclusiones más importantes extraídos de estos resultados.

Finalmente, en la Sección 7 se darán las conclusiones extraídas del trabajo y se propondrán líneas futuras de investigación o desarrollo relacionado con el mismo.

2. Contexto y estado del arte

El estudio de la detección y de la clasificación automática del discurso de odio mediante procesamiento de lenguaje natural (PLN) es un campo relativamente reciente, pero ha evolucionado rápidamente en los últimos años debido a su importancia (García-Díaz et al., 2022). En la Figura 1 mostramos una gráfica de Dimensions[4] para los términos de búsqueda “hate speech detection”, filtrado por las categorías "Information and Computing Sciences” y “Artificial Intelligence”, donde se puede apreciar un notable crecimiento en el número de publicaciones de trabajos relacionados con el discurso de odio a lo largo de los últimos años.

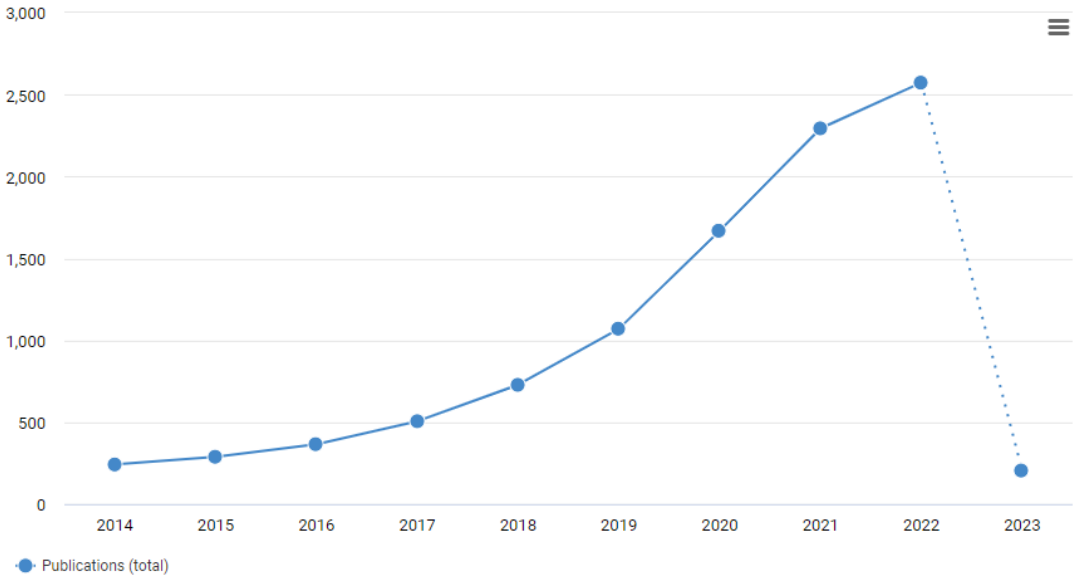

Figura 1: Gráfica de Dimensions con los términos de búsqueda de "hate speech detection” para las categorías “Information and Computer Sciences” y “Artificial Intelligence”, donde se muestra el número de publicaciones por año.

El interés en esta área ha aumentado a medida que las redes sociales y otras plataformas de internet han crecido en términos de influencia y adopción por parte de la gran mayoría de los usuarios (Arango et al., 2019).

En la presente sección haremos una revisión del estado del arte, donde comenzaremos destacando los principales eventos y talleres a nivel mundial, enfocados en la detección del discurso del odio. A continuación, listaremos algunos de los dataset más utilizados para dichas tareas. Finalmente, analizamos las diferentes técnicas de PLN utilizadas para extraer

[4] https://app.dimensions.ai/discover/publication

información de un texto, así como los modelos de aprendizaje automático empleados en el estado del arte, desde los modelos de machine learning (ML) clásicos hasta soluciones más modernas basadas en aprendizaje profundo y Transformers.

2.1. CONGRESOS RELATIVOS A LA DETECCIÓN DE ODIO EN TEXTOS

El impacto de las publicaciones nocivas online ha dado lugar a un gran número de estudios y eventos enfocados a la detección del odio y lenguaje ofensivo. Como ejemplo, se listan los siguientes talleres y congresos[5].

- SemEval[6], taller internacional sobre el procesamiento del lenguaje natural cuya misión es avanzar en el estado actual del arte. Cada año, este taller propone una serie de tareas compartidas en las que se presentan y comparan sistemas de análisis semántico computacional diseñados por diferentes equipos. Las tareas más destacadas para la detección de odio en internet son:
    - Identifying and Categorizing Offensive Language in Social Media (SemEval-2029, Tarea 12)
    - *Multilingual Offensive Language Identification in Social Media* (SemEval-2020, Tarea 12)

  El taller SemEval 2023 cuenta con la tarea 10 que trata sobre la detección de sexismo en internet. Este evento está actualmente en curso y los resultados obtenidos por los equipos participantes se publicarán a lo largo del año.

- *Workshop on Online Abuse and Harms* (WOAH[7]), que en el año 2022 celebró su sexta edición, cuyo objetivo es avanzar en la investigación para detectar, clasificar y modelar el contenido ofensivo y dañino en internet.
- *GermEval Shared Task*[8] (edición de 2018 y 2019), centrado en el procesamiento del lenguaje natural para detección de lenguaje ofensivo en el idioma alemán.

[5] Nótese que no todas las ediciones de cada evento están enfocadas a la detección del discurso de odio, sino que en cada año se plantean una o varias tareas a resolver mediante PLN.
[6] https://semeval.github.io/
[7] https://www.workshopononlineabuse.com/
[8] https://germeval.github.io/tasks/

- *PolEval*[9] (edición de 2019, tarea 6), sobre la detección automática del ciberacoso en Twitter para el lenguaje polaco.
- *HASOC*[10] (2019), sobre identificación de expresiones de odio y contenidos ofensivos en las lenguas indoeuropeas.
- *AMI*[11] *(2018),* taller para la identificación automática de la misoginia, para el idioma italiano y el inglés.

Con relación a los estudios sobre el discurso del odio en idioma español, observamos que no encontramos tanta variedad como los centrados en el idioma inglés. De hecho, los estudios que existen están relacionados mayoritariamente con la participación de *IberEval 2018 - Automatic Misogyny Identification* y la Tarea 5 del taller *SemEval 2019* (García-Díaz et al., 2022).

SemEval-2019, Tarea 5

Esta tarea tuvo como objetivo detectar contenidos de odio en los textos de las redes sociales en español, concretamente en las publicaciones de Twitter, contra dos objetivos: los inmigrantes y las mujeres. Además, la tarea implementaba una perspectiva multilingüe en la que se proporcionaron datos de los idiomas inglés y español (*HatEval*), para entrenar y probar los sistemas participantes. El conjunto de datos de *HatEval* estaba compuesto por 19.600 tuits, 13.000 en inglés y 6.600 en español. (Basile et al., 2019). Esta tarea se articulaba en torno a dos subtareas relacionadas:

- Subtarea A: Consistía en una detección básica de discurso de odio, en la que se pedía a los participantes que marcaran la presencia de odio en los tweets (clasificación binaria).
- *Subtarea B:* En esta segunda subtarea se pretendía ir más allá de la simple detección binaria de discurso de odio. De este modo, se trataba de determinar si el objetivo del mensaje era un individuo un grupo de personas, y si el contenido del mensaje contenía lenguaje agresivo.

[9] http://2019.poleval.pl/
[10] https://hasocfire.github.io/hasoc/2019/
[11] https://amievalita2018.wordpress.com/

IberEval 2018 (AMI)

Este taller estaba enfocado a la detección de tweets misóginos mediante PLN, con un dataset multilingüe, con 4.138 tuits escritos en español y 3.977 en inglés (Fersini et al., 2018). Del mismo modo que en el caso de *SemEval 2019 task 5,* IberEval 2018 estaba organizado en dos subtareas:

- Subtarea A: Consistía en una tarea de identificación binaria de mensajes misóginos.
- *Subtarea B:* En esta segunda subtarea había que determinar cuándo el objetivo del comentario misógino era un individuo concreto o un grupo.

2.2. DATASETS

En este apartado listamos algunos de los dataset más utilizados en el estado del arte para tareas de detección de discurso de odio en inglés.

- ***Waseem and Hovy***: Este conjunto de datos está compuesto por 16.000 tweets anotados como "sexistas", "racistas" y "sin odio" (Waseem & Hovy, 2016).

- **Davidson et al.**: Compuesto por 24.802 tuits anotados en tres clases: discurso de odio, ofensivo (pero no de odio), y ni ofensivo ni de odio (Davidson et al., 2017)

- **HatEval**: Este conjunto de datos se compone de 19.600 tweets, 13.000 en inglés y 6.600 en español (Basile et al., 2019).

- **Stormfront**: Dataset público sobre discurso de odio recopilado a través de mensajes de foros de Internet en idioma inglés. Este dataset está disponible en GitHub[12] . El foro de origen es Stormfront[13].

[12] https://github.com/Vicomtech/hate-speech-dataset
[13] https://www.stormfront.org/

- **TRAC-I**: Se trata de un dataset creado a partir de textos de Facebook y Twitter, en idioma hindi e inglés. Se compone de 12.000 mensajes clasificados en abiertamente agresivos (esta clase expresa abiertamente la agresión utilizando léxicos simbólicos típicos), encubiertamente agresivos (expresión sutil e indirecta de la agresión, incluyendo el sarcasmo, la sátira y las preguntas retóricas) y no agresivos (Kumar et al., 2018).

- **HS**: 4.575 tweets en hindi y en inglés etiquetados como discurso de odio (aquellos tuits que inducen al odio) y discurso normal (tuits que no inducen ninguna forma de odio) (Bohra et al., 2018).

- ***HOT:*** Al igual que el dataset HS, tiene texto en hindi e inglés. Consta de 3.679 tuits clasificados en tres categorías: No ofensivos, ofensivos (con objeto de herir los sentimientos del receptor) e inductores de odio (Mathur et al., 2018).

A continuación, se listan algunos de los datasets más importantes en idioma español, utilizados por distintos estudios del estado del arte.

- **HaterNet:** Dataset en idioma español construido a partir de Twitter, compuesto por 6.000 textos etiquetados, con 1.567 tweets anotados como odio y 4.433 anotados como no odio (Pereira-Kohatsu et al., 2019).

- **HatEval 2019:** Dataset construido a partir de Twitter compuesto por 6.600 textos en español, con 2.739 anotados como odio y 3.861 etiquetados como no odio (Basile et al., 2019).

- ***IberEval 2018 – AMI***: Dataset en español compuesto por 4.138 tweets, 2.064 anotados como mensajes misóginos y 2.074 como no misóginos (Fersini et al., 2018).

- **MisoCorpus 2020**: El conjunto de datos completo contiene 8.390 tweets y se divide en: (1) VARS, que considera la violencia hacia las mujeres en la política y los medios de

comunicación públicos; (2) SELA, sobre la comprensión de las diferencias en los mensajes misóginos en el español de España y el español de América Latina; y (3) DDSS, que contiene rasgos generales relacionados con la misoginia (García-Díaz et al., 2021).

2.3. TÉCNICAS Y MODELOS

El procedimiento que se suele seguir para realizar el análisis de un texto, ya sea con el objetivo de detectar odio o para cualquier otro, consta de tres pasos:

1. Preprocesado de texto: Cuyo objetivo es preparar el texto para el análisis haciendo uso de diferentes técnicas de PLN como las descritas en el apartado 2.3.1 Técnicas de preprocesado.
2. Extracción de características: El rendimiento de un sistema de aprendizaje de IA depende completamente de la correcta representación del problema. El objetivo aquí es extraer características del texto a analizar para obtener representaciones que sean manejables para su procesamiento (Plaza-del-Arco et al., 2021).
3. Clasificación mediante modelos IA: Una vez tengamos una representación de nuestros textos mediante la extracción de características, podemos entrenar modelos de inteligencia artificial (ya sea desde cero o apoyarnos en modelos pre-entrenados) que nos permitan clasificar textos nuevos con mayor o menor precisión. Las técnicas que pueden utilizarse para crear modelos de clasificación automática de un texto son muy variadas. Sin embargo, es posible agruparlas en tres tipos principales de técnicas: aprendizaje automático clásico, aprendizaje profundo y aprendizaje por transferencia.

2.3.1. Técnicas de preprocesado

Como es natural, el texto que nos llega en bruto puede presentar un formato que diste mucho de lo que podríamos considerar el formato correcto, compuesto por palabras incompletas, mal escritas o en otros idiomas, conteniendo espacios innecesarios, etc. Por ejemplo: p- e-r-r-o, n€gr0. Además, en nuestro texto origen existirán, casi con total seguridad, infinidad de palabras innecesarias que no nos aporten ningún valor.

Así pues, en primer lugar y antes de extraer características del texto y construir modelos a partir de esta información, debemos dedicar tiempo a las tareas de limpieza, formateo y

preparación de los datos. Estas tareas están presentes en el día a día de todos los proyectos de IA en general, y de procesamiento de lenguaje natural en particular (Urdaneta, 2019).

Existe una amplia variedad de librerías PLN Open Source para realizar estas tareas de preprocesado en diversos idiomas como son NLTK[14], Freeling[15], Pattern.es[16], Spacy[17] y Stanford NLP[18].

- **Tokenización:** Esta técnica consiste en la segmentación del texto en frases, palabras o incluso caracteres, es decir, segmentar el texto en unidades más pequeñas (tokens o n-gramas) que podamos manejar como referencia para extraer características que aporten valor a nuestro sistema. Además, eliminaremos todos aquellos tokens que no nos aporten valor, de modo que reduzcamos el número de elementos a tratar. Para facilitar la labor de eliminar los tokens innecesarios de nuestro corpus, se suelen utilizar listas de *stopwords***.** Estas listas constan de palabras que, por ser muy habituales en el idioma tratado o por cualquier otro motivo particular, aportan poco valor al problema que estamos tratando. Por ello, es interesante identificarlas y filtrarlas, por ejemplo: los determinantes, las conjunciones "y / e", "o / u", etc. Esta una forma de reducir los elementos de nuestro texto de entrada, pero también se pueden utilizar otros métodos como, por ejemplo, decidir eliminar todas las palabras de longitud menor o mayor a un umbral determinado.
- **Normalización**: Normalizar nuestro texto será una tarea importante si queremos que nuestras palabras sigan un formato estándar. Del paso anterior, nuestro tokenizador ha podido reconocer la misma palabra, pero escrita en mayúsculas y en minúsculas (por ejemplo, tres formas distintas de la misma palabra: hablar, HABLAR y Hablar). Si queremos tener solo una versión, será imprescindible normalizar nuestro texto.
- **POS (part-of-speech) tagging:** El POS es la técnica sintáctica para etiquetar a cada una de las palabras de un texto su categoría gramatical. De esta forma, logramos capturar características sintácticas del texto, es decir, tenemos en cuenta la relación de las palabras. Trabajos anteriores han probado a identificar el odio utilizando

[14] https://www.nltk.org/
[15] http://nlp.lsi.upc.edu/freeling/node/1
[16] https://www.clips.uantwerpen.be/pages/pattern-es
[17] https://spacy.io/
[18] https://stanfordnlp.github.io/stanfordnlp/

características sintácticas y léxicas, como los n-gramas (a nivel de carácter, palabra y frase) y el uso de una bolsa de palabras ofensivas. Por ejemplo, Warner and Hirschberg (2012) encontró que el trigrama "<DET> judío <SUSTANTIVO>" es la característica más significativa para detectar odio antisemita, mientras que Waseem and Hovy (2016) identificó n-gramas de caracteres predictivos mediante coeficientes de regresión logística. (Wang, 2018).

- **NER (Named Entity Recognition**): La detección de entidades permite identificar automáticamente determinadas palabras de un texto y clasificarlas en diferentes categorías: nombres propios, lugares, marcas, cantidades, etc.
- **Lematización**: Tras aplicar las técnicas de tokenización y normalización, habremos reducido considerablemente el número de elementos a tratar. Sin embargo, y debido a las peculiaridades del lenguaje, podemos seguir teniendo diferentes formas que representan la misma palabra. Por ejemplo, en español tenemos una gran variedad de conjugaciones de los verbos: *juego, juegas, juegan, jugaban*… todas estas palabras proceden del mismo verbo en infinitivo (*jugar)*. También sabemos que *perros, perrito, perrazo*, etc., son diferentes variantes del vocablo perro. La técnica de lematización lo que consigue es reducir todas estas palabras derivadas a su lema, que es la forma en la que encuentras la palabra en el diccionario.
- **Radicalización:** En inglés, se conoce como *stemming* al procedimiento de convertir palabras en raíces. Estas raíces son la parte invariable de palabras. Las raíces se diferencian del lema en que no tienen por qué ser palabras de un idioma. Por ejemplo, si utilizamos la función *Snowball Stemmer* de la librería NLTK de Python para obtener la raíz de las palabras *canta, cantas y cantamos*, veremos que la raíz resultante es la misma: "*cant".* Además del snowball, nltk permite usar otros algoritmos como el Porter Stemmer, muy utilizados en los estudios del estado del arte (Frenda et al., 2018) y (Davidson et al., 2017) .

2.3.2. Técnicas de extracción de características

En primer lugar, revisaremos las técnicas más simples de extracción de características, (también conocidas como técnicas superficiales), donde destacamos la bolsa de palabras y la

técnica TF-IDF. A continuación, pasaremos a explicar los word embeddings, una técnica más compleja capaz de representar las palabras de nuestro lexicón mediante vectores multidimensionales, capaces de capturar incluso relaciones semánticas entre palabras.

- **Bolsa de palabras**

La bolsa de palabras (BoW, de sus siglas en inglés) es una representación vectorial compuesta por un diccionario (lexicones) con las palabras de los textos con los que se quieren entrenar los modelos. En estos lexicones se representa la relevancia de cada elemento mediante métricas como, por ejemplo, si la palabra aparece en el texto (booleano), o la cantidad de veces que una palabra se repite en el texto.

A continuación, mostramos un ejemplo muy simple de una bolsa de palabras (Figura 2), donde dados 2 textos se cuenta la ocurrencia de cada palabra como métrica para la extracción de características.

Texto1: El gato es negro.

Texto2: El perro es blanco y es bonito.

Con este ejemplo, nuestro lexicón estaría compuesto por las siguientes 8 palabras:

[El gato es negro perro blanco y bonito]

	1 **EL**	**2** **gato**	**3** **es**	**4** **negro**	**5** **perro**	**6** **blanco**	**7** **y**	**8** **bonito**
Texto 1	1	1	1	1	0	0	0	0
Texto 2	1	0	2	0	1	1	1	1

Figura 2: Ejemplo de bolsa de palabras (BoW)

Se trata de un ejemplo muy simple donde la mayoría de palabras aparecen una vez o ninguna, a excepción de la palabra: "*es",* que aparece 2 veces en el texto 2.

- **TF-IDF**

TF-IDF (del inglés Term frequency – Inverse document frequency) (Luhn, 1957) se trata de una técnica muy popular y utilizada en el campo de la clasificación de texto

automático, como se puede comprobar en varios de los trabajos realizados en el taller de SemEval-2019 (Basile et al., 2019). TF-IDF es una técnica cuyo objetivo es encontrar el documento más relevante para cierto término dentro de una colección de documentos. Para ello, mide con qué frecuencia aparece un término o frase dentro de un documento determinado, y lo compara con el número de documentos que mencionan ese mismo término dentro de una colección entera de documentos. De esta forma, palabras muy utilizadas del lenguaje como son los determinantes o las conjunciones (que aparecen en casi todos los documentos) tendrán un valor bajo, ya que aportan muy poco valor. Sin embargo, palabras que se repiten mucho en uno o varios documentos, pero no aparecen en el resto del conjunto de documentos, obtendrán un valor alto de TF-IDF.

Estas técnicas superficiales se enfrentan a limitaciones en la detección de textos de discurso de odio, especialmente cuando estos textos no contienen palabras ofensivas, transmitiendo odio encubierto (Dinakar et al., 2011; Mathur et al., 2018). Lo mismo ocurre en caso contrario, cuando el texto contiene palabras ofensivas, insultos o cualquier expresión soez, pero que carece de odio debido al contexto en el que se está utilizando. Como ya sabemos, las palabras pueden adoptar distintos significados dependiendo del contexto en el que se encuentren, debido a elementos intrínsecos del propio lenguaje como son el sarcasmo o el humor.

Como parte positiva, es que las decisiones de clasificación de los modelos entrenados a partir de características a nivel superficial son modelos interpretables y, por tanto, satisfacen el principio de explicabilidad dentro del marco de las directrices europeas para una IA fiable (Hleg, 2019), permitiendo que los usuarios puedan comprender el proceso de toma de decisiones y poder confiar en resultados de estos algoritmos automáticos.

- **Word Embeddings**

Word Embedding (Firth, 1957; Mikolov et al., 2013) es una de las técnicas más populares para representar el vocabulario de un texto, y está presente en muchos de los estudios del estado del arte para detección de odio, como Melnyk, (2021) y Dash et al. (2021) . Esta técnica es capaz de capturar el contexto y la similitud semántica y sintáctica (género, sinónimos, etc.) de las palabras dentro de un texto. Cada palabra se

representa en forma de un vector n-dimensional basado en la situación en la que aparece junto con otras palabras. Esto nos permite generar vectores de palabras de forma que palabras similares tengan incrustaciones de palabras similares (Sachdeva et al., 2021).

Por ejemplo, si tenemos las palabras «perro», «gato» y «tomate», cabría esperar que las palabras perro y gato estuvieran representadas por vectores más cercanos entre sí en el espacio vectorial donde se definen estos vectores en relación al vector que representa la palabra tomate, que quedaría más alejado. Las representaciones de Word Embeddings pueden generarse a partir de representaciones pre-entrenadas como Word2vec (Mikolov et al., 2013), Glove (Pennington et al., 2014) y fastText (Bojanowski et al., 2017). Estos modelos son conceptualmente iguales, pero hay una pequeña diferencia: fastText opera a nivel de caracteres, mientras que Word2Vec y Glove lo hacen a nivel de palabras.

2.3.3. Machine Learning clásico

Entre las diversas técnicas convencionales de aprendizaje automático utilizadas en la tarea de la detección del discurso del odio en Internet, destacan las máquinas de vectores soporte (**SVM**), la **regresión logística** y los **Random Forest** (Burnap & Williams, 2015; Davidson et al., 2017; Nobata et al., 2016; Waseem & Hovy, 2016).

Sachdeva et al., 2021, muestra que estos tres modelos son los que proporcionan mejor rendimiento dentro del ML convencional en términos de *Accuracy, Precision, Recall y F1*. Por otro lado, en este estudio se concluye que el modelo K-Vecinos Más Cercanos (**KNN,** de sus siglas en inglés), obtuvo el peor rendimiento para la tarea de clasificación de textos.

El taller SemEval 2019, tarea 5 (que consistía en detectar discurso de odio en Twitter contra mujeres e inmigrantes), muestra que el modelo *SVM* es especialmente relevante, ya que los sistemas creados mediante este modelo obtuvieron los mejores resultados de la competición (Basile et al., 2019).

2.3.4. Deep Learning

Durante los últimos años, los métodos de Deep Learning (DL) o aprendizaje profundo, han despertado un gran interés a la hora de resolver el problema de la detección del discurso de

odio (Badjatiya et al., 2017; Gambäck & Sikdar, 2017; Gröndahl et al., 2018; Arango et al., 2019; Melnyk, 2021). Dentro de las técnicas de DL más utilizadas en la clasificación de textos, destacan las redes neuronales convolucionales y las redes neuronales recurrentes (García-Díaz et al., 2022).

Badjatiya et al. (2017) y Gambäck & Sikdar (2017) fueron los primeros en utilizar redes neuronales recurrentes y redes neuronales de convolución, respectivamente, para la detección del discurso del odio en los tuits.

- **CNN**

  Las redes neuronales convolucionales (CNN) son un tipo de red neuronal que procesa capas de forma jerárquica, lo que les permite diferenciar distintas características en las entradas recibidas (Roy et al., 2020). La capa más importante, y la que da nombre a la red, es la capa convolucional. Esta capa funciona a partir de unos filtros que van desplazándose por la imagen o el texto, dependiendo el problema a resolver, obteniendo las salidas de la capa mediante un producto escalar.

  En el caso de imágenes, las primeras capas pueden detectar formas básicas como líneas, esquinas o curvas y se van especializando hasta llegar a capas más profundas que reconocen formas complejas como el rostro de una persona o la silueta de un coche. Aunque se diseñaron inicialmente para la visión por computador, han sido eficaces también para tareas de PLN y de detección de odio (Wang, 2018). En la Figura 3 podemos observar la arquitectura de una red neuronal convolucional aplicada al problema de análisis de sentimiento de textos.

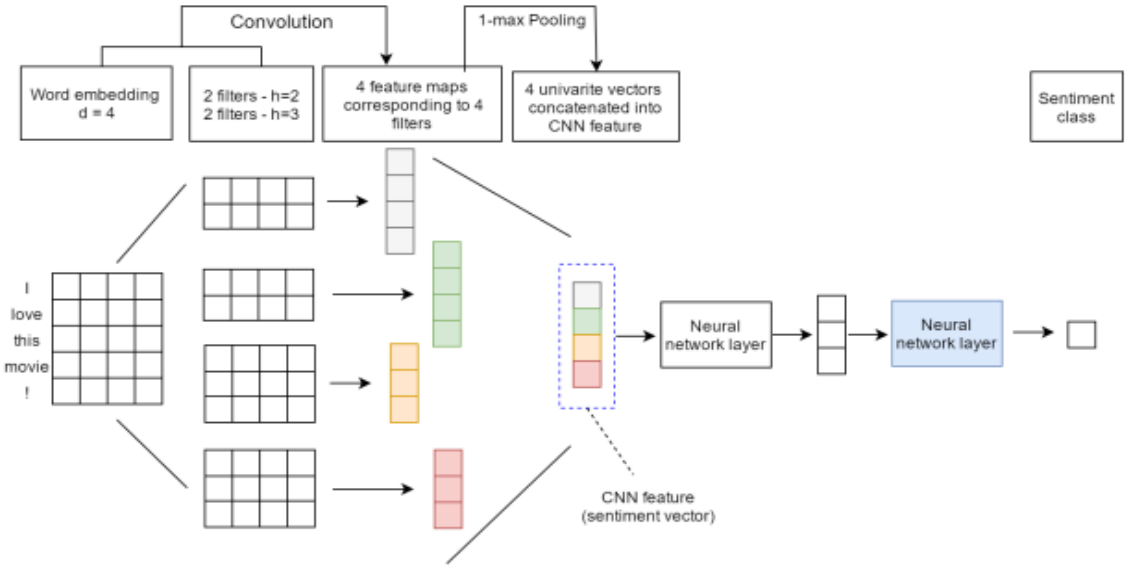

Figura 3: Arquitectura de una red neuronal convolucional[19]

Cuando utilizamos una red CNN aplicada a PLN, lo que procesamos son textos en lugar de imágenes. Estos textos tendrán una representación matricial, donde las filas representan las palabras codificadas mediante word embeddings con una dimensión d (espacio vectorial donde hemos embebido los textos). Por tanto, cada filtro de convolución tendrá una anchura igual a la longitud del embedding donde están incrustados los textos a procesar, en nuestro ejemplo d=4, de modo que cada filtro irá recorriendo las palabras en una sola dirección, de arriba abajo, en lugar de izquierda a derecha y de arriba abajo como sucede con las imágenes. En nuestro ejemplo observamos que tenemos 4 filtros, dos de altura h=2 y otros dos de altura h=3. Esto significa que queremos detectar características locales en grupos formados por dos y tres palabras, capturando diferentes niveles de correlación entre palabras. Así pues, cada filtro se encargará de capturar cierta característica de los datos.

Como estamos aplicando capas de convolución que son unidimensionales (recorremos la matriz de entrada de arriba a abajo), en lugar de las bidimensionales utilizadas en imágenes, la salida que obtenemos tras aplicar nuestro filtro es un vector en lugar de una matriz. Estos vectores serán nuestros mapas de características.

[19] Imagen extraída de (Nguyen et al., 2017)

En la fase de *max-Pooling* solo nos quedamos con un elemento, el resultado más grande de cada uno de los mapas de características, para reducir la dimensionalidad.
Finalmente, concatenamos los valores máximos obtenidos en la fase de *max-Pooling* para conformar la entrada de la siguiente capa, una *fully connected layer*. En nuestro ejemplo, tenemos dos capas densas como últimas capas. La última capa estará compuesta por una sola neurona para clasificación binaria.

- **RNN y LSTM**

Las redes neuronales recurrentes (RNN) son una clase de redes especializadas en analizar datos de series temporales. La principal característica de este tipo de redes radica en su capacidad de modelar relaciones temporales entre elementos de la secuencia a través de un estado interno de la red o hidden state, que podemos considerar como una memoria sobre lo que la red ha visto hasta ese momento. En esta arquitectura se aplica una fórmula recurrente sobre una secuencia de entrada de manera que, en cada paso dado, se depende del nuevo valor de entrada x y del estado interno h anterior. Por tanto, este tipo de arquitecturas permiten modelar relaciones entre palabras dentro de un texto.

Las LSTM (Long Short Term Memory) son un tipo especial de redes recurrentes (Vigna et al., 2017). Estas redes surgieron como una arquitectura encaminada a solucionar los problemas de memoria de las RNN tradicionales. En la práctica, estas últimas presentan problemas para aprender relaciones con elementos de time step lejanos (es decir, que no están cerca del time step actual). Esto limita en gran parte el potencial teórico de las RNN. Por ejemplo, dentro del campo del procesamiento de lenguaje natural, cuando analizamos un texto es importante mantener la información aprendida desde el inicio hasta el final de este, de modo que podamos extraer características y relaciones entre palabras dentro de un mismo texto.
Las LSTM están diseñadas para intentar solucionar este problema. En LSTM se establecen unos criterios para almacenar la información obtenida hasta el momento. El modelo aprende qué partes de la representación se deben olvidar para incluir las más importantes. Para ello, mantienen un estado interno *cell state* (c) además del *hidden state* (h), el cual representa una especie de autopista de información a lo largo del tiempo.

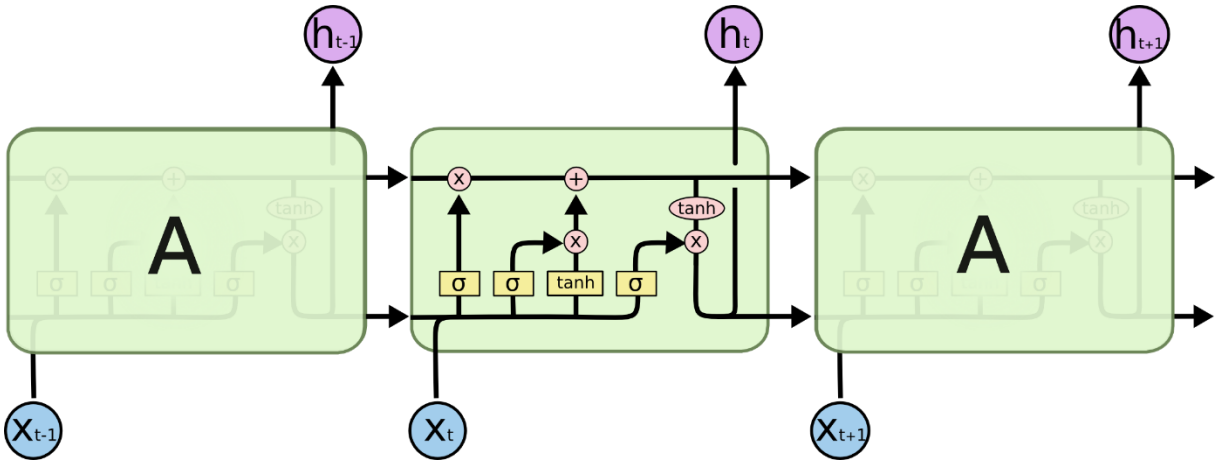

Figura 4: Arquitectura LSTM

Fuente: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

En la Figura 4 mostramos la arquitectura LSTM donde podemos observar que, en vez de calcular directamente el valor de salida *h*, ahora obtenemos cuatro vectores distintos conocidos como puertas o gates: *i, f, g y o*, que después se combinan para obtener el cell state *c* y el hidden state *h*. A continuación, explicamos brevemente la función de cada vector.

- **f (forget gate):** Decide qué información del cell state hay que olvidar. Para ello, toma el hidden state anterior y la entrada actual, los transforma y los lleva a una función de activación sigmoid. Si uno de los valores de este vector es 0, o cercano a 0, entonces la LSTM eliminará esa porción de información, mientras que si alcanza valores iguales o cercanos a 1 esta información se mantendrá y llegará a la celda de estado.

- **I (input gate):** Decide qué nueva información incorporamos al cell-state. Para ello, tomamos nuevamente el estado oculto anterior y la entrada actual, los transformamos y los llevamos de nuevo a una función de activación sigmoid. En este caso, los valores que queremos preservar en la memoria de la red serán aquellos cercanos a 1. Este resultado lo multiplicamos por el vector *g* que viene de aplicar una función tanh a la entrada actual, para obtener valores entre -1 y 1 que regulen la red.

- **Cell state (c)**: Teniendo ya los datos generados por las compuertas forget e input, ahora podemos actualizar la celda de estado (es decir, la memoria de la red LSTM). Para ello, primero debemos saber cuánto queremos olvidar. Para ello, multiplicamos el vector

del olvido f por el valor del cell state c. A continuación, sumamos lo anterior a lo calculado en el *input gate*, generando así la memoria actualizada.

- **Output Gate:** Finalmente debemos calcular el nuevo estado oculto, para lo cual usamos el output gate o puerta de salida. En primer lugar, escalamos el nuevo cell state para garantizar que esté en el rango de -1 a 1. Para ello usamos la función tanh. Por otro lado, tomamos nuevamente el estado oculto anterior y la entrada actual y los pasamos por una función sigmoid. Finalmente, multiplicamos los dos valores anteriores para obtener el nuevo estado oculto.

Existe una versión alternativa llamada Bi-LSTM (Bidirectional Long Short-Term Memory). Se trata de una arquitectura idéntica a la LSTM, solo que en este caso la red neuronal se entrenará con los mismos datos una segunda vez, recorriéndolos en orden inverso. Si bien las LSTM/BiLSTM suponen una mejora respecto a las RNN clásicas, ambos modelos comparten una arquitectura secuencial que limita en gran medida la paralelización de las ejecuciones y, por tanto, el rendimiento LSTM general. Por último, la arquitectura GRU (Gated Recurrent Unit), es una versión simplificada de LSTM introducida en 2014 por Chung et al. y utiliza un sistema similar de gates al visto en la LSTM. Las mayores diferencias con LSTM son que se combina el cell state y el hidden state en un solo elemento, así como la forget gate y la input gate en una sola puerta.

2.3.5. Transfer Learning

Utilizando como punto de partida modelos pre-entrenados, el Transfer Learning permite desarrollar rápidamente modelos eficaces y resolver problemas complejos de PLN o de visión por computador sin necesidad de tener que entrenar nuestro propio modelo de cero o de disponer de una inmensa cantidad de datos. De este modo, los modelos pre-entrenados se han convertido en un elemento básico en el ámbito del procesamiento del lenguaje natural.

En los últimos años, desde la introducción de la arquitectura Transformer, se han utilizado en muchas otras tareas diferentes de PLN, superando a modelos anteriores basados en redes neuronales recurrentes (Pérez et al., 2021). Los modelos Transformer tienen como principal

innovación la sustitución de las capas recurrentes, como las LSTMs que se venían usando hasta ese momento en PLN, por las denominadas capas de atención (Vaswani et al., 2017).

A nivel de arquitectura, los Transformers se basan en dos partes bien diferenciadas, un codificador y un decodificador. Si observamos la Figura 5, el primer bloque que aparece en la parte izquierda corresponde al codificador o encoder, mientras que el bloque de la derecha corresponde al decodificador o decoder. El encoder está compuesto por una pila de N = 6 capas idénticas. Cada capa tiene dos subcapas. La primera es un mecanismo de autoatención (multi-head attention), y la segunda es una red simple totalmente conectada. Por otro lado, el descodificador también se compone de una pila de N = 6 capas idénticas. Además de las dos subcapas de cada capa del codificador, el descodificador inserta una tercera subcapa multi-head attention, que se aplica sobre la salida de la pila del codificador.

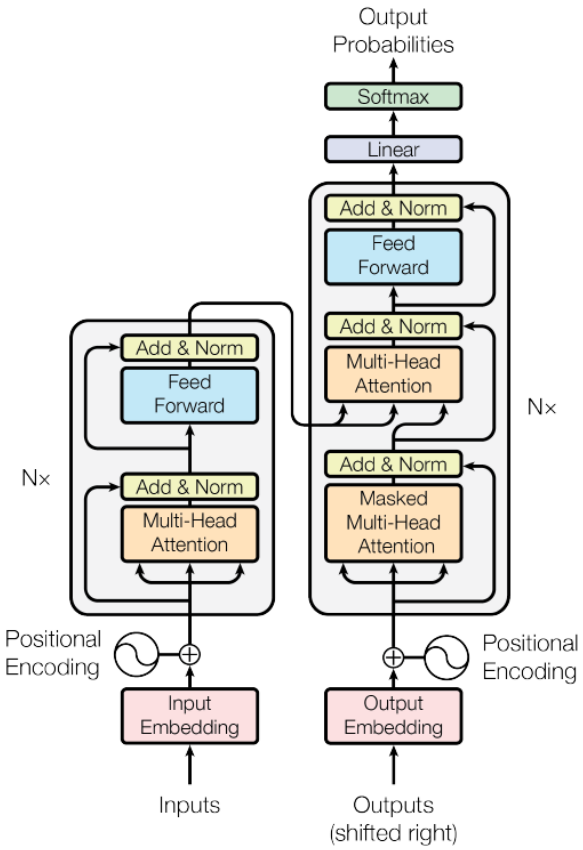

Figura 5: Arquitectura Transformer[20]

[20] Imagen extraida de (Vaswani et al., 2017).

Tanto el codificador como el decodificador trabajan sobre secuencias enteras de texto en lugar de palabra por palabra. De este modo, en lugar de analizar palabras sueltas, se obtiene un análisis global. A continuación, describimos los elementos más importantes de la arquitectura Transformer.

- **Positional encoding**

  Dado que nuestro modelo no contiene recurrencia ni convolución, para que el modelo pueda mantener el control sobre el orden de la secuencia, debemos incorporar alguna información sobre la posición relativa o absoluta de los tokens en la secuencia. Para ello, añadimos "codificaciones posicionales" a los embeddings de entrada en la parte inferior de las pilas del codificador y decodificador. Las codificaciones posicionales tienen la misma dimensión que los embeddings, por lo que ambas pueden sumarse. Así, para cada elemento en la secuencia habrá un correspondiente vector posicional único que permitirá el procesamiento en paralelo de la totalidad de la secuencia en los siguientes bloques que componen el Transformer, lo que supone una de las grandes innovaciones introducidas por esta arquitectura.

- **Self-attention**

  En la Figura 5, observamos que el elemento de entrada aparece tres veces suministrado al módulo de multi-head attention, tanto en el codificador como en el decodificador. Esto es un concepto que se llama auto-atención o self-attention, que básicamente es la clave del Transformer. Como hemos comentado, la arquitectura Transformer recibe todo el texto de una vez, siendo capaz de analizar como cada una de las palabras se relaciona con el resto de las palabras de ese mismo texto y, de este modo, recomponer o reconstruir la información según esas relaciones. Así, el mecanismo de self-attention recodificará los textos en las primeras etapas del Transformer.

- **Bloque residual y de normalización**

  El propósito de este módulo es preservar la información al pasar por el bloque de multi-head attention. Posteriormente, la salida de este bloque residual se lleva a un bloque de normalización.

- **Capa fully-connected**

  Además de las subcapas de atención, cada una de las capas de nuestro codificador y decodificador contiene una red feed-forward totalmente conectada, encargada de aprender a representar de manera optimizada la información proveniente de la capa anterior.

La mejora de rendimiento ofrecida por la arquitectura Transformer ha permitido el rápido desarrollo de modelos sobre conjuntos de datos tan grandes que anteriormente no era viable procesar, dando lugar al modelo BERT (Bidirectional Encoder Representations from Transformers) y a los GPT (Generative Pre-trained Transformer), estos últimos utilizados principalmente para generar textos que simulan la redacción humana.

- **Modelos BERT y RoBERTa**

BERT es un modelo Transformer bidireccional, pre-entrenado sobre una gran cantidad de datos sin etiquetar para aprender una representación del lenguaje que se puede utilizar para realizar fine-tuning y adaptarlo a tareas específicas de aprendizaje automático (Devlin et al., 2019; Pérez et al., 2021). **RoBERTa** (A Robustly Optimized BERT Pretraining Approach) es otro modelo basado en la arquitectura BERT (Liu et al., 2019). RoBERTa utiliza la misma arquitectura de BERT, pero aplicando pequeños cambios que mejoran notablemente el rendimiento del modelo en todas las tareas en comparación con BERT. RoBERTa también utiliza un vocabulario más amplio (50K, frente los 30K de BERT).

- **Modelos multilingües**

Dentro del campo de modelos multilingües, encontramos **m-BERT** (Devlin et al., 2019) y **XML-R** (Lample & Conneau, 2019). Estos dos modelos han impulsado el estado del arte en tareas de PLN multilingüe mediante el pre-entrenamiento en muchos idiomas, mostrando cómo un único modelo puede aprender de varios idiomas, estableciendo bases sólidas para tareas no relacionadas con el inglés (Cañete et al., 2020).

M-Bert (Multilingual BERT) ha sido pre-entrenado con el corpus Wikipedia en 104 idiomas, capaz de realizar una generalización multilingüe sorprendentemente bien (Pires et al., 2019).

Por su lado, XML-R (XLM-RoBERTa) XLM-RoBERTa es una versión multilingüe de RoBERTa. Está pre-entrenada en 2,5 TB de datos CommonCrawl filtrados que contienen 100 idiomas.

- **Modelos monolingües para el idioma español**

El primer modelo monolingüe disponible públicamente en español fue **BETO** (Cañete et al., 2020), un modelo BERT entrenado en su totalidad sobre un gran corpus en español, que mejora los resultados obtenidos por m-Bert para clasificar textos en español (García-Díaz et al., 2022), lo que demuestra que un modelo monolingüe con suficiente entrenamiento puede superar a un modelo multilingüe, incluso cuando se utilizan más recursos y entrenamiento para este último (Devlin et al., 2019). BETO tiene un tamaño similar al de un BERT-Base (BERT-base tiene 12 capas, mientras que BERT-large 24). Existen 2 versiones de BETO, la cased y la uncased. En la versión uncased, el texto con el que se le ha entrenado ha sido previamente transformado a minúsculas, mientras que en la versión cased, el texto con el que se le ha entrenado es el mismo que el de entrada (sin cambios). Asimismo, en la versión uncased se eliminan los acentos, mientras que en la versión cased se conservan.

Más recientemente, se han desarrollado otros modelos lingüísticos para el español, como **BERTIN** (de la Rosa et al., 2022) y **RoBERTuito** (Pérez et al., 2021) , ambos basados en la arquitectura RoBERTa.

3. Objetivos y metodología de trabajo

Considerando el estado del arte y los trabajos preliminares en el proyecto HATEMEDIA, se ha planteado los siguientes objetivos.

3.1. Objetivo general

Comparar el rendimiento de diferentes algoritmos de aprendizaje profundo y transfer learning sobre el dataset creado por el proyecto HATEMEDIA, con el objetivo de determinar cuál clasifica mejor y concluir si es posible la detección automática de expresiones de odio dentro de este caso de estudio.

3.2. Objetivos específicos

- Investigar las técnicas y métodos de aprendizaje automático profundo y transfer learning del estado del arte que abordan el problema de la detección del discurso del odio, para identificar qué técnicas y métodos nos conviene utilizar en nuestro estudio comparativo.
- Análisis exploratorio del dataset de HATEMEDIA con el objetivo de identificar potenciales problemas y oportunidades.
- Preprocesado y creación de diferentes versiones de nuestro dataset original; una versión completa con todos los registros preprocesados y otras versiones reducidas pero balanceadas.
- Entrenar los modelos seleccionados con las diferentes versiones de nuestro dataset y medir sus rendimientos.
- Evaluar los resultados obtenidos para determinar la viabilidad de detección de expresiones de odio y la preferencia de usar alguno de los modelos, si la hubiera.

3.3. Metodología del trabajo

La metodología del trabajo consistirá en seguir los pasos que se describen a continuación:

Tabla 1: Lista de tareas a realizar para la consecución de objetivos

	Tarea	Descripción
1	Lectura del estado del arte	Búsqueda de información sobre los recientes avances en técnicas y modelos de IA para la detección del discurso del odio.
2	Análisis exploratorio de los datos	Análisis exhaustivo de los datos disponibles en el dataset de Hatemadia, con el fin de entender las fortalezas y debilidades que nos ofrecen los datos de cara a resolver el problema planteado.
3	Selección de los modelos de aprendizaje profundo para realizar nuestra comparativa	De los algoritmos de aprendizaje profundo y transfer learning presentes en el estado del arte, decidir cuáles serán utilizados en nuestro estudio comparativo.
4	Preparación de los datos	Preparación de los datos necesarios para alimentar los algoritmos de aprendizaje profundo seleccionados, aplicando las transformaciones y normalizaciones necesarias.
5	Creación de diferentes versiones del dataset	Debido al desbalanceo de clases de nuestro conjunto de datos original, será necesario crear una nueva versión del dataset que contenga una proporción balanceada de etiquetas para poder comparar los resultados de las pruebas con cada dataset por separado.
6	Aplicación de técnicas seleccionadas sobre los datos disponibles	Utilizar los algoritmos y técnicas seleccionadas sobre los datasets disponibles para obtener resultados.
7	Identificar las métricas de evaluación	Determinar las métricas para la evaluación de los algoritmos seleccionados.
8	Análisis de resultados	Análisis comparativo de resultados para las distintas técnicas y modelos utilizados en el estudio.
9	Conclusiones y líneas futuras	Análisis de los resultados obtenidos y listar una serie de recomendaciones a aplicar en trabajos futuros.

4. Cómo detectar odio en medios de información social

En este trabajo queremos evaluar la viabilidad de utilizar técnicas de aprendizaje profundo y transfer learning sobre nuestro dataset de Hatemedia para obtener un modelo predictivo que permita la detección de expresiones de odio en castellano. Nuestra intención consiste en apoyarnos en estos datos para investigar, en primer lugar, si es viable entrenar un modelo de clasificación binario que permita detectar si un texto contiene odio (independientemente de su grado de intensidad) y, en caso afirmativo, determinaremos cuál de los modelos utilizados funciona mejor. De esta forma, podríamos utilizar este modelo para favorecer la detección y monitoreo de este tipo de expresiones en los entornos digitales. Por lo tanto, el objetivo de este trabajo no es conseguir desarrollar un algoritmo novedoso que resuelva total o parcialmente el problema tratado, sino estudiar la viabilidad de la aplicación de técnicas ya existentes para determinar, en caso afirmativo, cuál de los algoritmos utilizados es la mejor opción.

4.1. DATASET

El dataset utilizado proviene del proyecto Hatemedia, que ha centrado su estudio en los principales medios informativos profesionales de España (La Vanguardia, ABC, El País, El Mundo y 20Minutos), para analizar cómo se difunden las expresiones de odio en los entornos digitales asociados a este tipo de medios. En este dataset podemos encontrar más de 500.000 textos etiquetados según su grado de odio, textos procedentes tanto de publicaciones de medios informativos como de mensajes de usuarios que interactúan con estos desde sus cuentas sociales en Facebook, Twitter y en sus portales institucionales. A pesar de tratarse de un dataset con una buena cantidad de registros, tan solo una pequeña parte corresponden a textos de *ODIO*. Debido a esto, se ha decidido crear distintas versiones balanceadas del dataset original, de modo que podamos llevar a cabo diferentes pruebas en nuestra comparativa.

4.1.1. Dataset completo

Nuestro dataset original sufre del problema del desbalanceo, donde existe una clase que está representada en menor medida. De 574.760 registros, 12.296 están etiquetados como *ODIO* (el 2,1% de los datos), mientras que el 97,9% restante se corresponde con la etiqueta de *NO_ODIO* (Figura 6).

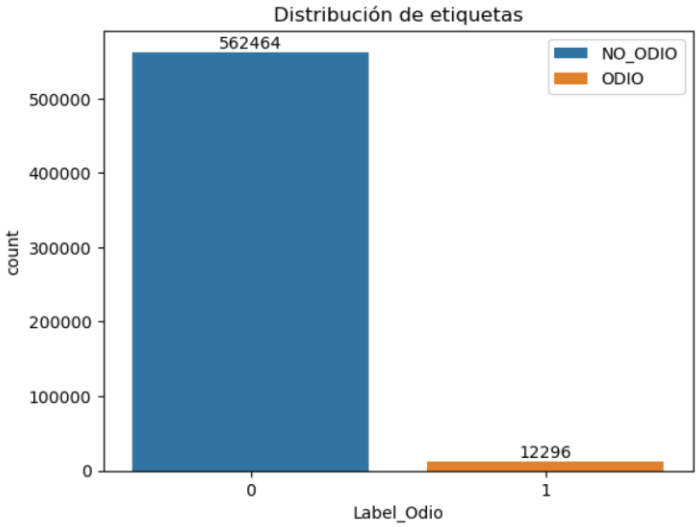

Figura 6: Distribución de etiquetas del dataset original

Por lo general, el desbalance de datos afecta a los algoritmos en su proceso de generalización, traduciéndose en que nuestro modelo entrenado no tenga una capacidad de predicción que nos sirva para su uso posterior (Chawla et al., 2004). Intentaremos paliar este problema mediante la creación de datasets alternativos a partir del original, con un número balanceado de clases, y compararemos los resultados obtenidos por separado.

4.1.2. Datasets balanceados

Crearemos 3 subconjuntos distintos de datos a partir del dataset original, prestando atención al número de muestras de cada clase para obtener un dataset balanceado. Para ello, seleccionaremos todos los mensajes etiquetados como *ODIO* y añadiremos la misma cantidad de mensajes etiquetados como *NO_ODIO*, atendiendo a diferentes criterios para cada uno de los nuevos datasets. Llamaremos a estas versiones de los datasets V1, V2 y V3 respectivamente.

- **Selección aleatoria de textos (V1)**: Tomaremos todos los textos etiquetados como *ODIO* y añadiremos aleatoriamente la misma cantidad de textos de *NO_ODIO*.

- **Selección de textos de longitud homogénea (V2)**: En nuestro dataset original tenemos textos que van desde 1 sola palabra hasta una longitud máxima de 3.044. A la hora de entrenar un algoritmo para que pueda aprender a clasificar textos en *ODIO* y *NO_ODIO*, será importante conocer si obtener un subconjunto de textos de longitud homogénea supone alguna mejora en el rendimiento. Para ello crearemos un nuevo dataset balanceado, consistente en textos de longitud homogénea.

- **Selección de textos correspondientes a un mismo medio (V3)**: La detección de expresiones de odio en textos de internet es un problema complejo, tal y como hemos comprobado en la sección **2. Contexto y estado del arte**. Acotar el ámbito de estos textos podría mejorar el rendimiento de los modelos, y eso es precisamente lo que vamos a analizar con este dataset, donde escogeremos textos relacionados con un solo medio de entre todos los disponibles (EL PAÍS, EL MUNDO, LA VANGUARDIA, 20MIN y ABC). Elegiremos el medio en función de cual tenga el mejor balance entre muestras *ODIO* y *NO_ODIO* y, dependiendo de los resultados obtenidos por el dataset anterior, seleccionaremos o no únicamente textos de longitud homogénea.

4.2. MODELOS DE APRENDIZAJE PROFUNDO PARA LA DETECCIÓN DEL ODIO

Para realizar nuestra comparativa, hemos seleccionado un total de 4 modelos predictivos (3 modelos de deep learning y 1 modelo de transfer learning) de los mencionados en el apartado **2.3. Técnicas y modelos**. Para decidir el diseño final de los modelos a utilizar, como el número de capas de convolución para la CNN, número y tamaño de los filtros, añadir o no más de una capa densa de neuronas, decidir si incluir capas de dropout, etc, hemos realizado pruebas tomando distintas combinaciones, entre ellas las configuraciones presentadas en el trabajo de Benítez-Andrades et al. (2022), donde se realiza un análisis comparativo de modelos con el objetivo de detectar racismo y xenofobia en twitter usando redes CNN, LSTM y transfer

learning. Finalmente, hemos optado por las arquitecturas más sencillas posibles que se describen a continuación, debido a que arquitecturas más complejas aumentaban considerablemente el tiempo de ejecución sin aumentar apenas el rendimiento, probablemente por sobre ajustarse demasiado a los datos de entrenamiento (overfitting).

- **SNN:** En primer lugar, utilizaremos un clasificador basado en un modelo de red neuronal simple (SNN, simple neural network en inglés). Este sencillo modelo consistirá en una primera capa de embedding que será posteriormente aplanada y conectada directamente a una capa densa de 1 neurona con una función de activación *sigmoid*, que será la encargada de devolver el resultado de la clasificación binaria. Este modelo SNN nos servirá de línea base o *baseline*, pues la capacidad predictiva en este caso residirá en la capa de embedding, cuya salida proveerá vectores bidimensionales que serán las representaciones de cada uno de nuestros textos. La capa densa de una neurona será la encargada de devolver como salida un valor entre 0 y 1, que será el que utilizaremos para determinar si el texto se clasifica como *ODIO* (> 0,5) o *NO_ODIO* (<=0,5).

- **CNN:** En segundo lugar, utilizaremos un modelo CNN, con una primera capa de embedding, seguida por 1 capa convolucional 1D (probaremos diferente número y tamaño de filtros para seleccionar la mejor combinación). La función de activación utilizada en esta capa será la función ReLU (Unidad Lineal Rectificada), que en la actualidad es la función de activación con más éxito y más utilizada en redes de neuronas profundas (Ramachandran et al., 2017). A la salida de esta capa de convolución se le aplicará una función de MaxPooling para reducir el tamaño de las muestras, y el resultado se conectará a una capa densa de 1 neurona con una función sigmoid.

- **LSTM:** En tercer lugar, seleccionamos para realizar nuestra comparativa el modelo recurrente LSTM, donde utilizaremos en primer lugar una capa de embedding, seguida de una capa LSTM (probaremos diferente número de neuronas para poder seleccionar la mejor opción). La salida irá conectada, al igual que en los casos anteriores, a una capa densa de 1 neurona con función de activación sigmoid.

- **BETO:** Finalmente, utilizaremos en nuestra comparativa el modelo Transformers monolingüe para el idioma español BETO, tanto la versión *cased* como *uncased (*"dccuchile/bert-base-spanish-wwm-cased" y "dccuchile/bert-base-spanish-wwm-uncased" respectivamente*)*. Estos modelos se pueden encontrar en la web de Hugging Face [22], y son accesibles desde el código a través de la biblioteca Transformers[23]. La librería Hugging Face, además de soportar una variedad de diferentes modelos de Transformers pre-entrenados, incluye versiones preconstruidas adaptadas a una tarea específica, como por ejemplo clasificación de texto. Para nuestras pruebas utilizaremos *BertForSequenceClassification*[24].

4.3. MÉTRICAS DE EVALUACIÓN

Como métricas para comparar los distintos modelos entrenados vamos a utilizar:

- **Accuracy** (Exactitud): Esta métrica indica el número de muestras correctamente clasificadas para todas las clases sobre el total de muestras. En nuestro caso, al tener conjuntos de datos muy desequilibrados, este parámetro por sí solo no nos es suficiente ya que podemos clasificar muy bien la clase mayoritaria, teniendo valores altos de exactitud y, sin embargo, detectar muy mal la clase minoritaria, en este caso los textos de *ODIO*. La fórmula para calcular el accuracy es la siguiente:

$$Acc = \frac{muestras\_correctas}{\text{total muestras}} = \frac{TP+TN}{TP+TN+FP+FN}$$

- **Precisión**: nos indica lo precisa que es nuestra clasificación, es decir, de las muestras reconocidas en una clase cuántas son correctas.

[22] https://huggingface.co/dccuchile
[23] https://huggingface.co/docs/transformers
[24] https://huggingface.co/transformers/v3.0.2/model_doc/bert.html#bertforsequenceclassification

$$\text{Precisión} = \frac{\text{muestras clasificadas correctamente en una clase}}{\text{total muestras clasificadas en esa clase}} = \frac{TP}{TP+FP}$$

- **Recall** (Exhaustividad): Esta métrica es también conocida como el ratio de verdaderos positivos y es utilizada para saber cuántos valores positivos son correctamente clasificados.

$$Recall = \frac{muestras\ clasificadas\ correctamente\ en\ una\ clase}{total\ muestras\ de\ la\ clase} = \frac{TP}{TP+FN}$$

Donde,

TP = True Positive o muestra clasificada en una clase de forma correcta.

TN = True Negative o muestra no clasificada en una clase correctamente.

FP = False Positive o muestra clasificada en una clase cuando no pertenece a ella.

FN = False Negative o muestra no clasificada en una clase cuando sí pertenece a ella.

- **F1-score**: Una métrica que combina Precisión y Recall.

$$\text{F1} = \frac{2*Precisión*Recall}{Precisión+Recall}$$

- **Macro-F1**: Se trata de la media no ponderada de las puntuaciones F1-score.

$$\text{Macro-F1} = \frac{sum(F1-scores)}{número\ de\ clases}$$

5. DESARROLLO DE MODELOS DE APRENDIZAJE PROFUNDO PARA LA DETECCIÓN DE ODIO

A continuación, se detalla el trabajo realizado en este estudio, desde la preparación de datos hasta el entrenamiento y evaluación de los modelos y su posterior comparativa.

5.1. ANÁLISIS Y PREPARACIÓN DE LOS DATOS

En un primer vistazo, observamos que nuestro dataset contiene 9 columnas (Figura 7).

	medio	soporte	url	tipo_mensaje	texto	intensidad	tipo_odio	tono_humoristico	modificador
0	EL PAÍS	WEB	https://elpais.com/deportes/2021-01-20/alcoyan...	COMENTARIO	el barça nunca acaeza ante un segundo b ni ant...	3.0	Otros	NaN	NaN
1	EL PAÍS	WEB	https://elpais.com/deportes/2021-01-20/alcoyan...	COMENTARIO	el real madrid ha puesto punto y final a su an...	0.0	NaN	NaN	NaN
2	EL PAÍS	WEB	https://elpais.com/espana/2021-01-18/comienza-...	COMENTARIO	cristina cifuentes podría haber sido la presid...	3.0	Ideológico	NaN	NaN
3	EL PAÍS	WEB	https://elpais.com/espana/2021-01-18/comienza-...	COMENTARIO	habría que reabrir el caso. el supremo se dedi...	3.0	Ideológico	NaN	NaN
4	EL PAÍS	WEB	https://elpais.com/espana/2021-01-18/comienza-...	COMENTARIO	me parece un poco exagerado pedir más de tres ...	3.0	Ideológico	Si	NaN

Figura 7: Extracto de las cinco primeras filas del dataset original.

- **Medio**: Indica el medio digital de donde se ha extraído el texto. En nuestro dataset tenemos 5 valores diferentes (El PAÍS, EL MUNDO, LA VANGUARDIA, 20MIN y ABC).
- **Soporte**: Indica el soporte del medio (Web, Twitter).
- **Url:** Link al texto. Hemos comprobado casos en el que los links no corresponden con el texto al que debería apuntar, por lo que consideramos que esta columna es poco fiable.
- **Tipo_mensaje:** Tipo mensaje puede tomar 3 valores (COMENTARIO, NOTICIA y TITULAR_NOTICIA).
- **Texto:** Es el texto para clasificar. Nuestra variable objetivo.
- **Intensidad:** Indica la intensidad de la expresión de odio, con siete posibles valores que van desde 0.0 hasta 6.0. El valor 0.0 indica que el texto no contiene odio, mientras que el resto de los valores corresponden a una intensidad de odio. Como en nuestro estudio sólo nos interesa realizar una clasificación binaria, transformamos esta columna a valores 0 y 1 (*NO_ODIO* y *ODIO* respectivamente), y la renombramos a *label_odio*.

- **Tipo_odio:** Indica el tipo de odio del texto. Este campo solo es aplicable a textos etiquetados como *ODIO*. Existen 7 posibles valores (Racismo, Sexual, Misoginia, Religioso, Xenofobia, Ideológico y Otros), además de combinaciones entre ellos (por ejemplo, "Racismo, Misoginia") hasta un total de 72 combinaciones distintas.
- **Tono_humorístico:** Es un booleano que indica si existe humor en el texto etiquetado como *ODIO*. (Solo aplicable a textos etiquetados como *ODIO*).
- **Modificador:** Contiene los valores: "Humor", "Atenuador", "Intensificador" y la combinación "Intensificador, Atenuador". (Solo aplicable a textos etiquetados como *ODIO*).

A continuación, realizamos un análisis pormenorizado de los datos disponibles en el dataset de Hatemedia, con el fin de entenderlos en profundidad y comprobar la calidad de los mismos.

5.1.1. Tratamiento de los valores nulos.

En este apartado perseguimos dos objetivos principales: 1) eliminar registros cuando el campo "contenido" o "intensidad" es nulo; 2) si tenemos nulos en otras columnas, decidir qué hacer con ellos.

En la Figura 8 mostramos el número de campos nulos para cada columna de nuestro dataset. Los 562.467 registros con valor nulo en las columnas "tipo_odio" y "tono_humorístico" y los 574.410 de "modificador" son esperados, ya que se trata de columnas que, de tomar un valor, solo lo toman cuando el texto en cuestión es etiquetado como *ODIO*. Para el resto de los textos, su valor debe ser siempre nulo.

```
cols = df.columns.values
for columna in cols:
    print ('Columna', columna ,':', df[df[columna].isnull()].shape[0],'nulos.')
```

```
Columna medio : 0 nulos.
Columna soporte : 0 nulos.
Columna url : 0 nulos.
Columna tipo_mensaje : 1 nulos.
Columna texto : 0 nulos.
Columna intensidad : 0 nulos.
Columna tipo_odio : 562467 nulos.
Columna tono_humoristico : 574617 nulos.
Columna modificador : 574410 nulos.
Columna label_odio : 0 nulos.
```

Figura 8: Conteo de campos nulos por columna

Si embargo, observamos también que tenemos un valor nulo en la columna tipo_mensaje.

Para decidir qué hacer con este registro, hemos seguido los siguientes pasos.

1. En primer lugar, hemos accedido a la url asociada a este registro: *https://twitter.com/1022840547118145536/status/1347858101454704642*
   Sin embargo, a pesar de que hace referencia a un tweet del medio "ABC", hemos comprobado que esta url no se corresponde con el texto en cuestión, si no con otro distinto. Tras hacer la prueba con otras urls del fichero, confirmamos que esta columna no tiene datos fiables.
2. En segundo lugar, hemos estudiado el valor que contiene el campo tipo_mensaje para otros registros cuyo medio es "ABC" y su soporte es "Twitter", para intentar deducir qué valor podría ser el más probable para nuestro campo nulo. Los valores que toma este campo para otros registros similares son: "COMENTARIO", "NOTICIA" o "NaN". Inicialmente, podríamos pensar que los textos catalogados como "NOTICIA" son textos más largos y elaborados, mientras que los de tipo "COMENTARIO" podrían corresponderse a textos significativamente más cortos. Sin embargo, comprobamos que no es así, puesto que podemos observar algunos ejemplos de tipo_mensaje = "NOTICIA" que están compuestos por textos de muy pocas palabras. Así pues, consideramos la hipótesis de que la columna tipo_mensaje (al menos para los registros con medio y soporte "ABC" y "TWITTER") hace la distinción de cuándo un comentario se escribe en contestación a una noticia directamente, y cuándo se escribe en contestación a otro comentario. Es decir, cuando un usuario escribe un comentario en referencia a una noticia, su campo tipo_mensaje sería "NOTICIA" y en el caso de que un usuario responda a otro comentario, su texto se catalogaría como tipo_mensaje = "COMENTARIO". En cualquier caso, esto es solo una suposición.

Como no somos capaces de determinar con certeza el valor de tipo_mensaje para este caso y teniendo en cuenta que el campo label_odio es 0 (correspondiente a la etiqueta de *NO_ODIO*, que es la etiqueta mayoritaria en nuestro dataset), decidimos eliminar el mensaje.

5.1.2. Análisis exploratorio y visualización de los datos

Una vez hemos terminado el tratamiento de los valores nulos en el dataset, nos disponemos a realizar un análisis exploratorio de los datos. Actualmente sabemos que el dataset está claramente desbalanceado (ver Figura 1), con 562.464 observaciones de *NO_ODIO*, frente a 12.296 de *ODIO*. A continuación, realizaremos un estudio de cómo se distribuyen los datos en función de las distintas variables, para entender un poco mejor el dataset.

En primer lugar, estudiamos la distribución de los datos en relación a la variable SOPORTE. En la Figura 9 podemos observar que el 60% del dataset corresponde a textos de soporte WEB, mientras que el 40% corresponde a Twitter. Podría parecer que nuestros datos están bien representados en ambos soportes. Sin embargo, si atendemos únicamente a los textos de *ODIO,* podemos apreciar que el 77% están vinculados a Twitter. Este dato es interesante, porque siendo los textos de odio tan solo un 2% de los textos totales del dataset, el 77% de esta minoría están asociados a la plataforma Twitter.

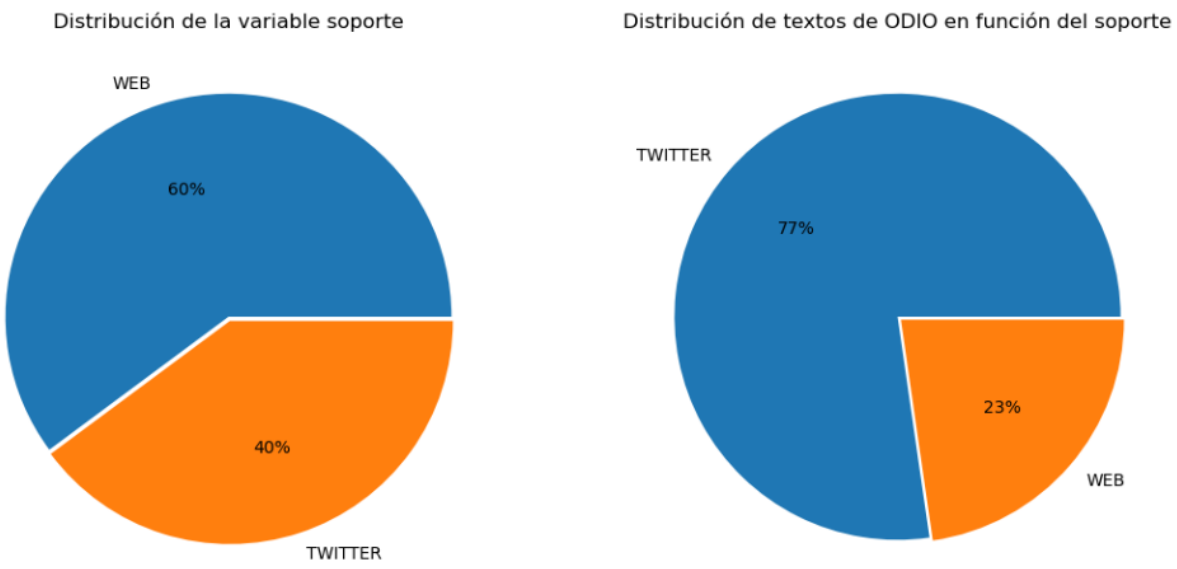

Figura 9: Distribución de los datos en función de la variable soporte (izquierda), y la misma distribución pero considerando solo los textos de ODIO (derecha)

Estudiaremos ahora la distribución de la variable MEDIO con respecto a la variable SOPORTE:

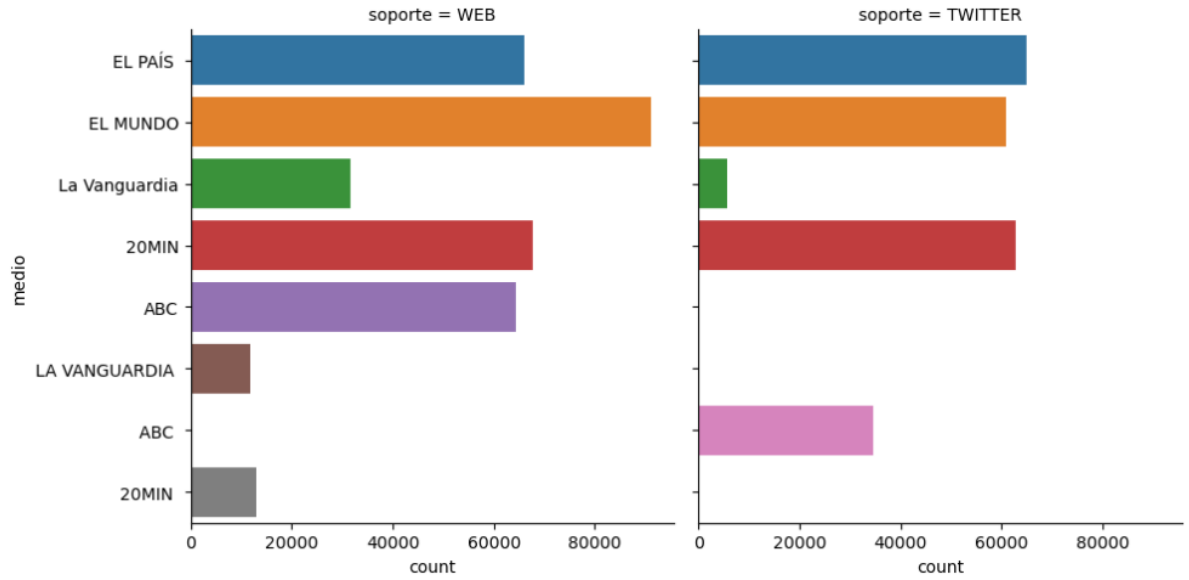

Figura 10: Distribución de la variable MEDIO con respecto a la variable SOPORTE (con medios duplicados)

En la Figura 10 podemos ver que tenemos medios duplicados: "La Vanguardia", "20MIN" y "ABC". Normalizamos el nombre de los medios duplicados o con espacios sobrantes.

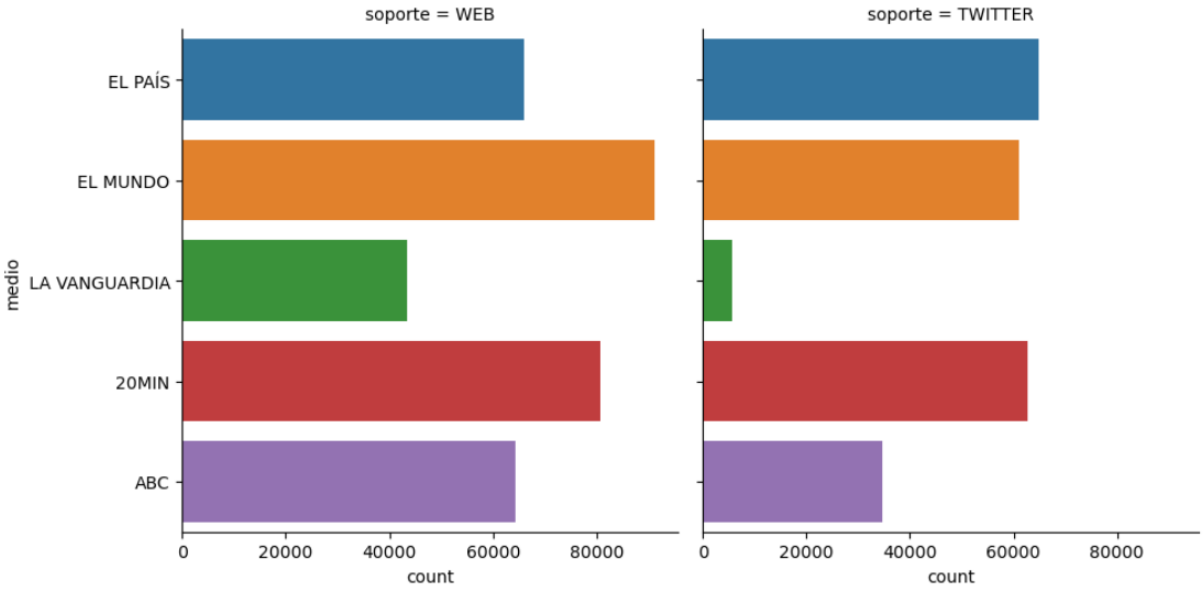

Figura 11: Distribución de la variable MEDIO con respecto a la variable SOPORTE (tras normalización de los nombres de los medios)

La Figura 11 muestra un gráfico con los nombres de los medios normalizados donde podemos apreciar que todos los medios tienen presencia en ambos soportes (WEB y Twitter).

Comparando los medios entre sí, podemos comprobar que "La Vanguardia" es el medio con menos textos en nuestro dataset. Sin embargo, este hecho no debería suponer ningún problema para nuestro estudio.

Ahora estudiamos la distribución de los datos únicamente en relación con la variable MEDIO.

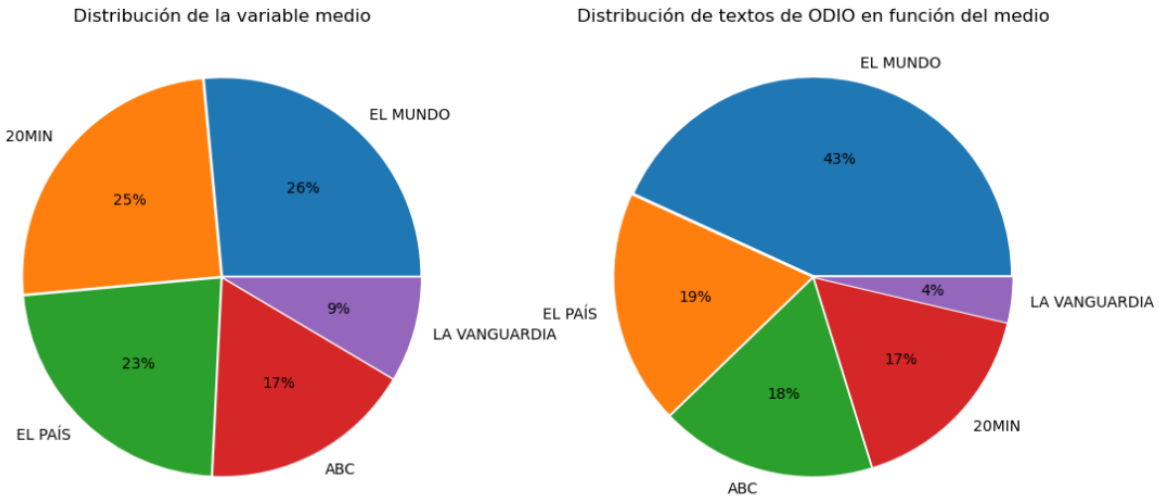

Figura 12: Distribución de la variable medio (izquierda), y la misma distribución pero considerando solo los textos de ODIO (derecha)

En la Figura 12 podemos apreciar que EL MUNDO no solo es el medio con mayor presencia en nuestro dataset (26%), sino que además es de donde se generan la mayor parte de los textos de *ODIO* (un 43% del total). Por lo tanto, EL MUNDO parece la mejor opción si decidimos crear un dataset balanceado con textos provenientes de un mismo medio para añadir a nuestra comparativa.

Analizamos ahora la distribución de la variable tipo_mensaje con respecto a la etiqueta label_odio (Figura 13).

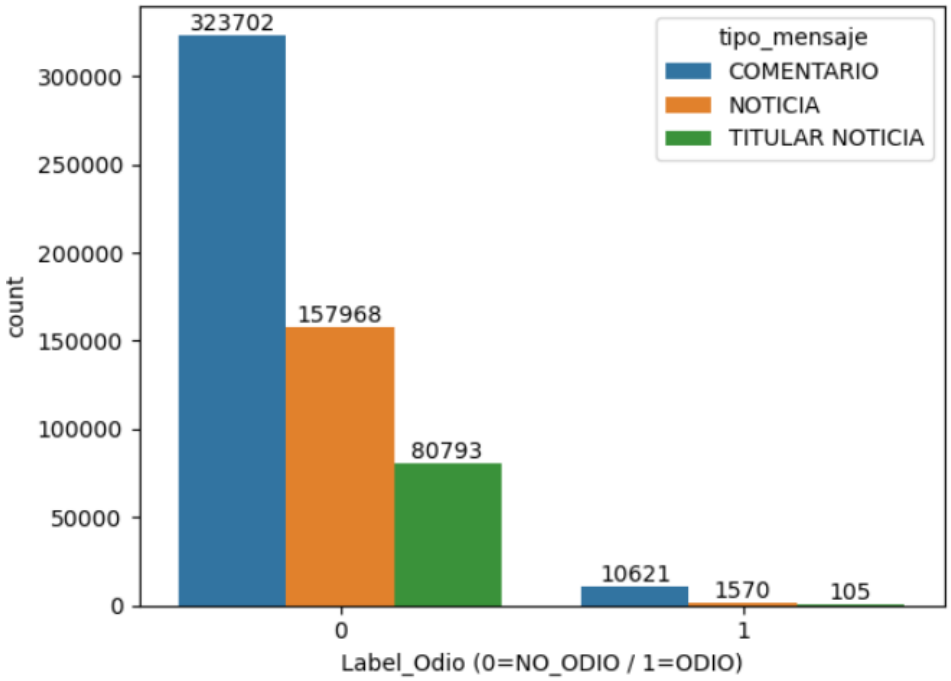

Figura 13: Distribución de la variable tipo_mensaje con respecto a la etiqueta label_odio

Vemos que, para ambos casos (*ODIO* y *NO_ODIO*), la categoría más común es “COMENTARIO”, y la menos común es “TITULAR_NOTICIA”.

Las tres columnas restantes de nuestro dataset: tipo_odio, tono_humorístico y modificador, no serán necesarias para nuestro estudio, ya que estas columnas únicamente dan información adicional para los mensajes de odio (para el resto de los casos toman el valor nulo). A nosotros nos bastará con conocer la etiqueta *ODIO* y *NO_ODIO* para cada texto, con el objetivo de entrenar nuestro modelo, sin necesidad de entrar en más detalle sobre el tipo de odio. Aun así, hemos realizamos un pequeño análisis y estos han sido los resultados:

- Tipo_odio: Observamos que, de los 12.296 textos de odio presentes en nuestro dataset, la gran mayoría están catalogados con tipo_odio = "Otros" (73%), seguido por tipo_odio = "Ideológico" (14%).

- Tono_humorístico: De los 12.296 textos de odio, tan solo 143 (1%) han sido catalogados con tono_humorístico = ”Si”. El resto tienen valor nulo.

- Modificador: Tan solo el 3% de los mensajes de odio contienen valor en este campo. Los valores presentes son “Atenuador” (114 observaciones), “Intensificador” (233), la tupla “Intensificador, Atenuador” (2) y Humor (1).

Tras este análisis exploratorio, decidimos eliminar las columnas tipo_odio, tono_humorístico y modificador por no aportar valor a nuestro estudio. También decidimos borrar el registro que contiene el campo “tipo_mensaje” a nulo, de modo que dejamos un dataset libre de valores nulos.

5.1.3. Preparación de la columna Texto

El siguiente paso será tratar la columna texto de nuestro dataset. La columna texto es la que contiene los mensajes a clasificar como *ODIO / NO_ODIO*, por lo que debemos realizar un preprocesado de modo que aseguremos que el contenido de esta columna es óptimo para poder entrenar nuestros modelos de inteligencia artificial.

Para ello, creamos una función que implemente un flujo de limpieza y preprocesado de los datos, consistente en los siguientes pasos:

- Limpieza de URLs
- Eliminación de @ y su mención
- Eliminación de los caracteres especiales
- Eliminación de palabras con longitud <2
- Eliminación de espacios en blanco adicionales
- Tokenización
- Lematización

```python
def limpiar_texto(texto):
    """Función que dado un texto devuelve ese mismo texto preprocesado, realizando las siguientes tareas:"""

    # Eliminamos los links de las URLs
    texto = re.sub(r'''(?i)\b((?:https?://|www\d{0,3}[.]|[a-z0-9.\-]+[.][a-z]{2,4}/)(?:[^\s()<>]+|\(([^\s()<>]+|(\([^\s()<>]+\)))

    #Eliminamos la @ y su mención
    texto = re.sub(r"@[A-Za-z0-9À-ÿ\u00f1\u00d1]+", ' ', texto)

    # Eliminamos los caracteres especiales.
    texto = re.sub(r'\W', ' ', texto)

    # Eliminamos caracteres sueltos (longitud <2)
    texto = re.sub(r'\b\w{1}\b', ' ', texto)

    # Eliminamos espacios en blanco adicionales
    texto = re.sub(r" +", ' ', texto)

    # Eliminamos espacio en blanco del principio (en caso de que hubiera)
    texto = re.sub(r'^\s', '', texto)

    # Eliminamos espacio en blanco del final (en caso de que hubiera)
    texto = re.sub(r'\s+$', '', texto)

    return texto
```

Figura 14: Extracción de código que implementa el flujo de preprocesado de la columna texto

En la Figura 14 mostramos el código de la función *limpiar_texto*, que comienza por eliminar las urls, que no nos van a dar ningún valor a lo hora de hacer nuestra clasificación. Así mismo, eliminamos las menciones propias de los tweets (expresiones que empiezan por @).
Continuamos por eliminar todos los caracteres especiales de los textos utilizando la expresión regular re.sub(r'\W', ' ', texto). Tras ello, eliminamos los caracteres sueltos (longitud <2), y también eliminamos los espacios en blanco adicionales que nos hayan podido quedar, tanto en el interior del texto, como al principio y al final.

```python
nlp = spacy.load('es_core_news_sm')
def normalizar (t):
    """Función que dada una lista de textos, devuelve esa misma lista de textos
       con los textos normalizados, realizando las siguientes tareas:
       1.- Pasamos la palabra a minúsculas
       2.- Lematizamos la palabra
       3.- Eliminamos todas las palabras que no nos aporten valor desde el punto de vista de su categoria gramatical.
    """

    #POS -> incluimos PROPN para mantener nombres propios, AUX porque en spacy español perderíamos formas verbales
            #si no lo incluimos (por ejemplo estoy, sería y será son etiquetados como AUX por spacy).
            #También incluimos INTJ (interjecciones), porque hemos detectado que spacy no hace un uso correcto de esta etiqueta
            #y adjetivos como "traidor" son consideradas INTJ.

    for index, texto  in enumerate(tqdm(t)):
        texto = nlp(texto.lower())
        t[index] = " ".join([word.lemma_ for word in texto if word.pos_ in ['NOUN', 'ADJ', 'VERB', 'ADV','PROPN', 'AUX', 'INTJ']]


    return t
```

Figura 15: Extracción de código que implementa el flujo de normalización y lematización de la columna texto

En la Figura 15 podemos ver el código de la función normalizar. Esta función se ayuda de la librería spacy para pasar el texto a minúsculas y lematizar cada palabra. Por último, nos quedamos solo con las palabras que pertenezcan a ciertas categorías gramaticales que

consideramos útiles para nuestro estudio, como nombres, adjetivos, verbos y adverbios. Hemos decidido quedarnos con las etiquetas gramaticales AUX (verbo auxiliar) e INTJ (interjección), porque en el idioma español, las palabras etiquetadas como AUX son formas verbales que deseamos mantener, como por ejemplo “estoy”, “sería” y “será”. En el idioma inglés, las palabras con la etiqueta AUX normalmente se eliminarían del estudio. En el caso de la etiqueta INTJ, hemos detectado que spacy no hace un uso correcto de esta etiqueta para el idioma español en todos los casos, de modo que durante nuestra investigación hemos encontrado que adjetivos como "traidor" son consideradas INTJ. Como no queremos perder estas palabras, decidimos mantener todas las palabras etiquetadas como INTJ.

Tras realizar el preprocesado de los datos, mostramos algunos ejemplos del resultado:

```
Texto sin procesar:
 a mí me la trae floja el resultado. esta señora no daba la talla. como otros ya aludidos.
Texto procesado:
 traer flojo resultado señora no dar talla ya aludido

********************

Texto sin procesar:
 actuar si que actuó. presuntamente coaccionando a las “profesoras”. lo de buena fe ya es de traca.
Texto procesado:
 actuar actuar presuntamente coaccionar profesora buena fe ya ser traca

********************

Texto sin procesar:
 hostia!!! será que por fin va a reconocer los derechos de sus hijos extramaritales!!!

españa merece un rey bastardo!!!
Texto procesado:
 hostia ser fin ir reconocer derecho hijo extramarital españa merecer rey bastardo
```

Figura 16: Comparación del texto antes y después de aplicar preprocesado

En la Figura 16 podemos comprobar que, tras el preprocesado del texto, hemos eliminados signos de puntación, espacios en blanco sobrantes, saltos de línea y cualquier otro carácter especial. Asimismo, las palabras han sido normalizadas y lematizadas.

Tras la realización del preprocesado, observamos que hemos perdido 23 textos de *ODIO* y 2.522 de *NO_ODIO*, todos ellos comentarios compuestos por palabras intranscendentes o mal escritas que se han convertido en textos nulos, tal y como podemos ver en la Figura 17.

texto_procesado	texto_sin_procesar
NaN	esa misma
NaN	quo
NaN	¡uf!
NaN	@g2reven
NaN	@lavanguardia
NaN	d.e.p
NaN	d.e.p
NaN	d. e. p.
NaN	d. e. p.
NaN	y?

Figura 17: Textos procesados nulos vs textos sin procesar

Eliminamos todas esas filas con textos nulos, ya que no nos aportarán nada para nuestra investigación.

5.1.4. Estudio de la longitud de los textos

Analizar la longitud de los textos como una variable más nos revelará información importante sobre nuestros datos, como la longitud máxima y mínima, así como su distribución. Los modelos de aprendizaje profundo pueden mostrar un comportamiento muy diferente en función de las longitudes de los textos que se les proporciona, tanto desde el punto de vista de rendimiento como de tiempo de ejecución.

Para poder analizar la longitud de los textos programamos una función lambda que cuente las palabras de cada uno de ellos y las almacenamos en una nueva columna de nuestro dataset llamada “num_palabras”. En la Figura 18 mostramos como queda nuestro dataset procesado, tras añadir la nueva columna “num_palabras”, eliminar las filas y columnas innecesarias y con los textos preprocesados.

```
df_join['num_palabras'] = df_join['texto'].apply(lambda t: len(t.split(" ")))
df_join
```

	texto	label_odio	medio	soporte	tipo_mensaje	num_palabras
0	barça nunca acaezar segundo tercero ya estar a...	1	EL PAÍS	WEB	COMENTARIO	14
1	real madrid haber poner punto final andadura c...	0	EL PAÍS	WEB	COMENTARIO	103
2	cristinar cifuent poder haber ser presidenta m...	1	EL PAÍS	WEB	COMENTARIO	74
3	haber reabrir caso supremo dedicar proteger si...	1	EL PAÍS	WEB	COMENTARIO	9
4	parecer poco exagerado pedir más año prisión c...	1	EL PAÍS	WEB	COMENTARIO	78
...	...	...	...	...	...	...
574755	sabéis estar abultado tamaño pelota leer tonte...	1	EL PAÍS	TWITTER	COMENTARIO	10
574756	sólo dinero público invertir vuestro panfleto ...	1	EL PAÍS	TWITTER	COMENTARIO	14
574757	panfleto criticar gasto públco sois puta mierd...	1	EL PAÍS	TWITTER	COMENTARIO	19
574758	servicio enfermo estar problema oler izquierdo...	1	EL PAÍS	TWITTER	COMENTARIO	7
574759	así ser iréis mierda ya no valeis panfleto	1	EL PAÍS	TWITTER	COMENTARIO	8

572214 rows × 6 columns

Figura 18: Creación columna num_palabras que contiene la longitud de los textos

A continuación, realizamos un estudio detallado basado en nuestro nuevo campo "num_palabras". El objetivo es conocer cómo se distribuye esta variable para entender si existe algún patrón de correlación y sacar conclusiones que puedan ayudarnos a abordar nuestro problema. Para ello, obtendremos estadísticas basadas en la longitud de los textos como la longitud media, mínima y máxima, primero de forma general (teniendo en cuenta todos los textos) y a continuación centrándonos únicamente en los textos de odio. Asimismo, obtendremos estadísticas de la longitud de textos en función de las variables TIPO_MENSAJE y MEDIO.

5.1.4.1. Estudio general de longitud de los textos

En la Figura 19 se muestran las estadísticas relacionadas con la longitud de los textos de nuestro dataset. Observamos que tenemos textos que van desde 1 una sola palabra (estos textos tan cortos probablemente no nos aporten información útil a la hora de entrenar nuestro clasificador), hasta 3.044 palabras, con una media global de 60 palabras por texto.

```
Longitud general de los textos

La longitud media de los textos es: 60 palabra(s)
La longitud min de los textos es: 1 palabra(s)
La longitud max de los textos es: 3044 palabra(s)

Estadística detallada para la longitud de los textos del dataset:

count    572214.000000
mean         60.087796
std         145.839963
min           1.000000
25%           5.000000
50%          10.000000
75%          26.000000
max        3044.000000
Name: num_palabras, dtype: float64
```

Figura 19: Estadísticas para la longitud general de los textos del dataset

Probablemente, los textos que contienen una única palabra no aporten información útil a la hora de entrenar el clasificador. Es muy probable también que la diferencia tan grande que existe entre los textos de longitud mínima y máxima perjudique el rendimiento del clasificador ya que, tras el proceso de padding, los textos más cortos se representarán como un vector de unos pocos enteros al inicio y más de 3000 ceros al final. Por este motivo, una parte de nuestra investigación será trabajar con un dataset compuesto de textos de longitud homogénea para poder comparar los resultados obtenidos.

También podemos apreciar en estas estadísticas que el 75% de los textos de nuestro dataset está compuesto por 26 palabras o menos. Esto quiere decir que la longitud máxima de 3044 es un valor atípico. Más adelante analizaremos la longitud de los textos en función de otras variables como TIPO_MENSAJE o MEDIO, de modo que podamos entender si los textos más largos guardan alguna relación con cierto tipo de variables.

5.1.4.2. Longitud en función de etiqueta label_odio

En la Figura 20 mostramos el mismo estudio anterior pero esta vez atendiendo a la etiqueta label_odio. Comprobamos que, de media, los mensajes de *ODIO* son significativamente más cortos que los etiquetados como *NO_ODIO,* 9 palabras frente a 61. El 75% de los textos de odio contienen 11 palabras o menos. Cuando construyamos nuestro dataset balanceado en base a la longitud de las palabras, tendremos que tener en cuenta estas estadísticas que

devuelven los mensajes de odio para crear un dataset donde la media de la longitud de los textos sea alrededor de nueve palabras.

```
Longitud de los mensajes en función de etiqueta ODIO / NO_ODIO:

Para mensajes de ODIO
La longitud media de los mensajes de ODIO es: 9 palabra(s)
La longitud min de los mensajes de ODIO es: 1 palabra(s)
La longitud max de los mensajes de ODIO es: 1301 palabra(s)

Para mensajes de NO_ODIO
La longitud media de los mensajes de NO_ODIO es: 61 palabra(s)
La longitud min de los mensajes de NO_ODIO es: 1 palabra(s)
La longitud max de los mensajes de NO_ODIO es: 3044 palabra(s)

Estadística detallada para la longitud de los mensajes de ODIO:

count    12273.000000
mean         9.035199
std         18.127268
min          1.000000
25%          3.000000
50%          6.000000
75%         11.000000
max       1301.000000
Name: num_palabras, dtype: float64

Estadística detallada para la longitud de los mensajes de NO_ODIO:

count    559941.000000
mean         61.206786
std         147.207007
min           1.000000
25%           5.000000
50%          10.000000
75%          27.000000
max        3044.000000
Name: num_palabras, dtype: float64
```

Figura 20: Estadísticas para la longitud de los textos en función de su etiqueta label_odio

5.1.4.3. Longitud en función de tipo_mensaje y medio

Ahora nos disponemos a analizar la distribución de la variable NUM_PALABRAS en función de TIPO_MENSAJE y MEDIO, tanto para los textos en general, como únicamente para los textos etiquetados como *ODIO*. Los resultados son mostrados a continuación en las Figuras 21, 22 y 23:

Estadísticas de la longitud de los textos en función del TIPO_MENSAJE:

TITULAR NOTICIA	NOTICIA	COMENTARIO
count 80849.000000	count 159143.000000	count 332222.000000
mean 14.532054	mean 155.218703	mean 25.603982
std 56.672585	std 232.669502	std 62.377650
min 1.000000	min 1.000000	min 1.000000
25% 5.000000	25% 3.000000	25% 5.000000
50% 7.000000	50% 25.000000	50% 10.000000
75% 9.000000	75% 251.000000	75% 21.000000
max 1832.000000	max 3044.000000	max 2820.000000
Name: num_palabras, dtype: float64	Name: num_palabras, dtype: float64	Name: num_palabras, dtype: float64

Estadísticas de la longitud de los textos en función del TIPO_MENSAJE (solo para textos de odio):

TITULAR NOTICIA	NOTICIA	COMENTARIO
count 105.000000	count 1570.000000	count 10598.000000
mean 5.914286	mean 10.831847	mean 8.799962
std 5.282617	std 43.349364	std 10.069870
min 1.000000	min 1.000000	min 1.000000
25% 2.000000	25% 3.000000	25% 3.000000
50% 4.000000	50% 6.000000	50% 6.000000
75% 9.000000	75% 11.000000	75% 11.000000
max 34.000000	max 1301.000000	max 115.000000
Name: num_palabras, dtype: float64	Name: num_palabras, dtype: float64	Name: num_palabras, dtype: float64

Figura 21: Estadísticas de la longitud de los textos en función del tipo_mensaje (arriba) y las mismas estadísticas, pero centradas únicamente en los textos de odio (abajo)

En base a estas estadísticas observamos que los textos correspondientes a TITULAR_NOTICIA son significativamente más cortos que las NOTICIAS y COMENTARIOS, tanto para todos los textos en general como para los de odio. Si nos centramos únicamente en los textos de *ODIO,* podemos observar que el 75% de ellos tienen 11 palabras o menos para cualquier valor de TIPO_MENSAJE.

Estadísticas de la longitud de los textos en función del MEDIO

ABC	EL PAÍS	20MIN
count 98612.000000	count 130309.000000	count 143036.000000
mean 64.884456	mean 98.298529	mean 60.867761
std 146.945856	std 206.289317	std 114.070845
min 1.000000	min 1.000000	min 1.000000
25% 5.000000	25% 6.000000	25% 3.000000
50% 9.000000	50% 10.000000	50% 9.000000
75% 29.000000	75% 33.000000	75% 55.000000
max 2820.000000	max 2993.000000	max 2139.000000
Name: num_palabras, dtype: float64	Name: num_palabras, dtype: float64	Name: num_palabras, dtype: float64

LA VANGUARDIA	EL MUNDO
count 48738.000000	count 151519.000000
mean 41.238766	mean 29.430850
std 119.535639	std 102.311496
min 1.000000	min 1.000000
25% 4.000000	25% 6.000000
50% 8.000000	50% 11.000000
75% 15.000000	75% 22.000000
max 2870.000000	max 3044.000000
Name: num_palabras, dtype: float64	Name: num_palabras, dtype: float64

Figura 22: Estadísticas de la longitud de textos en función del MEDIO

Estadísticas de la longitud de los textos en función del MEDIO (solo para textos de odio):

ABC	EL PAÍS	20MIN
count 2154.000000 mean 8.473538 std 14.688545 min 1.000000 25% 3.000000 50% 5.000000 75% 10.000000 max 503.000000 Name: num_palabras, dtype: float64	count 2340.000000 mean 10.928632 std 34.993550 min 1.000000 25% 3.000000 50% 7.000000 75% 13.000000 max 1301.000000 Name: num_palabras, dtype: float64	count 2025.000000 mean 7.967901 std 9.020697 min 1.000000 25% 3.000000 50% 5.000000 75% 10.000000 max 92.000000 Name: num_palabras, dtype: float64

LA VANGUARDIA	EL MUNDO
count 453.000000 mean 8.048565 std 6.930587 min 1.000000 25% 3.000000 50% 6.000000 75% 10.000000 max 47.000000 Name: num_palabras, dtype: float64	count 5301.000000 mean 8.919638 std 9.766324 min 1.000000 25% 3.000000 50% 6.000000 75% 11.000000 max 92.000000 Name: num_palabras, dtype: float64

Figura 23: Estadísticas de la longitud de textos en función del MEDIO, pero únicamente centrado en los textos de odio

Las estadísticas relacionadas con la longitud de textos en función del medio no nos aportan mayor conocimiento para textos de *ODIO*, seguimos viendo que el 75% están compuestos por 13 palabras o menos, variando el valor medio en pocas palabras según el MEDIO.

Observamos que el estudio de la variable NUM_PALABRAS con respecto a tipo_mensaje y medio no nos aporta información adicional con respecto a lo visto en el estudio general, si bien nos confirma coherencia en nuestros datos, siendo los comentarios y los títulos de las noticias textos más cortos que la propia noticia.

5.1.5. Proceso de Tokenización

Antes de poder entrenar nuestros modelos, necesitamos transformar nuestros datos para que estos sean legibles para nuestros modelos. Los modelos BERT tienen su propia forma de tokenizar los datos, por lo que explicamos de forma separada este proceso.

5.1.5.1. Tokenización en Deep Learning

Para realizar el proceso de tokenización en los modelos de deep learning, en primer lugar, usamos la clase Tokenizer[25] del módulo Keras. Esta clase permite vectorizar el corpus de texto, convirtiendo cada texto en una secuencia de números enteros. Para ello, primero utilizamos la función *fit_on_texts()* para crear un diccionario de palabra-índice. En este diccionario, cada palabra de nuestro corpus es usada como índice, mientras que los valores son un índice único para cada palabra. Esta tupla será utilizará posteriormente por la función *text_to_sequences()* para convertir cada texto en una secuencia de enteros, donde cada entero corresponde a una única palabra del corpus.

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X)
X_token = tokenizer.texts_to_sequences(X)
print (X [45])
print (X_token[45])
```

```
entonces ser estar forrado hacienda comunista ir rico
[184, 1, 5, 20882, 1778, 1204, 12, 1128]
```

Figura 24: Extracto donde se muestra el uso de la clase Tokenizer del módulo keras

Si exploramos ahora la variable X_token (Figura 24), veremos que tenemos una lista cuyo contenido son números. Estos números son las representaciones de las palabras del texto original. En nuestro ejemplo, el texto compuesto por 8 palabras: “entonces ser estar forrado hacienda comunista ir rico” se ha convertido en una lista con 8 enteros [184, 1, 5, 20882, 1778, 1204, 12, 1228], cada entero correspondiente a una palabra, por ejemplo, el valor 184 corresponde a la palabra “entonces”.

También podemos observar que la longitud de las listas es variable. Esto supone un problema si queremos alimentar una red neuronal, que necesita vectores de longitud fija. Por ello, debemos establecer un valor máximo que llamaremos MAX_LONG que, en nuestro caso, será

[25] https://keras.io/api/keras_nlp/tokenizers/tokenizer/

la longitud del texto más largo de nuestro corpus. Los textos cuya longitud sea inferior a MAX_LONG, se rellenarán al final con ceros. Este proceso se conoce como *padding* y permite generar una lista de textos de longitud fija.

En la Figura 25 determinamos el tamaño del vocabulario y, a continuación, realizamos el proceso de padding.

```
MAX_LONG = df.num_palabras.max()

# Añadimos +1 al tamaño del vocabulario para reservar el índice 0 necesario para el padding.
vocab_size = len(tokenizer.word_index) + 1

#definimos longitud maxima
MAX_LONG = df.num_palabras.max()

#Aplicamos padding a nuestros textos
X_token = pad_sequences(X_token, padding='post', maxlen=MAX_LONG)

#vocab_size = 387186 palabras
print ('MAX_LONG:', MAX_LONG)
print ('Tamaño Vocabulario:',vocab_size)
print(X_token [45])
print('Tamaño vector con padding:',len (X_token [45])) #Longitud es MAX_LONG
```

```
MAX_LONG: 3044
Tamaño Vocabulario: 387186
[184   1   5 ...   0   0   0]
Tamaño vector con padding: 3044
```

Figura 25: Extracto de código donde se calculan MAX_LONG, vocab_size y se aplica padding a los textos

5.1.5.2. Tokenización en BERT

En esta sección, prepararemos nuestro conjunto de datos al formato en el que se puede entrenar BETO. Para poder alimentar el modelo con nuestros textos, hay que dividirlos previamente en tokens y, a continuación, asignar estos tokens a su índice en el vocabulario del tokenizador.

Antes de tokenizar, tenemos que cumplir con los requisitos de formato que nos exige la arquitectura BERT (Devlin et al., 2019).

- Añadir tokens especiales al principio y al final de cada texto. En concreto, debemos anteponer el símbolo especial [CLS] al principio de cada texto y añadir el símbolo especial [SEP] al final del texto.

- Homogeneizar la longitud de los textos mediante la definición de una longitud fija, truncando los textos más largos y aplicando padding a los textos más cortos. El relleno o padding se realiza con un token especial [PAD]. La longitud máxima de los textos que permite la arquitectura BERT es de 512 tokens.

- Diferenciar explícitamente los tokens que aportan valor real de los tokens de relleno con la "máscara de atención" o "attention mask". Esta "máscara de atención" es una matriz de 1s y 0s que indica qué tokens son de relleno y cuáles no. Esta máscara indica al mecanismo de "autoatención" de BERT que no incorpore estos tokens especiales [PAD] a su interpretación del texto.

La tokenización la realizaremos con el tokenizador propio de BERT (*BertTokenizer,* de la librería Transformers). *BertTokenizer* se encarga de asignar a cada token (cada palabra) un índice del vocabulario del tokenizador.

Tras esto, hacemos uso de la función *tokenizer.encode_plus()*, encargada de realizar los siguientes pasos:

1. Divide la frase en tokens.
2. Añade los tokens especiales [CLS] y [SEP].
3. Asigna los tokens a sus ID.
4. Rellena o trunca los textos para dejarlos con la misma longitud (rellena con 1s hasta completar la longitud máxima establecida).
5. Crea las máscaras de atención que diferencian los tokens que aportan valor de los tokens de relleno [PAD].

De esta forma, obtenemos los vectores input_ids (realizado en los primeros 4 pasos de la función *encode_plus*) y attention_mask (paso 5), necesarios para poder entrenar posteriormente nuestro modelo BETO.

5.1.6. Creación de conjunto de datos de entrenamiento y de test

En esta sección explicaremos cómo ha sido el proceso de división de los datos para obtener los conjuntos de entrenamiento y test para cada uno de los datasets.

5.1.6.1. Dataset completo

Para poder realizar nuestras pruebas, primero debemos dividir nuestro dataset en conjuntos de entrenamiento y de test. Fijamos el tamaño del conjunto de test en el 20% de todo el conjunto de datos. Seguidamente imprimimos el número de muestras de cada conjunto, utilizando la función *Counter*(), incluida en el paquete collections (Figura 26).

```
X =  X_token
y= df['label_odio']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=100)
print(f"Distribución de las muestras en conjunto de entrenamiento: {Counter(y_train)}")
print(f"Distribución de las muestras en conjunto de test: {Counter(y_test)}")
```

```
Distribución de las muestras en conjunto de entrenamiento: Counter({0: 447917, 1: 9854})
Distribución de las muestras en conjunto de test: Counter({0: 112024, 1: 2419})
```

Figura 26: Distribución de datos en conjunto de entrenamiento y test

5.1.6.2. Dataset balanceado

Un conjunto de datos equilibrado o balanceado es un conjunto de datos en el que cada clase objetivo está representada aproximadamente por el mismo número de muestras.

Para lograr el equilibrio vamos a aplicar una técnica conocida como *undersampling*, cuyo objetivo es reducir las muestras dominantes (en nuestro caso las etiquetas de *NO_ODIO*), de modo que los ejemplos *ODIO* y *NO_ODIO* queden balanceados en nuestro dataset (Figura 27).

```
from imblearn.under_sampling import RandomUnderSampler

under_sampler = RandomUnderSampler(random_state=42)

X_balanceado, y_balanceado = under_sampler.fit_resample(X.values.reshape(-1, 1), y.values.reshape(-1, 1))

print(f"Nuestro Dataset ya está balanceado: {Counter(y_balanceado)}")
print (f"Tamaño total del dataset: {len(y_balanceado.tolist())}")
```

```
Nuestro Dataset ya está balanceado: Counter({0: 12273, 1: 12273})
Tamaño total del dataset: 24546
```

Figura 27: Uso de la técnica de undersampling para lograr un dataset balanceado

Observamos que ahora el número de palabras de nuestro vocabulario es menor al que teníamos con el dataset completo. Ahora tenemos 57.306 palabras (añadimos +1 para reservar el índice 0 necesario para el padding).

En la Figura 28 mostramos cómo quedan distribuidas las etiquetas tanto para el conjunto de entrenamiento como para el conjunto de test. Se puede apreciar que estos conjuntos están balanceados.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=100)
print(f"Distribución de las muestras en conjunto de entrenamiento: {Counter(y_train)}")
print(f"Distribución de las muestras en conjunto de test: {Counter(y_test)}")

print(f"Tamaño del dataset: {len(y_train)+len(y_test)}")
```

```
Distribución de las muestras en conjunto de entrenamiento: Counter({1: 9851, 0: 9785})
Distribución de las muestras en conjunto de test: Counter({0: 2488, 1: 2422})
Tamaño del dataset: 24546
```

Figura 28: Distribución de datos en conjunto de entrenamiento y test para dataset V1

Para crear los 2 dataset balanceados restantes, uno con textos de longitud homogénea (V2) y otro con textos pertenecientes al medio “El MUNDO” (V3), seguimos un procedimiento análogo.

Para la creación del dataset V2, se ha filtrado previamente por la columna num_palabras para seleccionar textos con una longitud máxima 32 palabras. Como resultado obtenemos un dataset balanceado de tamaño 23.932 (Figura 29), lo que quiere decir que hemos perdido 614 registros sin comparamos con el dataset balanceado de selección aleatoria. El motivo de esta reducción es que 307 textos etiquetados como *ODIO* se han filtrado por superar 32 palabras. Por lo tanto, al crear el dataset balanceado, tenemos 307 muestran menos de cada clase (614 registros).

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=100)
print(f"Distribución de las muestras en conjunto de entrenamiento: {Counter(y_train)}")
print(f"Distribución de las muestras en conjunto de test: {Counter(y_test)}")

print(f"Tamaño del dataset: {len(y_train)+len(y_test)}")
```

```
Distribución de las muestras en conjunto de entrenamiento: Counter({1: 9582, 0: 9563})
Distribución de las muestras en conjunto de test: Counter({0: 2403, 1: 2384})
Tamaño del dataset: 23932
```

Figura 29: Distribución en conjunto de entrenamiento y test para dataset V2

Para la creación del dataset con textos pertenecientes a "El MUNDO", se ha realizado un filtro tanto por num_palabras como por medio. Se ha decidido filtrar por num_palabras porque los resultados para el modelo LSTM son mucho mejores cuando obtenemos un dataset de longitud acotada, como veremos en la siguiente sección **5.2 Entrenamiento y evaluación de los modelos**. Como resultado, obtenemos un dataset balanceado de tamaño 10.268 (Figura 30), lo que indica que hemos perdido 14.278 textos con respecto al dataset balanceado de selección aleatoria (24.546 - 10.268). Esto es debido a que en nuestro dataset original tan solo 5.134 textos de *ODIO* pertenecen al medio "EL MUNDO" y además contienen menos de 32 caracteres.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=100)
print(f"Distribución de las muestras en conjunto de entrenamiento: {Counter(y_train)}")
print(f"Distribución de las muestras en conjunto de test: {Counter(y_test)}")

print(f"Tamaño del dataset: {len(y_train)+len(y_test)}")
```

```
Distribución de las muestras en conjunto de entrenamiento: Counter({0: 4123, 1: 4091})
Distribución de las muestras en conjunto de test: Counter({1: 1043, 0: 1011})
Tamaño del dataset: 10268
```

Figura 30: Distribución en conjunto de entrenamiento y test para dataset V3

5.2. ENTRENAMIENTO Y EVALUACIÓN DE LOS MODELOS

Para cada uno de los modelos seleccionados, realizaremos experimentos con las distintas versiones del dataset descritos en el apartado **4. Cómo detectar odio en los medios de información social**. Todos los experimentos han sido ejecutados desde la versión gratuita colab, configurando el entorno para hacer uso del modo de ejecución GPU (Tesla T4), lo que nos ha permitido ejecutar nuestros modelos hasta 10 veces más rápido que desde el entorno básico.

5.2.1. SNN (Simple Neural Network)

Comenzamos nuestro experimento con una red neuronal simple que usaremos a modo de línea base. Para ello creamos un modelo Sequential() y, a continuación, creamos nuestra capa de embedding. La capa de embedding tendrá una longitud de entrada de MAX_LONG. Para la dimensión del vector y tras probar varias alternativas, el valor seleccionado es 50. El tamaño del vocabulario será de VOCAB_SIZE, calculado en el apartado **5.1.5.1 Tokenización en Deep Learning**. A continuación, como estamos conectando directamente nuestra capa de embedding a una capa fully-connected, aplanamos la capa de embedding. Por último, añadimos una capa densa con función de activación sigmoid, que es la más adecuada para problemas de clasificación binaria. Para compilar nuestro modelo, usaremos el algoritmo de descenso de gradiente eficiente Adam Optimizer, utilizado en Benítez-Andrades et al. (2022). Para nuestra función de pérdida usaremos binary_crossentropy, por tratarse de un problema de clasificación binaria. Finalmente, como métrica queremos medir la Accuracy (Figura 31).

```
model_SNN = Sequential()
embedding_layer = Embedding(vocab_size, 50, input_length=MAX_LONG, trainable=True)
model_SNN.add(embedding_layer)

model_SNN.add(Flatten())
model_SNN.add(Dense(1, activation='sigmoid'))

model_SNN.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
```

Figura 31: Extracto del código python del modelo SNN

5.2.1.1. Dataset Completo

Nuestra primera prueba la realizaremos con el dataset completo. Este dataset, en el que ya hemos aplicado el preprocesamiento visto en el apartado **5.1 Análisis y preparación de los datos**, consta de 572.214 registros y 6 columnas (Figura 6).

Como hay 387.186 palabras en nuestro corpus (valor de VOCAB_SIZE) y cada palabra se representa como un vector de 50 dimensiones, el número de parámetros será de 387.186 x 50 = 19.359.300 en la capa de embedding. En la capa de aplanamiento, simplemente multiplicamos las filas (longitud de cada vector de entrada o MAX_LONG) y las columnas o dimensiones del embedding (3044 x 50 = 152200). Por último, en la capa densa, el número de parámetros es de 152201, 152200 provenientes de la capa de aplanamiento y 1 del parámetro de sesgo (Figura 32).

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_2 (Embedding)      (None, 3044, 50)          19359300

 flatten_2 (Flatten)          (None, 152200)            0

 dense_2 (Dense)              (None, 1)                 152201

=================================================================
Total params: 19,511,501
Trainable params: 19,511,501
Non-trainable params: 0
```

Figura 32: Resumen del modelo SNN compilado para el dataset completo

Utilizamos el método *fit*() de la librería scikit-learn para entrenar nuestra red neuronal, seleccionando diferentes valores de batch_size (25, 50, 100) y epochs (2, 3, 5) para poder comparar rendimientos. Finalmente, indicamos un *validation_split* de 0,1 para que el 10% de los datos de entrenamiento se utilice como datos de validación. Finalmente, evaluamos nuestro modelo con el método *evaluate*().

Observamos que con el dataset completo obtenemos una accuracy del 99% para los datos de entrenamiento y un 98% para los datos de validación. Finalmente, y tras la evaluación del modelo con los datos de test, obtenemos un accuracy del 98%.

Estos resultados podrían parecer buenos, pues una accuracy de 98% para los datos de test son un muy buen valor. Sin embargo, debemos analizar los valores ofrecidos por la matriz de confusión para estar seguros de que nuestro modelo está funcionando correctamente. En la Figura 33 mostramos la matriz de confusión obtenida, donde observamos que todas las predicciones se corresponden con la etiqueta *NO_ODIO,* con 112.024 instancias predichas correctamente y 2.419 instancias etiquetadas de forma incorrecta.

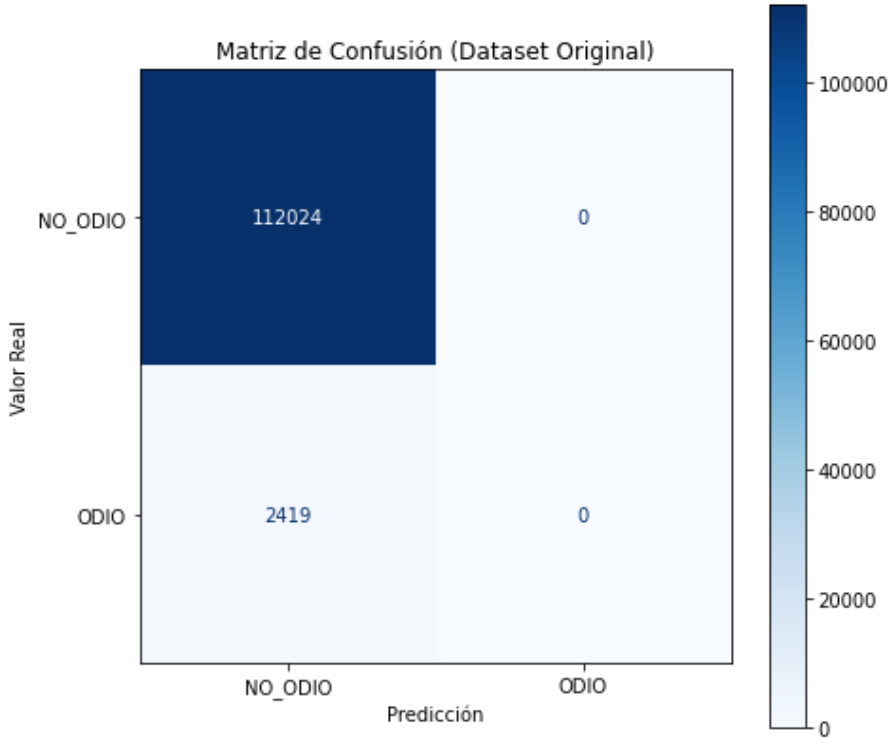

Figura 33: Matriz de confusión con dataset completo

Las pruebas realizadas para el dataset completo muestran unos resultados propios de un dataset desbalanceado (Tabla 2). Al final, al modelo le basta con predecir siempre la clase dominante para conseguir un 98% de accuracy (acertando siempre los textos de *NO_ODIO* conseguimos acertar un 98% de las ocasiones). Sin embargo, esto implica que nunca predice la etiqueta *ODIO*, y esto se traduce en un recall del 0% para esta clase. Por lo tanto, estos modelos entrenados con el dataset completo no son útiles en absoluto para resolver nuestro problema.

Tabla 2: Resultados en test para todos los modelos con dataset completo

	Precision	Recall	F1-score	n_registros
NO_ODIO	98%	100%	99%	112024
ODIO	0%	0%	0%	2419
Accuracy	98%			114443

Estos mismos resultados se han obtenido para los diferentes valores de *batch_size* y *epochs*. Asimismo, los resultados de la matriz de confusión para el dataset completo son exactamente los mismos para todos los modelos utilizados en nuestra comparativa por lo que de aquí en adelante centraremos nuestras pruebas en las versiones de los datasets balanceados.

5.2.1.2. Datasets Balanceados

A continuación, mostramos los resultados obtenidos por el modelo SNN para cada una de las versiones del dataset balanceado (V1, V2 y V3). La columna "Tiempo" se refiere al tiempo empleado en entrenar y validar el modelo.

- **SSN y V1 (Muestras Aleatorias)**

A diferencia de lo que ocurría con el dataset completo, para V1 los resultados muestran unos valores razonables para todas las métricas (Tabla 3). Observamos que, con este dataset balanceado, el modelo es capaz de recuperar un 84% de los textos de *ODIO* (recall) con una accuracy del 86%.

Tabla 3: Resultados en test para SNN con dataset V1

	Precision	Recall	F1-score	n_registros
NO_ODIO	85%	87%	86%	2488
ODIO	87%	84%	85%	2422
Accuracy	86%			4910
Tiempo	13s			

En la Figura 34 mostramos la matriz de confusión obtenida para SNN y V1, donde observamos 2.173 instancias correctamente clasificadas como *NO_ODIO* y 2.041 correctamente clasificadas como *ODIO*.

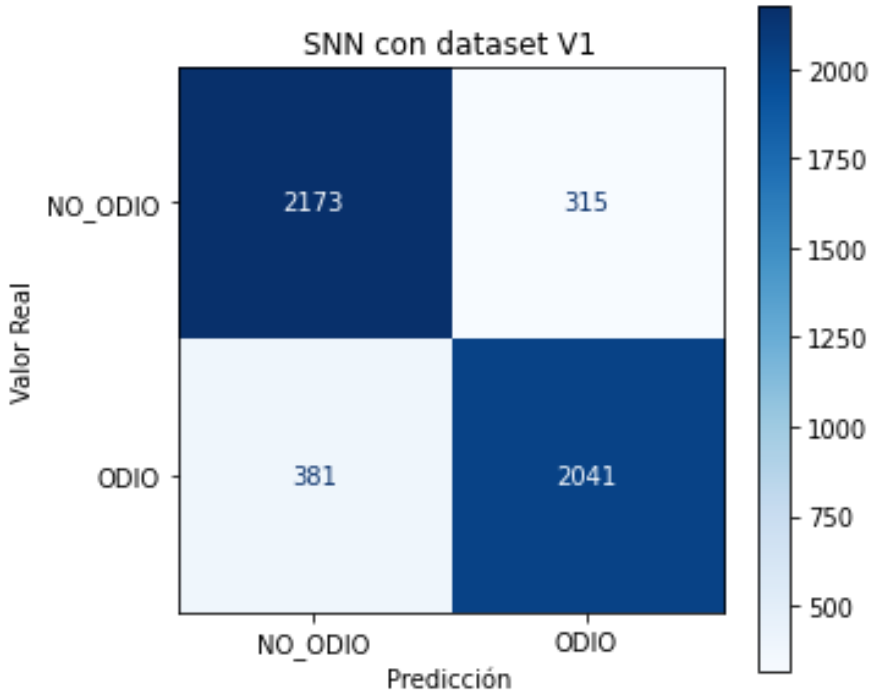

Figura 34: Matriz de confusión obtenida modelo SNN y dataset V1

Estos resultados se han conseguido con un batch_size de tamaño 50 y 2 épocas. Hemos comprobado que más allá de la tercera *epoch* perdemos *accuracy* para el conjunto de test debido al fenómeno *overfitting* o sobreajuste del modelo al conjunto de entrenamiento.

- **SSN y V2 (Longitud homogénea)**

En la Tabla 4 observamos que para el dataset balanceado V2 empeoramos ligeramente todas las métricas con respecto al dataset V1, alcanzando un accuracy del 83%. Esto indica que el hecho de tener unos textos de longitud homogénea no ha ayudado en este sentido al modelo SSN a predecir mejor. Por el contrario, debemos mencionar que el tiempo de ejecución sí que mejoró con respecto a los tiempos obtenidos con el dataset V1, reduciéndose aproximadamente un 30%. De este modo comprobamos que, al acotarse la longitud de los textos, los tiempos de cómputo se reducen notablemente.

Tabla 4: Resultados en test para SNN con dataset V2

	Precision	Recall	F1-score	n_registros
NO_ODIO	80%	87%	83%	2403
ODIO	86%	78%	82%	2384
Accuracy	83%			4787
Tiempo	9s			

A continuación, mostramos la matriz de confusión obtenida para SNN y V2 (Figura 35). Observamos que las instancias predichas correctamente han disminuido ligeramente con respecto al dataset V1, con 2.099 para la etiqueta de *NO_ODIO* y 1.856 para la etiqueta *ODIO*.

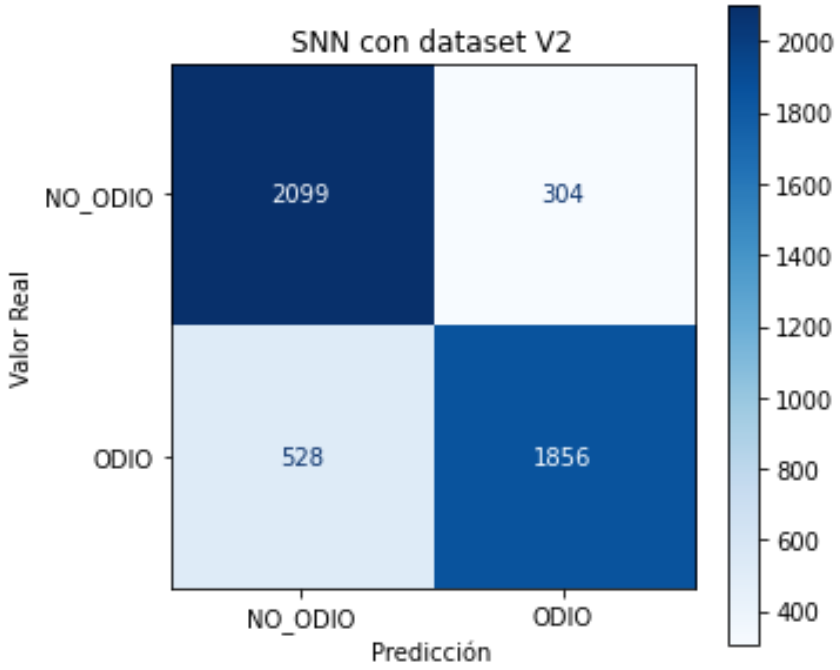

Figura 35: Matriz de confusión obtenida modelo SNN y dataset V2

- **SNN y V3 (Medio "El Mundo")**

Vamos a realizar las mismas pruebas, pero con el dataset V3. En esta ocasión, hemos decidido seleccionar textos únicamente del medio El MUNDO (además de filtrarlos por longitud < 32), para comprobar si seleccionar textos de un mismo medio contribuye a mejorar el rendimiento del modelo.

En la Tabla 5 mostramos los resultados obtenidos:

Tabla 5: Resultados en test para SNN con dataset V3

	Precision	Recall	F1-score	n_registros
NO_ODIO	73%	81%	77%	1011
ODIO	79%	71%	75%	1043

Accuracy	76%	2054
Tiempo	3s	

Observamos que hemos obtenido unas métricas por debajo de lo conseguido con el dataset V2. Estos no son los resultados esperados, pues confiábamos en que seleccionar textos de un mismo medio ayudara al modelo a predecir mejor. Sin embargo, creemos que estos resultados se deben a la reducción del número total de registros de entrenamiento (hemos pasado de 23.932 registros en V2 frente a 10.268 registros en V3). Al tener menos muestras para aprender, el modelo cae en overfitting, sobre ajustándose a los datos de entrenamiento, y no es capaz de generalizar correctamente. De hecho, nuestro modelo alcanza un 99% de accuracy para el conjunto de entrenamiento tras 2 épocas, mientras que se queda en un 76% para el conjunto de test. También apreciamos un menor tiempo de ejecución con respecto a V2, pero este hecho es normal al tener que procesar menos registros.

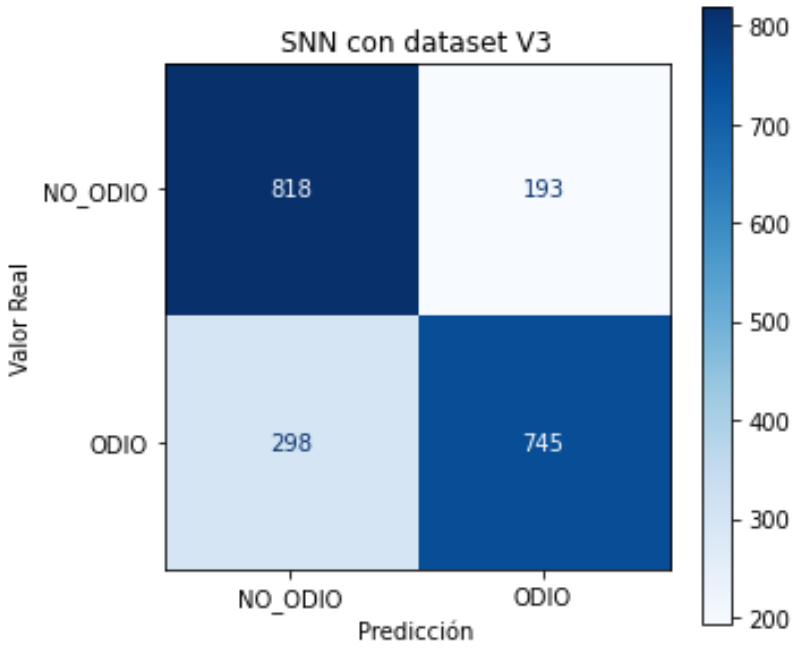

Figura 36: Matriz de confusión obtenida modelo SNN y dataset V3

5.2.2. CNN (Convolutional Neural Network)

A continuación, vamos a realizar las mismas pruebas con la red neuronal convolucional. Como hemos comentado, solo mostraremos los resultados obtenidos con los datasets balanceados, pues tras realizar el experimento con el dataset completo comprobamos que el resultado es exactamente el mismo que con la SNN (solo predice la clase dominante debido al pronunciado desbalanceo de clases).

Con el fin de garantizar que los resultados obtenidos por la red neuronal sean lo más elevados posibles, se han realizado una serie de pruebas en las que se ha comprobado el rendimiento del modelo en función del valor de determinados parámetros que mostramos en la Tabla 6.

Tabla 6: Parámetros seleccionados para CNN

Parámetro	Opciones probadas	Opción seleccionada
Batch size	25, 50, 100	**50**
Epochs	2, 3, 5	**2**
N_Filtros	32, 64, 128	**64**
Tamaño Filtro	3, 4, 5	**3**
Optimizador	Adam, SGD	**Adam**
Learning rate	1e−2, 1e−3	**1e−3**

Nuestra red neuronal convolucional contendrá una capa de embedding seguida de una capa convolucional con función de activación RELU y 1 capa max_pooling para reducir el tamaño de las características, cuya salida irá conectada a una capa densa de 1 neurona con función de activación sigmoid (Figura 37). Para compilar nuestro modelo, usaremos el Adam Optimizer y para nuestra función de pérdida usaremos binary_crossentropy.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_10 (Embedding)     (None, 32, 50)            1220700

 conv1d_8 (Conv1D)            (None, 30, 64)            9664

 global_max_pooling1d_8 (Glo  (None, 64)                0
 balMaxPooling1D)

 dense_14 (Dense)             (None, 1)                 65

=================================================================
Total params: 1,230,429
Trainable params: 1,230,429
Non-trainable params: 0
_________________________________________________________________
```

Figura 37: Resumen del modelo CNN

A continuación, mostramos los resultados obtenidos para cada una de las versiones del dataset balanceado (V1, V2 y V3).

- **CNN y V1 (Muestras Aleatorias)**

Comenzamos con los resultados obtenidos por el modelo CNN con el dataset V1. En la Tabla 7 podemos observar que este modelo alcanza un 87% de accuracy, superando ligeramente al modelo SNN, que obtuvo un 86% con este mismo dataset.

Tabla 7: Resultados en test para CNN con dataset V1

	Precision	Recall	F1-score	n_registros
NO_ODIO	88%	86%	87%	2488
ODIO	86%	88%	87%	2422

Accuracy	87%	4910
Tiempo	18s	

En la Figura 38 mostramos la matriz de confusión obtenida para CNN y V1. Comprobamos que este modelo es capaz de predecir correctamente 2.134 instancias para la etiqueta de *NO_ODIO* y 2.122 instancias para la etiqueta *ODIO*.

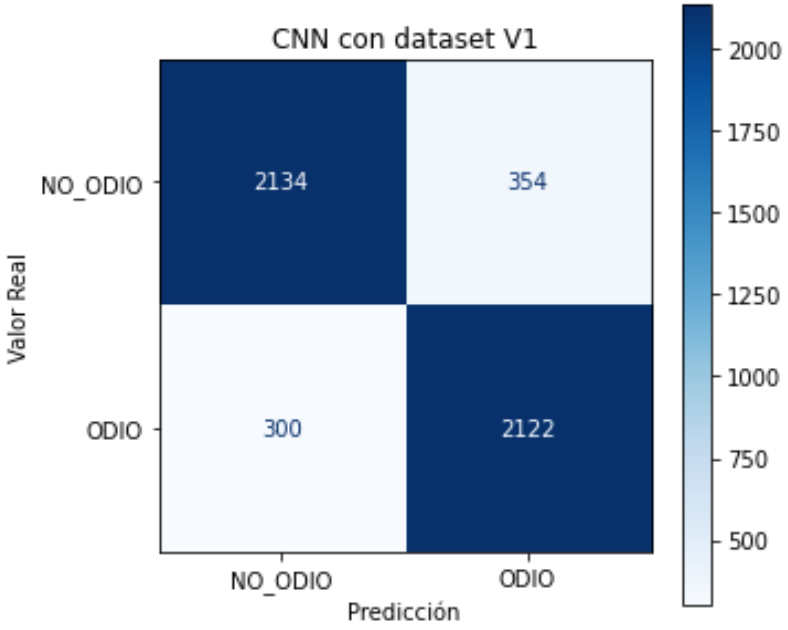

Figura 38: Matriz de confusión obtenida modelo CNN y dataset V1

- **CNN y V2 (Longitud homogénea)**

A continuación, mostramos los resultados para CNN y V2 (Tabla 8), donde alcanzamos un accuracy del 85%, superando ligeramente los resultados obtenidos por SNN con este mismo dataset (83%).

Tabla 8: Resultados en test para CNN con dataset V2

	Precision	Recall	F1-score	n_registros
NO_ODIO	85%	84%	85%	2403
ODIO	84%	85%	85%	2384

Accuracy	85%	4787
Tiempo	13s	

A continuación, mostramos la matriz de confusión obtenida para CNN y V2 (Figura 39). Comprobamos que este modelo es capaz de predecir correctamente 2.026 instancias para la etiqueta de *NO_ODIO* y 2.031 instancias para la etiqueta *ODIO*.

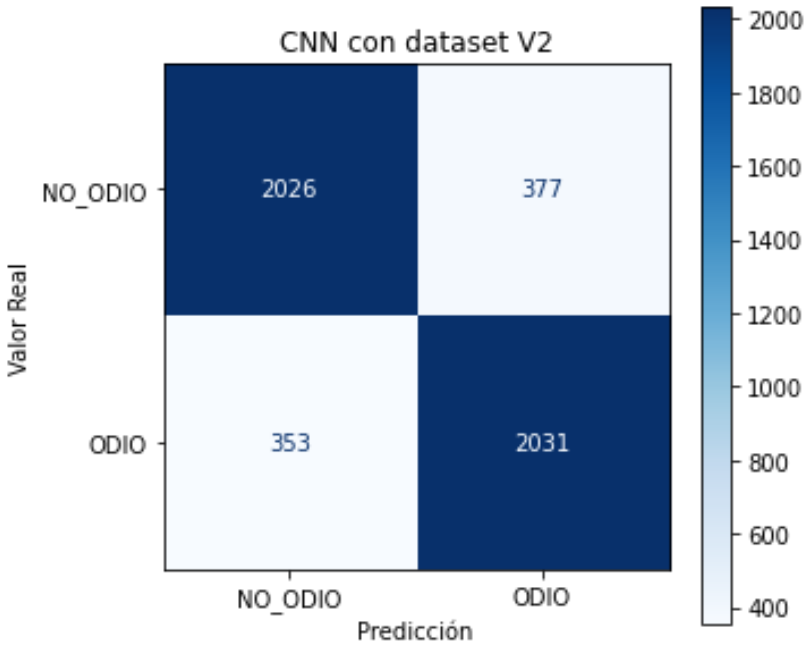

Figura 39: Matriz de confusión obtenida modelo CNN y dataset V2

- **CNN y V3 (Medio "El Mundo")**

Finalmente, mostramos los resultados obtenidos por el modelo CNN con el dataset V3:

Tabla 9: Resultados en test para CNN con dataset V3

	Precision	Recall	F1-score	n_registros
NO_ODIO	79%	87%	83%	1011
ODIO	86%	78%	82%	1043

Accuracy	83%	2054
Tiempo	5s	

En la Figura 40 mostramos la matriz de confusión obtenida para CNN y V3, obteniendo 884 instancias etiquetadas correctamente para la etiqueta de *NO_ODIO* y 813 para la etiqueta *ODIO.*

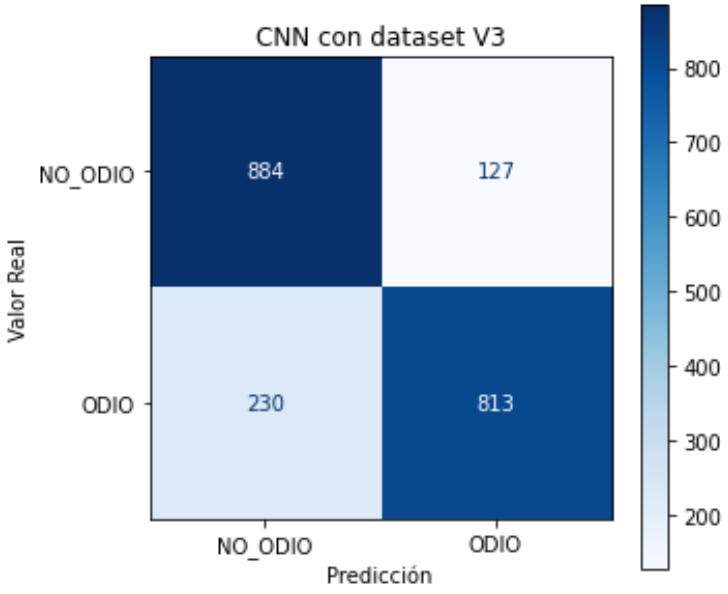

Figura 40: Matriz de confusión obtenida modelo CNN y dataset V3

Como podemos observar, los resultados de CNN son relativamente buenos para todos los datasets, siendo la versión V1 la que alcanza los mejores registros (87% tanto para la métrica accuracy como para F1-score. Por lo tanto, para CNN la reducción de la longitud de los textos no ha supuesto ningún beneficio en términos de accuracy, precisión, recall o F1. Sin embargo, para V2 sí que hemos mejorado en términos de tiempo de ejecución, teniendo en cuenta que ambos datasets cuentan con un número similar de registros.

El rendimiento obtenido con el dataset V3 es el más bajo de todos los datasets balanceados. El motivo de este hecho es el mismo que se explicó con modelo SNN. Al tener menos muestras para aprender, el modelo cae en overfitting, sobre ajustándose a los datos de entrenamiento, y no es capaz de generalizar correctamente.

Como ocurrió en SNN, los mejores resultados para el entrenamiento se han conseguido con 2 épocas. Hemos comprobado que, más allá de la tercera epoch perdemos accuracy para el conjunto de test debido al fenómeno overfitting o sobreajuste del modelo al conjunto de entrenamiento.

5.2.3. LSTM (Short Term Memory)

A continuación, vamos a realizar las mismas pruebas para la red neuronal LSTM.

En la Tabla 10 mostramos la selección de los mejores parámetros para nuestra red. Como podemos observar, en LSTM no tenemos tamaño de filtro como parámetro. Por lo demás, las configuraciones más óptimas son idénticas a las de CNN.

Tabla 10: Parámetros seleccionados para LSTM

Parámetro	Opciones probadas	Opción seleccionada
Batch size	25, 50, 100	**50**
Epochs	2, 3, 5	**2**
N_Filtros	32, 64, 128	**64**
Optimizador	Adam, SGD	**Adam**
Learning rate	1e−2, 1e−3	**1e−3**

Nuestra red LSTM contendrá inicialmente una capa de embedding, tal y como hemos hecho en los casos anteriores. A continuación, creamos una capa LSTM con 64 neuronas conectada a una capa densa de 1 neurona con función de activación sigmoid (Figura 41). Para compilar nuestro modelo, usaremos el Adam Optimizer y para nuestra función de pérdida usaremos binary_crossentropy.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_2 (Embedding)      (None, 32, 50)            744200

 lstm (LSTM)                  (None, 64)                29440

 dense_2 (Dense)              (None, 1)                 65

=================================================================
Total params: 773,705
Trainable params: 773,705
Non-trainable params: 0
```

Figura 41: Resumen del modelo LSTM

A continuación, mostramos los resultados obtenidos para cada una de las versiones del dataset balanceado (V1, V2 y V3).

- **LSTM y V1 (Muestras Aleatorias)**

Dados los resultados mostrados por la matriz de confusión (Figura 42), es evidente que el modelo LSTM no funciona bien para este dataset, pues únicamente predice la clase de *ODIO*. Uno de los posibles motivos de este comportamiento es la longitud máxima de los textos (3.044 caracteres) que hace que los mensajes más cortos estén compuestos mayoritariamente por valores 0 (tras el proceso de padding), perjudicando el rendimiento de nuestro algoritmo basado en la arquitectura RNN.

Tabla 11: Resultados en test para LSTM con dataset V1

	Precision	Recall	F1-score	n_registros
NO_ODIO	0%	0%	0%	2488
ODIO	49%	100%	66%	2422

Accuracy	49%	4910
Tiempo	70s	

Los resultados mostrados en la Tabla 11 confirman este mal comportamiento del modelo LSTM con V1, donde observamos un 0% en las métricas de precisión, recall y F1-score para la clase *NO_ODIO*. Además, podemos extraer que se ha necesitado un tiempo de ejecución aproximadamente 3 veces superior al necesitado para ejecutar la red CNN.

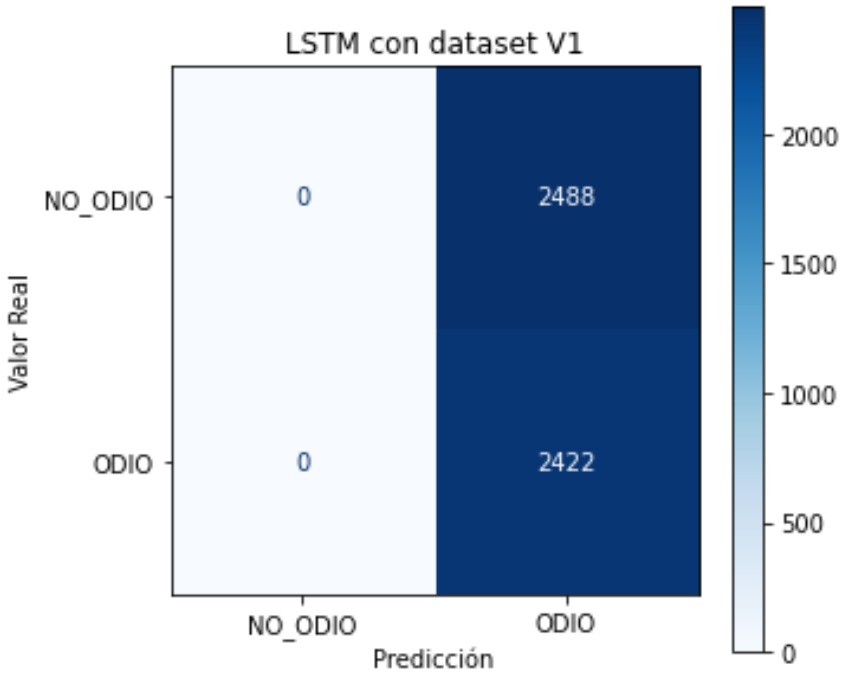

Figura 42: Matriz de confusión obtenida modelo LSTM y dataset V1

- **LSTM y V2 (Longitud homogénea)**

Los resultados obtenidos al entrenar la red LSTM con el dataset V2 son bastante buenos, cercanos incluso a los obtenidos por CNN. Esto confirma que la longitud de los textos (o la falta de homogeneidad de estos) era el motivo por el cual la red recurrente no estaba funcionando correctamente.

Tabla 12: Resultados en test para LSTM con dataset V2

	Precision	Recall	F1-score	n_registros
NO_ODIO	83%	86%	84%	2403
ODIO	85%	82%	83%	2384
Accuracy	84%			4787
Tiempo	29s			

Otro dato interesante que se puede extraer de la Tabla 12 es que LSTM requiere aproximadamente el doble de tiempo de ejecución que CNN.

A continuación, mostramos la matriz de confusión obtenida para el modelo LSTM con el dataset V2.

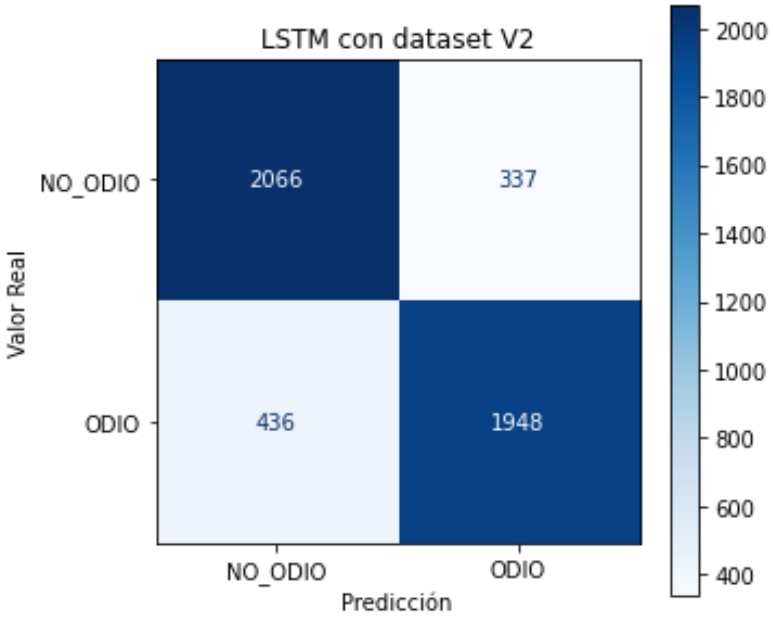

Figura 43: Matriz de confusión obtenida modelo LSTM y dataset V2

- **LSTM y V3 (Medio "El Mundo")**

Al igual que ocurría con los modelos anteriores, los resultados obtenidos para el dataset V3 son ligeramente inferiores a los obtenidos con el dataset V2 (80% de accuracy frente a los 84% obtenidos con V2).

Tabla 13: Resultados en test para LSTM con dataset V3

	Precision	Recall	F1-score	n_registros
NO_ODIO	80%	78%	79%	1011
ODIO	79%	81%	80%	1043

Accuracy	80%	2054
Tiempo	13s	

El motivo que podemos dar es el mismo que el comentado para los casos anteriores. Al disponer de menor cantidad de datos para el entrenamiento, el modelo se ve afectado por el fenómeno de overfitting o sobreajuste, impidiéndole generalizar correctamente.

En la Figura 44 mostramos la matriz de confusión obtenida para el modelo LSTM con el dataset V3.

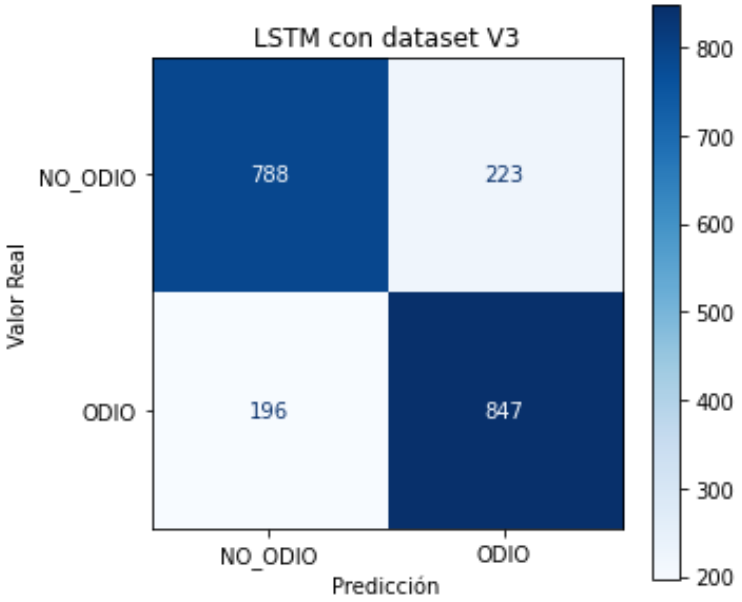

Figura 44: Matriz de confusión obtenida modelo LSTM y dataset balanceado V3

5.2.4. BETO

A continuación, explicaremos el proceso de entrenamiento del modelo BETO y mostraremos los resultados en conjunto de test. Para garantizar que los resultados obtenidos por este modelo de transfer learning sean los mejores posibles, se han realizado una serie de pruebas en las que se ha evaluado el rendimiento del modelo en función del valor que toman distintos parámetros, del mismo modo que se hace en el trabajo de Benítez-Andrades et al. (2022). Podemos ver los valores finales seleccionados para cada uno en la Tabla 14.

Tabla 14: Parámetros seleccionados para BETO

Parámetro	Opciones probadas	Opción seleccionada
Tipo de Modelo	cased, uncased	cased
Batch size	25, 50, 100	50
Epochs	2, 3, 5	2
Optimizador	Adam, SGD	Adam
Learning rate	2e−5, 3e−5, 4e−5	**2e−5**

Una vez que nuestros datos de entrada están formateados correctamente y hemos obtenido los vectores de input_ids y attention_mask descritos en la sección **5.1.5.2 Tokenización en BERT**, ya podemos prepararnos para adaptar el modelo BETO pre-entrenado para nuestra tarea de clasificación.

En primer lugar, crearemos un iterador para nuestro conjunto de datos utilizando la clase DataLoader de *torch.utils.data*. El objeto DataLoader necesita saber nuestro tamaño de nuestro batch para el entrenamiento. En nuestro caso, indicamos un batch_size de 50.
A continuación, creamos los DataLoaders para nuestros conjuntos de entrenamiento y validación (Figura 45)

```
# Creamos los DataLoaders para nuestros conjuntos de entrenamiento y validación.
# Tomaremos muestras de entrenamiento en orden aleatorio.
train_dataloader = DataLoader(
            train_dataset,
            sampler = RandomSampler(train_dataset), # Seleccionamos batches al azar.
            batch_size = batch_size # Indicamos el batch_size
        )

# Para la validación el orden no importa, así que los leeremos secuencialmente.
validation_dataloader = DataLoader(
            val_dataset,
            sampler = SequentialSampler(val_dataset), # Seleccionamos los batches secuencialmente.
            batch_size = batch_size
        )
```

Figura 45: Creación Dataloader para conjunto de entrenamiento y validación

Tanto train_dataloader como validation_dataloader son tuplas que contienen los siguientes elementos que son necesarios para entrenar nuestro modelo:

- input_ids (tensor de tamaño batch_size x max_sequence_length),
- attention_mask (tensor de tamaño batch_size x max_sequence_length)
- labels (tensor de tamaño batch_size x n_labels)

La implementación de Hugging Face pytorch incluye un conjunto de interfaces diseñadas para diversas tareas de PLN[26]. Nosotros utilizaremos el modelo *BertForSequenceClassification* (Figura 46). Se trata de la arquitectura BERT con una capa lineal añadida al final que

[26] https://huggingface.co/transformers/v2.2.0/model_doc/bert.html

utilizaremos como clasificador de nuestros textos. A medida que alimentamos los datos de entrada, todo el modelo BERT pre-entrenado y la capa de clasificación adicional no entrenada se entrenan para nuestra tarea específica.

```
# Cargamos BertForSequenceClassification, el modelo BERT preentrenado con una única
# capa de clasificación lineal añadida al final.

modelo = 'dccuchile/bert-base-spanish-wwm-cased'
model = BertForSequenceClassification.from_pretrained(
    modelo,   #modelo Beto (case)
    num_labels = 2, # 2 etiquetas para clasificación binaria
    output_attentions = False, # Indica si el modelo devuelve attentions-weights.
    output_hidden_states = False, # Indica si el modelo devuelve todas las hidden-states.
)
```

Figura 46: Carga del modelo BETO (cased)

Una vez que tenemos nuestro modelo cargado, establecemos los valores de los hiperparámetros: *Optimizador*, *Learning Rate* y *Epochs* indicados en la Tabla 14. Finalmente, para llevar a cabo el entrenamiento del modelo nos hemos basado en el código del script run_glue.py[27] proporcionado por Hugging Face.

A continuación, mostramos los resultados obtenidos para cada una de las versiones del dataset balanceado (V1, V2 y V3).

- **Muestras Aleatorias (V1)**

Comenzamos mostrando los resultados obtenidos por el modelo BETO con el dataset V1. En la Tabla 15 observamos como BETO es capaz de alcanzar un 89% de accuracy, superando los resultados obtenidos por los modelos anteriores de aprendizaje profundo, donde los resultados fueron de 86% (SNN), 87% (CNN) y 49% (LSTM). Como parte negativa, destacamos un tiempo de entrenamiento de 392 segundos, superando notablemente los tiempos requeridos por los modelos anteriores (13s, 18s y 70s respectivamente).

[27] https://github.com/huggingface/transformers/blob/5bfcd0485ece086ebcbed2d008813037968a9e58/examples/run_glue.py#L128

Tabla 15: Resultados en test para BETO con dataset V1

	Precision	Recall	F1-score	n_registros
NO_ODIO	92%	85%	88%	2483
ODIO	86%	92%	89%	2427

Accuracy	89%	4910
Tiempo	392s	

En la Figura 47 mostramos la matriz de confusión obtenida para BETO y V1. Comprobamos que este modelo de transfer learning es capaz de predecir correctamente 2.112 instancias para la etiqueta de *NO_ODIO* y 2.238 instancias para la etiqueta *ODIO*.

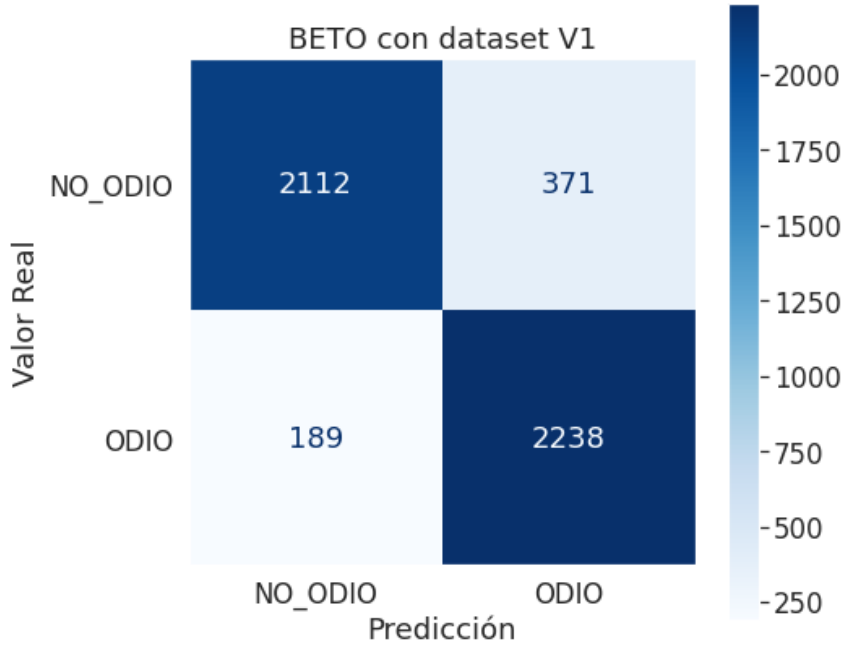

Figura 47: Matriz de confusión obtenida modelo BETO y dataset balanceado V1

- **Longitud homogénea (V2)**

A continuación, mostramos los resultados obtenidos por el modelo BETO y el dataset V2 (Tabla 16), donde alcanzamos un accuracy del 87%.

Tabla 16: Resultados en test para BETO con dataset V2

	Precision	Recall	F1-score	n_registros
NO_ODIO	89%	84%	86%	2360
ODIO	85%	89%	87%	2427
Accuracy	87%			4787

En la Figura 48 mostramos la matriz de confusión obtenida para BETO y V2. En este caso, BETO es capaz de predecir correctamente 1.989 instancias para la etiqueta de *NO_ODIO* y 2.170 instancias para la etiqueta *ODIO*.

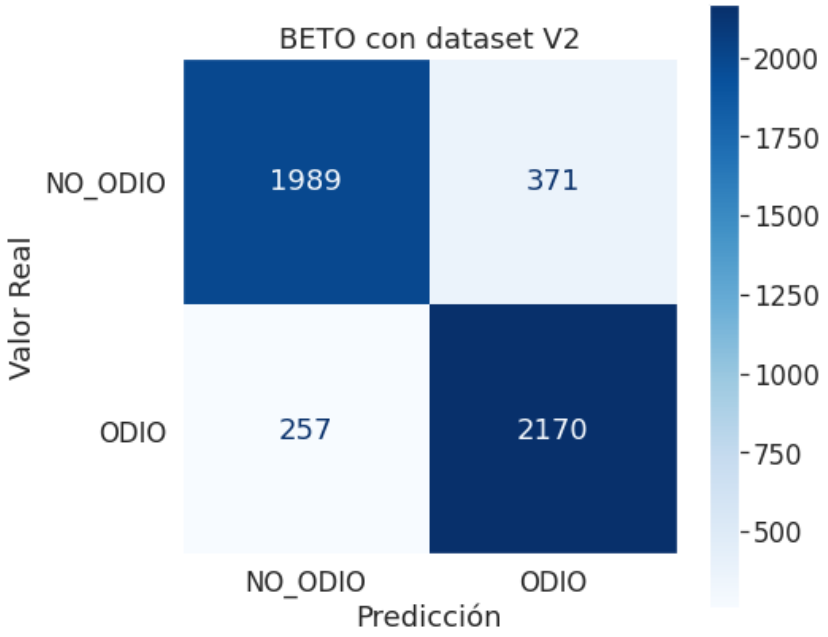

Figura 48: Matriz de confusión obtenida modelo BETO y dataset balanceado V2

- **Medio "El Mundo" (V3)**

En la Tabla 17 podemos observar que el dataset V3 es el que ofrece los peores resultados, en este caso un 84% de accuracy. Tal y como sucedió con el resto de los modelos utilizados en esta comparativa, la reducción del número de registros de entrenamiento ha penalizado la capacidad de generalización de BETO.

Tabla 17: Resultados en test para BETO con dataset V3

	Precision	Recall	F1-score	n_registros
NO_ODIO	87%	81%	84%	1056
ODIO	81%	87%	84%	998
Accuracy	84%			2054

Finalizamos mostrando la matriz de confusión correspondiente a BETO y el dataset V3. En este caso, nuestro modelo ha sido capaz de etiquetar correctamente 856 instancias de la clase *NO_ODIO* y 870 de la clase *ODIO*.

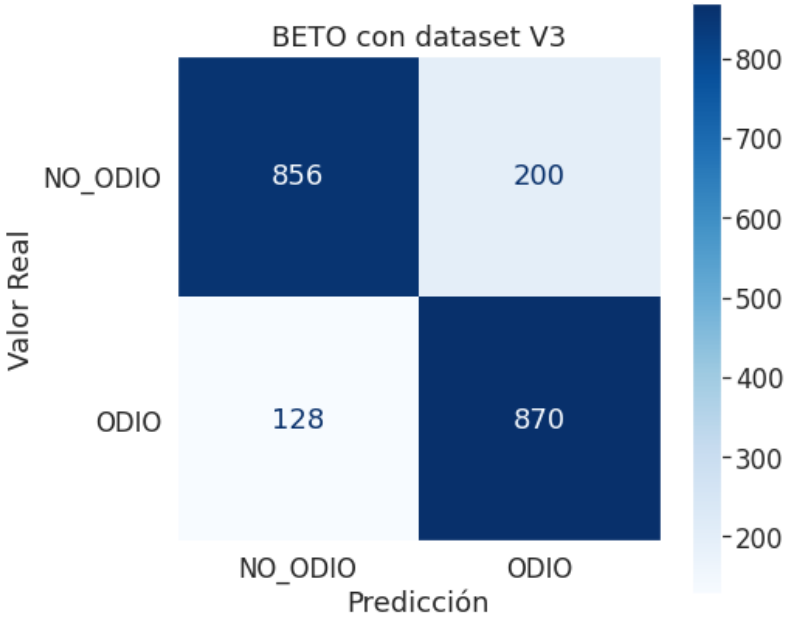

Figura 49: Matriz de confusión obtenida modelo BETO y dataset balanceado V3

6. Discusión y análisis de resultados

En primer lugar, debemos comentar que el dataset original presenta un desbalanceo de clases tan acuciado que resulta inservible, pues todos los modelos entrenados con este dataset terminan prediciendo siempre la clase dominante, consiguiendo un 98% de accuracy, pues devolviendo siempre *NO_ODIO* conseguimos acertar un 98% de las ocasiones. Sin embargo, esto implica que nunca predice la etiqueta *ODIO*, y esto se traduce en un recall del 0% para esta clase. Por lo tanto, estos modelos entrenados con el dataset completo no son útiles para resolver nuestro problema.

Si nos centramos en las pruebas realizadas con los tres datasets balanceados (V1, V2 y V3), observamos que los mejores resultados en términos de accuracy y F1 los encontramos con BETO (cased) entrenado con el dataset V1 (selección aleatoria), seguido por CNN entrenado también con V1 y empatado con BETO entrenado con V2. En la Tabla 18 hemos resaltado los los porcentajes más altos por cada dataset. En este punto, habría que resaltar que CNN requiere mucho menos tiempo que BETO para completar su entrenamiento, 18 segundos frente a 392 segundos respectivamente para el dataset V1.

Tabla 18: Comparativa de resultados en test

		SNN	CNN	LSTM	BETO (cased)
Accuracy	**V1**	86%	87%	49%	**89%**
	V2	83%	85%	84%	**87%**
	V3	76%	83%	80%	**84%**
F1-score (NO_ODIO)	**V1**	86%	87%	0%	**88%**
	V2	83%	85%	84%	**86%**
	V3	77%	83%	79%	**84%**
F1-score (ODIO)	**V1**	85%	87%	66%	**89%**
	V2	82%	85%	83%	**87%**
	V3	75%	82%	80%	**84%**

Estos resultados nos podrían llevar a pensar que un dataset balanceado de selección aleatoria (V1) es la mejor opción para entrenar nuestros modelos de deep learning y transfer learning. Sin embargo, este dataset entraña una serie de desventajas: por ejemplo, el modelo LSTM no

funciona correctamente con este dataset debido a la longitud máxima de los textos (3.044 palabras) que hace que los mensajes más cortos estén compuestos mayoritariamente por valores 0 (padding), perjudicando el rendimiento de nuestro algoritmo basado en la arquitectura de red recurrente. Se ha comprobado que, con textos de longitud homogénea, este modelo alcanza una exactitud cercana a la obtenida por CNN (aunque con tiempos de entrenamiento entre 2 y 3 tres veces superiores en LSTM).

Atendiendo a los tiempos de ejecución de cada dataset, los tiempos con V1 son aproximadamente un 50% mayores con respecto a los tiempos de ejecución con V2, tanto para la red SSN como para CNN. Para el caso de LSTM, esta diferencia de tiempos se acrecienta, siendo el tiempo de ejecución en V1 más del doble que en V2 (70 segundos frente a 29 segundos). Este hecho hace que debamos cuestionarnos si la mejora en las métricas obtenidas en V1 compensan el tiempo y esfuerzo invertido. Con un dataset de entrenamiento como el nuestro (alrededor de 20K registros) esta decisión no es crítica, pues ambos tiempos de ejecución son asumibles. Sin embargo, si se desea trabajar con un dataset de grandes dimensiones, este hecho debe ser tenido en cuenta.

La Tabla 19 muestra los resultados del estado del arte para la detección de discurso de odio en español obtenidos a partir de los conjuntos de datos de HaterNet y HatEval, además de un dataset ad-hoc creado en el trabajo de Amores et al. (2021). A continuación, vamos a comparar estos resultados con nuestra mejor propuesta en términos de macro-F1, el modelo BETO cased entrenado con el dataset balanceado V1.

Para el conjunto de datos HaterNet, el modelo de Plaza-del-Arco et al. (2021) presenta un modelo BETO (cased) que supera a nuestra propuesta en términos de F1 para la etiqueta *NO_ODIO*, alcanzando un 89% frente a nuestro 88%. Sin embargo, en términos de macro-F1, nuestro modelo BETO mejora los resultados de Plaza-del-Arco et al. (2021) en un 14%. El motivo es que nuestro modelo es capaz de predecir mejor la etiqueta de odio, seguramente a consecuencia de haber sido entrenado con un dataset mejor balanceado y con un mayor número de muestras de textos de odio. En la Tabla 20 mostramos la distribución de cada dataset.

En el estudio de Amores et al. (2021) se generaron un total de ocho modelos predictivos: seis usando algoritmos de aprendizaje superficial, uno generado a partir de los votos de esos modelos anteriores y otro usando aprendizaje profundo. Los mejores resultados los obtuvo este último modelo, basado en una red neuronal recurrente GRU, alcanzando un macro-F1 del 77%. Nuestra propuesta mejora este resultado en un 16%.

En cuanto al conjunto de datos HatEval, Plaza-del-Arco et al. (2021) y Pérez et al. (2021) superaron el mejor resultado obtenido en la competición de SemEval-2019 Task 5, probando un modelo BETO (cased) y Robertuito (uncased) respectivamente. El modelo Robertuito es el que alcanza mejor rendimiento en términos de Macro-F1, con un 80%. Nuestro modelo BETO cased entrenado sobre el dataset V1 supera los resultados de Pérez et al. (2021) con una mejora del 11%.

Tabla 19: Resultados del estado del arte para la detección de discurso de odio en español

Modelo	Dataset	F1 (NO_ODIO)	F1 (ODIO)	Macro-F1
SVM (SemEval-2019 Task 5)	**HatEval**	76%	70%	73%
RNN-GRU (Amores et al., 2021)	**Ad-hoc**	-	-	77%
BETO$_{cased}$ (Plaza-del-Arco et al., 2021)	**HaterNet**	**89%**	66%	78%
BETO$_{cased}$ (Plaza-del-Arco et al., 2021)	**HatEval**	80%	76%	78%
Robertuito$_{uncased}$ (Pérez et al., 2021)	**HatEval**	-	-	80%
BETO$_{cased}$ (Nuestra propuesta)	**Hatemedia (V1)**	88%	**89%**	**89%**

Tabla 20: Distribución de los datasets en español

Dataset	n_registros	NO_ODIO	ODIO
HatEval	**6.600**	**3.861** (59%)	**2.739** (41%)
HaterNet	**6.000**	**4.433** (74%)	**1.567** (26%)
Ad-hoc (Amores et al. 2021)	**10.213**	**3879** (38%)	**6334** (62%)
Hatemedia (V1)	**24.546**	**12.273** (50%)	**12.273** (50%)

Comparando estos resultados destacamos la importancia de entrenar los modelos sobre un dataset balanceado con el mayor número posible de registros, con el fin de obtener un modelo que reajuste sus parámetros en base a la información del dataset sin caer en el fenómeno de overfitting, de modo que a posteriori sea capaz de generalizar con la llegada de nuevos datos.

7. Conclusiones y trabajo futuro

7.1.Conclusiones

Con este trabajo pretendíamos comparar el rendimiento de diferentes algoritmos de aprendizaje profundo y transfer learning sobre el dataset creado por el proyecto HATEMEDIA, con el fin de determinar cuál clasifica mejor y concluir si la detección automática de expresiones de odio era viable dado nuestro conjunto de datos.

Para conseguirlo, el primer objetivo planteado que consistía en investigar las técnicas y métodos de aprendizaje automático profundo y transfer learning disponibles que abordan el problema de la detección del discurso del odio, se ha desarrollado en el apartado 2.3, abordando no solo los diferentes modelos, sino también las técnicas de preprocesado y extracción de características más utilizadas en el estado del arte.

En este punto enlazamos con el segundo objetivo, el análisis exploratorio del dataset de HATEMEDIA, desarrollado con detalle en el apartado 5.1. Llevar a cabo este análisis exploratorio reveló que el dataset original sufría de un fuerte desbalanceo de clases que lo hacía inservible. Esto nos llevó al siguiente objetivo: crear diferentes versiones balanceadas de nuestro dataset original para solventar este problema y poder realizar una posterior comparativa.

Los objetivos relativos al entrenamiento y evaluación de los modelos para medir sus rendimientos con los diferentes datasets se han desarrollado en el apartado 5.2, donde hemos realizado un análisis de la parametrización a utilizar, construyendo modelos precisos (especialmente BETO y CNN, con 89% y 87% de accuracy respectivamente) en la detección de odio. Aquí, una dificultad que encontramos fue la falta de homogeneidad de los textos (textos compuestos desde 1 palabra hasta 3.044). Este hecho produjo que el modelo LSTM no funcionara correctamente, si bien este problema fue solventado al construir el dataset de longitud homogénea (V2).

Así, los resultados obtenidos son muy satisfactorios y los modelos generados a partir de los datasets balanceados tienen unas prestaciones más que aceptables, concluyendo que la detección automática de expresiones de odio es viable para estos datasets balanceados, siendo BETO (versión cased) el modelo que mejor resultados ha ofrecido en términos de accuracy y F1-score, seguido de cerca por el modelo CNN, este último además con tiempos de entrenamiento hasta 22 veces más eficientes que BETO.

Dejando a un lado el modelo SNN (que se ha utilizado como línea base), LSTM es el modelo que peor rendimiento ha ofrecido. Tras analizar los resultados obtenidos con del dataset V1, no recomendaríamos LSTM para análisis de textos extensos, al menos cuando la longitud de los textos del dataset no sea homogénea. Por el contrario, BETO y CNN han respondido satisfactoriamente a todas las versiones de los datasets balanceados, por lo que los convierte en los modelos más versátiles.

7.2. Líneas de trabajo futuro

Nuestro dataset original contenía más de 570.000 registros, pero resultó inservible debido al problema del desbalanceo de clases. Como líneas futuras planteamos realizar las mismas pruebas presentadas en este trabajo, pero con un dataset balanceado con mayor cantidad de registros. Nuestros datasets balanceados han permitido generar un conjunto de entrenamiento del orden de 20K registros, lo que ha dificultado la capacidad de los modelos para generalizar, cayendo en el problema de overfitting o sobreajuste tras las primeras épocas de entrenamiento.

Otra alternativa interesante a explorar sería utilizar embeddings pre-entrenados como Word2vec, Glove o fasttext, en lugar de la capa de embedding de keras actual para comprobar si utilizar estos algoritmos suponen una mejora sustancial en términos de accuracy y F1-score.

Por otro lado, hemos identificado textos en nuestro dataset original que, por sí solos, no se podrían considerar textos de odio, pero que han sido etiquetados en función de su contexto (por ejemplo, conversaciones cruzadas entre usuarios de internet). Mostramos tres ejemplos de textos etiquetados como *ODIO* que, como textos aislados, no deberían serlo:

- Ejemplo 1: *"¿la de falconetti acaso?*"
- Ejemplo 2: "*cela.... camilo? y dices que no daba vergüenza ajena?*"
- Ejemplo 3: "*el muerto, culpable, pues.*"

Consideramos que estos ejemplos etiquetados como *ODIO* no hacen más que introducir ruido al modelo, por lo que para mejoras futuras a la hora de crear el dataset, habría que incluir únicamente textos completos de los que se pudiera inferir odio por sí mismos de manera aislada, y que no dependieran del contexto en el que se encuentren.

Bibliografía

Alkomah, F., & Ma, X. (2022). A Literature Review of Textual Hate Speech Detection Methods and Datasets. En *Information (Switzerland)* (Vol. 13, Issue 6). MDPI. https://doi.org/10.3390/info13060273

Amores, J. J., Blanco-Herrero, D., Sánchez-Holgado, P., & Frías-Vázquez, M. (2021). Detecting ideological hatred on Twitter. Development and evaluation of a political ideology hate speech detector in tweets in Spanish. *Cuadernos.Info*, *49*, 98-124. https://doi.org/10.7764/cdi.49.27817

Arango, A., Pérez, J., & Poblete, B. (2019). Hate Speech Detection is Not as Easy as You May Think: A Closer Look at Model Validation. *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 45-54. https://doi.org/10.1145/3331184.3331262

Badjatiya, P., Gupta, S., Gupta, M., & Varma, V. (2017). Deep Learning for Hate Speech Detection in Tweets. *Proceedings of the 26th International Conference on World Wide Web Companion - WWW 17 Companion*. https://doi.org/10.1145/3041021.3054223

Basile, V., Bosco, C., Fersini, E., Nozza, D., Patti, V., Rangel Pardo, F. M., Rosso, P., & Sanguinetti, M. (2019). SemEval-2019 Task 5: Multilingual Detection of Hate Speech Against Immigrants and Women in Twitter. *Proceedings of the 13th International Workshop on Semantic Evaluation*, 54-63. https://doi.org/10.18653/v1/S19-2007

Benítez-Andrades, J. A., González-Jiménez, Á., López-Brea, Á., Aveleira-Mata, J., Alija-Pérez, J. M., & García-Ordás, M. T. (2022). Detecting racism and xenophobia using deep learning models on Twitter data: CNN, LSTM and BERT. *PeerJ Computer Science*, *8*. https://doi.org/10.7717/PEERJ-CS.906

Bohra, A., Vijay, D., Singh, V., Akhtar, S. S., & Shrivastava, M. (2018). *A Dataset of Hindi-English Code-Mixed Social Media Text for Hate Speech Detection*. https://github.com/deepanshu1995/HateSpeech-Hindi-

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, *5*, 135-146. https://doi.org/10.1162/tacl_a_00051

Burnap, P., & Williams, M. (2015). Cyber Hate Speech on Twitter: An Application of Machine Classification and Statistical Modeling for Policy and Decision Making: Machine Classification of Cyber Hate Speech. *Policy & Internet*, *7*. https://doi.org/10.1002/poi3.85

Cañete, J., Chaperon, G., Fuentes, R., Ho, J.-H., Kang, H., & Pérez, J. (2020). *SPANISH PRE-TRAINED BERT MODEL AND EVALUATION DATA*. https://github.com/josecannete/spanish-corpora

Chawla, N. v, Japkowicz, N., & Kotcz, A. (2004). Editorial: Special Issue on Learning from Imbalanced Data Sets. *SIGKDD Explor. Newsl.*, *6*(1), 1-6. https://doi.org/10.1145/1007730.1007733

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. http://arxiv.org/abs/1412.3555

Dash, S., Grover, R., Shekhawat, G., Kaur, S., Mishra, D., & Pal, J. (2021). *Insights Into Incitement: A Computational Perspective on Dangerous Speech on Twitter in India*. http://arxiv.org/abs/2111.03906

Davidson, T., Warmsley, D., Macy, M., & Weber, I. (2017). *Automated Hate Speech Detection and the Problem of Offensive Language*. http://arxiv.org/abs/1703.04009

de la Rosa, J., Ponferrada, E. G., Villegas, P., Salas, P. G. de P., Romero, M., & Grandury, M. (2022). *BERTIN: Efficient Pre-Training of a Spanish Language Model using Perplexity Sampling*. http://arxiv.org/abs/2207.06814

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 4171-4186. https://doi.org/10.18653/v1/N19-1423

Dinakar, K., Reichart, R., & Lieberman, H. (2011, octubre). *Modeling the Detection of Textual Cyberbullying*.

Fersini, E., Rosso, P., & Anzovino, M. (2018). *Overview of the Task on Automatic Misogyny Identification at IberEval 2018*. https://figure-eight.com/

Filippo, M., Fulper, R. S., Ferrara, E. la, Ahn, Y., Flammini, A., Lewis, B., & Rowe, K. K. (2015). *Misogynistic Language on Twitter and Sexual Violence*.

Firth, J. R. (1957). A synopsis of linguistic theory 1930-55. *Studies in Linguistic Analysis (Special Volume of the Philological Society)*, *1952-59*, 1-32.

Frenda, S., Montes, M., Ghanem, B., & Montes-Y-Gómez, M. (2018). *Exploration of Misogyny in Spanish and English tweets Text Mining in Semi-structured data sets View project Hate Speech Detection View project Exploration of Misogyny in Spanish and English tweets*. https://www.researchgate.net/publication/326838153

Gambäck, B., & Sikdar, U. K. (2017). Using Convolutional Neural Networks to Classify Hate-Speech. *Proceedings of the First Workshop on Abusive Language Online*, 85-90. https://doi.org/10.18653/v1/W17-3013

García-Díaz, J. A., Cánovas-García, M., Colomo-Palacios, R., & Valencia-García, R. (2021). Detecting misogyny in Spanish tweets. An approach based on linguistics features and word embeddings. *Future Generation Computer Systems*, *114*, 506-518. https://doi.org/https://doi.org/10.1016/j.future.2020.08.032

García-Díaz, J. A., Jiménez-Zafra, S. M., García-Cumbreras, M. A., & Valencia-García, R. (2022). Evaluating feature combination strategies for hate-speech detection in Spanish using linguistic features and transformers. *Complex and Intelligent Systems*. https://doi.org/10.1007/s40747-022-00693-x

Gröndahl, T., Pajola, L., Juuti, M., Conti, M., & Asokan, N. (2018). *All You Need is «Love»: Evading Hate-speech Detection*. http://arxiv.org/abs/1808.09115

Kumar, R., Reganti, A. N., Bhatia, A., Maheshwari, T., & Rao, B. (2018). *Aggression-annotated Corpus of Hindi-English Code-mixed Data*.

Lample, G., & Conneau, A. (2019). *Cross-lingual Language Model Pretraining*. http://arxiv.org/abs/1901.07291

Lingiardi, V., Carone, N., Semeraro, G., Musto, C., D'Amico, M., & Brena, S. (2020). Mapping Twitter hate speech towards social and sexual minorities: a lexicon-based approach to semantic content analysis. *Behaviour and Information Technology*, *39*(7), 711-721. https://doi.org/10.1080/0144929X.2019.1607903

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. http://arxiv.org/abs/1907.11692

Luhn, H. P. (1957). A Statistical Approach to Mechanized Encoding and Searching of Literary Information. *IBM Journal of Research and Development*, *1*(4), 309-317. https://doi.org/10.1147/rd.14.0309

Mathur, P., Sawhney, R., Ayyar, M., & Ratn Shah, R. (2018). *Did you offend me? Classification of Offensive Tweets in Hinglish Language*. www.github.com/pmathur5k10/

Melnyk, L. (2021). Hate speech targets in COVID-19 related comments on Ukrainian news websites. *Journal of Computer-Assisted Linguistic Research*, *5*(1), 47-75. https://doi.org/10.4995/jclr.2021.15966

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). *Distributed Representations of Words and Phrases and their Compositionality*. arXiv. https://doi.org/10.48550/ARXIV.1310.4546

Müller, K., & Schwarz, C. (2021). Fanning the Flames of Hate: Social Media and Hate Crime. *Journal of the European Economic Association*, *19*(4), 2131-2167. https://doi.org/10.1093/jeea/jvaa045

Nguyen, H.-T., Nguyen, M., & Le. (2017, enero). *An Ensemble Method with Sentiment Features and Clustering Support*.

Nobata, C., Tetreault, J., Thomas, A., Mehdad, Y., & Chang, Y. (2016). Abusive language detection in online user content. *25th International World Wide Web Conference, WWW 2016*, 145-153. https://doi.org/10.1145/2872427.2883062

Papcunová, J., Martončik, M., Fedáková, D., Kentoš, M., Bozogáňová, M., Srba, I., Moro, R., Pikuliak, M., Šimko, M., & Adamkovič, M. (2021). Hate speech operationalization: a preliminary examination of hate speech indicators and their structure. *Complex and Intelligent Systems*. https://doi.org/10.1007/s40747-021-00561-0

Pennington, J., Socher, R., & Manning, C. (2014). GloVe: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1532-1543. https://doi.org/10.3115/v1/D14-1162

Pereira-Kohatsu, J. C., Quijano-Sánchez, L., Liberatore, F., & Camacho-Collados, M. (2019). Detecting and Monitoring Hate Speech in Twitter. *Sensors*, *19*(21). https://doi.org/10.3390/s19214654

Pérez, J. M., Furman, D. A., Alemany, L. A., & Luque, F. (2021). *RoBERTuito: a pre-trained language model for social media text in Spanish*. http://arxiv.org/abs/2111.09453

Pires, T., Schlinger, E., & Garrette, D. (2019). How Multilingual is Multilingual BERT? *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 4996-5001. https://doi.org/10.18653/v1/P19-1493

Plaza-del-Arco, F. M., Molina-González, M. D., Ureña-López, L. A., & Martín-Valdivia, M. T. (2021). Comparing pre-trained language models for Spanish hate speech detection. *Expert Systems with Applications*, *166*. https://doi.org/10.1016/j.eswa.2020.114120

Ramachandran, P., Zoph, B., & Le, Q. v. (2017). *Searching for Activation Functions*. http://arxiv.org/abs/1710.05941

Roy, P. K., Tripathy, A. K., Das, T. K., & Gao, X. Z. (2020). A framework for hate speech detection using deep convolutional neural network. *IEEE Access*, *8*, 204951-204962. https://doi.org/10.1109/ACCESS.2020.3037073

Sachdeva, J., Chaudhary, K. K., Madaan, H., & Meel, P. (2021). Text Based Hate-Speech Analysis. *Proceedings - International Conference on Artificial Intelligence and Smart Systems, ICAIS 2021*, 661-668. https://doi.org/10.1109/ICAIS50930.2021.9396013

Urdaneta, L. A. (2019, abril). *Reducir el número de palabras de un texto: lematización y radicalización (stemming) con Python*. https://medium.com/qu4nt/reducir-el-n%C3%BAmero-de-palabras-de-un-texto-lematizaci%C3%B3n-y-radicalizaci%C3%B3n-stemming-con-python-965bfd0c69fa

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. ukasz, & Polosukhin, I. (2017). Attention is All you Need. En I. Guyon, U. von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems* (Vol. 30). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Vigna, F. del, Cimino, A., Dell'orletta, F., Petrocchi, M., & Tesconi, M. (2017). *Hate me, hate me not: Hate speech detection on Facebook*. https://curl.haxx.se

Wang, C. (2018). *Interpreting Neural Network Hate Speech Classifiers*.

Waseem, Z., & Hovy, D. (2016). Hateful Symbols or Hateful People? Predictive Features for Hate Speech Detection on Twitter. *NAACL*.

Anexo. Artículo de investigación

Comparativa de modelos de aprendizaje profundo para la detección de odio en castellano en medios de información social

Carlos Simón Gallego

Universidad Internacional de la Rioja, Logroño (España)

8 de febrero de 2023

Resumen

Con este trabajo tratamos de determinar la viabilidad que existe en la detección automática de expresiones de odio en castellano mediante la aplicación de Deep Learning (DL) sobre el dataset del proyecto Hatemedia. Para ello realizamos una comparativa de soluciones para determinar qué modelo de DL ofrece mejor rendimiento para esta tarea. Se han realizado las mismas pruebas con diferentes versiones del dataset; una versión con todos los registros y otras versiones reducidas y balanceadas. Tras el trabajo comparativo, encontramos que el dataset original resulta inservible debido al problema de desbalanceo de clases, que hace que todos los modelos acaben prediciendo únicamente la clase dominante. Para los datasets balanceados, el modelo BETO (versión cased) es el que mejor rendimiento ofrece, superando los resultados obtenidos por otros modelos del estado del arte entrenados con diferentes datasets. Finalizamos exponiendo todas las dificultades encontradas y ofreciendo alternativas de mejora para trabajos futuros.

Palabras Clave

Discurso de odio, Aprendizaje profundo, BETO, Procesamiento de lenguaje natural, Clasificación de texto

I. Introducción

Hoy en día, el auge de las redes sociales y medios informativos online genera una enorme cantidad de información y proliferación de contenidos (desinformativos o no) que en muchas ocasiones ponen en entredicho la tolerancia, civismo y respeto a determinados colectivos.

Dada esta enorme cantidad de contenidos generados, no es factible confiar únicamente en la supervisión humana para combatir el discurso de odio en internet. Sin embargo, nos encontramos ante el problema de que no existe una definición única para el discurso de odio, lo que complica en gran medida la labor de crear datasets etiquetados y algoritmos que detecten el odio automáticamente y con precisión en un texto.

Este estudio pretende contribuir a la detección automática del discurso de odio en español. Para ello, hacemos uso del corpus etiquetado por el equipo del proyecto Hatemedia[1] y comparamos varias técnicas de clasificación basadas en modelos de aprendizaje profundo.

En el apartado II se hará un análisis del contexto y el estado del arte, donde repasaremos los talleres y eventos más relevantes de los últimos años enfocados a tratar el problema de la detección de expresiones de odio en textos, así como los datasets y sistemas basados en inteligencia artificial más conocidos que se utilizan para intentar abordar este complejo problema. Los objetivos generales y específicos son descritos en el apartado III, donde también se detallarán los pasos necesarios para la consecución de estos. En el apartado IV se describe el procedimiento que se va a seguir para llevar a cabo la comparativa. Esto comprende desde la descripción de las versiones del dataset que se van a utilizar, hasta los modelos seleccionados y las métricas de evaluación utilizadas. En el apartado V pasaremos a describir con todo detalle el desarrollo del trabajo, mostrando los resultados y mediciones obtenidos, para continuar en el apartado VI con una discusión sobre la relevancia de los resultados, identificando los datos más importantes extraídos de estos resultados. Finalmente, en el apartado VII se darán las conclusiones extraídas del trabajo y se propondrán líneas futuras de investigación o desarrollo relacionado con el mismo

II. Estado del Arte

El estudio de la detección y clasificación automática del discurso de odio mediante procesamiento de lenguaje natural (PLN) es un campo relativamente reciente, pero el interés en esta área ha aumentado a medida que las redes sociales y otras plataformas de internet han crecido en términos de influencia y adopción por parte de la gran mayoría de los usuarios [1].

En la presente sección haremos una revisión del estado del arte, donde comenzaremos destacando los principales eventos

[1] https://www.hatemedia.es/

y talleres a nivel mundial enfocados en la detección del discurso del odio. A continuación, listaremos algunos de los dataset más utilizados para dichas tareas. Finalmente, analizamos las diferentes técnicas de PLN utilizadas para extraer información de un texto, así como los modelos de aprendizaje automático empleados en el estado del arte, desde los modelos de machine learning (ML) clásicos hasta soluciones más modernas basadas en aprendizaje profundo y Transformers.

2.1 Congresos relativos a la detección de odio en textos

El impacto de las publicaciones nocivas online ha dado lugar a un gran número de estudios y eventos enfocados a la detección del odio y lenguaje ofensivo. Como ejemplo, se listan los siguientes talleres y congresos.

- SemEval[2]: Taller internacional sobre el procesamiento del lenguaje natural cuya misión es avanzar en el estado actual del arte. Cada año, este taller propone una serie de tareas compartidas en las que se presentan y comparan sistemas de análisis semántico computacional diseñados por diferentes equipos

- Workshop on Online Abuse and Harms (WOAH[3]), que en el año 2022 celebró su sexta edición, cuyo objetivo es avanzar en la investigación para detectar, clasificar y modelar el contenido ofensivo y dañino en internet.

- GermEval Shared Task[4] (edición de 2018 y 2019), centrado en el procesamiento del lenguaje natural para detección de lenguaje ofensivo en el idioma alemán.
- PolEval[5] (edición de 2019, tarea 6), sobre la detección automática del ciberacoso en Twitter para el lenguaje polaco.

- HASOC[6] (2019), sobre identificación de expresiones de odio y contenidos ofensivos en las lenguas indoeuropeas.

- AMI[7] (2018), taller para la identificación automática de la misoginia, para el idioma italiano y el inglés.

En relación a los estudios sobre el discurso del odio en idioma español, observamos que no encontramos tanta variedad como los centrados en el idioma inglés. De hecho, los estudios que existen están relacionados mayoritariamente con la participación de IberEval 2018 - Automatic Misogyny Identification y la Tarea 5 del taller SemEval 2019 [2].

- *SemEval-2019, Tarea 5*

Esta tarea tuvo como objetivo detectar contenidos de odio en los textos de las redes sociales en español, concretamente en las publicaciones de Twitter, contra dos objetivos: los inmigrantes y las mujeres. Además, la tarea implementaba una perspectiva multilingüe en la que se proporcionaron datos de los idiomas inglés y español (HatEval), para entrenar y probar los sistemas participantes. El conjunto de datos de HatEval estaba compuesto por 19.600 tuits, 13.000 en inglés y 6.600 en español [3]. Esta tarea se articulaba en torno a dos subtareas relacionadas:

- Subtarea A: Consistía en una detección básica de discurso de odio, en la que se pedía a los participantes que marcaran la presencia de odio en los tweets (clasificación binaria).

- Subtarea B: En esta segunda subtarea se trataba de determinar si el objetivo del mensaje era un individuo un grupo de personas, y si el contenido del mensaje contenía lenguaje agresivo.

- *IberEval 2018 (AMI)*

Este taller estaba enfocado a la detección de tweets misóginos mediante PLN, con un dataset multilingüe, con 4.138 tuits escritos en español y 3.977 en inglés [4]. Del mismo modo que en el caso de SemEval 2019 task 5, IberEval 2018 estaba organizado en dos subtareas:

- Subtarea A: Consistía en una tarea de identificación binaria de mensajes misóginos.

- Subtarea B: En esta segunda subtarea había que determinar cuándo el objetivo del comentario misógino era un individuo concreto o un grupo.

2.2 Datasets

En este apartado listamos algunos de los dataset más utilizados en el estado del arte para tareas de detección de discurso de odio en inglés.

- Waseem and Hovy: Este conjunto de datos está compuesto por 16.000 tweets anotados como "sexistas", "racistas" y "sin odio" [5].

- Davidson et al.: Compuesto por 24.802 tuits anotados en tres clases: discurso de odio, ofensivo (pero no de odio), y ni ofensivo ni de odio [6].

- HatEval: Este conjunto de datos se compone de 19.600 tweets, 13.000 en inglés y 6.600 en español [3].

- HS: 4.575 tweets en hindi y en inglés etiquetados como discurso de odio (aquellos tuits que inducen al odio) y discurso normal (tuits que no inducen ninguna forma de odio) [7].

A continuación, se listan algunos de los datasets más importantes en idioma español:

- HaterNet: Dataset en idioma español construido a partir de Twitter, compuesto por 6.000 textos etiquetados, con 1.567 tweets anotados como odio y 4.433 anotados como no odio [8].

[2] https://semeval.github.io/
[3] https://www.workshopononlineabuse.com/
[4] https://germeval.github.io/tasks/
[5] http://2019.poleval.pl/
[6] https://hasocfire.github.io/hasoc/2019/
[7] https://amievalita2018.wordpress.com/

- HatEval 2019: Dataset construido a partir de Twitter compuesto por 6.600 textos en español, con 2.739 anotados como odio y 3.861 etiquetados como no odio [3].

- IberEval 2018 – AMI: Dataset en español compuesto por 4.138 tweets, 2.064 anotados como mensajes misóginos y 2.074 como no misóginos [4].

2.3 Técnicas y modelos

El procedimiento que se suele seguir para realizar el análisis de un texto, ya sea con el objetivo de detectar odio o para cualquier otro, consta de tres pasos: 1. Preprocesado de texto, 2. Extracción de características, 3. Clasificación mediante modelos IA.

2.3.1. Técnicas de preprocesado

Como es natural, el texto que nos llega en bruto puede presentar un formato que diste mucho de lo que podríamos considerar el formato correcto, compuesto por palabras incompletas, mal escritas o en otros idiomas, conteniendo espacios innecesarios, etc. Además, en nuestro texto origen existirán, casi con total seguridad, infinidad de palabras innecesarias que no nos aporten ningún valor.

Así pues, en primer lugar y antes de extraer características del texto y construir modelos a partir de esta información, debemos dedicar tiempo a las tareas de limpieza, formateo y preparación de los datos. Estas tareas están presentes en el día a día de todos los proyectos de IA en general, y de procesamiento de lenguaje natural en particular [9].

Algunas de las técnicas de preprocesado más habituales son las siguientes:

- Tokenización, que consiste en segmentar el texto en unidades más pequeñas (tokens o n-gramas) que podamos manejar como referencia para extraer características que aporten valor a nuestro sistema. Además, eliminaremos todos aquellos tokens que no nos aporten valor, de modo que reduzcamos el número de elementos a tratar.

- Normalización: Será una tarea importante si queremos que nuestras palabras sigan un formato estándar. Del paso anterior, nuestro tokenizador ha podido reconocer la misma palabra, pero escrita en mayúsculas y en minúsculas, por ejemplo. Si queremos tener solo una versión, será imprescindible normalizar nuestro texto.

- POS (part-of-speech) tagging: El POS es la técnica sintáctica para etiquetar a cada una de las palabras de un texto su categoría gramatical.

- Lematización: La técnica de lematización lo que consigue es reducir todas las palabras derivadas a su lema, que es la forma en la que encuentras la palabra en el diccionario.

- NER (Named Entity Recognition): La detección de entidades permite identificar automáticamente determinadas palabras de un texto y clasificarlas en diferentes categorías.

2.3.2. Técnicas de extracción de características

Las técnicas más simples de extracción de características, (también conocidas como técnicas superficiales), son la bolsa de palabras (BoW, de sus siglas en inglés) y la técnica TF-IDF(del inglés Term frequency – Inverse document frequency) [10]. BoW es una representación vectorial compuesta por un diccionario (lexicones) con las palabras de los textos con los que se quieren entrenar los modelos. En estos lexicones se representa la relevancia de cada elemento mediante métricas como, por ejemplo, si la palabra aparece en el texto (booleano), o la cantidad de veces que una palabra se repite en el texto. TF-IDF es una técnica cuyo objetivo es encontrar el documento más relevante para cierto término dentro de una colección de documentos. Para ello, mide con qué frecuencia aparece un término o frase dentro de un documento determinado, y lo compara con el número de documentos que mencionan ese mismo término dentro de una colección entera de documentos.

Una técnica más compleja son los Word Embeddings [11] [12], utilizadas para representar las palabras de nuestro lexicón mediante vectores multidimensionales, capaces de capturar incluso relaciones semánticas entre palabras. Esta técnica está presente en muchos de los estudios del estado del arte para detección de odio, como [13] y [14]. Las representaciones de Word Embeddings pueden generarse a partir de representaciones pre-entrenadas como Word2vec [12], Glove [15]) y fastText [16].

2.3.3. Machine Learning clásico

Entre las diversas técnicas convencionales de aprendizaje automático utilizadas en la tarea de la detección del discurso del odio en Internet, destacan las máquinas de vectores soporte (SVM), la regresión logística y los Random Forest [17] [6] [18]; [5]).

[19] muestra que estos tres modelos son los que proporcionan mejor rendimiento dentro del ML convencional en términos de Accuracy, Precision, Recall y F1. Por otro lado, en este estudio se concluye que el modelo K-Vecinos Más Cercanos (KNN, de sus siglas en inglés), obtuvo el peor rendimiento para la tarea de clasificación de textos.

El taller SemEval 2019, tarea 5 (que consistía en detectar discurso de odio en Twitter contra mujeres e inmigrantes), muestra que el modelo SVM es especialmente relevante, ya que los sistemas creados mediante este modelo obtuvieron los mejores resultados de la competición [3].

2.3.3.1. Deep Learning

Dentro de las técnicas de DL más utilizadas en la clasificación de textos, destacan las redes neuronales convolucionales (CNN) y las redes neuronales recurrentes (RNN) [2].

[20] y [21] fueron los primeros en utilizar redes neuronales recurrentes y redes neuronales de convolución, respectivamente, para la detección del discurso del odio en los tuits.

CNN

Las CNN son un tipo de red neuronal que procesa capas de forma jerárquica, lo que les permite diferenciar distintas características en las entradas recibidas. La capa más importante, y la que da nombre a la red, es la capa convolucional. Esta capa funciona a partir de unos filtros que van desplazándose por la imagen o el texto, dependiendo el problema a resolver, obteniendo las salidas de la capa mediante un producto escalar.

Aunque se diseñaron inicialmente para la visión por computador, han sido eficaces también para tareas de PLN y de detección de odio [22]. En la Figura 1 podemos observar la arquitectura de una red neuronal convolucional aplicada al problema de análisis de sentimiento de textos.

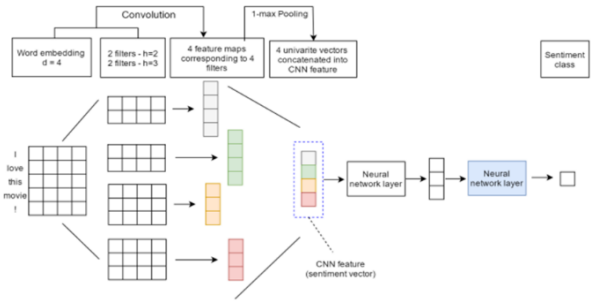

Figura. 1. Arquitectura de una CNN extraída de [38]

Cuando utilizamos una red CNN aplicada a PLN, lo que procesamos son textos en lugar de imágenes. Estos textos tendrán una representación matricial, donde las filas representan la las palabras codificadas mediante word embeddings con una dimensión d (espacio vectorial donde hemos embebido los textos). Por tanto, cada filtro de convolución tendrá una anchura igual a la longitud del embedding donde están incrustados los textos a procesar, en nuestro ejemplo d=4, de modo que cada filtro irá recorriendo las palabras en una sola dirección, de arriba abajo, en lugar de izquierda a derecha y de arriba abajo como sucede con las imágenes.

En nuestro ejemplo observamos que tenemos 4 filtros, dos de altura h=2 y otros dos de altura h=3. Esto significa que queremos detectar características locales en grupos formados por dos y tres palabras, capturando diferentes niveles de correlación entre palabras. Así pues, cada filtro se encargará de capturar cierta característica de los datos. Como estamos aplicando capas de convolución que son unidimensionales (recorremos la matriz de entrada de arriba a abajo), en lugar de las bidimensionales utilizadas en imágenes, la salida que obtenemos tras aplicar nuestro filtro es un vector en lugar de una matriz. Estos vectores serán nuestros mapas de características.

En la fase de max-Pooling solo nos quedamos con un elemento, el resultado más grande de cada uno de los mapas de características, para reducir la dimensionalidad.

Finalmente, concatenamos los valores máximos obtenidos en la fase de max-Pooling para conformar la entrada de la siguiente capa, una fully connected layer. En nuestro ejemplo, tenemos dos capas densas como últimas capas. La última capa estará compuesta por una sola neurona para clasificación binaria.

RNN y LSTM

Las redes neuronales recurrentes (RNN) son una clase de redes especializadas en analizar datos de series temporales. La principal característica de este tipo de redes radica en su capacidad de modelar relaciones temporales entre elementos de la secuencia a través de un estado interno de la red o hidden state, que podemos considerar como una memoria sobre lo que la red ha visto hasta ese momento. En esta arquitectura se aplica una fórmula recurrente sobre una secuencia de entrada de manera que, en cada paso dado, se depende del nuevo valor de entrada x y del estado interno h anterior. Por tanto, este tipo de arquitecturas permiten modelar relaciones entre palabras dentro de un texto.

Las LSTM (Long Short Term Memory) son un tipo especial de redes recurrentes [23]. Estas redes surgieron como una arquitectura encaminada a solucionar los problemas de memoria de las RNN tradicionales. En la práctica, estas últimas presentan problemas para aprender relaciones con elementos de time step lejanos (es decir, que no están cerca del time step actual). Esto limita en gran parte el potencial teórico de las RNN. Por ejemplo, dentro del campo del procesamiento de lenguaje natural, cuando analizamos un texto es importante mantener la información aprendida desde el inicio hasta el final del mismo, de modo que podamos extraer características y relaciones entre palabras dentro de un mismo texto. Las LSTM están diseñadas para intentar solucionar este problema. En LSTM se establecen unos criterios para almacenar la información obtenida hasta el momento. El modelo aprende qué partes de la representación se deben olvidar para incluir las más importantes.

Existe una versión alternativa llamada Bi-LSTM (Bidirectional Long Short-Term Memory). Se trata de una arquitectura idéntica a la LSTM, solo que en este caso la red neuronal se entrenará con los mismos datos una segunda vez, recorriéndolos en orden inverso. Si bien las LSTM/BiLSTM suponen una mejora respecto a las RNN clásicas, ambos modelos comparten una arquitectura secuencial que limita en gran medida la paralelización de las ejecuciones y, por tanto, el rendimiento LSTM general. Por último, la arquitectura GRU (Gated Recurrent Unit), es una versión simplificada de LSTM introducida en 2014 por [24].

Transfer learning

Utilizando como punto de partida modelos pre-entrenados, el Transfer Learning permite desarrollar rápidamente modelos eficaces y resolver problemas complejos de PLN o de visión por computador sin necesidad de tener que entrenar nuestro propio modelo de cero o de disponer de una inmensa cantidad de datos. De este modo, los modelos pre-entrenados se han convertido en un elemento básico en el ámbito del procesamiento del lenguaje natural.

En los últimos años, desde la introducción de la arquitectura Transformer, se han utilizado en muchas otras tareas diferentes de PLN, superando a modelos anteriores basados en redes neuronales recurrentes [25]. Los modelos Transformer tienen como principal innovación la sustitución de las capas recurrentes, como las LSTMs que se venían usando hasta ese momento en PLN, por las denominadas capas de atención [26].

A nivel de arquitectura, los Transformers se basan en dos partes bien diferenciadas, un codificador o encoder y un decodificador o decoder. El encoder está compuesto por una pila de N = 6 capas idénticas. Cada capa tiene dos subcapas. La primera es un mecanismo de autoatención (multi-head attention), y la segunda es una red simple totalmente conectada. Por otro lado, el descodificador también se compone de una pila de N = 6 capas idénticas. Además de las dos subcapas de cada capa del codificador, el descodificador inserta una tercera subcapa multi-head attention, que se aplica sobre la salida de la pila del codificador. Tanto el codificador como el decodificador trabajan sobre secuencias enteras de texto en lugar de palabra por palabra. De este modo, en lugar de analizar palabras sueltas, se obtiene un análisis global.

La mejora de rendimiento ofrecida por la arquitectura Transformer ha permitido el rápido desarrollo de modelos sobre conjuntos de datos tan grandes que anteriormente no era viable procesar, dando lugar al modelo BERT (Bidirectional Encoder

Representations from Transformers) y a los GPT (Generative Pre-trained Transformer), estos últimos utilizados principalmente para generar textos que simulan la redacción humana

Modelos BERT y RoBERTa

BERT es un modelo Transformer bidireccional, pre-entrenado sobre una gran cantidad de datos sin etiquetar para aprender una representación del lenguaje que se puede utilizar para realizar fine-tuning y adaptarlo a tareas específicas de aprendizaje automático [27] [25]. RoBERTa (A Robustly Optimized BERT Pretraining Approach) es otro modelo basado en la arquitectura BERT [28]. RoBERTa utiliza la misma arquitectura de BERT, pero aplicando pequeños cambios que mejoran notablemente el rendimiento del modelo en todas las tareas en comparación con BERT. RoBERTa también utiliza un vocabulario más amplio (50K, frente los 30K de BERT).

Modelos multilingües

Dentro del campo de modelos multilingües, encontramos m-BERT [27] y XML-R [29]. Estos dos modelos han impulsado el estado del arte en tareas de PLN multilingüe mediante el pre-entrenamiento en muchos idiomas, mostrando cómo un único modelo puede aprender de varios idiomas, estableciendo bases sólidas para tareas no relacionadas con el inglés [30].

M-Bert (Multilingual BERT) ha sido pre-entrenado con el corpus Wikipedia en 104 idiomas, capaz de realizar una generalización multilingüe sorprendentemente bien [31]. Por su lado, XML-R (XLM-RoBERTa) XLM-RoBERTa es una versión multilingüe de RoBERTa. Está pre-entrenada en 2,5 TB de datos CommonCrawl filtrados que contienen 100 idiomas.

Modelos monolingües para el idioma español

El primer modelo monolingüe disponible públicamente en español fue BETO [30], un modelo BERT entrenado en su totalidad sobre un gran corpus en español, que mejora los resultados obtenidos por m-Bert para clasificar textos en español [2], lo que demuestra que un modelo monolingüe con suficiente entrenamiento puede superar a un modelo multilingüe, incluso cuando se utilizan más recursos y entrenamiento para este último [27]. BETO tiene un tamaño similar al de un BERT-Base (BERT-base tiene 12 capas, mientras que BERT-large 24). Existen 2 versiones de BETO, la cased y la uncased. En la versión uncased, el texto con el que se le ha entrenado ha sido previamente transformado a minúsculas, mientras que en la versión cased, el texto con el que se le ha entrenado es el mismo que el de entrada (sin cambios). Asimismo, en la versión uncased se eliminan los acentos, mientras que en la versión cased se conservan.

Más recientemente, se han desarrollado otros modelos lingüísticos para el español, como BERTIN [32] y RoBERTuito [25], ambos basados en la arquitectura RoBERTa.

III. Objetivos y Metodología

El objetivo general de este trabajo es comparar el rendimiento de diferentes algoritmos de aprendizaje profundo y transfer learning sobre el dataset creado por el proyecto HATEMEDIA, con el objetivo de determinar cuál clasifica mejor y concluir si es posible la detección automática de expresiones de odio dentro de este caso de estudio.

Los objetivos específicos y metodología necesarios para llevar a cabo el objetivo general consistirán en: realizar un estudio del estado del arte para identificar qué técnicas y métodos nos conviene utilizar en nuestra comparativa. Realizar un análisis exploratorio de los datos disponibles en el dataset de Hatemadia con el objetivo de identificar potenciales problemas y oportunidades. Preprocesar los datos y creación de diferentes versiones de nuestro dataset original; una versión completa y otras versiones reducidas pero balanceadas. Entrenar los modelos seleccionados con las diferentes versiones de nuestro dataset y medir sus rendimientos. Evaluar los resultados obtenidos para determinar la viabilidad de detección de expresiones de odio y la preferencia de usar alguno de los modelos, si la hubiera.

El desarrollo será iterativo siguiendo los objetivos específicos marcados en el trabajo.

IV. Contribución

En este trabajo queremos evaluar la viabilidad de utilizar técnicas de aprendizaje profundo y transfer learning sobre nuestro dataset de Hatemedia para obtener un modelo predictivo que permita la detección de expresiones de odio en castellano. Nuestra intención consiste en apoyarnos en estos datos para investigar, en primer lugar, si es viable entrenar un modelo de clasificación binario que permita detectar si un texto contiene odio (independientemente de su grado de intensidad) y, en caso afirmativo, determinaremos cuál de los modelos utilizados funciona mejor.

4.1 Dataset

El dataset utilizado proviene del proyecto Hatemedia, que ha centrado su estudio en los principales medios informativos profesionales de España (La Vanguardia, ABC, El País, El Mundo y 20Minutos), para analizar cómo se difunden las expresiones de odio en los entornos digitales asociados a este tipo de medios. Este dataset está compuesto por 574.760 registros. A pesar de tratarse de un dataset con una buena cantidad de registros, sufre del problema del desbalanceo de clases, donde existe una etiqueta que está representada en menor medida. Del total de registros, 12.296 están etiquetados como *ODIO* (el 2,1% de los datos), mientras que el 97,9% restante se corresponde con la etiqueta de *NO_ODIO*.

Por lo general, el desbalance de datos afecta a los algoritmos en su proceso de generalización, traduciéndose en que nuestro modelo entrenado no tenga una capacidad de predicción que nos sirva para su uso posterior [33]. Debido a este problema, se ha decidido crear tres versiones balanceadas del dataset original, de modo que podamos llevar a cabo diferentes pruebas en nuestra comparativa. Para ello, seleccionaremos todos los mensajes etiquetados como *ODIO* y añadiremos la misma cantidad de mensajes etiquetados como *NO_ODIO*, atendiendo a diferentes criterios para cada uno de los nuevos datasets. Llamaremos a estas versiones de los datasets V1, V2 y V3 respectivamente.

- Selección aleatoria de textos (V1): Tomaremos todos los textos etiquetados como *ODIO* y añadiremos aleatoriamente la misma cantidad de textos de *NO_ODIO*.

- Selección de textos de longitud homogénea (V2): En nuestro dataset original tenemos textos que van desde 1 sola palabra hasta una longitud máxima de 3044. A la hora de entrenar un algoritmo para que pueda

aprender a clasificar textos en *ODIO* y *NO_ODIO*, será importante conocer si obtener un subconjunto de textos de longitud homogénea supone alguna mejora en el rendimiento. Para ello crearemos un nuevo dataset balanceado, consistente en textos de longitud homogénea.

- Selección de textos correspondientes a un mismo medio (V3): Escogeremos textos relacionados con un solo medio de entre todos los disponibles (EL PAÍS, EL MUNDO, LA VANGUARDIA, 20MIN y ABC). Elegiremos el medio en función de cual tenga el mejor balance entre muestras *ODIO* y *NO_ODIO* y, dependiendo de los resultados obtenidos por el dataset anterior, seleccionaremos o no únicamente textos de longitud homogénea

4.2. Análisis y preparación de los datos

El primer paso en nuestro estudio consistió en un análisis pormenorizado de los datos disponibles en el dataset de Hatemadia, con el fin de entenderlos en profundidad y comprobar la calidad de los mismos. Inicialmente se realizó un estudio de los valores nulos con el objetivo principal de identificar estos registros y decidir cómo proceder con ellos. Finalmente, se eliminaron todos aquellos datos que no aportaban valor a nuestro estudio. Una vez terminamos el tratamiento de los valores nulos en el dataset, se realizó un análisis exploratorio de los datos, donde nos deshicimos de columnas innecesarias para nuestro estudio y pudimos comprobar el problema de desbalanceo de clases que sufre nuestro dataset, con 562.464 observaciones de *NO_ODIO*, frente a 12.296 de *ODIO*. Tras ello, preparamos implementamos una función que aplicaba un flujo de limpieza y preprocesado de los datos, realizando tareas como la eliminación de urls y caracteres especiales, eliminación de palabras de longitud<2, eliminación de espacios en blanco sobrantes, tokenización y lematización, de modo que los datos quedaran correctamente preparados para alimentar nuestros modelos. Asimismo, se realizó un estudio de la longitud de los textos. Analizar la longitud de los textos como una variable más nos reveló información importante sobre nuestros datos, como la longitud máxima y mínima, así como su distribución. Finalmente, tuvimos que aplicar un proceso de tokenización a los datos, de modo que fueran legibles por los modelos. La arquitectura BERT tiene su propia forma de tokenizar los datos, por lo que tuvimos que tratar de forma separada este proceso: por un lado, para los tres modelos de deep learning y, por otro, BETO.

4.3. Modelos de DL para la detección de odio

Para realizar nuestra comparativa, hemos seleccionado un total de 4 modelos predictivos, una red neuronal simple (SNN), una red convolucional (CNN), una red LSTM y el Transformer para el idioma español BETO.

Para decidir el diseño final de los modelos a utilizar, como el número de capas de convolución para la CNN, número y tamaño de los filtros, añadir o no más de una capa densa de neuronas, decidir si incluir capas de dropout, etc, hemos realizado pruebas tomando distintas combinaciones, entre ellas las configuraciones presentadas en el trabajo de [34], donde se realiza un análisis comparativo de modelos con el objetivo de detectar racismo y xenofobia en twitter usando redes CNN, LSTM y transfer learning. Finalmente, hemos optado por las arquitecturas que se describen a continuación:

SNN: Este sencillo modelo consistirá en una primera capa de embedding que será posteriormente aplanada y conectada directamente a una capa densa de 1 neurona con una función de activación sigmoid, que será la encargada de devolver el resultado de la clasificación binaria.

CNN: Esta red estará compuesta por una primera capa de embedding, seguida por una capa convolucional 1D (probaremos diferente número y tamaño de filtros para seleccionar la mejor combinación). La función de activación utilizada en esta capa será la función ReLU (Unidad Lineal Rectificada), que en la actualidad es la función de activación con más éxito y más utilizada en redes de neuronas profundas [35]. A la salida de esta capa de convolución se le aplicará una función de MaxPooling para reducir el tamaño de las muestras, y el resultado se conectará a una capa densa de 1 neurona con una función sigmoid.

LSTM: Utilizaremos en primer lugar una capa de embedding, seguida de una capa LSTM (probaremos diferente número de neuronas para poder seleccionar la mejor opción). La salida irá conectada, al igual que en los casos anteriores, a una capa densa de 1 neurona con función de activación sigmoid.

BETO: Finalmente, utilizaremos en nuestra comparativa el modelo transformer monolingüe para el idioma español BETO, tanto la versión cased como uncased ("dccuchile/bert-base-spanish-wwm-cased" y "dccuchile/bert-base-spanish-wwm-uncased" respectivamente). Estos modelos se pueden encontrar en la web de Hugging Face[8], y son accesibles desde el código a través de la biblioteca Transformers[9].

4.4 Parametrización

Con el fin de garantizar que los resultados obtenidos por la red neuronal sean lo más elevados posibles, se han realizado una serie de pruebas en las que se ha comprobado el rendimiento del modelo en función del valor de determinados parámetros que mostramos en las Tablas 1, 2 y 3.

Tabla 1: Parámetros seleccionados para CNN

Parámetro	Opciones probadas	Opción seleccionada
Batch size	25, 50, 100	**50**
Epochs	2, 3, 5	**2**
N_Filtros	32, 64, 128	**64**
Tamaño Filtro	3, 4, 5	**3**
Optimizador	Adam, SGD	**Adam**
Learning rate	1e−2, 1e−3	**1e−3**

Tabla 2: Parámetros seleccionados para LSTM

Parámetro	Opciones probadas	Opción seleccionada
Batch size	25, 50, 100	**50**
Epochs	2, 3, 5	**2**
N_Filtros	32, 64, 128	**64**
Optimizador	Adam, SGD	**Adam**
Learning rate	1e−2, 1e−3	**1e−3**

[8] https://huggingface.co/dccuchile

[9] https://huggingface.co/docs/transformers

Tabla 3: Parámetros seleccionados para BETO

Parámetro	Opciones probadas	Opción seleccionada
Tipo de Modelo	cased, uncased	cased
Batch size	25, 50, 100	50
Epochs	2, 3, 5	2
Optimizador	Adam, SGD	Adam
Learning rate	2e−5, 3e−5, 4e−5	**2e−5**

4.4 Métricas de evaluación

Como métricas para comparar los distintos modelos vamos a utilizar la accuracy (Acc) o exactitud, que indica el número de muestras correctamente clasificadas para todas las clases sobre el total de muestras, la F1-score, una métrica que combina Precisión y Recall, y a Macro-F1, que se trata de la media no ponderada de las puntuaciones F1-score.

$$Acc = \frac{TP+TN}{TP+TN+FP+FN};\ \text{Precisión} = \frac{TP}{TP+FP};\ Recall = \frac{TP}{TP+FN}$$

$$\text{F1-score} = \frac{2*\text{Precisión}*\text{Recall}}{\text{Precisión}+\text{Recall}};\ \text{Macro-F1} = \frac{\text{sum(F1-scores)}}{\text{número de clases}}$$

Donde: TP representa las muestras negativas correctamente clasificadas, FP las muestras positivas clasificadas como negativas, FN las negativas clasificadas como positivas y FP las positivas clasificadas como negativas.

V. Evaluación y Resultados

Para cada uno de los modelos seleccionados, realizaremos experimentos con las distintas versiones del dataset descritos en el apartado anterior. Todos los experimentos han sido ejecutados desde la versión gratuita colab, configurando el entorno para hacer uso del modo de ejecución GPU (Tesla T4), lo que nos ha permitido ejecutar nuestros modelos hasta 10 veces más rápido que desde el entorno básico.

Resultados con dataset completo

Las pruebas realizadas para el dataset completo muestran unos resultados propios de un dataset desbalanceado (Figura 2). Al final, al modelo le basta con predecir siempre la clase dominante para conseguir un 98% de accuracy (acertando siempre los textos de *NO_ODIO* conseguimos acertar un 98% de las ocasiones). Sin embargo, esto implica que nunca predice la etiqueta *ODIO*, y esto se traduce en un recall del 0% para esta clase. Por lo tanto, estos modelos entrenados con el dataset completo no son útiles en absoluto para resolver nuestro problema.

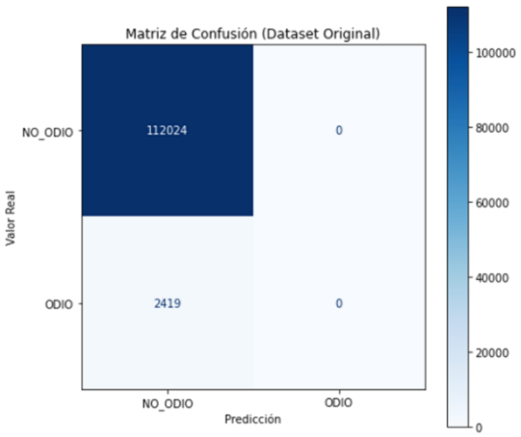

Figura 2: Matriz de confusión con dataset completo para todos los modelos

Estos mismos resultados se han obtenido para los diferentes valores de batch_size y epochs.

Resultados con datasets balanceados (V1, V2, V3)

La Tabla 4 muestra los resultados obtenidos por los modelos para cada una de las versiones del dataset balanceado V1, V2, y V3. Estos resultados se han conseguido con un batch_size de tamaño 50 y 2 épocas. Hemos comprobado que más allá de la tercera epoch perdemos accuracy para el conjunto de test debido al fenómeno overfitting o sobreajuste del modelo al conjunto de entrenamiento.

Tabla 4: Comparativa de resultados en test

		SNN	CNN	LSTM	BETO (cased)
Accuracy	V1	86%	87%	49%	**89%**
	V2	83%	85%	84%	**87%**
	V3	76%	83%	80%	**84%**
F1-score (NO_ODIO)	V1	86%	87%	0%	**88%**
	V2	83%	85%	84%	**86%**
	V3	77%	83%	79%	**84%**
F1-score (ODIO)	V1	85%	87%	66%	**89%**
	V2	82%	85%	83%	**87%**
	V3	75%	82%	80%	**84%**

A diferencia de lo que ocurría con el dataset completo, para los datasets balanceados se obtienen unos valores razonables para todas las métricas, con excepción del modelo LSTM al entrenarse con el dataset V1. En este caso, nuestro modelo no pasa de un 49% de accuracy y un 0% de F1 para la clase de *NO_ODIO*. Es evidente que el modelo LSTM no funciona bien para este dataset V1, pues únicamente predice la clase de *ODIO*. El motivo de este comportamiento es la longitud máxima de los textos (3044 caracteres) que hace que los mensajes más cortos estén compuestos mayoritariamente por valores 0 (tras el proceso de padding), perjudicando el rendimiento de nuestro algoritmo basado en la arquitectura RNN. Confirmamos este punto con los resultados de los datasets V2 y V3, donde obtenemos unos resultados bastante buenos, cercanos a los obtenidos por CNN.

Por otro lado, observamos que con dataset balanceado V2 empeoramos ligeramente todas las métricas con respecto al dataset V1. Esto indica que el hecho de tener unos textos de longitud homogénea no ha ayudado en este sentido a los modelos a predecir mejor. Sin embargo, debemos comentar que el tiempo de ejecución sí mejoró, reduciéndose aproximadamente un 30% al acotarse la longitud de los textos. Para el dataset V3, observamos unas métricas por debajo de lo conseguido con el dataset V2. Creemos que estos resultados se deben a la reducción del número total de registros de entrenamiento (hemos pasado de 23.932 registros en V2 frente a 10.268 registros en V3). Al tener menos muestras para aprender, los modelos caen en overfitting, sobre ajustándose a los datos de entrenamiento, y no es capaz de generalizar correctamente

VI. Discusión

En primer lugar, debemos comentar que el dataset original presenta un desbalanceo de clases tan acuciado que resulta inservible, pues todos los modelos entrenados con este dataset terminan prediciendo siempre la clase dominante.

Si nos centramos en las pruebas realizadas con los tres datasets balanceados, observamos que los mejores resultados

en términos de accuracy y F1 los encontramos con BETO (cased) entrenado con el dataset V1, seguido por CNN entrenado también con V1 y empatado con BETO entrenado con V2 (Tabla 4).

Estos resultados nos podrían llevar a pensar que un dataset balanceado de selección aleatoria (V1) es la mejor opción para entrenar nuestros modelos. Sin embargo, este dataset entraña una serie de desventajas: por ejemplo, el modelo LSTM no funciona correctamente con este dataset debido a la longitud máxima de los textos (3.044 palabras) que hace que los mensajes más cortos estén compuestos mayoritariamente por valores 0 (padding), perjudicando el rendimiento de nuestro algoritmo basado en la arquitectura de red recurrente. Se ha comprobado que, con textos de longitud homogénea, este modelo alcanza una exactitud cercana a la obtenida por CNN (aunque con tiempos de entrenamiento entre 2 y 3 tres veces superiores en LSTM).

La Tabla 5 muestra los resultados del estado del arte (SOA) para la detección de discurso de odio en español obtenidos a partir de los conjuntos de datos de HaterNet y HatEval, además de un dataset ad-hoc creado en el trabajo de [36]. Comparamos estos resultados con nuestra mejor propuesta en términos de macro-F1, el modelo BETO cased entrenado con el dataset balanceado V1.

Para el conjunto de datos HaterNet, el modelo de [37] presenta un modelo BETO (cased) que supera a nuestra propuesta en términos de F1 para la etiqueta *NO_ODIO*, alcanzando un 89% frente a nuestro 88%. Sin embargo, en términos de macro-F1, nuestro modelo BETO mejora los resultados de [37], en un 14%. El motivo es que nuestro modelo es capaz de predecir mejor la etiqueta de odio, seguramente a consecuencia de haber sido entrenado con un dataset mejor balanceado y con un mayor número de muestras de textos de odio. En la Tabla 6 mostramos la distribución de cada dataset.

En el estudio de [36] se generaron un total de ocho modelos predictivos: seis usando algoritmos de aprendizaje superficial, uno generado a partir de los votos de esos modelos anteriores y otro usando aprendizaje profundo. Los mejores resultados los obtuvo este último modelo, basado en una red neuronal recurrente GRU, alcanzando un macro-F1 del 77%. Nuestra propuesta mejora este resultado en un 16%.

En cuanto al conjunto de datos HatEval, [37] y [25] superaron el mejor resultado obtenido en la competición de SemEval-2019 Task 5, probando un modelo BETO (cased) y Robertuito (uncased) respectivamente. El modelo Robertuito es el que alcanza mejor rendimiento en términos de Macro-F1, con un 80%. Nuestro modelo BETO cased entrenado sobre el dataset V1 supera los resultados de [25] con una mejora del 11%.

Tabla 5: Resultados SOA para detección de discurso de odio en español

Modelo	Dataset	F1 (NO_ODIO)	F1 (ODIO)	Macro-F1
SVM (SemEval-2019 Task 5)	**HatEval**	76%	70%	73%
RNN-GRU [36]	**Ad-hoc**	-	-	77%
BETO$_{cased}$ [37]	**HaterNet**	**89%**	66%	78%
BETO$_{cased}$ [37]	**HatEval**	80%	76%	78%
Robertuito$_{uncased}$ [25]	**HatEval**	-	-	80%
BETO$_{cased}$ (Nuestra propuesta)	**Hatemedia (V1)**	88%	**89%**	**89%**

Comparando estos resultados destacamos la importancia de entrenar estos modelos de transfer learning sobre un dataset balanceado con el mayor número posible de registros, con el fin de obtener un modelo que reajuste sus parámetros en base a la información del dataset sin caer en el fenómeno de overfitting, de modo que a posteriori sea capaz de generalizar con la llegada de nuevos datos.

Tabla 6: Distribución de los datasets en español

Dataset	n_registros	NO_ODIO	ODIO
HatEval	**6.600**	**3.861** (59%)	**2.739** (41%)
HaterNet	**6.000**	**4.433** (74%)	**1.567** (26%)
Ad-hoc [36]	**10.213**	**3879** (38%)	**6334** (62%)
Hatemedia (V1)	**24.546**	**12.273** (50%)	**12.273** (50%)

VII. Conclusiones

Si bien los resultados a partir del dataset original no son buenos debido al problema del desbalanceo de clases, los modelos generados a partir de los datasets balanceados tienen unas prestaciones más que aceptables, concluyendo que la detección automática de expresiones de odio es viable para estos datasets balanceados, siendo BETO (versión cased) el modelo que mejor resultados ofrece en términos de accuracy y F1-score, seguido muy de cerca por el modelo CNN. No recomendaríamos LSTM para análisis de textos extensos, al menos cuando la longitud de los textos del dataset no sea homogénea, a la vista de los resultados obtenidos por este modelo con V1. Por el contrario, tanto BETO como CNN han respondido bien a todas las versiones de los datasets balanceados, por lo que los convierte en los modelos más versátiles

Finalmente, como líneas futuras planteamos. realizar las mismas pruebas presentadas en este trabajo, pero con un dataset balanceado con mayor cantidad de registros. Nuestros datasets balanceados han permitido generar un conjunto de entrenamiento del orden de 20K registros, lo que ha dificultado la capacidad de los modelos para generalizar, cayendo en el problema de overfitting o sobreajuste tras las primeras épocas de entrenamiento.

Referencias

[1] Arango, A., Pérez, J., & Poblete, B. (2019). Hate Speech Detection is Not as Easy as You May Think: A Closer Look at Model Validation. Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, 45-54. https://doi.org/10.1145/3331184.3331262

[2] García-Díaz, J. A., Jiménez-Zafra, S. M., García-Cumbreras, M. A., & Valencia-García, R. (2022). Evaluating feature combination strategies for hate-speech detection in Spanish using linguistic features and transformers. Complex and Intelligent Systems. https://doi.org/10.1007/s40747-022-00693-x

[3] Basile, V., Bosco, C., Fersini, E., Nozza, D., Patti, V., Rangel Pardo, F. M., Rosso, P., & Sanguinetti, M. (2019). SemEval-2019 Task 5: Multilingual Detection of Hate Speech Against Immigrants and Women in Twitter. Proceedings of the 13th International Workshop on Semantic Evaluation, 54-63. https://doi.org/10.18653/v1/S19-2007

[4] Fersini, E., Rosso, P., & Anzovino, M. (2018). Overview of the Task on Automatic Misogyny Identification at IberEval 2018. https://figure-eight.com/

[5] Waseem, Z., & Hovy, D. (2016). Hateful Symbols or Hateful

People? Predictive Features for Hate Speech Detection on Twitter. NAACL.

[6] Davidson, T., Warmsley, D., Macy, M., & Weber, I. (2017). Automated Hate Speech Detection and the Problem of Offensive Language. http://arxiv.org/abs/1703.04009

[7] Bohra, A., Vijay, D., Singh, V., Akhtar, S. S., & Shrivastava, M. (2018). A Dataset of Hindi-English Code-Mixed Social Media Text for Hate Speech Detection. https://github.com/deepanshu1995/HateSpeech-Hindi-

[8] Pereira-Kohatsu, J. C., Quijano-Sánchez, L., Liberatore, F., & Camacho-Collados, M. (2019). Detecting and Monitoring Hate Speech in Twitter. Sensors, 19(21). https://doi.org/10.3390/s19214654

[9] Urdaneta, L. A. (2019, abril). Reducir el número de palabras de un texto: lematización y radicalización (stemming) con Python. https://medium.com/qu4nt/reducir-el-n%C3%BAmero-de-palabras-de-un-texto-lematizaci%C3%B3n-y-radicalizaci%C3%B3n-stemming-con-python-965bfd0c69fa

[10] Luhn, H. P. (1957). A Statistical Approach to Mechanized Encoding and Searching of Literary Information. IBM Journal of Research and Development, 1(4), 309-317. https://doi.org/10.1147/rd.14.0309

[11] Firth, J. R. (1957). A synopsis of linguistic theory 1930-55. Studies in Linguistic Analysis (Special Volume of the Philological Society), 1952-59, 1-32.

[12] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. arXiv. https://doi.org/10.48550/ARXIV.1310.4546

[13] Melnyk, L. (2021). Hate speech targets in COVID-19 related comments on Ukrainian news websites. Journal of Computer-Assisted Linguistic Research, 5(1), 47-75. https://doi.org/10.4995/jclr.2021.15966

[14] Dash, S., Grover, R., Shekhawat, G., Kaur, S., Mishra, D., & Pal, J. (2021). Insights Into Incitement: A Computational Perspective on Dangerous Speech on Twitter in India. http://arxiv.org/abs/2111.03906

[15] Pennington, J., Socher, R., & Manning, C. (2014). GloVe: Global Vectors for Word Representation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 1532-1543. https://doi.org/10.3115/v1/D14-1162

[16] Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching Word Vectors with Subword Information. Transactions of the Association for Computational Linguistics, 5, 135-146. https://doi.org/10.1162/tacl_a_00051

[17] Burnap, P., & Williams, M. (2015). Cyber Hate Speech on Twitter: An Application of Machine Classification and Statistical Modeling for Policy and Decision Making: Machine Classification of Cyber Hate Speech. Policy & Internet, 7. https://doi.org/10.1002/poi3.85

[18] Nobata, C., Tetreault, J., Thomas, A., Mehdad, Y., & Chang, Y. (2016). Abusive language detection in online user content. 25th International World Wide Web Conference, WWW 2016, 145-153. https://doi.org/10.1145/2872427.2883062

[19] Sachdeva, J., Chaudhary, K. K., Madaan, H., & Meel, P. (2021). Text Based Hate-Speech Analysis. Proceedings - International Conference on Artificial Intelligence and Smart Systems, ICAIS 2021, 661-668. https://doi.org/10.1109/ICAIS50930.2021.9396013

[20] Badjatiya, P., Gupta, S., Gupta, M., & Varma, V. (2017). Deep Learning for Hate Speech Detection in Tweets. Proceedings of the 26th International Conference on World Wide Web Companion - WWW 17 Companion. https://doi.org/10.1145/3041021.3054223

[21] Gambäck, B., & Sikdar, U. K. (2017). Using Convolutional Neural Networks to Classify Hate-Speech. Proceedings of the First Workshop on Abusive Language Online, 85-90. https://doi.org/10.18653/v1/W17-3013

[22] Wang, C. (2018). Interpreting Neural Network Hate Speech Classifiers.

[23] Vigna, F. del, Cimino, A., Dell'orletta, F., Petrocchi, M., & Tesconi, M. (2017). Hate me, hate me not: Hate speech detection on Facebook. https://curl.haxx.se

[24] Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. http://arxiv.org/abs/1412.3555

[25] Pérez, J. M., Furman, D. A., Alemany, L. A., & Luque, F. (2021). RoBERTuito: a pre-trained language model for social media text in Spanish. http://arxiv.org/abs/2111.09453

[26] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. ukasz, & Polosukhin, I. (2017). Attention is All you Need. En I. Guyon, U. von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), Advances in Neural Information Processing Systems (Vol. 30). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[27] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), 4171-4186. https://doi.org/10.18653/v1/N19-1423

[28] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. http://arxiv.org/abs/1907.11692

[29] Lample, G., & Conneau, A. (2019). Cross-lingual Language Model Pretraining. http://arxiv.org/abs/1901.07291

[30] Cañete, J., Chaperon, G., Fuentes, R., Ho, J.-H., Kang, H., & Pérez, J. (2020). SPANISH PRE-TRAINED BERT MODEL AND EVALUATION DATA. https://github.com/josecannete/spanish-corpora

[31] Pires, T., Schlinger, E., & Garrette, D. (2019). How Multilingual is Multilingual BERT? Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, 4996-5001. https://doi.org/10.18653/v1/P19-1493

[32] de la Rosa, J., Ponferrada, E. G., Villegas, P., Salas, P. G. de P., Romero, M., & Grandury, M. (2022). BERTIN: Efficient Pre-Training of a Spanish Language Model using Perplexity Sampling. http://arxiv.org/abs/2207.06814

[33] Chawla, N. v, Japkowicz, N., & Kotcz, A. (2004). Editorial:

Special Issue on Learning from Imbalanced Data Sets. SIGKDD Explor. Newsl., 6(1), 1-6. https://doi.org/10.1145/1007730.1007733

[34] Benítez-Andrades, J. A., González-Jiménez, Á., López-Brea, Á., Aveleira-Mata, J., Alija-Pérez, J. M., & García-Ordás, M. T. (2022). Detecting racism and xenophobia using deep learning models on Twitter data: CNN, LSTM and BERT. PeerJ Computer Science, 8. https://doi.org/10.7717/PEERJ-CS.906

[35] Ramachandran, P., Zoph, B., & Le, Q. v. (2017). Searching for Activation Functions. http://arxiv.org/abs/1710.05941

[36] Amores, J. J., Blanco-Herrero, D., Sánchez-Holgado, P., & Frías-Vázquez, M. (2021). Detecting ideological hatred on Twitter. Development and evaluation of a political ideology hate speech detector in tweets in Spanish. Cuadernos.Info, 49, 98-124. https://doi.org/10.7764/cdi.49.27817

[37] Plaza-del-Arco, F. M., Molina-González, M. D., Ureña-López, L. A., & Martín-Valdivia, M. T. (2021). Comparing pre-trained language models for Spanish hate speech detection. Expert Systems with Applications, 166. https://doi.org/10.1016/j.eswa.2020.114120

[38] Nguyen, H.-T., Nguyen, M., & Le. (2017, enero). An Ensemble Method with Sentiment Features and Clustering Support.