Escuela Politécnica Superior de Jaén
Grado en Ingeniería Informática

Universidad de Jaén
Escuela Politécnica Superior
de Jaén

TRABAJO FIN DE GRADO

DETECCIÓN DE LA OFENSIVIDAD EN REDES SOCIALES MEDIANTE LENGUAJE NATURAL E INFORMACIÓN CONTEXTUAL

Alumna

María Estrella Vallecillo Rodríguez

Tutores

Arturo Montejo Ráez
(Departamento de Informática)

Flor Miriam Plaza del Arco
(Departamento de Informática)

Noviembre, 2022

(Página intencionalmente en blanco)

Universidad de Jaén

Departamento de Informática

Don Arturo Montejo Ráez y Doña Flor Miriam Plaza del Arco, tutores del Trabajo Fin de Grado titulado: '**Detección de la ofensividad en redes sociales mediante lenguaje natural e información contextual**', que presenta Doña María Estrella Vallecillo Rodríguez, otorgan el visto bueno para su entrega y defensa en la Escuela Politécnica Superior de Jaén.

Jaén, Noviembre de 2022

La alumna:

VALLECILLO RODRIGUEZ MARIA ESTRELLA - 26053388T

Firmado digitalmente por VALLECILLO RODRIGUEZ MARIA ESTRELLA - 26053388T
Fecha: 2022.11.04 12:23:11 +01'00'

María Estrella Vallecillo Rodríguez

Los tutores:

Firmado digitalmente por MONTEJO RAEZ ARTURO - ***2658** el día 04/11/2022

PLAZA DEL ARCO FLOR MIRIAN - 15519730C

Digitally signed by PLAZA DEL ARCO FLOR MIRIAN - 15519730C
Date: 2022.11.04 14:54:56 +01'00'

Arturo Montejo Ráez

Flor Miriam Plaza del Arco

(Página intencionalmente en blanco)

Agradecimientos

Me gustaría dar las gracias a todas aquellas personas que han estado a mi lado en esta etapa y que me han aportado su granito de arena para hacerme la persona que soy hoy en día. En especial a mi pareja, por su paciencia infinita, su confianza y su apoyo, y a mi madre y mi hermana porque juntas hemos disfrutado los buenos momentos y nos hemos unido aún más en los malos.

Por otro lado, quiero agradecer a todos y cada uno de mis profesores a lo largo de toda mi vida académica, ya que de una manera u otra han conseguido que hoy esté aquí. En especial, a mis dos tutores, Flor Miriam Plaza del Arco y Arturo Montejo Ráez, por dedicarme parte de sus vacaciones, por confiar en mi, guiarme, apoyarme y darme todos esos consejos durante el desarrollo de este trabajo, pero sobre todo por mostrarme esa parte de la informática que no conocía y que me ha terminado encantando.

Finalmente, quiero acordarme de todos mis compañeros, con los que he podido compartir muchas horas de café, de prácticas y de estudio. Destacando a Clara Díaz y David de la Rosa, con los que he tenido muchísima suerte de tener a mi lado estos cuatro años y sin ellos la carrera no habría sido lo mismo.

Sin duda, este trabajo no habría sido posible sin todos vosotros. Muchísimas gracias.

Ficha del Trabajo Fin de Título	
Titulación	Grado en Ingeniería Informática
Modalidad	Trabajo teórico/Experimental
Especialidad (solo TFG)	Tecnología de la Información
Mención (solo TFG)	Sin mención
Idioma	Español
Tipo	Específico
TFT en equipo	No
Autor/a	María Estrella Vallecillo Rodríguez
Fecha de asignación	03/11/2022

Descripción corta	El propósito de este proyecto, es estudiar la manera de integrar información contextual en los sistemas que tratan de detectar el lenguaje ofensivo. Para ello, se plantean diversos experimentos, en los que se desarrollan sistemas que introducen estas características contextuales de diversas maneras, además de otros sistemas más tradicionales que no tienen en cuenta esta información contextual. Finalmente, se extraen conclusiones de los resultados obtenidos.

NORMAS APLICADAS EN ESTE DOCUMENTO	
LOCALES	
TFT-UJA:2017	Normativa de Trabajos Fin de Grado, Fin de Máster y otros Trabajos Fin de Título de la Universidad de Jaén (Normativa marco UJA aprobada en Consejo de Gobierno)
TFT-EPSJ:2017	Normativa sobre Trabajos Fin de Grado y Fin de Máster en la Escuela Politécnica Superior de Jaén (Normativa EPSJ aprobada en Junta de Escuela)
TFT-EPSJ	Criterios de evaluación y normas de estilo para TFG y TFM de la Escuela Politécnica Superior de Jaén
NACIONALES E INTERNACIONALES	
ISO 2145:1978	Documentación - Numeración de divisiones y subdivisiones en documentos escritos
UNE 50132:1994	Traducción de la ISO 2145
APA 6ª edición	Estilo de referencias y citas de APA (American Psychological Association)

NORMAS UTILIZADAS COMO BASE O REFERENCIA	
NACIONALES	
UNE 157001:2014	Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico
UNE 157801:2007	Criterios generales para la elaboración de proyectos de sistemas de información
Estas normas se han utilizado como base o referencia para la inclusión de algunos contenidos y definiciones sobre elaboración de proyectos, entendiendo como proyecto la documentación consensuada entre una empresa y un cliente, que da lugar al perfeccionamiento de un contrato para la elaboración de una obra o la prestación de un servicio. Por consiguiente, no debe esperarse la aplicación de estas normas en cuanto a la completitud de los contenidos ni a la organización de los mismos.	

Contenido

Índice de Ilustraciones

Índice de Tablas

1 ESPECIFICACIÓN DEL TRABAJO

1.1 Introducción

El Procesamiento del Lenguaje Natural, más conocido como PLN, es una rama de la Inteligencia Artificial que busca herramientas y técnicas para facilitar la interacción hombre-máquina por medio del lenguaje natural, ya que este es el lenguaje que los humanos hablamos, siendo más flexible y menos rígido que los lenguajes formales como los lenguajes de programación y las fórmulas matemáticas. El PLN se puede aplicar a una gran cantidad de tareas como, por ejemplo, traducción automática, recuperación de información, sistemas de búsqueda de respuesta, extracción de conocimiento, reconocimiento y generación del habla, clasificación de texto, entre otras.

En los últimos años, esta disciplina ha aumentado el interés de las empresas, gobiernos e investigadores, pues cada día se mueve una gran cantidad de datos en Internet, y disponer de técnicas que nos ayuden a analizarla de forma automática, supone un ahorro de tiempo y costes.

En este trabajo, nos vamos a centrar en la tarea de clasificación de texto, y más concretamente en la detección del lenguaje ofensivo en redes sociales, ya que hoy en día, bien debido al anonimato o la comunicación online, el comportamiento nocivo cada vez se da más en estos medios, dando lugar incluso a casos de bullying o ciberacoso, especialmente entre los más jóvenes. El hecho de poder tener perfiles en una red social sin necesidad de identificarte, hace que sea más fácil ofender a desconocidos sin temor a las represalias que esto pueda ocasionar, por lo que disponer de sistemas que detecten automáticamente cuando un comentario puede ser ofensivo para alguien sería muy útil para intentar frenar esta práctica y así evitar que la toxicidad de un comentario ofensivo se expanda por la red con mensajes similares dirigidos hacia la misma persona o el mismo colectivo.

El PLN dispone de diversas técnicas a la hora de detectar la ofensividad, por ejemplo, tradicionalmente se elaboraban lexicones o bolsas de palabras para ver si los textos contenían palabras ofensivas, posteriormente se pasó a las redes neuronales tradicionales como las máquinas de vectores de soporte, SVM, random forest, decision tree classifier… Hasta que llegaron los modelos Transformers, una de las técnicas más novedosas de los últimos años en la disciplina del PLN, basadas en redes neuronales profundas. Más adelante, explicaremos en profundidad en los modelos Transformers.

1.2 Objetivos del trabajo

En este apartado, vamos a explicar cuáles son los objetivos que queremos cumplir en el desarrollo del proyecto y que, en parte, motivaron su propuesta.

1.2.1 Objetivos generales

Los objetivos principales con los que se planteó el trabajo fueron:

- Estudio del estado de la cuestión para conocer las técnicas apropiadas para abordar la clasificación de textos, y más concretamente la detección de la ofensividad.
- Desarrollar un sistema que fuese capaz de detectar aquellos mensajes de texto que son ofensivos, para el idioma español.
- Investigar sobre los modelos del lenguaje existentes y cuáles resultan más adecuados para resolver esta tarea.
- Búsqueda de un corpus que contenga datos suficientes para entrenar los sistemas planteados y poder evaluarlos.

- Analizar el contexto en el que se producen los mensajes ofensivos y ver de qué manera podemos integrar un sistema de clasificación automático para ayudar a la detección de estos comentarios.

1.2.2 Objetivos específicos

Los objetivos específicos que queremos lograr en este proyecto son:

- Realizar un ajuste de modelos que ya han sido pre-entrenados.
- Realizar una búsqueda óptima de hiper-parámetros para obtener modelos con el mejor rendimiento posible.
- Explorar diferentes técnicas de integración de características no lingüísticas que puedan ayudar a realizar la clasificación.
- Investigar la importancia del formato de integración de los datos contextuales, viendo si es mejor introducir textos tal cual se producen en las redes sociales o intentando eliminar aquello que no es útil para la clasificación y quitando así, el ruido que llevan estos textos como por ejemplo, enlaces a otras páginas web, menciones a otros usuarios o hashtags.

1.3 Antecedentes y estado del arte

La detección de la ofensividad de forma automática trata de elaborar algoritmos que sean capaces de reconocer cuando un texto es ofensivo. Para realizar esta tarea, los algoritmos deben identificar si un texto es ofensivo o no y asignarle una etiqueta que nos diga si pertenece al grupo de los ofensivos o no ofensivos. Por lo que, estamos ante una tarea de clasificación de textos, en la que tenemos distintas categorías y un texto sólo puede pertenecer a una de estas.

Para la realización de este trabajo, situamos el punto de partida en la tarea MeOffendES 2021, organizada en Iberian Languages Evaluation Forum (IberLEF [1])

de 2021, enmarcado en el 37º congreso anual de la Sociedad Española de Procesamiento del Lenguaje Natural (SEPLN 2021). En esta tarea, el objetivo principal era fomentar la detección del lenguaje ofensivo en las distintas variantes del español. Nos encontramos con cuatro tareas a realizar, dos tareas destinadas a reconocer el lenguaje ofensivo en textos extraídos de distintas redes sociales para el español y otras dos tareas destinadas a la variante mexicana del español.

En este trabajo nos vamos a enfocar en las dos primeras tareas, debido a que son las que se centran en el castellano, lengua para la que queremos desarrollar nuestro sistema, y no en la variante mexicana del español que son las dos tareas restantes. En dichas tareas, los participantes debían de realizar una clasificación multiclase de textos extraídos de distintas redes sociales, con cuatro clases entre las que podemos clasificar los comentarios, donde sólo una de ellas es la correcta. La primera tarea se basaba en reconocer la clase de cada texto sin tener en cuenta información extra (influencer al que se dirige el comentario y su género, red social en la que este se publica), lo que nosotros llamaremos metadatos. En la segunda tarea, sí se proporciona información contextual del mensaje. Para la resolución de estas tareas, los distintos participantes implementaron redes neuronales tradicionales y distintas arquitecturas Transformers. A continuación, describimos qué hizo cada equipo en la competición. El equipo que obtuvo los mejores resultados usó un modelo Transformer multilingüe, este modelo necesitaba mucha memoria, ya que establece los tamaños de lote (más adelante veremos qué es esto) muy grandes y modificó algunos de los hiper-parámetros para obtener el resultado más óptimo. El equipo que quedó en segundo lugar exploró distintas características de los textos, como características léxicas, de negación, incrustaciones de palabras, entre otras. También estudiaron de qué manera podían añadir dichas características a un perceptrón multicapa final. Un perceptrón multicapa es una red neuronal artificial, y hablaremos más adelante sobre esto. El equipo que quedó en tercer lugar hizo un estudio para ver si funcionaba mejor un sistema de clasificación de secuencias o un modelo de lenguaje, concretamente BERT, para obtener la salida de este modelo. Esta propuesta agrupa el *max pooling* (una técnica de reducción de los datos que se van a usar) de todas las capas del modelo con la codificación de los tokens obtenidos en la última capa. Además, integra dos técnicas, una en la que

entrenaban el modelo una vez y con la salida de este primer entrenamiento volvía a entrar en el proceso de aprendizaje para obtener un conjunto más largo. La otra técnica fue usar la técnica de “pérdida focal” para disminuir el desequilibrio que hay entre clases en el conjunto de datos. El equipo en cuarto lugar, estudió diferentes modelos de aprendizaje profundo basado en redes neuronales y un modelo del lenguaje , en este caso el modelo BERT, obteniendo los mejores resultados cuando utilizan modelos del lenguaje. Finalmente, el equipo en último lugar elaboró un algoritmo de aprendizaje automático clásico, basado en una máquina de soporte de vectores (SVM). Según podemos observar, los modelos de redes neuronales superan con diferencia a los algoritmos clásicos, por lo que nosotros nos centraremos en estos modelos de redes neuronales para desarrollar nuestro sistema e intentaremos superar los mejores resultados para el modelo que quedó en primer lugar en esta competición.

Para evaluar nuestro sistema y poderlo comparar con el resto, hacemos uso de las métricas más empleadas para la clasificación de textos en PLN y además propuestas por los organizadores de la competición de MeOffendES:

- Precisión: muy útil para ver la calidad de los modelos a la hora de clasificar. Trata de identificar el porcentaje de comentarios de cada clase pertenecen a esa clase, de entre todos los predichos para esa clase. En la Ilustración 1.1 podemos ver la forma de calcular la precisión, dónde TP son los comentarios que se han predicho correctamente y FP aquellos textos que se han predicho en la clase a analizar y que verdaderamente no pertenecen a la clase predicha.

$$precision = \frac{TP}{TP + FP}$$

Ilustración 1.1 Fórmula de la precisión **[31]**

- Recall: esta métrica nos informa sobre la cantidad de comentarios de cada clase que el modelo es capaz de reconocer. Es decir, el número de

comentarios correctamente predichos entre el número de comentarios que verdaderamente pertenecen a esa clase. En la Ilustración 1.2 podemos ver la fórmula con la que se calcula el recall, dónde TP son los comentarios que se han predicho correctamente y FN son los comentarios que verdaderamente pertenecen a la clase y no se predicen como pertenecientes a ella.

$$recall = \frac{TP}{TP + FN}$$

***Ilustración 1.2 Fórmula del recall* [31]**

- F1-score: esta métrica combina la precisión y el recall en un solo valor, mediante una media armónica de ambas. Esto nos permite poder comparar el rendimiento combinado entre la precisión y el recall. En la imagen inferior (Ilustración 1.3), podemos ver la fórmula usada para calcular el F1-score.

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

***Ilustración 1.3 Fórmula del F1-score* [31]**

Es importante señalar, que cuanto más queramos afinar un sistema en cuanto a recall, más nos alejaremos de un sistema con buena precisión, por lo que es fundamental encontrar un buen equilibrio de ambas.

Por otro lado, nos encontramos tres formas de calcular estas métricas:

- Macro: esta estrategia de cálculo de métricas, nos indica cómo de bien se clasifica en todas las clases, ya que es una media aritmética de todas las puntuaciones obtenidas de la métrica a medir por clase.

- Micro: calcula una puntuación de la métrica a tener en cuenta, de forma global. Esto hace que cuanto mejor se clasifique en la clase más abundante en el conjunto de datos, mejor será este tipo de métricas.
- Weighted: esta estrategia, realiza una media ponderada, viendo el peso de cada clase en el conjunto de datos.

1.4 Alcance

Con los objetivos ya definidos previamente, vamos a explicar qué entregables y resultados esperamos al finalizar el proyecto:

- **Código fuente.** Se adjuntará el código fuente con el que se realizaron todos estos experimentos.
- **Resultados.** Junto con el código también se incluirán los resultados obtenidos en cada experimento.
- **Memoria.** Contiene toda la información del desarrollo del proyecto, así como un manual para poder reproducir los resultados del experimento en el caso de que así se requiera. La memoria es este documento.

1.5 Hipótesis y restricciones

1.5.1 Hipótesis

Al tratarse de un TFG de experimentación seguiremos el método científico. Una vez sabemos lo que queremos estudiar, debemos fijar las hipótesis de partida para las cuales realizaremos los distintos experimentos y finalmente veremos si estas hipótesis iniciales se cumplen y estamos ante un planteamiento acertado. Las hipótesis que planteamos son:

- Los modelos a los que se le realice un preprocesamiento de los datos funcionarán mejor que aquellos modelos que se entrenan con datos que contienen mucho ruido.
- Los modelos que hayan sido entrenados en un dominio más cercano al estudio (por ejemplo, modelos entrenados con datos de Twitter), funcionarán mejor que los modelos que han sido entrenados con datos más generales.
- Pensamos que los modelos que integran características contextuales (como los datos de la red social en la que se producen los comentarios, la persona a la que se le realiza el comentario y su género), obtendrán un rendimiento mayor que aquellos que no disponen de esta información, ya que con la información contextual, los modelos serán capaces de modelar mejor el lenguaje ofensivo.
- En cuanto a la estrategia utilizada para insertar los datos contextuales, tenemos dos arquitecturas planteadas, una que concatena estos datos al inicio de los textos para que los modelos detecten la información más importante y otra que una vez que se han obtenido los datos más relevantes de los textos para realizar la clasificación, utiliza las características contextuales para complementar la información más importante de los textos. Consideramos que la segunda estrategia será la que obtenga un mejor funcionamiento, pues los metadatos son aspectos fundamentales del ámbito en el que se producen los comentarios y así nos aseguramos de que el modelo tenga en cuenta esta información.
- Aquellos modelos que tengan mayor tamaño de muestras (*batch-size*) para realizar su entrenamiento funcionarán mejor que aquellos en los que, para entrenar el modelo, el número de muestras sea menor.
- Creemos que los modelos *Transformers* serán buenos detectores de la presencia del lenguaje ofensivo.
- Finalmente, pensamos que los modelos que se entrenan con más datos funcionan mejor.

1.5.2 Restricciones

En primer lugar, a la hora de plantear el desarrollo de este proyecto, se estimó el tiempo de trabajo que se le iba a dedicar al mismo. En ese planteamiento, se vio que teníamos claramente una restricción de tiempo, pues este ya parte de un límite aproximado de trescientas horas.

En segundo lugar, contamos con las limitaciones de los modelos utilizados, pues, estos se han entrenado con datos que están influenciados por los seres humanos. Estos datos, que ya están sesgados, se pasan al modelo como entrada y este aprende de ellos. Además, la percepción de la ofensividad de un mensaje es subjetiva y depende de la persona que lo interprete o de quién haga ese comentario.

Otra restricción encontrada a la hora de realizar este proyecto ha sido la memoria disponible a la hora de ejecutar los modelos, pues cuando ejecutamos los modelos en Google Colaboratory, el tamaño de la GPU cambia, obteniendo como máximo 16 GB de RAM. Esto nos limita al establecer los tamaños de *batch-size* (más adelante, veremos qué es esto) ya que, si el tamaño del modelo es muy grande, no se puede ejecutar el código al re-entrenar los modelos por errores de desbordamiento. Por otro lado, en el clúster de supercomputación ADA, teníamos una capacidad superior de RAM, ya que disponíamos de 192 GB.

Además de lo expuesto con anterioridad, contamos con la restricción del estilo de búsqueda de *Optuna (*herramienta que vamos a utilizar para la búsqueda de hiper-parámetros), pues al ir buscando en un rango de manera aleatoria, no sabemos si hemos obtenido el mejor valor de todo el rango establecido para los parámetros (máximo global) o solo los valores que dan un máximo, dentro de los valores aleatorios obtenidos (máximo local).

1.6 Estudio de alternativas y viabilidad

Inicialmente, a la hora de plantear los experimentos, tuvimos que hacer una búsqueda de los distintos modelos de lenguaje basados en la arquitectura *Transformer* que existen y seleccionar aquellos con los que experimentar. Existen una gran cantidad de modelos del lenguaje para trabajar, por lo que vamos a explicar los más conocidos:

- **Generative Pre-Trained Transformers 3 (GPT-3):** Es un modelo de lenguaje usado sobre todo para generar textos y creado por la empresa OpenAI [9]. Este modelo es capaz de reconocer patrones en los datos y aprender a través de los ejemplos proporcionados. Es capaz de desarrollar textos a través de un simple enunciado, mantener conversaciones, elaborar resúmenes… Este modelo fue entrenado con una gran cantidad de información. Los datos de entrenamiento venían de organizaciones que se dedicaban a obtener datos públicos de internet, *WebText2*, una colección de textos extraídos de Wikipedia y otros sitios webs, además de distintos libros. En total, su corpus de entrenamiento tiene más de 1000 millones de palabras. Su arquitectura basada en *Transformers*, consta de un codificador que toma como entrada una secuencia de palabras y produce una representación vectorial de cada una de ellas que luego pasa por un mecanismo de atención para predecir la siguiente palabra en la secuencia. El decodificador toma esta representación vectorial y produce una distribución de probabilidad de todas las palabras posibles dadas las entradas anteriores.
- **Bidirectional Encoder Representation from Transformers (BERT [2]):** es un modelo de lenguaje creado por Google, su objetivo es el de interpretar el lenguaje que usamos en el buscador de google de una forma más natural, mediante el uso de una red neuronal. Debido a que es bidireccional, analiza la frase en los dos sentidos, es decir, a la izquierda y a la derecha de la palabra clave, consiguiendo así entender el contexto y la temática de cada oración. A raíz de la creación de este modelo, surgieron muchos derivados, como veremos en los siguientes puntos. Para entrenar a este modelo, se usó un corpus masivo de texto. Una ventaja de

este modelo es que puede ampliarse para resolver tareas diferentes, como clasificación, resumen de textos, responder a preguntas, resolución de entidades… Además, a raíz de la creación de este modelo se han creado muchos más inspirados en su arquitectura, como por ejemplo BETO [12], una adaptación de BERT para el español.

- **Text-To-Text Transfer Transformers (T5 [7]):** este modelo propone una reforma de todas las tareas a realizar en el PLN en un formato de texto a texto unificado, donde la entrada al modelo y la salida sean siempre cadenas de texto, a diferencia de los modelos BERT o derivados que solo pueden generar una etiqueta de clase o intervalo de entrada. La estrategia de este modelo permite usar la misma función de pérdida, hiperparametros e incluso el modelo en sí en cualquier tarea del PLN (resumen de documentos, traducción automática de textos, tareas de clasificación…) Otra ventaja es que se puede adaptar a tareas de regresión, mediante un entrenamiento para predecir la representación de una cadena de un número en lugar de un número como tal. Este modelo se entrenó con una versión depurada de Common Crawl.
- **Robustly Optimized BERT Pretraining Approach (RoBERTa [3]):** se creó con la misma arquitectura de BERT, modificando ciertos hiper-parámetros claves, entre ellos, entrenar con *batch-size* y *learning-rate* más grandes y eliminando el objetivo de entrenamiento previo de la siguiente oración. Esto permite mejorar el objetivo del lenguaje enmascarado de BERT y que haya un mayor rendimiento en las diferentes tareas en comparación con BERT.
- **Extra Large Multilingual RoBERTa (XLMRoBERTa [8]):** tiene la misma implementación que RoBERTa, con la diferencia de que ha sido entrenado con un conjunto más grande de datos y datos de 100 lenguajes diferentes. No necesita que se le especifique un lenguaje concreto para saber de qué lenguaje se trata.
- **BERTIN [4]:** Este modelo surge de la necesidad de crear modelos del lenguaje con pocos recursos. Su proyecto trata de elaborar modelos basados en BERT para el español. Su estrategia se fundamenta en reducir

el pre-entrenamiento de modelos del lenguaje en la mitad del número de pasos y usando aproximadamente una quinta parte de los datos. Según diversos experimentos, los modelos creados alcanzan el estado del arte y en algunas tareas lo superan.

- **BART** [5]**:** es un modelo que tiene un codificador automático para eliminar el ruido de modelos secuencia a secuencia de pre-entrenamiento. Su funcionamiento se basa en romper el texto con una función de eliminación de ruido y aprendiendo a reconstruir el texto original. En su arquitectura, usa un codificador bidireccional como BERT y un decodificador de izquierda a derecha como GPT-3. Las tareas fundamentales para las que se usa este tipo de modelos son la comprensión, traducción y generación del lenguaje natural.
- **A Little BERT (ALBERT** [6]**):** tiene una arquitectura parecida a BERT, con la misma cantidad de capas ocultas. Lo diferente de este modelo es que usa dos técnicas para reducir parámetros del modelo BERT y así disminuir la memoria necesaria y aumentar la velocidad de entrenamiento. Estas técnicas se basan en la división de la matriz de embeddings en dos matrices más pequeñas y en la aplicación de capas repetitivas divididas entre diversos grupos.

Tras este resumen de los modelos más conocidos dentro del PLN, nosotros escogimos los que más se ajustan a tareas de clasificación y estaban adaptados al español, como son BETO y RoBERTa. Adicionalmente, usamos BERTIN para comprobar si funciona bien en la detección del lenguaje ofensivo.

Para obtener los modelos seleccionados, usamos Hugging Face [49], una plataforma que tiene más de sesenta mil modelos, seis mil conjuntos de datos y seis mil aplicaciones de demostración, todas estas de código abierto. Además proporciona métodos e información para crear tu propio modelo o re-ajustar los modelos que puedes obtener de esta y mediante foros permite la colaboración entre distintas personas, para que cualquiera pueda experimentar y explorar con todo lo que proporciona.

Una vez elegidos los modelos del lenguaje, se pasó a una búsqueda de hiper-parámetros de entrenamiento, para intentar buscar el modelo más óptimo y con un mejor funcionamiento. Encontramos tres alternativas viables para hacer esto:

- **Elaborar nuestro propio código**, que recorriera todas las posibles combinaciones de hiper-parámetros que se querían probar para cada modelo. Se descartó porque iba a ser muy costoso en cuanto a tiempo de ejecución y además había ya un software especializado en este tipo de trabajos.
- **Ray-Tune** [34]**:** Es una biblioteca de python, elaborada fundamentalmente para la ejecución de experimentos y el ajuste de hiper-parámetros. Su estrategia de búsqueda de hiper-parámetros se basa en algoritmos de entrenamiento basados en población (PBT) e hiperbanda. El entrenamiento basado en algoritmos PBT aprende de los valores de los hiper-parámetros y de los distintos pesos de la red. En él, hay varios procesos independientes que usan distintos hiper-parámetros y aquellos que dan peor resultado son sustituidos por otros modelos cuyos valores o pesos han sido ligeramente modificados de los mejores modelos obtenidos previamente. Por otro lado, el algoritmo de hiperbanda está diseñado para espacios de búsqueda grandes, basado en la detección temprana, comenzando por una búsqueda aleatoria de hiper-parámetros y que realiza una poda asíncrona de aquellos modelos con peor rendimiento, administrandole así más recursos a los modelos más prometedores.
- **Optuna** [22]**:** Framework de optimización de hiper-parámetros de código abierto para realizar una búsqueda de hiper-parámetros de forma automática. Tiene una serie de muestreadores que tratan de reducir el espacio de búsqueda mediante el uso de un registro de parámetros sugerido y los valores objetivo que se evalúan. Por defecto, usa un estimador con estructura de árbol, ajustando los hiper-parámetros con dos gaussianas distintas, una para los valores objetivo y otra para los valores de parámetros restantes, escogiendo el valor que maximiza el ratio de la

primera gaussiana entre la segunda. Lo que ha motivado el uso de este software, frente a Ray-Tune, ha sido ver en Internet la cantidad de investigaciones que lo usan y su fácil integración en Hugging Face para adaptar el código de entrenamiento a esta búsqueda de hiper-parámetros, además de que si se tienen las dos bibliotecas instaladas, Hugging Face por defecto usa Optuna.

1.7 Descripción de la solución propuesta

La solución que planteamos para este proyecto es utilizar técnicas de procesamiento de lenguaje natural para poder detectar el lenguaje ofensivo para el español. Utilizaremos modelos *Transformers* pues son los que están más relacionados con el estado de la cuestión. Dentro de los modelos seleccionados, tenemos el modelo XLMRoBERTa [45] por ser el que mejores resultados obtuvo de la competición en tareas de detección de la ofensividad y los modelos BERT [39][40], RoBERTa [41][42] y Bertin [43][44], de dos formas distintas, una con su forma base y otra con un ajuste al dominio de *Twitter,* pues el lenguaje ofensivo se suele producir en redes sociales, dónde el lenguaje utilizado es más informal, se cometen faltas de ortografía, se ponen emoticonos, URLs… Los modelos tienen unos parámetros que debemos definir a la hora de realizar un entrenamiento, y queremos optimizar estos parámetros para obtener los mejores resultados posibles. Más adelante, explicaremos un poco más sobre estos modelos y sus parámetros de entrenamiento.

Además queremos ver de qué manera podemos hacer uso de las características contextuales de los textos para así proporcionar más información a los modelos y ayudarlos a detectar el lenguaje soez.

Otra cosa que queremos analizar, es si realiza una limpieza previa de los datos puede ayudar a los modelos a clasificar, dentro de esta limpieza entraría sustituir las URLs y menciones a otros usuarios por una etiqueta, url y @user respectivamente y eliminar la almohadilla de los hashtags.

Para analizar toda esta información, realizaremos distintos experimentos basados en el entrenamiento de los modelos citados con anterioridad con el uso de distintas tecnologías explicadas en el punto anterior (Sección 1.6), almacenando los resultados obtenidos en una plataforma llamada MLflow[1], ya que nos permite ver los resultados de forma clara, permitiendo ordenar los valores según la métrica elegida, o realizar filtrados de datos, en el caso de que nos interese ver el resultado de un modelo o experimento concreto.

1.8 Material y métodos

En cuanto a los materiales utilizados en este proyecto, nos encontramos el conjunto de datos y los modelos entrenados.

El conjunto de datos es OffendES [11], está dividido en tres partes: unos datos que sirven para entrenamiento, otros de validación, usados para realizar la optimización de hiper parámetros o para entrenar si los juntamos con los de entrenamiento. La tercera sirve para evaluar el funcionamiento del modelo final. Esta colección, contiene comentarios en español que se realizaron en las redes sociales *Youtube, Instagram* y *Twitter*. Los comentarios que contiene están dirigidos a distintos influencers y se proporciona información del influencer al que se le hizo el comentario, el sexo de ese influencer, la red social en la que se realizó el comentario y luego una etiqueta que nos indica en qué clase estaría el comentario. Las clases en las que se podría clasificar un comentario son: OFP para comentarios ofensivos dirigidos a una persona, OFG para comentarios ofensivos dirigidos a un grupo o colectivo, NO para comentarios que son no ofensivos y NOM para comentarios que son no ofensivos pero que contienen palabras malsonantes. El motivo de elección de este conjunto de datos fue por la cantidad de metadatos que nos proporcionaba, ya que nos permitía conocer un poco más el contexto en el que se producía la ofensividad.

[1] **MLflow:** https://mlflow.org/

En el apartado 1.6 podemos ver los tipos de modelos que hay y una pequeña justificación de cuáles van a ser utilizados, por lo que aquí detallaremos de una forma un poco más concreta, de todos los disponibles en el repositorio de Hugging Face, cuáles son los utilizados en este trabajo. Los modelos seleccionados son:

- **BETO Twitter** [39] es un modelo BETO pre-entrenado con datos de *Twitter* que pueden contener hashtags y menciones a otros usuarios.
- **BETO Normal** [40] es un modelo BERT entrenado con datos en español. Es uno de los modelos más usados para tareas de PLN en español. El modelo fue pre-entrenado con un gran corpus de datos en español con datos de *Wikipedia* en español, de la DGT (Dirección General de Tráfico), entre otras muchas fuentes.
- **RoBERTa Normal** [41] pertenece al proyecto MarIA impulsado por la Secretaría de Estado de Digitalización e Inteligencia Artificial en el marco del Plan de Tecnología del Lenguaje. El objetivo de este proyecto es generar recursos para el español. Está entrenado con textos muy variados del español, obtenidos de los fondos documentales digitales de la BNE.
- **RoBERTa Twitter** [42] es un modelo pre-entrenado con datos de Twitter que tiene como base un modelo RoBERTa. Su tokenizador ya está adaptado para hacer una limpieza de texto de esta red social, teniendo en cuenta cuando hay referencias a usuarios y uso de hashtags.
- **RoBERTa Cardiff** [45] tiene como base un modelo XLMRoBERTa, entrenado en una gran cantidad de lenguajes, sin necesidad de especificar para qué lenguaje va a ser utilizado este modelo. Lo hemos seleccionado por ser el mejor modelo de la competición MeOffendEs. Los datos de pre-entrenamiento de este modelo provienen de *Twitter*.
- **BERTIN Normal** [43] es un nuevo modelo base, entrenado con datos en español de la colección multilingüe Common Crawl. Este conjunto de datos es conocido como *mC4*.
- **BERTIN Twitter** [44] utiliza como base el modelo BERTIN anterior, pero pre-entrenado en datos de *Twitter*.

En cuanto a la metodología a utilizar, usamos lo más nuevo del PLN que son los modelos Transformers y para los cuales Hugging Face nos proporciona gran cantidad de modelos a elegir y métodos para saber cómo realizar los entrenamientos y demás pasos de procesamiento. Todo estos métodos se explicarán en los apartados siguientes.

1.9 Tecnologías utilizadas

Para la realización de este proyecto, se optó por el uso de Python como lenguaje de programación, pues es uno de los lenguajes más populares y más utilizados para la realización de tareas dentro del PLN. Además, Python nos proporciona multitud de bibliotecas muy útiles para este campo, entre las que podemos encontrar las siguientes, usadas en el desarrollo de este proyecto:

- **Numpy** [17]**:** biblioteca muy utilizada en computación científica, pues proporciona un objeto de matriz multi-dimensional y otros derivados de este, además de una gran variedad de funciones para realizar operaciones con estos objetos. Necesaria para trabajar con las matrices de forma más sencilla.
- **Pandas** [13]**:** proporciona herramientas para la manipulación de tablas de datos de manera rápida y eficiente. Además, nos facilita herramientas para leer y escribir archivos en distintos formatos. Necesaria para leer y almacenar datos durante este trabajo.
- **Matplotlib** [14]**:** nos permite crear una gran variedad de gráficos de alta calidad, de una forma bastante sencilla y con la posibilidad de modificar el estilo visual y diseño de estos, además de permitir la exportación de los gráficos realizados a muchos formatos de archivo. Será la biblioteca utilizada para analizar los datos iniciales y los resultados de los experimentos.

- **WordCloud** [23]**:** proporciona herramientas para visualizar las palabras más frecuentes en un texto, donde las palabras más frecuentes tienen un tamaño más grande. Usada para hacer un análisis inicial de los datos con los que trabajaremos.
- **TextFlow** [33]**:** una biblioteca que se dedica a la extracción de distintas características de los textos, como volumetría, extracción de lexemas, análisis de la polaridad o la ironía de los comentarios…
- **NLTK** [15]**:** una de las plataformas más usada para trabajar con datos relacionados con el lenguaje humano, proporciona gran cantidad de corpus, recursos léxicos y un conjunto de bibliotecas de procesamiento de texto para distintas tareas, como por ejemplo, clasificación, lematización, tokenización, razonamiento semántico… En este caso la hemos usado para obtener palabras que no tienen carga semántica en el español, conocidas como “palabras vacías”.
- **Sklearn** [16]**:** biblioteca que proporciona herramientas simples para hacer análisis predictivo de datos. En nuestro proyecto será muy útil para el cálculo de las distintas métricas de evaluación de los modelos implementados.
- **Optuna** [22]**:** herramienta de código abierto que te permite realizar una optimización de hiper-parámetros, de forma automática. Gracias a esta biblioteca, realizaremos una optimización de hiper-parámetros de nuestros modelos.
- **Datasets** [19]**:** biblioteca que te permite acceder y descargar conjuntos de datos y algunas métricas de evaluación para tareas relacionadas con PLN, visión por ordenador y pistas de audio. Actualmente contiene más de 2658 datasets y más de 34 métricas de evaluación. De aquí obtendremos nuestro dataset para realizar los distintos experimentos.
- **Transformers** [46]**:** proporciona una Interfaz de Programación de Aplicaciones (API) facilitando la descarga y el entrenamiento de una gran variedad de modelos. Puedes realizar un pre-entrenamiento del modelo elegido para adaptarlo a tareas de distintas modalidades, entre las que

podemos encontrar la clasificación del texto. Además *Transformers* establece el estado del arte actual en tareas del PLN.

Una vez presentadas las bibliotecas necesarias para la realización de este trabajo, vamos a explicar un poco qué servicios web y plataformas de computación hemos usado y por qué los hemos elegido:

- **Google Colaboratory** [20]**:** herramienta que te permite programar y ejecutar Python desde el navegador, sin tener que configurar nada y con acceso gratuito a GPUs, las cuales son muy útiles para realizar un entrenamiento de modelos más rápido que mediante el uso de CPU. El acceso a la GPU tiene limitaciones de tiempo, permitiendo un uso continuado de 8 horas como máximo y no siendo posible su uso entre las siguientes 12 y 24 horas.
- **Clúster Ada** [47]**:** un sistema de computación de altas prestaciones ofrecido por el CEATIC de la Universidad de Jaén a sus investigadores. Cuenta con procesadores Intel Xeon Silver 4208 y tarjetas gráficas NVIDIA Tesla V100 y NVIDIA GeForce RTX 2080Ti. Necesario para realizar la optimización de hiper-parámetros, pues con las limitaciones de Google Colaboratory, esta optimización no llegaba a completarse.
- **MLflow** [18]**:** plataforma de código abierto que te permite administrar de una forma más sencilla, el ciclo de vida del machine learning (ML). Actualmente te permite administrar experimentos, la reproducibilidad, implementación y registro de modelos. En este trabajo, la hemos usado para gestionar el resultado de los distintos experimentos.
- **Ngrok** [21]**:** es un servicio que nos permite exponer a Internet una URL generada de forma dinámica, la cual apunta a un servicio Web que se está ejecutando en local en nuestra máquina. Muy útil para acceder a MLflow desde Google Colaboratory.

De todas estas tecnologías hay dos que consideramos fundamentales para el desarrollo de este proyecto, por lo que le vamos a dedicar dos subapartados para explicarlas con más detalle.

1.9.1 Optuna

Para que se comprenda mejor el funcionamiento de Optuna, es necesario hablar de la arquitectura que tiene para optimizar los hiper-parámetros de los modelos en la fase de entrenamiento. Podríamos decir que sigue un esquema como el que vemos en la Ilustración 1.4, en el que los datos se dividen en dos particiones, la de entrenamiento y la de validación. Los datos de entrenamiento permiten el aprendizaje de un modelo. Los datos de validación se usan para evaluar el modelo y, con los resultados obtenidos, optar por un reajuste de los hiper-parámetros para pasar a un nuevo ciclo de entrenamiento y evaluación.

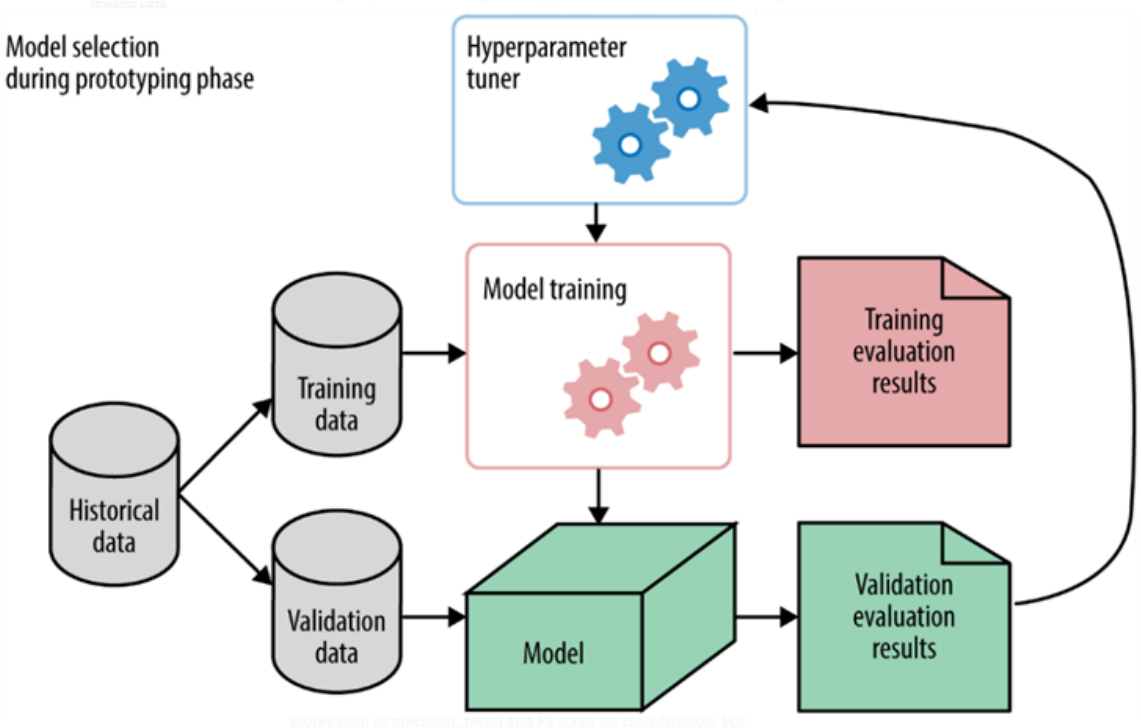

Ilustración 1.4 Funcionamiento Optuna [55]

Si entramos en detalles más técnicos, mirando la imagen inferior (Ilustración 1.5), Optuna cuenta con una función objetivo a optimizar, buscando que el valor resultante de esta se maximice o minimice (según el objetivo que establezcamos). Para ello, Optuna cuenta con una API que aplica el algoritmo deseado entre *suggest* o *pruning*. La diferencia entre ambos algoritmos radica en que, con el primero, se elige el valor de los parámetros en un rango determinado y, en el segundo, con el método *should_prune(),* se establece si se debe realizar una poda o no cuando se prueban distintos valores para los hiper-parámetros, pues optuna trabaja con árboles a la hora de asignar valores a los parámetros de entrenamiento. Optuna, tiene distintos *workers*, para que apliquen el algoritmo utilizado para optimizar hiper-parámetros. Cada trabajador va actualizando los valores almacenados de forma simultánea. Además, esta API cuenta con funciones que nos generan informes sobre los cambios en los valores a optimizar o que nos devuelven los valores finales. Todo esto se registra y se obtiene del almacenamiento.

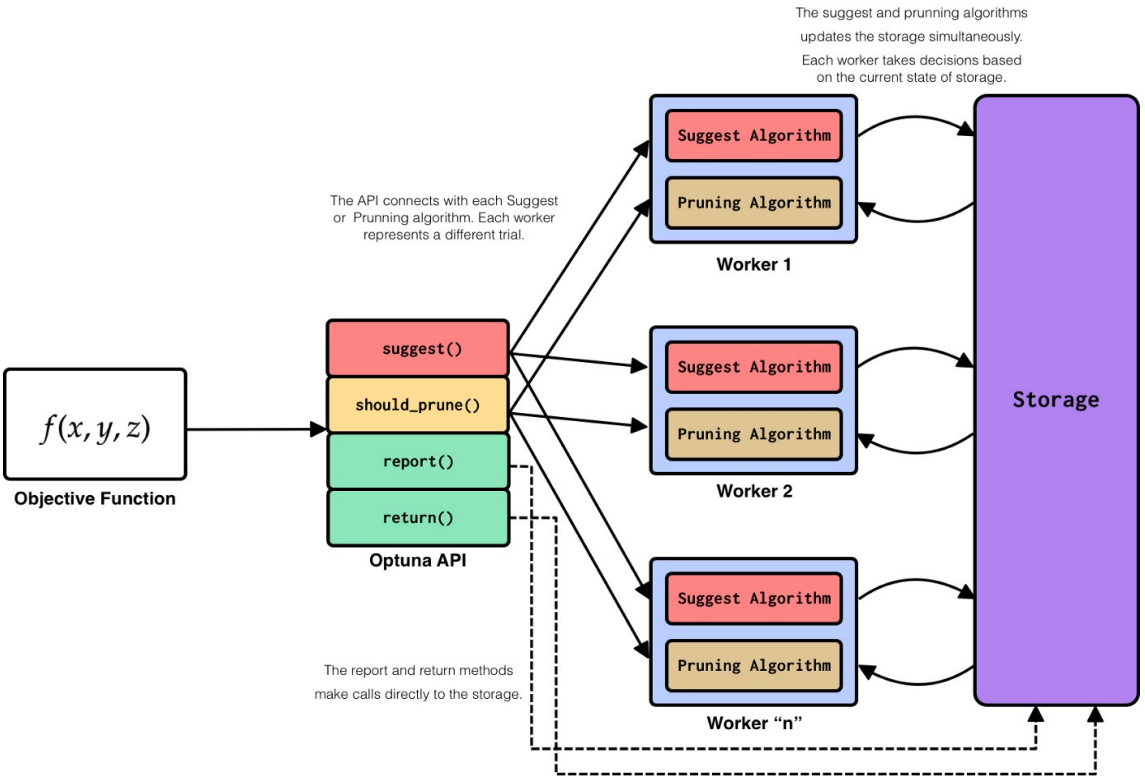

Ilustración 1.5 Arquitectura Optuna [54]

1.9.2 Transformers

En este proyecto, Transformers se ha utilizado por los modelos que nos proporciona y los métodos que implementa para entrenar y ajustar los distintos modelos, un aspecto base para la elaboración de este trabajo de investigación. Tal y como se vio en la Sección 1.6 tenemos distintos tipos de modelos, cada uno con una arquitectura diferente. Para entender cómo funcionan los modelos *Transformers*, es necesario explicar las redes neuronales sobre las que se fundamentan estos modelos para, a continuación, detallar el funcionamiento de los modelos *Transformers* en general, y más concretamente de cada modelo elegido.

1.9.2.1 Funcionamiento de una red neuronal

Una red neuronal está formada por diferentes capas, las cuales contienen una o más neuronas. Tenemos tres tipos de capas, en inglés conocidas como *layers*:

- ***Input layer* o capa de entrada:** esta capa se encarga de representar los datos con los que la red neuronal será alimentada.
- ***Hidden layer* o capa oculta:** puede haber varias capas de este tipo dentro de una red neuronal. En esta capa, todas las neuronas están conectadas con las neuronas de la siguiente capa.
- ***Output layer* o capa de salida:** es la capa que se encarga de proporcionarnos los resultados. Si se encarga de resolver tareas de clasificación, el número de neuronas será igual al número de clases que contienen los datos (cuatro, en nuestro caso).

En la imagen inferior (Ilustración 1.6), podemos ver una red neuronal con sus distintas capas, donde los círculos representan a las neuronas, las de color verde serían las neuronas de la capa de entrada, las naranjas de la capa oculta y las azules de la capa de salida.

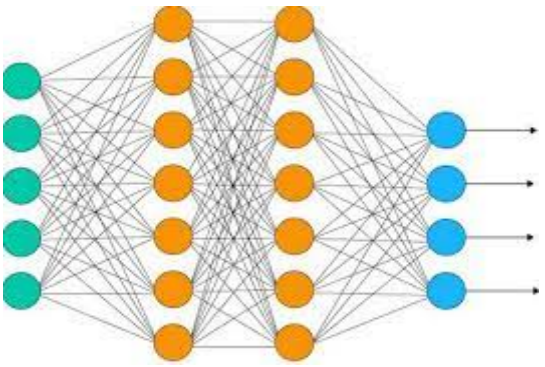

Ilustración 1.6 Red Neuronal [57]

En una red neuronal, cada neurona tiene una función con un peso que depende de cada neurona, y un parámetro adicional que es igual entre las neuronas de la misma capa. El resultado se envía a una función de activación, y el valor obtenido se envía a la siguiente capa.

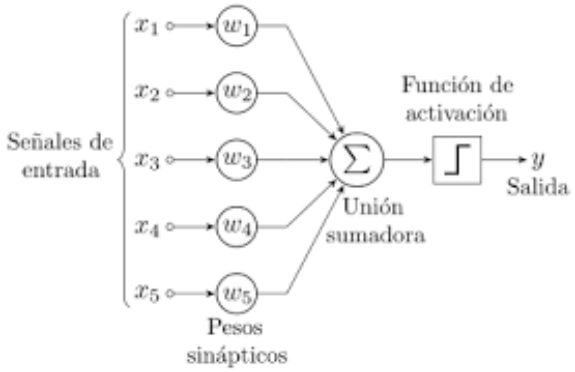

Ilustración 1.7 Funcionamiento de una neurona [58]

En la imagen superior (Ilustración 1.7), se puede ver el funcionamiento de una neurona que recibe datos de entrada o de salida de otras neuronas de capas anteriores con un peso establecido, al que le aplica su función activación antes de obtener el valor de salida.

1.9.2.2 Funcionamiento Modelos Transformers

La arquitectura de un modelo Transformers tiene dos componentes clave, un codificador y un decodificador. La tarea del codificador es recibir una entrada y construir una representación de esta, lo que quiere decir que el modelo adquiere conocimiento a partir de esta entrada. En cuanto al decodificador, obtiene la salida del codificador junto con otras entradas y genera una secuencia objetiva. Así, el modelo es capaz de generar resultados. Según la tarea que queramos realizar, podemos usar el codificador, decodificador o ambos. Por ejemplo, para tareas que requieran el conocimiento de la entrada, como clasificación de entidades o de texto, es bueno usar modelos de solo codificador. En tareas de generación de texto, lo mejor es usar modelos de solo decodificador, y para tareas generativas que necesiten una entrada como puede ser la generación de resúmenes o de traducción, sería bueno usar ambos.

Los modelos Transformers hacen uso de un aspecto clave, y es que estos modelos se construyen sobre capas de atención, que son las encargadas de que el modelo aprenda que ciertos tokens deben atender específicamente a otros tokens en la oración para tratar de representarla. Esto se debe a que las palabras tienen un significado que está afectado por el contexto que les rodea.

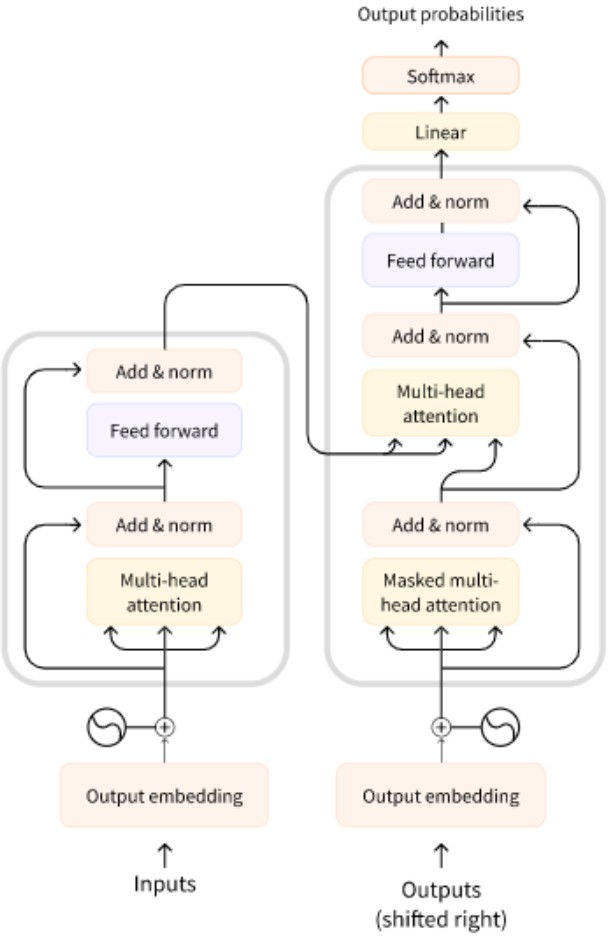

Ilustración 1.8 Arquitectura Modelo Transformers [56]

En la Ilustración 1.8 podemos ver una arquitectura de un modelo transformers con el codificador a la izquierda y el decodificador a la derecha. Esta arquitectura fue diseñada para la traducción, de forma que el codificador recibía las entradas en un idioma y el decodificador las mismas frases en el idioma al que queríamos traducir dichas frases. En el codificador, las capas de atención podían usar todas las palabras de una oración. Sin embargo, el decodificador solo puede prestar atención a aquellas palabras de las que ya ha obtenido una traducción, es decir, a las palabras anteriores a las palabra que se está traduciendo ahora.

Un aspecto a tener en cuenta es que la primera capa de atención del decodificador presta atención a todas las entradas que le pasamos, pero la segunda capa usa la salida del codificador. Por tanto, puede acceder a la oración de entrada completa y así predecir mejor la palabra actual.

Otro aspecto fundamental de este modelo que debemos saber, es que aunque aparentemente, nosotros le pasemos el texto tal cual, las máquinas no pueden trabajar con este, ya que no entienden nuestro lenguaje. Lo que hacen estos modelos es usar un tokenizador que convierte el texto en una representación adecuada para que así el ordenador pueda entender lo que le estamos pasando. El objetivo de los tokenizadores es encontrar la representación más idónea de los textos para los distintos modelos y a ser posible, la más pequeña para que ocupe menos espacio. Hay diferentes tipos de tokenizadores, por lo que en las siguientes subsecciones veremos cuál usa cada modelo.

1.9.2.2.1 Funcionamiento BERT

Los modelos Beto Normal y Beto Twitter, utilizados en este trabajo son modelos basados en una arquitectura BERT, la única diferencia entre un BERT y BETO, es que BETO es un modelo BERT entrenado en español. El objetivo fundamental de BERT es codificar texto, para obtener una representación numérica que permita su correcta interpretación. Como solo queremos codificar el texto, el modelo BERT tendrá sólo la parte de codificación que vimos en la Ilustración 1.9 de la arquitectura *Transformers*. Concretamente, un modelo BERT básico tiene 12 codificadores.

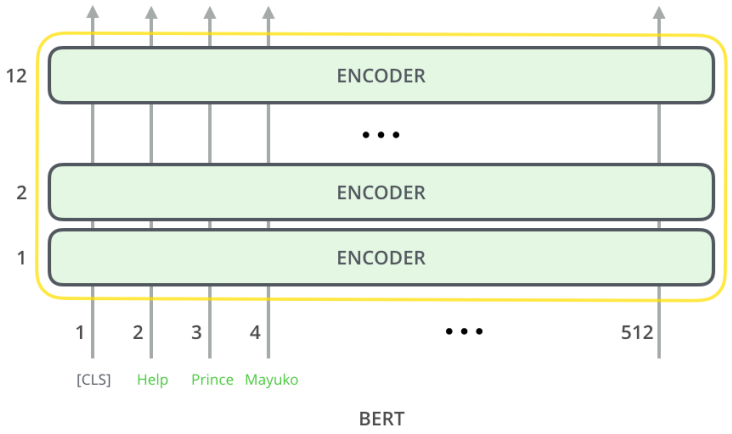

Ilustración 1.9 Imagen del modelo BERT [60]

Como hemos dicho antes, necesitamos establecer una representación del texto que la red pueda entender y procesar, es por esto que usamos un tokenizador, que convierte un texto en una secuencia de tokens. El tokenizador que utiliza BERT está basado en el algoritmo *Word-Piece*, este algoritmo inicializa el vocabulario para incluir todos los caracteres que tiene el conjunto de datos de entrenamiento y va progresivamente aprendiendo una gran cantidad de reglas de combinación. A diferencia de otros tokenizadores, no se elige el par más frecuente, sino que busca maximizar la probabilidad del conjunto de entrenamiento una vez añadido al vocabulario. Para maximizar la probabilidad de los datos de entrenamiento, lo que hace es encontrar el par de símbolos cuya probabilidad dividida por la probabilidad del primer símbolo seguido del segundo es la más alta de todos los pares de símbolos.

Este tokenizador tiene dos fases. En la primera, tokeniza el texto e incluye un token de inicio [CLS] al comienzo de este y un token llamado [SEP] al final del texto de entrada o para separar distintas entradas de texto, en el caso de que la tarea a realizar necesite de dos o más entradas. Una vez que hemos obtenido los tokens del texto de entrada, se generan tres representaciones (ver Ilustración 1.10), las cuales son:

- Un *embedding*, que representa a cada token como un vector.

- Otro *embedding* con la codificación de la palabra según su posición en el texto. Esta posición es importante tenerla en cuenta, para que el codificador tenga en cuenta la posición relativa de cada token, pues luego la red preprocesa las palabras de forma simultánea.
- Además, añadirá un nuevo *embedding* que indicará a qué segmento pertenece la frase. Cada segmento está separado por un token *SEP* y se codifica de forma distinta.
- Estas tres representaciones se suman y obtenemos los vectores de entrada a la red neuronal

Ilustración 1.10 Imagen del funcionamiento tokenizador BERT [59]

BERT se pre-entrena usando colecciones de textos muy grandes y aprende a analizar el texto de forma bidireccional, y a codificar cada palabra según su contexto. Esto lo hace para identificar de forma precisa cada palabra, ya que una misma palabra puede tener significados distintos según su contexto.

Hay un banco de peces en el fondo del mar.
Fui al banco a sacar dinero.

Ilustración 1.11 Bidireccionalidad de BERT

El pre-entrenamiento de BERT se hace sobre dos tareas *semi-supervisadas*, es decir, sobre tareas sobre textos sin que éstos tengan por qué estar anotados. Una de las tareas consiste en entrenar el modelo para que complete una frase a la que le falta una palabra. Esta palabra puede estar en cualquier posición en la frase. De esta manera, se fuerza al modelo a que aprenda el lenguaje de forma bidireccional

(Ilustración 1.11). La segunda tarea consiste en intentar que el modelo diga si una frase puede ir detrás de otra.

Una vez que ya hemos realizado el pre-entrenamiento, lo siguiente será adaptar cada modelo a la tarea objetivo (clasificación textos, anotación de palabras, generación de respuestas…), para eso agregamos al modelo una red neuronal y una capa softmax (Ilustración 1.12). Una vez añadidas estas capas, será necesario volver a entrenar el modelo con los textos específicos para la tarea a realizar. Este nuevo entrenamiento será más corto que el anterior, pues el modelo ya ha aprendido del lenguaje en el primer entrenamiento.

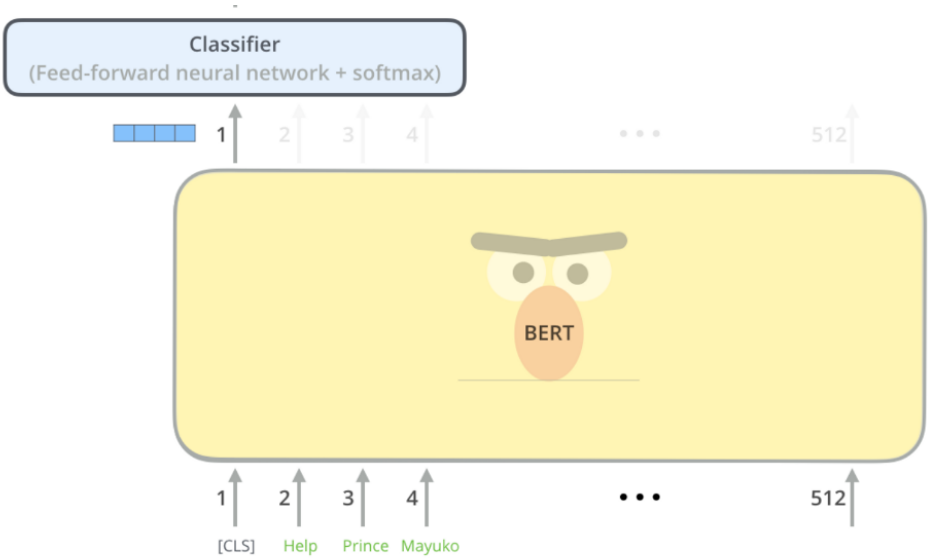

Ilustración 1.12 Imagen fine-tuning BERT [60]

En nuestro caso, como tenemos una tarea de clasificación, solo necesitamos analizar el token de clasificación a la salida del modelo, pues BERT, al ser bidireccional, concentra la información necesaria para la clasificación en un único token.

Añadir que, en nuestro caso, el pre-entrenamiento ya estaba realizado y solo necesitábamos hacer un ajuste del modelo a la tarea de clasificación de textos según la ofensividad detectada.

1.9.2.2.2 Funcionamiento RoBERTa

Para los modelos elegidos en este proyecto, tenemos RoBERTa Normal, RoBERTa Twitter y RoBERTa Cardiff, que siguen esta arquitectura. RoBERTa se basa en un modelo BERT y difiere en cuatro aspectos: entrena el modelo durante más tiempo y con lotes más grandes y más datos, elimina el objetivo de predicción de la siguiente frase, entrena en secuencias más largas y cambia de forma dinámica el patrón de enmascaramiento de los datos. Esto le permite obtener mejores resultados que BERT para la predicción del lenguaje enmascarado, es decir, predecir palabras ubicadas en cualquier parte de la frase. Además, como entrenamiento usan otro conjunto de datos diferente al usado para entrenar el BERT original.

En cuanto a la generación de patrones dinámicos en el pre-entrenamiento de este modelo, RoBERTa cambia la palabra enmascarada para cada frase cada vez que le proporcionan una secuencia al modelo. Al eliminar como objetivo la predicción de la siguiente frase, esto no tiene en cuenta en la función de pérdida del modelo.

En cuanto al tokenizador, RoBERTa usa un tokenizador de tipo Byte-Pair Encoder (BPE), este algoritmo sustituye el par más común de símbolos consecutivos y lo sustituye por un byte que no está presente en los datos iniciales. De esta forma, este algoritmo trata de representar las palabras más frecuentes como un único token, mientras que las palabras más raras se dividen en dos o más tokens, A diferencia de wordPiece, Byte-Pair sí coje el par más frecuente.

1.9.2.2.3 Funcionamiento BERTIN

BERTIN es un modelo que surge de la necesidad de generar modelos para el español, a partir de pocos recursos computacionales. Se basa en un modelo RoBERTa. Para realizar este tipo de modelo, basaron su estrategia en lo que ellos llaman “muestreo de perplejidad”, que usa los datos justos para pre-entrenar un modelo, reduciendo este conjunto inicial de datos de pre-entrenamiento a un quinto del necesario para pre-entrenar el modelo base (RoBERTa) y usando al menos la mitad de pasos para dicho entrenamiento. Este modelo tiene, por tanto, la misma arquitectura que RoBERTa y usa su mismo tokenizador, ya que solo difiere en el volumen de datos y la calidad de estos y el reducir el número de pasos de entrenamiento, pero no la arquitectura.

1.10 Metodología de desarrollo de software

Al empezar un proyecto de esta envergadura, es necesario realizar un trabajo de estructuración, planificación y control de desarrollo del mismo. Es por esto que inicialmente debemos estudiar y seleccionar una metodología a seguir para su desarrollo.

En nuestro caso, desde el inicio se evaluaron distintas opciones y finalmente se optó por una metodología de desarrollo de software iterativa. Se caracteriza por dividir el proyecto en pequeños bloques temporales, denominados iteraciones. En cada iteración se realiza un trabajo determinado y al terminar este, se obtienen los resultados y se comprueba si estos son correctos o cumplen con los objetivos. De esta forma, si al final de una iteración algo no va bien, se pueden plantear medidas correctivas o soluciones alternativas.

Hemos dividido el proyecto en distintas fases y en el cambio de cada fase se han mantenido reuniones para ver si el proyecto iba bien y los resultados eran los esperados. En el caso de que se viese algún valor inesperado se revisaba todo por si debíamos de mantenernos en la misma fase y arreglar el error.

Las ventajas de usar esta metodología son:

- **Control:** como el proyecto está dividido en pequeñas partes independientes, es fácil saber en qué estado se encuentra el proyecto.
- **Disminución de la propagación de errores:** al finalizar cada fase, se comprueba si hay algún fallo, gracias a esto conseguimos pasar de etapas solucionando los errores que se hayan detectado al finalizar la fase actual.
- **Gestión de cambios:** ya que si en algún momento nuestro proyecto debe ser modificado, podemos gestionar de forma más sencilla los cambios pertinentes.
- **Comunicación:** genera una interacción constante entre los desarrolladores y los clientes, en este caso, entre el alumno y los tutories, debido a que al final de cada iteración se realizan reuniones para que ambas partes reciban una retroalimentación del estado del proyecto. Además de que permite integrar al cliente en el desarrollo.
- **Aumento de la productividad y optimización del proceso de desarrollo:** como los desarrolladores saben en qué fase se encuentran y lo que deben hacer para finalizarla, son más eficientes. Además como antes de cada iteración se corrigen los errores que se detectan, conseguimos que al final del desarrollo no se acumulen los errores y haya que cambiar aspectos básicos de la estructura del proyecto, por lo que estamos optimizando la implementación del producto.

1.11 Estimación del tamaño y esfuerzo

Como este proyecto es un TFG, no tenemos restricciones económicas, sino temporales. Sabemos que como máximo, este trabajo debe tener una duración de 300 horas aproximadamente y que va a estar desarrollado por una sola persona, por lo que el esfuerzo necesario para desarrollar este proyecto será trescientas horas por persona.

Para estimar el tamaño que tendrá el desarrollo de este trabajo, usaremos una técnica conocida como Líneas de Código (LOC). Esta técnica se basa en medir de forma directa el tamaño de software a implementar y para calcularlo es tan simple como contar las instrucciones de código fuente que necesita cada componente del producto final, sin contar los comentarios que el código pueda tener ni las líneas que se queden en blanco entre funciones. Como estamos al inicio del proyecto, no sabemos cuántas líneas de código vamos a tener exactamente, por lo que debemos realizar una estimación de cuanto creemos que nos va a ocupar el software que necesario. Para calcular esta estimación, nos basamos en la fórmula que aparece en la imagen de abajo (Ilustración 1.13), dónde a es el número LOC óptimo, m es el valor probable de este y b el valor pesimista.

$$E = \frac{a + 4 \cdot m + b}{6}$$

Ilustración 1.13 Fórmula para el cálculo LOC

A continuación, tenemos un cuadro con los valores optimistas, pesimistas y más probables de líneas de código, para cada funcionalidad que queremos tener en nuestro proyecto. Además de una columna con la estimación final.

Función	Optimista	Más Probable	Pesimista	Estimación
Análisis de los datos iniciales	500	600	850	625
Optimización Hiper-parámetros	100	125	150	125
Realización de los experimentos planteados	125	200	300	205
Total				955

Tabla 1.1 Estimaciones LOC para el proyecto actual

1.12 Planificación temporal

Al inicio del proyecto, se planteó una planificación de las distintas fases del proyecto en el tiempo, para así, poder saber sin mucho esfuerzo en que fase se encuentra nuestro proyecto. Además, podemos identificar todas las tareas necesarias para su desarrollo, qué ocurre si una sufre un retraso y si se retrasa, cómo solucionarlo para llegar a la fecha deseada de entrega o finalizarlo lo antes posible.

Es importante remarcar que cada día vamos a dedicarle 5 horas y media, ya que tenemos 300 horas a repartir en 55 días laborales.

Pasemos a ver una tabla (Tabla 1.2) que resume cada fase de forma más visual:

Fase	Fecha de Inicio	Objetivo
Fase 1	6/06/2022	Investigar acerca de los distintos modelos Transformers existentes, y sobre cómo entrenar u optimizar hiper-parámetros en los distintos modelos.
Fase 2	15/06/2022	Realizamos un análisis inicial de los datos que tenemos. Esto es muy importante, pues de aquí podremos obtener información muy importante para continuar en el proyecto.
Fase 3	24/06/2022	Planificar los distintos experimentos que se van a realizar, es decir, que modelos y con qué datos

Fase 4	29/06/2022	Optimizar los hiper-parámetros de los distintos modelos a utilizar.
Fase 5	20/07/2022	Desarrollo de los distintos experimentos planificados. En esta fase, entrenaremos los modelos con los mejores hiper-parámetros, seleccionados de la fase anterior.
Fase 6	17/08/2022	Análisis de los resultados obtenidos y elaboración de las conclusiones.
Fase 7	22/08/2022	Elaboración de la memoria.

Tabla 1.2 Planificación temporal del proyecto

En la imagen inferior (Ilustración 1.14), podemos visualizar un diagrama de Gantt, donde se visualiza de forma más clara, cuando empezaremos una fase y cuando la terminaremos:

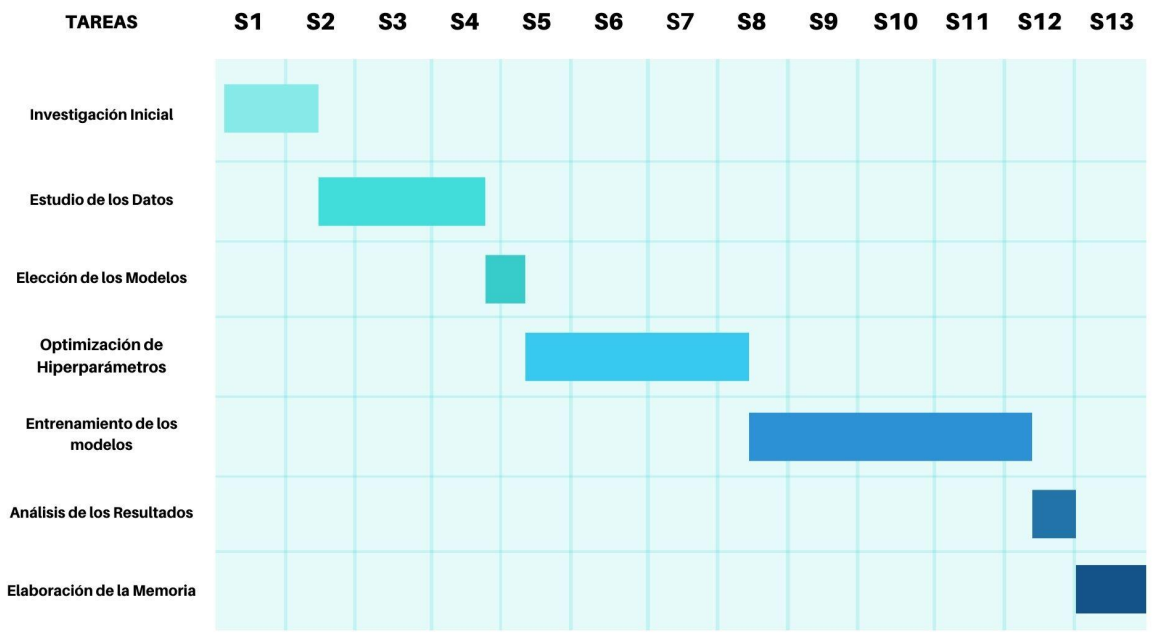

Ilustración 1.14 Diagrama de Gantt

1.13 Presupuesto

En este apartado se presenta una estimación del coste del proyecto, sabiendo las tecnologías que se van a usar, las tareas a realizar, el personal necesario para desarrollar dichas tareas y los servicios necesarios a contratar para su desarrollo. Además, para el cálculo total del coste, se ha tenido en cuenta el tiempo de duración del proyecto que son 55 días laborales, lo que ocupa en total, aproximadamente, dos meses y medio. Veamos una tabla (Tabla 1.3) con el coste del proyecto de una forma más general:

Tipo de Presupuesto	Coste
Presupuesto Software	84,63 €
Presupuesto Hardware	1.094,15 €
Presupuesto de Personal e Instalaciones	23.470,1 €
Total	**24.648,88 €**

Tabla 1.3 Presupuesto Total del proyecto

A continuación, desglosaremos este apartado en tres subapartados, para que se visualice de forma más clara, el coste de los productos software, hardware y de contratación del personal e instalaciones necesarios para el desarrollo de este trabajo.

1.13.1 Presupuesto software

Debemos tener en cuenta el coste que suponen aquellos productos o herramientas software que se utilizan en el desarrollo de este proyecto.

Como podemos ver a lo largo de todo este documento, todo el software utilizado es gratuito, a excepción del sistema operativo utilizado. Al tratarse de un bien amortizable, debemos calcular su amortización. Para ello, dividiremos el coste entre el tiempo de vida útil en meses. Veamos la Tabla 1.4 para visualizar los gastos de manera más detallada:

Software	Función	Coste	Tiempo de Vida Útil	Coste Final
Microsoft Windows 10 Home	Sistema Operativo del equipo utilizado para el desarrollo del proyecto.	135,39€	4 años	33,85€/mes
Total				84,63 €

Tabla 1.4 Presupuesto Software

1.13.2 Presupuesto hardware

En este apartado, explicaremos de una manera detallada el coste de los distintos productos hardware que necesitamos para este trabajo (Tabla 1.5), con su correspondiente amortización y una suma total del coste hardware:

Hardware	Función	Coste	Tiempo de Vida Útil	Coste Final
Portátil HP Omen 15-dc0xxx	Ordenador utilizado para el desarrollo del proyecto. Al ser un ordenador portátil, ya	949,99€	4 años	19,80€/mes

	lleva incluido el precio del teclado, ratón, monitor, placa base, memoria RAM Y ROM...			
Clúster ADA	Se ha usado el nodo TESLA Volta del clúster ADA	20057,23€*	4 años	417,86€/mes
Total				1094,15 €

Tabla 1.5 Presupuesto Hardware

*El coste de ese nodo del clúster se ha hecho de manera aproximada, buscando cuánto cuesta cada componente que dicho nodo tiene y haciendo una suma total del coste de todos los componentes. Las especificaciones de cada nodo del clúster, se pueden ver en el siguiente enlace: https://www.ujaen.es/centros/ceatic/servicios/supercomputacion/cluster-ada

1.13.3 Presupuesto de personal a contratar e instalaciones

Para la elaboración de este trabajo, hay que tener en cuenta el personal que lo va a desarrollar y en qué trabajos va a estar ese personal, pues depende de la tarea que realice tendrá un sueldo u otro. Es importante incluir aquí el gasto que se hace en las instalaciones en las que vamos a desarrollar el trabajo.

A continuación se muestran las tablas que detallan todos estos costes (Tablas 1.6 y 1.7):

Personal	Función	Coste	Coste Final
Analista de Datos	Persona encargada de analizar y procesar	30.000 €/año	2.500 €/mes

	los datos con los que se trabaja en este proyecto. Además de generar las gráficas y los informes de estos datos.		
Desarrollador de software	Persona con experiencia en el desarrollo del sistema, capaz de tomar decisiones de diseño, herramientas y plataformas a utilizar, siguiendo los estándares técnicos y de codificación establecidos.	32.500 €/año	2.708,34 €/mes
Jefe de Proyecto	Persona que dirigirá y gestionará el proyecto.	48,971 €/año	4.080,92 €/mes
Total			23.223,15€

Tabla 1.6 Presupuesto del Personal

Instalaciones	Función	Coste	Coste Final
Luz	Necesaria para poder usar el hardware requerido para este trabajo.	54,78 €/mes	54,78 €/mes
Internet	Tarifa mensual de conexión a Internet, con el que	44 €/mes	44 €/mes

	se ha estado trabajando. Tanto para investigar, como para desarrollar el proyecto.		
Total			246,95 €

Tabla 1.7 Presupuesto Servicios

2 Diseño inicial

2.1 Análisis y diseño del sistema

Para entender el proyecto, es necesario saber cómo se diseñó este. Y para ello es necesario conocer cuál es el diseño inicial del sistema, las integraciones de las distintas herramientas y tecnologías utilizadas, y pequeñas modificaciones que se le hacen a los modelos para realizar los distintos experimentos.

2.1.1 Arquitectura Base

Si miramos la imagen inferior (Ilustración 2.1), veremos el diseño básico para ajustar un modelo pre-entrenado. Inicialmente partimos de ese modelo que ya ha sido previamente entrenado, a continuación con unos hiper-parámetros necesarios para ajustar este modelo y un conjunto de datos de entrenamiento, re-entrenaremos el modelo inicial, logrando así un modelo ajustado a la detección de la ofensividad. Con el modelo ya ajustado, realizaremos una predicción con los datos del conjunto test, obteniendo unas métricas finales, que nos indican cómo de bien ha realizado la tarea tras su ajuste.

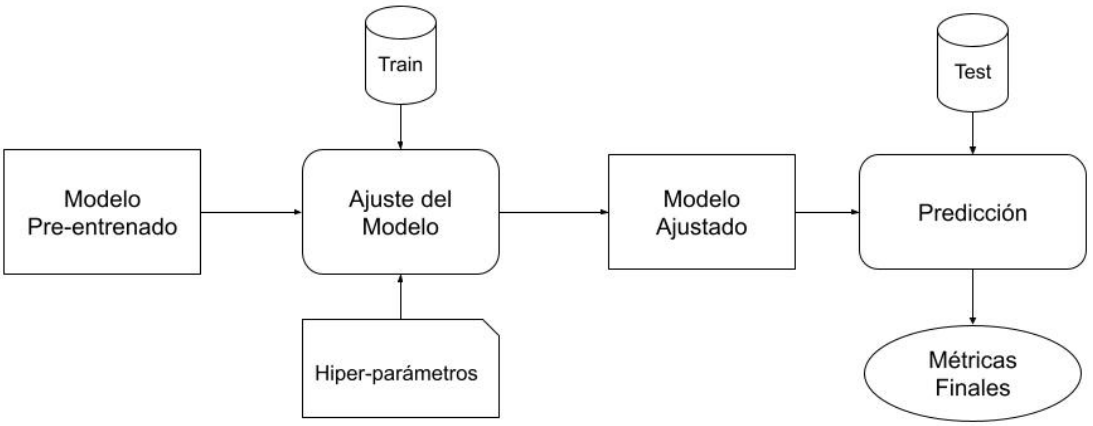

Ilustración 2.1 Arquitectura base

2.1.2 Integración de Optuna y MLflow en nuestra arquitectura

Como en nuestro trabajo vamos a utilizar distintas herramientas, Optuna para la optimización de hiper-parámetros y MLflow para almacenar los resultados tanto de la optimización de hiper-parámetros como de los resultados de los experimentos, debemos diseñar una arquitectura que nos muestre la integración del sistema con estas dos plataformas. En la imagen inferior (Ilustración 2.2), se puede ver esta arquitectura, en la que tenemos un modelo pre-entrenado al que se le hace un ajuste y vamos evaluando el modelo con los datos de validación, obteniendo unas métricas que le pasamos a Optuna, para que fije unos nuevos hiper-parámetros realizar un nuevo ajuste, continuando en este bucle hasta que se cumpla un mínimo de interacciones o que los resultados del ajuste no mejoren. Cuando salimos de ese bucle, Optuna nos devuelve los hiper-parámetros más óptimos de la búsqueda realizada y a continuación con estos hiper-parámetros ajustamos el modelo con la arquitectura que deseemos realizando una predicción y obteniendo un resultado final. Es importante tener en cuenta que tanto las métricas que obtenemos de la evaluación de los modelos en el ajuste de los hiper-parámetros como los resultados finales de la predicción del modelo, se pasan a MLflow para que este nos las muestre y las almacene.

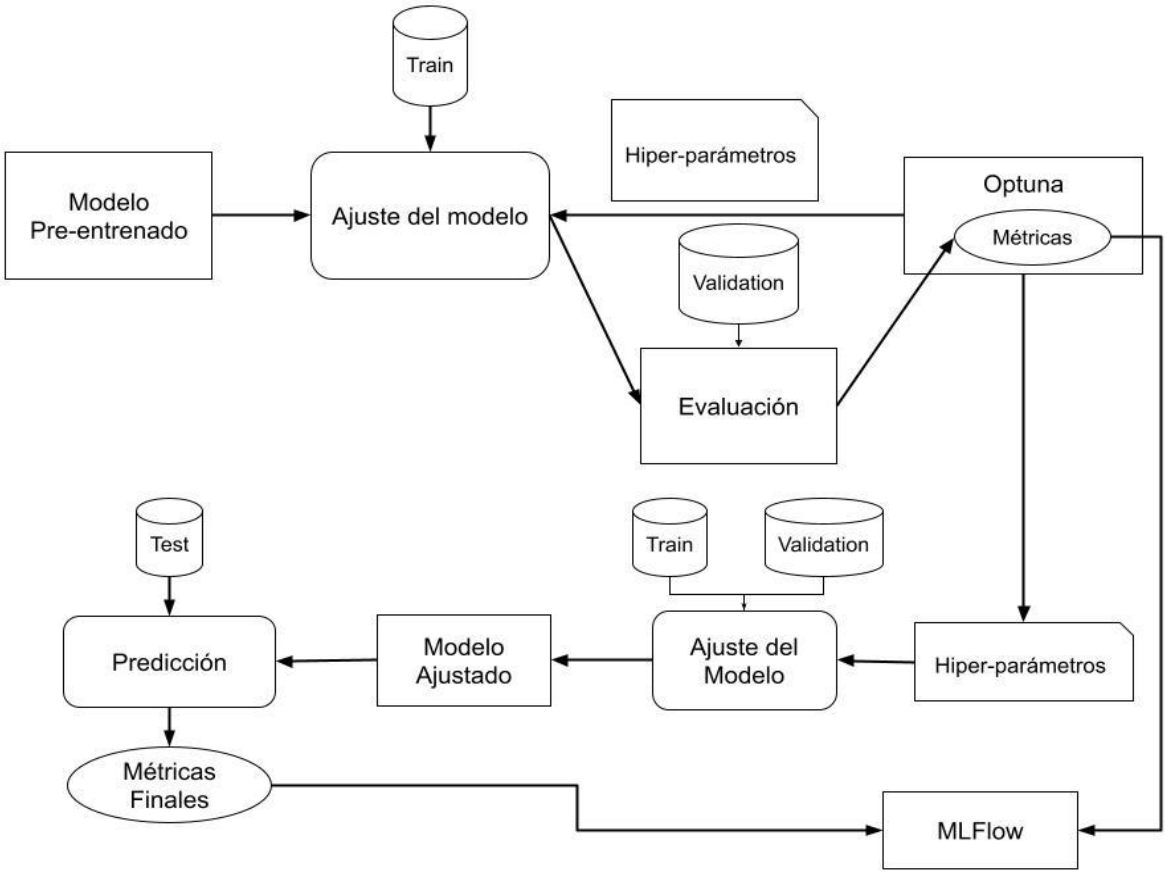

Ilustración 2.2 Arquitectura con la integración de Optuna y MLflow

2.1.3 Arquitecturas planteadas para los clasificadores

Para realizar los ajustes de los modelos con los que vamos a experimentar, una vez que ya obtenemos los hiper-parámetros óptimos, hay veces que necesitamos modificar la arquitectura de estos modelos para adaptarlos a los experimentos planteados. Nosotros nos vamos a centrar en modificar la parte del clasificador que tienen los modelos internamente para resolver la tarea de detección de la ofensividad.

Teniendo en cuenta que el conjunto de datos de la tarea a resolver incorpora metadatos para cada texto, relativos al *influencer* y la red social de donde procede el texto. Podemos considerar esto como información contextual. A la hora de realizar los distintos experimentos se plantean dos estrategias de insertar los datos

contextuales de cada comentario, por lo que cada una de las formas seguirá una arquitectura determinada. Veamos estas dos soluciones:

- **Insertar la información contextual al inicio del texto.** Esto hace que se trate de igual manera al texto que a los metadatos. En la imagen inferior (Ilustración 2.3), podemos ver un esquema de esta estrategia.

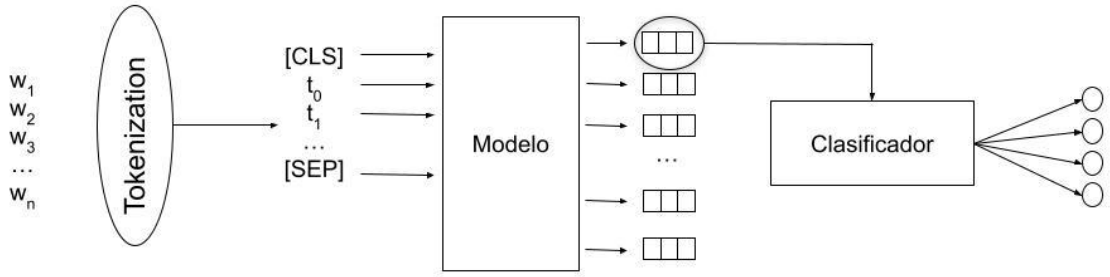

Ilustración 2.3 Imagen de la arquitectura con los metadatos integrados en el texto

- **Insertar la información contextual antes de entrar al clasificador.** Para esto obtenemos los metadatos en formato de texto, y los codificamos con un *one-hot encoding.* Así, le asignaremos a cada dato un vector formado por unos y ceros, donde el 1 representará la palabra a la que hace referencia el dato y el cero serán todos los demás valores. En la imagen inferior (Ilustración 2.4) podemos ver cómo funciona un *one-hot encoding*. A continuación, concatenamos el token de clasificación que sale del modelo con los metadatos que se obtienen a la salida del *one-hot encoding*. Estos datos concatenados entran en un clasificador, que nos dará a su salida una predicción sobre la clase a la que pertenece el texto inicial (ver Ilustración 2.5).

Ilustración 2.4 Imagen del funcionamiento del one-hot encoding

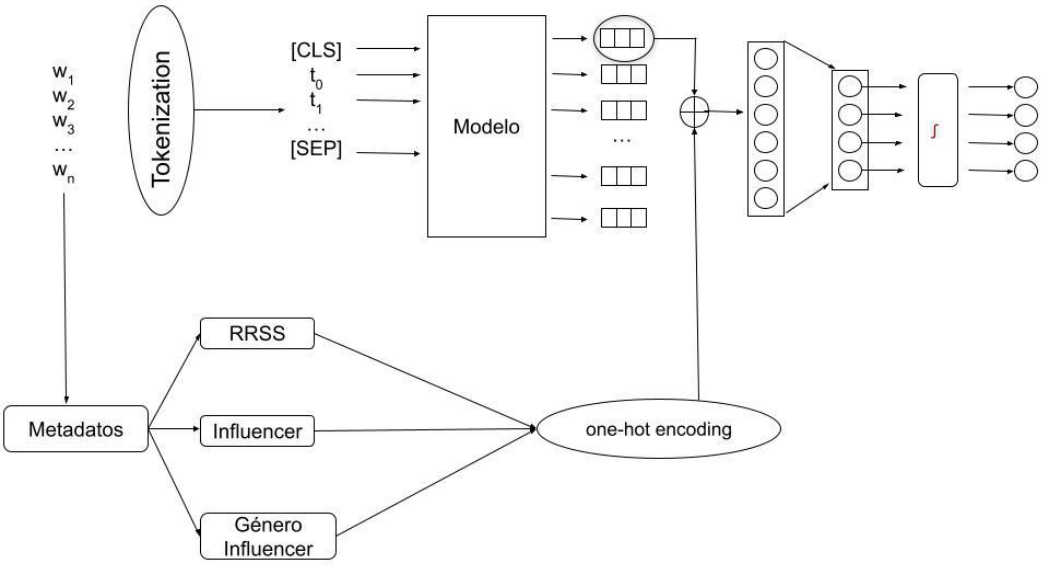

Ilustración 2.5 Imagen de la arquitectura con los metadatos concatenados antes de entrar al clasificador.

3 DESARROLLO

3.1 Primera iteración: Investigación

Antes de iniciar el desarrollo del proyecto fue necesario realizar una investigación sobre cómo desarrollar un sistema de detección de lenguaje ofensivo utilizando tecnologías de vanguardia de PLN, es decir, ¿Qué es un modelo del lenguaje?, ¿Qué modelos nos encontramos? ¿Qué arquitectura tienen estos

modelos? Además de conocer las formas que existen de entrenar estos modelos y qué arquitecturas podemos modificar, por ejemplo, el clasificador.

3.1.1 Modelos del lenguaje

Los modelos del lenguaje son redes neuronales artificiales que han sido entrenadas con una gran cantidad de textos de manera semi-supervisada, es decir, no necesita de personas a la hora de etiquetar los datos, ya que el objetivo del entrenamiento se calcula a partir de las entradas del modelo de forma automática. A través de este entrenamiento, el modelo desarrolla una comprensión estadística del lenguaje en el que es entrenado. Una vez que los modelos ya tienen un entrenamiento general, se pueden volver a entrenar para adaptarlos a temáticas o tareas más específicas. Para ello, los modelos sufren lo que se llama *transfer learning*. En estos casos, los datos de pre-entrenamiento con los que vamos a ajustar un modelo, o lo que es lo mismo, realizar un *fine-tuning*, sí que están anotados por humanos.

La estrategia que se sigue para obtener un mayor rendimiento de estos modelos es aumentar su tamaño y la cantidad de datos con la que se entrenan. El problema es que entrenar un modelo desde cero necesita de una gran cantidad de datos, recursos computacionales y tiempo. Debido a esto, lo que se hace es compartir los modelos.

Por lo tanto, cuando queremos utilizar un modelo, tenemos dos opciones: realizar un pre-entrenamiento o hacer un *fine-tuning*. En el caso del pre-entrenamiento, los pesos del modelo se inicializan aleatoriamente y el entrenamiento del modelo comienza sin tener ningún tipo de conocimiento. Este pre-entrenamiento necesita de conjuntos de datos muy grandes y en cuanto a tiempo, puede tardar varias semanas. Por otro lado, el *fine-tuning* es el entrenamiento que se le hace a un modelo después de que este ya haya tenido un entrenamiento previo, así el modelo realiza un aprendizaje extra con el conjunto de datos adaptado a la tarea a realizar. Las ventajas que obtenemos del *fine-tuning* es

que, como el modelo ya ha sido pre-entrenado, puede usar el conocimiento adquirido en los nuevos datos. Además, requiere menos tiempo, datos y recursos que al inicio para obtener buenos resultados. Como podemos observar, el conocimiento previo se transfiere al nuevo modelo de ajuste fino, de ahí que se llame transferencia del aprendizaje, en inglés *transfer-learning*. En nuestro trabajo optaremos por realizar un fine-tuning por todas las ventajas enumeradas anteriormente (utilización del conocimiento previo obtenido del pre-entrenamiento, necesidad de menor tiempo, datos y recursos).

3.1.2 Arquitecturas de los modelos

Como hemos dicho al inicio de este subapartado, se investigaron las diferentes arquitecturas que tenían los modelos elegidos. Sobre los modelos elegidos y las razones de su elección se habla con más detalle en la Sección 1.6. En cuanto a la investigación inicial que se realizó, podemos ver el Capítulo 2, en la que se explican las distintas arquitecturas investigadas y su funcionamiento.

3.2 Segunda iteración: Análisis de los datos

Inicialmente, seleccionamos los datos que íbamos a utilizar, seleccionamos OffendES [11] ya que era el que más información contextual nos proporcionaba, además de poder descargarlos a través de Hugging Face[2]. Después, analizamos el conjunto de datos que íbamos a usar. OffendES tiene tres divisiones, una llamada *Train*, utilizada para entrenar los modelos, otra llamada *Test*, utilizada para evaluar los modelos, y otra llamada *Validation*, qué es la utilizada para optimizar los hiper-parámetros o en algunos experimentos para entrenar junto con la división de entrenamiento.

Los datos presentan los siguientes campos:

- **Comment_id:** establece un identificador para el comentario.

[2] https://huggingface.co/datasets/fmplaza/offendes

- **Comment:** es el comentario a clasificar, este contiene el texto tal cual se produce en la red social en la que se publica, contiene emoticonos, menciones, URLs, sílabas repetidas...
- **Influencer:** el nombre del influencer al que se le hace el comentario. Los influencer considerados son: *dalas, dulceida, javioliveira, jpelirrojo, lauraescane, miare, nauterplay, nosoymia, soyunapringada, wildhater, windygirk* y *wismichu.*
- **Influencer_gender:** es el género del influencer, *hombre* o *mujer*.
- **Media:** es la red social en la que se publica el comentario, en este caso tenemos como redes sociales *Youtube*, *Instagram* y *Twitter*.
- **Label:** Es la etiqueta que tienen estos comentarios, a la hora de clasificar, estas etiquetas se sustituyen por números del 0 al 3, ya que el dataset inicial lo tiene en número. Las etiquetas posibles son:
  - OFP (0): Se trata de la etiqueta asignada para referirnos a comentarios ofensivos dirigidos a una persona.
  - OFG (1): Esta etiqueta es la que llevan los comentarios ofensivos dirigidos a un grupo o colectivo.
  - NO (2): Esta etiqueta es la asignada a comentarios que no son ofensivos.
  - NOM (3): Esta etiqueta es la que se refiere a comentarios que no son ofensivos pero contienen palabras malsonantes.

Una vez que sabemos el contenido del conjunto de datos a utilizar, en las siguientes secciones, vamos a mostrar distintas gráficas que nos ayuden a comprender mejor los datos, a través de una serie de estadísticas.

3.2.1 Comentarios según las divisiones del conjunto de datos

En primer lugar, queremos analizar el número de comentarios que tenemos de cada tipo, en cada división del conjunto de datos y el porcentaje de cada tipo de comentarios en comparación con el número de comentarios que tiene cada división:

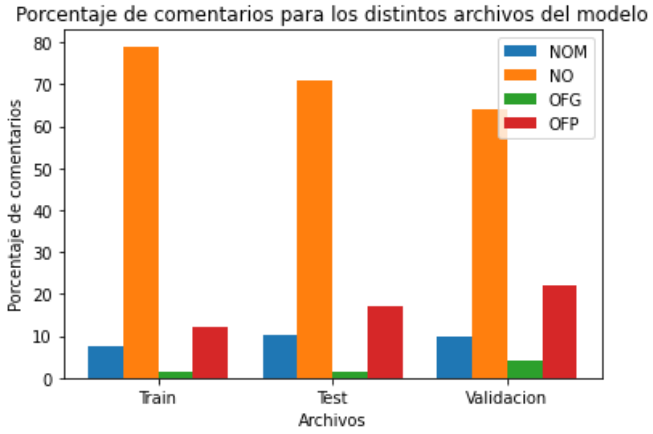

Ilustración 3.1 Porcentaje de comentarios para cada división del dataset.

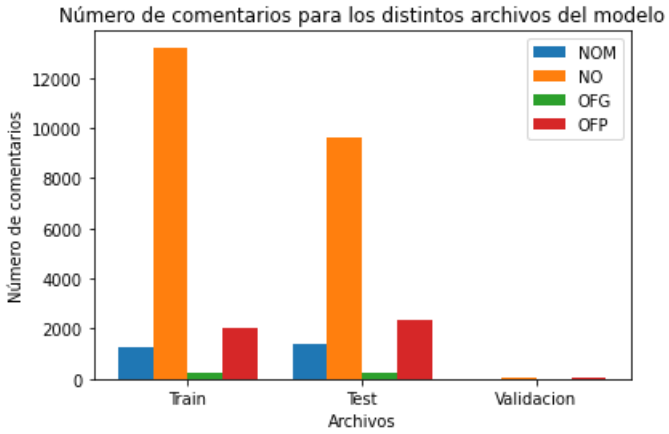

Ilustración 3.2 Número de comentarios para cada división del dataset.

Si observamos la primera imagen de arriba (Ilustración 3.1), podemos ver que, en cuanto al porcentaje de comentarios de cada tipo, todas las divisiones están desbalanceadas, es decir, no contienen el mismo porcentaje de comentarios por clase. La clase minoritaria son los comentarios ofensivos dirigidos a un grupo,

seguido de los no ofensivos malsonantes, ofensivos dirigidos a personas y, finalmente, la clase mayoritaria es la de comentarios no ofensivos. Además, las divisiones no tienen el mismo porcentaje de comentarios de cada clase, es decir, mientras que los comentarios no ofensivos en train son casi un 80%, para test esta clase de comentarios es un poco más del 70% y, en validación, estos no llegan al 70%. Si ahora nos fijamos en la Ilustración 3.2, vemos cómo el conjunto que más datos tiene es “Train”, seguido de “Test” y finalmente “Validación” que contiene tan pocos datos en comparación con el resto, que las barras casi no se ven.

3.2.2 Comentarios según el Influencer

Si observamos el porcentaje de comentarios que tenemos para cada Influencer dentro del dataset OffendES, podemos observar que los influencers que tienen más comentarios ofensivos dirigidos hacia una persona, son *miare*, seguida de *windygirk*, *soyunapringada, javioliveira* y *nauterplay*. Por otro lado, *dulceida* casi no recibe comentarios de este tipo. Si prestamos atención a los comentarios ofensivos grupales, los que más reciben comentarios de este tipo son *javioliveira, miare* y *dalas* y los que menos *jpelirojo*, que no tiene ningún comentario de este tipo y *dulceida* que recibe pero el porcentaje es muy pequeño. Si nos fijamos, los comentarios no ofensivos son los predominantes para todos los influencers, pero los que más destacan por tener casi todos sus comentarios no ofensivos, son *dulceida*, *nosoymia, jpelirrojo* y *lauraescane* con más del 90% de los comentarios no ofensivos. Finalmente, echemos un vistazo a los comentarios que son no ofensivos malsonantes: todos los influencers reciben comentarios de este tipo, pero los que más lo hacen son *soyunapringada* y *wismichu* y los que menos *dulceida* y *lauraescane*.

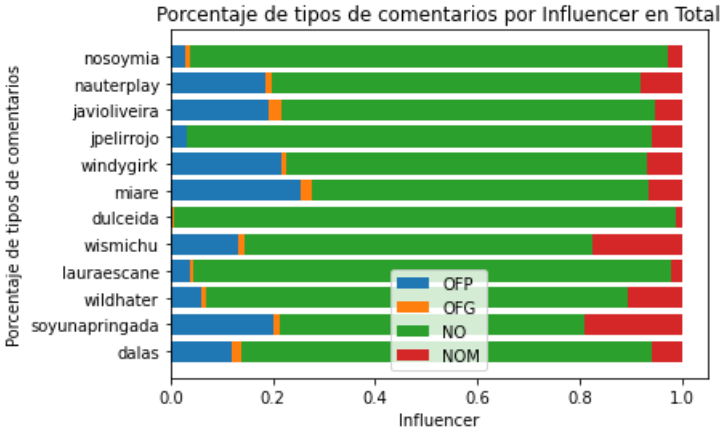

Ilustración 3.3 Porcentajes de comentarios por influencer

3.2.3 Comentarios según el género del Influencer

Por otro lado, si analizamos la clase de comentarios que reciben los influencers según su género, mirando la gráfica inferior, podemos ver que las mujeres reciben un porcentaje de comentarios dirigidos a una persona (OFP), un poco superior al porcentaje de los hombres y un poco menos de comentarios que son no ofensivos. El resto de clases están igualadas.

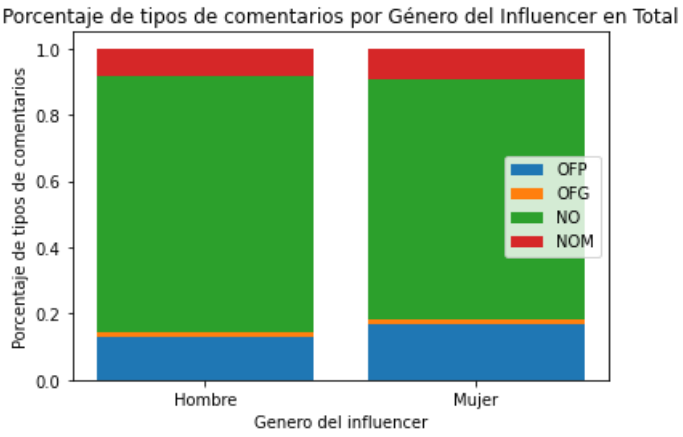

Ilustración 3.4 Porcentajes de comentarios por género

3.2.4 Comentarios según la red social de procedencia.

Si visualizamos ahora en qué red social se producen más comentarios de cada tipo, vemos que la red social en la que se producen más comentarios pertenecientes a la clase OFP es *Youtube*, de la clase OFG y NO, tiene más comentarios *Twitter* y finalmente de la clase NOM, el mayor porcentaje lo tiene la red social *Youtube*. De todas estas redes sociales, podríamos decir que según los datos obtenidos, la menos tóxica (la que menos ofensividad tiene) es Instagram.

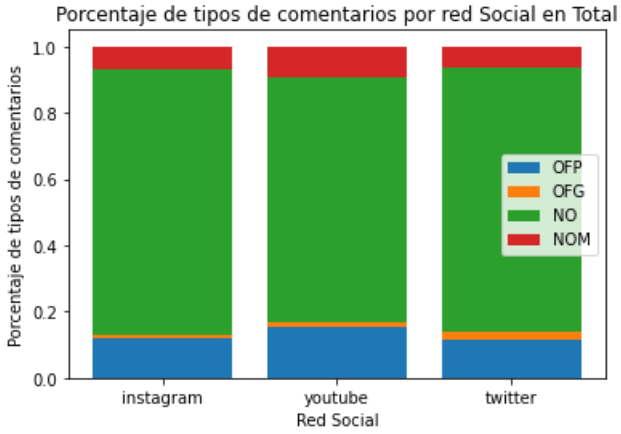

Ilustración 3.5 Porcentajes de comentarios según la Red Social

3.2.5 Palabras más frecuentes según el tipo de comentario.

Al inicio, también pensamos en realizar un análisis de nube de palabras, para visualizar cuáles eran las más utilizadas según la clase de comentarios que se realizaban. Para ello utilizamos nubes de palabras cuyo tamaño depende de su frecuencia de aparición en el texto. Obtuvimos las siguientes conclusiones según la clase:

- Si visualizamos la Ilustración 3.6 podemos ver que hay palabras normales que no suponen ningún tipo de insulto, pero luego también vemos palabras que resultan ofensivas como “mierda”, “gorda”, “basura”, “falsa”...

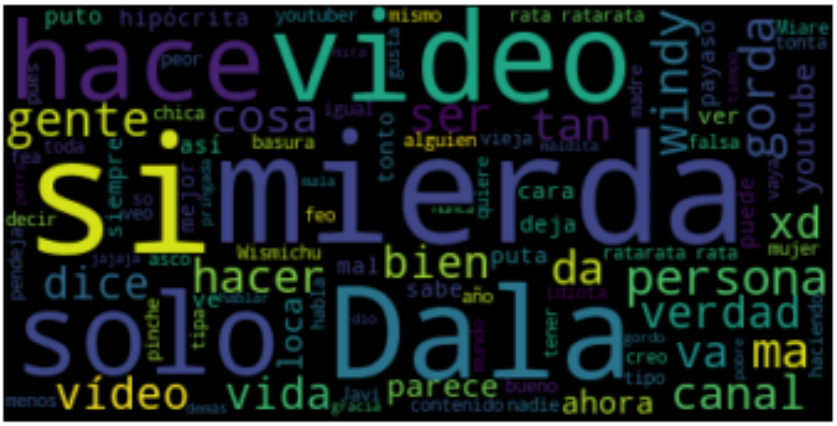

Ilustración 3.6 Nube de palabras de textos OFP

- En la Ilustración 3.7, observamos que hay palabras como gente o persona que son muy frecuentes y palabras de uso común como “familia”, “vídeo”, “comentario”, pero además de esto nos encontramos con palabras ofensivas como “puto idiota”, cosa que nos sorprende, ya que parecen un insulto dirigido solo a una persona, además de insultos como “payasos”, “gilipollas”...

Ilustración 3.7 Nube de palabras de textos OFG

- En la Ilustración 3.8, referida a los comentarios no ofensivos, vemos que la palabra “Franco” se repite dos veces, seguramente porque lleve al lado algún carácter oculto y por eso la herramienta usada la identifica como dos palabras distintas. Por lo demás, visualizamos palabras de uso común que

no incitan a pensar que haya ofensividad, como “amigo”, “gracias”, “vídeo”, “gente”, “hace”, “bien”…

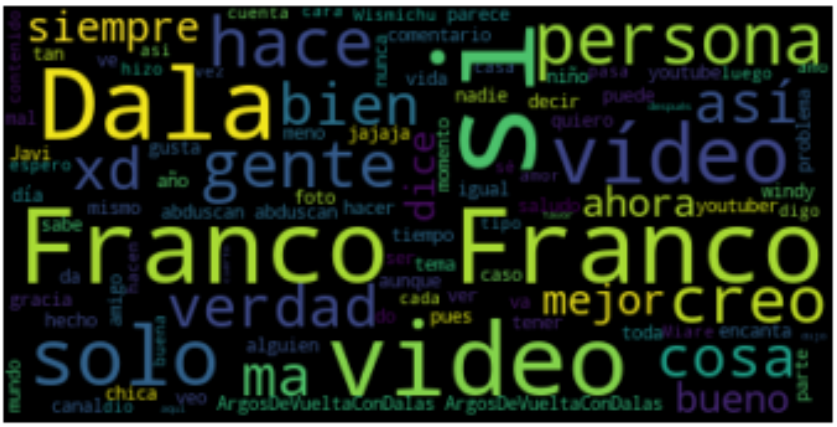

Ilustración 3.8 Nube de palabras de textos NO

- Finalmente, la Ilustración 3.9, es la imagen en la que más palabras malsonantes podemos observar, lo que no nos sorprende, pues tal y como dice el nombre de la clase, son comentarios que llevan palabras malsonantes, pero no conllevan al insulto.

Ilustración 3.9 Nube de palabras de textos NOM

3.2.6 Análisis con TextFlow

TextFlow es una herramienta que permite segmentar los textos en diferentes niveles, según se desee. Por ejemplo, se puede dividir un texto a nivel de párrafo y

estos a su vez en oraciones y las oraciones en palabras. El hecho de tener una arquitectura que nos facilite las distintas divisiones de los textos, nos ayuda para poder aplicar diferentes analizadores a los textos y extraer características esenciales de cada nivel e incluso agrupar las características de niveles inferiores en niveles superiores. Actualmente, cuenta con analizadores que son capaces de extraer más de 50 características de los textos, proporcionándonos gran cantidad de información, que se puede analizar de forma sencilla y sin gran esfuerzo. Todo lo relacionado con TextFlow se puede ver en la referencia [33].

Aunque hemos obtenido todas las características que TextFlow nos proporcionaba, vamos a ver en una tabla (Tabla 3.1) la emoción predominante, la polaridad predominante, el número medio de palabras, la etiqueta del *part-of-speech* (PoS) más frecuente, la ironía, el número medio de emojis en cada comentario y los emojis más frecuentes, según la clase de los comentarios.

Característica	OFP	OFG	NO	NOM
Emoción predominante	Enfado	Enfado	Otra	Enfado
Polaridad Predominante	Negativa	Negativa	Negativa	Negativa
Ironía o No Ironía	No	No	Ironía	No Ironía
Número Medio de Palabras	20	33	35	17
Etiqueta POS predominante	Nombre	Nombre	Nombre	Nombre
Número medio de emoticonos	0,2261	0,2927	0,7638	0,3201

en cada comentario				
Cuatro emoticonos más frecuentes	Cara llorando de risa, cara revolviéndose de la risa, cara de payaso y cara vomitando.	Cara llorando de risa, caca con ojos, cara revolviéndose de la risa y cara cabreada	Cara llorando de risa, corazón rojo, cara sonriendo con ojos de corazón, cara lanzando un beso	Cara llorando de risa, cara revolviéndose de la risa, corazón rojo y fuego

Tabla 3.1 Resultados del análisis de los textos con TextFlow.

Como podemos ver en la tabla superior (Tabla 3.1), el número medio de palabras es superior en los textos no ofensivos. Si nos fijamos, las etiquetas PoS (son etiquetas que indican la categoría gramatical de las palabras de un texto, como por ejemplo, nombres, adjetivos, verbos, pronombres, adverbios…) más predominantes en las cuatro clases son los nombres. El emoticono más usado es la cara llorando de la risa para las cuatro clases, pero si nos fijamos en los siguientes emoticonos más usados, podemos ver que las clases ofensivas (OFP y OFG) tienen emoticonos negativos como la cara de payaso, la cara vomitando, la cara enfadada o la caca con ojos, mientras que los emoticonos de los comentarios no ofensivos (NO y NOM) tienen significados positivos, como son los corazones rojos, la cara con ojos de corazón, caras lanzando un beso o el fuego. Otra cosa a tener en cuenta es que, de media, estos textos no suelen contener emoticonos. En cuanto a emociones, vemos que en casi todos la emoción predominante es el enfado y todos los tipos tienen como análisis de sentimientos textos negativos en su mayoría. Pero dentro de todo este análisis hay algo que nos llama la atención y es que, si analizamos la ironía de los textos, la mayoría de los comentarios de la clases OFP, OFG Y NOM son no irónicos y por el contrario, la mayoría de los que pertenecen a la clase NO sí que presentan cierta ironía, según el modelo usado para realizar este análisis.

3.3 Tercera iteración: Planteamiento de los experimentos

En esta fase, una vez hemos realizado las investigaciones de los distintos modelos existentes y su arquitectura, realizamos la elección de los modelos y los datos con los que experimentar.

Se plantean los siguientes experimentos, que explicaremos con más detalle en el apartado 4.1:

- Los modelos se entrenan con datos que no contienen información contextual.
- Los modelos se entrenan con datos que contienen información contextual integrada en el texto.
- Los modelos se entrenan con datos preprocesados que no contienen metadatos.
- Los modelos reciben como entrada de texto, los metadatos integrados en el comentario preprocesado.
- El modelo recibe la información contextual antes de pasar por el clasificador.
- El modelo recibe la información contextual antes de pasar por el clasificador y los textos están preprocesados.

3.4 Cuarta iteración: Optimización de los hiper-parámetros de los modelos

Para esta iteración, primero determinamos los hiper-parámetros a optimizar de entre todos los posibles y en qué rango situar la búsqueda. Entre los hiper-parámetros a optimizar en los distintos modelos, nos encontramos con:

- ***Learning-rate:*** hiper-parámetro que sirve para controlar cuánto cambia el modelo en respuesta a un error estimado cada vez que se produce una actualización de los pesos del modelo. Es difícil de realizar una elección de este valor, ya que un número muy alto puede hacer que el aprendizaje

de un conjunto de pesos sea inestable y un valor demasiado bajo, puede hacer que el entrenamiento se atasque. Por eso decidimos intentar optimizar este valor y así obtener un valor que proporcione un buen rendimiento. Inicialmente, situamos el rango de optimización entre $4 \cdot 10^{-5}$ y 0.01. El problema de este rango es que daba saltos muy grandes y hacía que la red no aprendiese, por lo que decidimos hacer el rango más pequeño y establecer unos valores más bajos. Así, finalmente nos quedamos con el rango establecido entre $7 \cdot 10^{-7}$ y $4 \cdot 10^{-5}$.

- ***Weight-decay:*** es un parámetro que ayuda a evitar que se produzca un sobreajuste de la red neuronal a los datos de entrenamiento. Para ello, añade una penalización a la función de coste, reduciendo los pesos de la red durante la retropropagación. Establecimos los valores de búsqueda para este hiper-parámetro entre 0,01 y $4 \cdot 10^{-5}$, ya que el valor de 0,01 es el más utilizado en la literatura científica.
- ***Batch-size:*** es el número de datos que procesa cada iteración de entrenamiento de la red, esto es útil porque la red neuronal actualiza los pesos de manera más estable. Cuando se tienen grandes cantidades de datos se necesitan computadores con más memoria, pero la red neuronal tarda menos en ejecutar cada ciclo. Por otro lado, si el número de datos es más pequeño, ahorramos memoria pero entrenamos más despacio. Debido a nuestra limitación de memoria, fue necesario realizar una optimización de este parámetro y elegimos como tamaño 8, 16 o 32.
- ***Epoch:*** es el número de veces que se ejecutan los algoritmos de aprendizaje sobre todo el conjunto de datos de entrenamiento. En cada época, todos los datos de entrenamiento pasan por la red neuronal para que esta aprenda de dichos datos. En este caso no hacemos optimización de este hiper-parámetro, pero establecemos un valor de parada para que, si tras tres ciclos no mejoran los resultados, el modelo finalice su entrenamiento. A este valor de parada se le llama *Early Stop Patience*.

Tras ver qué hiper-parámetros íbamos a optimizar y en qué rango establecíamos la búsqueda de los mejores valores, investigamos qué herramienta

de optimización íbamos a usar, tal y como se explica en la Sección 1.6. Finalmente, seleccionamos Optuna, por las razones especificadas en dicho apartado. Para el uso de esta herramienta, lo primero que debemos hacer es crear un estudio, con un nombre y una dirección de optimización. Ponerle un nombre al estudio facilita luego buscarlo en la memoria de nuestro ordenador por si necesitáramos revisarlo. La dirección de optimización se refiere a si queremos encontrar el valor más alto (maximizar) o más bajo (minimizar) de nuestro objetivo a optimizar. Una vez que hemos creado este estudio, es el momento de implementar la función objetivo que queremos optimizar y es necesario que esta devuelva un único valor. Además, dentro de esta función, necesitaremos llamar a las funciones que establecen los rangos en las variables a optimizar. En nuestro caso, esta función objetivo es el entrenamiento de un modelo y el valor que retorna la función es el valor de pérdida del entrenamiento del modelo, por lo que la dirección del estudio será intentar minimizar la pérdida. Una vez hemos implementado la función objetivo, podemos llamar a la función de optimización del estudio, especificando el número de pruebas (o *trial* como lo llaman los desarrolladores de Optuna) que se quiere realizar. También se le puede añadir una función *callback*, así, si no mejora en un número determinado de pruebas, finaliza la optimización del estudio.

Finalmente, una vez que hemos obtenido todos los resultados de las optimizaciones realizadas, las subimos a MLflow para poder visualizar mejor los resultados y organizar los experimentos. En la Ilustración 3.10 podemos ver cómo se verían estos resultados en la plataforma utilizada.

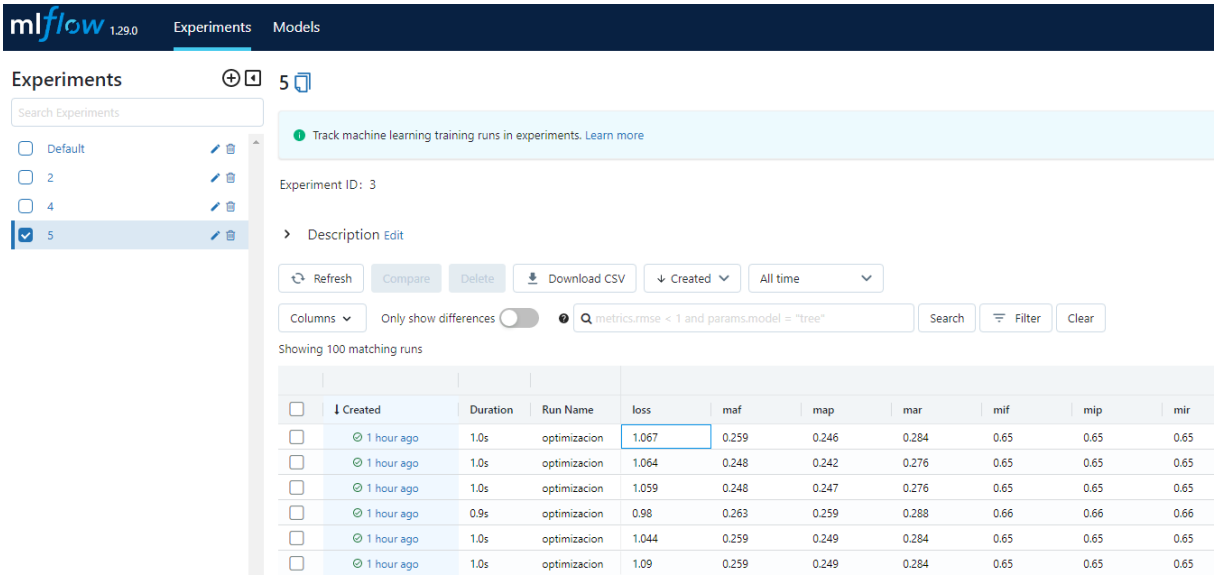

Ilustración 3.10 Imagen de la plataforma MLflow utilizada para ver los resultados de los hiper-parámetros.

La siguiente tabla (Tabla 3.2) muestra, a modo de resumen, los mejores parámetros para los modelos y los datos con los que se han entrenado. Recordemos que para optimizar los hiper-parámetros de los modelos y evaluar sus efectos utilizamos el conjunto de datos de validación.

Optuna, por tanto, realiza un aprendizaje completo a partir de una combinación de hiper-parámetros generada dentro de los márgenes indicados para cada uno de ellos. Una vez completadas todas las épocas sobre el conjunto de entrenamiento, el modelo resultante se evalúa contra el conjunto de validación. De esta forma obtenemos el rendimiento de cada combinación de hiper-parámetros, devolviendo Optuna aquella que genera los mejores resultados.

Modelo	Batch-size	Learning-rate	Weight-decay
Beto Twitter	8	$2{,}3 \cdot 10^{-5}$	$3{,}57 \cdot 10^{-3}$
Beto Normal	16	$3{,}89 \cdot 10^{-6}$	$5{,}23 \cdot 10^{-5}$

RoBERTa Twitter	16	$2{,}72 \cdot 10^{-5}$	$1{,}27 \cdot 10^{-3}$
RoBERTa Normal	8	$3{,}14 \cdot 10^{-6}$	$1{,}2 \cdot 10^{-5}$
RoBERTa Cardiff	8	$1{,}11 \cdot 10^{-5}$	$3{,}66 \cdot 10^{-3}$
BERTIN Twitter	8	$6{,}26 \cdot 10^{-6}$	$1 \cdot 10^{-4}$
BERTIN Normal	8	$7{,}82 \cdot 10^{-7}$	$8{,}7 \cdot 10^{-4}$

Tabla 3.2 Valores óptimos encontrados para cada modelo

3.5 Quinta iteración: Entrenamiento de los modelos

Para el desarrollo de esta iteración, llevamos a cabo un entrenamiento final para las distintas arquitecturas. Tenemos dos tipos de entrenamiento, uno para todos los modelos que cargamos desde el repositorio de *Hugging Face*, y otro con la modificación del modelo que cargamos desde *Hugging Face* para insertar los metadatos antes de la entrada del clasificador, que son las dos estrategias exploradas en esta investigación

Estrategia 1: arquitectura original

En cuanto al primer tipo de entrenamiento, nos encontramos con un sistema sencillo, muy parecido al de la arquitectura original con el que optimizamos los hiper-parámetros en *Optuna*. Para implementar este sistema hemos seguidos estos pasos:

1. Descargar y establecer el modelo que vamos a utilizar, especificando el número de clases en las que va a clasificar el modelo seleccionado.
2. A continuación, accedemos al nombre y al identificador que va a tener cada clase en la que nuestro modelo va a clasificar y le cambiamos el nombre, para que se corresponda con nuestras etiquetas (OFP, OFG, NO, NOM con identificadores de 0 a 3, respectivamente).

3. Una vez que hemos cargado el modelo, cargamos el dataset OffendEs, según el experimento a realizar, ya que si tenemos los datos integrados en el texto o vamos a utilizar los datos preprocesados, usaremos unos archivos u otros que ya contienen las modificaciones necesarias.
4. Tras esto necesitaremos definir el tokenizador, el cual es descargado desde el hub de Hugging Face, y con una función map contenida en el dataset, tokenizamos los comentarios de texto aplicando truncamiento (acorta los comentarios más largos, para que todos los comentarios tengan la misma longitud) y relleno (agrega tokens de relleno para que todos los comentarios tokenizados tengan la misma longitud, util para los comentarios más cortos).
5. Tras esto, necesitamos un recolector de datos, que nos formará un lote, utilizando una lista de *dataset* como entrada. Además de elaborar una función con las distintas métricas con las que evaluaremos nuestro modelo.
6. Después, haremos uso de la API trainer que tiene *Hugging Face* desarrollada, en la que definimos los argumentos de entrenamiento (hiper-parámetros a utilizar, la dirección en la que vamos a almacenar el modelo entrenado, o la estrategia a seguir) y ya concretaremos el entrenamiento a realizar, especificando el modelo, el conjunto de datos de entrenamiento, el de evaluación, las métricas a calcular y los *callback*, que es donde establecemos el valor de parada para que si no mejora en 3 ciclos, se finalice el entrenamiento.
7. Finalmente, ya podemos llamar a la función de entrenamiento y obtener los resultados de evaluación.

Estrategia 2: integración de metadatos antes de la capa de clasificación

Para el segundo tipo de entrenamiento, la estrategia es más o menos parecida, la única diferencia es que en vez de cargar un modelo desde Hugging Face, debemos especificar nosotros la red neuronal que vamos a tener. Para eso:

1. Crearemos una clase que, en su constructor, cargará el modelo de *Hugging Face* para usarlo como modelo base, establecerá el número de clases en las que vamos a clasificar con nuestro modelo, además del *dropout* necesario (el

dropout es una técnica de regularización que se basa en la eliminación de neuronas de las capas de los modelos) y la capa de clasificación lineal necesaria junto con sus dimensiones, las cuales serán, como entrada el número de dimensiones del modelo y de los vectores a concatenar (768+17) y como valor de salida, el número de etiquetas.

2. Lo siguiente a realizar en la clase de nuestro nuevo modelo es re-escribir el método *forward()*, pues es el método que calcula la función de pérdida de nuestro modelo. En dicha función, obtendremos las salidas del modelo, y a continuación concatenamos esa salida con las características obtenidas de los metadatos. Es importante remarcar que según el modelo, se utilizará como salida del modelo *pooled_output* o *sequence_output*. En caso de tratarse de un modelo BERT, usaremos *pooled_output*, ya que es el que usa para clasificar en su método forward original, mientras que RoBERTa y BERTIN, usarán el *sequence_output*, pero, ¿Cuál es la verdadera diferencia entre estas dos salidas? Pues bien, la diferencia reside en que cada modelo usa una estrategia distinta, BERT obtiene la salida de la última capa del token de clasificación de *sequence_output* y la pasa por una capa lineal y una función de activación *Tanh* para realizar la clasificación final y RoBERTa simplemente se queda con el token de clasificación de la última capa oculta (*sequence_output*) para realizar la clasificación final, sin necesidad de pasar por otras capas o funciones de activación.
3. Tras esto, a la salida concatenada con los metadatos se pasan por el *dropout* y la capa lineal de clasificación y finalmente, se calcula la función de pérdida, que al tratarse de un modelo de clasificación multiclase, será del tipo *Cross Entropy*. El valor que debe devolver el método *forward* debe ser de tipo *SequenceClasiffierOutput*.
4. En cuanto al algoritmo de entrenamiento, lo único que hay que modificar es que una vez cargados los datos, pasaremos los metadatos por un one-hot encoding para que el método forward los reconozca y los pueda concatenar con el token de clasificación. Esto lo haremos mediante la función *map* del *dataset*, que sirve para preprocesar los datos de forma rápida.

3.6 Sexta iteración: Análisis de los resultados

En esta iteración, ya en las fases finales del proyecto, nos dedicamos a analizar todos los resultados que obtuvimos de los experimentos realizados, elaborando un informe con tablas y gráficas para sacar las conclusiones de cuál es la mejor estrategia a seguir y qué modelo funciona mejor según las distintas métricas. Para esta iteración hemos dedicado todo un apartado (véanse Secciones 4.2 y 4.3).

3.7 Pruebas finales

Para ver si los modelos finales obtenidos funcionan bien, deberemos realizar una serie de pruebas, que veremos a continuación en los subapartados correspondientes:

3.7.1 Pruebas de verificación del sistema

La verificación en este caso sería comprobar si el sistema es capaz de clasificar los comentarios en alguna de las clases. Y como podemos ver en los resultados obtenidos, esto lo hacen todos los modelos.

Conjunto Entrenamiento	Modelo	%OFP	%OFG	%NO	%NOM
Train	BETO Twitter	12,27	1,62	76,79	9,33
	BETO Normal	15,42	1,48	74,81	8,3
	RoBERTa Twitter	14,55	1,73	74,18	9,55
	RoBERTa Normal	14,19	0,07	77,19	8,54
	RoBERTa Cardiff	14,33	1,89	74,43	9,35

	BERTIN Twitter	11,64	0	78,47	9,89
	BERTIN Normal	13,69	0	78,18	8,14
Train + Validación	BETO Twitter	15,67	1,3	75,44	7,58
	BETO Normal	14,26	1,37	74,92	9,44
	RoBERTa Twitter	16,3	1,89	72,88	8,93
	RoBERTa Normal	14,43	0,55	76,37	8,64
	RoBERTa Cardiff	14,98	1,47	73,78	9,78
	BERTIN Twitter	10,22	0	80,80	8,99
	BERTIN Normal	12,72	0	78,65	8,63

Tabla 3.3 Tabla Verificación Sin Metadatos (Estrategia Base)

Conjunto Entrenamiento	Modelo	%OFP	%OFG	%NO	%NOM
Train	BETO Twitter	13,08	1,55	76,07	9,30
	BETO Normal	15,27	1,32	75,17	8,25
	RoBERTa Twitter	15,32	1,73	74,1	8,85
	RoBERTa Normal	13,44	0,55	79,47	8,9
	RoBERTa Cardiff	15,42	1,62	73,43	9,53
	BERTIN Twitter	14,72	0	76,42	8,86
	BERTIN Normal	11,99	0	79,47	8,53

Train + Validación	BETO Twitter	13,64	1,21	75,67	9,47
	BETO Normal	13,44	1,32	75,55	9,69
	RoBERTa Twitter	16,11	1,52	73,03	9,33
	RoBERTa Normal	14,22	0,31	75,58	9,89
	RoBERTa Cardiff	14,79	1,56	73,27	10,39
	BERTIN Twitter	10,58	0	80,46	8,96
	BERTIN Normal	14,49	0	76,94	8,57

Tabla 3.4 Tabla Verificación Con Metadatos en el Texto (Estrategia MT)

Conjunto Entrenamiento	Modelo	%OFP	%OFG	%NO	%NOM
Train	BETO Twitter	13,57	1,12	75,18	10,12
	BETO Normal	14,6	1,4	75,53	8,47
	RoBERTa Twitter	15,43	1,85	72,96	9,76
	RoBERTa Normal	13,87	0,06	77,19	8,88
	RoBERTa Cardiff	14,21	1,35	74,3	10,14
	BERTIN Twitter	12,92	0	78,69	8,39
	BERTIN Normal	13,79	0	78	8,22
Train + Validación	BETO Twitter	15,46	1,64	75,6	7,31
	BETO Normal	14,37	1,38	74,89	9,36
	RoBERTa Twitter	17,76	1,79	70,87	9,57

	RoBERTa Normal	14,35	0,57	76,35	8,72
	RoBERTa Cardiff	17,22	1,62	72,6	8,56
	BERTIN Twitter	14,89	1,55	73,25	10,3
	BERTIN Normal	12,36	0	79,24	8,4

Tabla 3.5 Tabla Verificación Sin Metadatos y con Textos Preprocesados (Estrategia P)

Conjunto Entrenamiento	Modelo	%OFP	%OFG	%NO	%NOM
Train	BETO Twitter	13,54	1,38	75,58	9,5
	BETO Normal	15,06	1,32	75,37	8,25
	RoBERTa Twitter	14,82	1,51	74,88	8,8
	RoBERTa Normal	13,74	0,41	77,00	8,85
	RoBERTa Cardiff	15,28	1,58	73,49	9,65
	BERTIN Twitter	14,55	0	76,43	9,02
	BERTIN Normal	11,74	0	79,72	8,53
Train + Validación	BETO Twitter	13,55	1,34	75,79	9,33
	BETO Normal	13,46	1,37	75,51	9,66
	RoBERTa Twitter	16,99	1,68	72,14	9,18
	RoBERTa Normal	13,79	0,6	76,39	9,23
	RoBERTa Cardiff	16,4	1,58	73,06	8,96

	BERTIN Twitter	10,31	0	80,71	8,97
	BERTIN Normal	13,25	0	78,33	8,42

Tabla 3.6 Tabla Verificación Con Metadatos en el Texto Preprocesado (Estrategia MTP)

Conjunto Entrenamiento	Modelo	%OFP	%OFG	%NO	%NOM
Train	BETO Twitter	11,94	1,4	76,97	9,7
	BETO Normal	15,58	1,42	74,68	8,32
	RoBERTa Twitter	15,28	1,46	74,65	8,61
	RoBERTa Normal	14,02	0,86	76,39	8,74
	RoBERTa Cardiff	14,68	1,47	74,75	9,09
	BERTIN Twitter	9,51	0	81,44	9,05
	BERTIN Normal	13,38	0	78,45	8,17
Train + Validación	BETO Twitter	16,84	1,34	74,2	7,61
	BETO Normal	13,99	1,33	74,92	9,75
	RoBERTa Twitter	18,47	1,98	70,80	8,75
	RoBERTa Normal	14,61	0,26	76,49	8,64
	RoBERTa Cardiff	14,8	1,33	74,20	9,14
	BERTIN Twitter	13,46	0	78,94	8,98
	BERTIN Normal	13,13	0	77,85	8,90

Tabla 3.7 Tabla Verificación Con Metadatos Antes del Clasificador (Estrategia MC)

Conjunto Entrenamiento	Modelo	%OFP	%OFG	%NO	%NOM
Train	BETO Twitter	12,41	1,34	76,77	9,47
	BETO Normal	15,64	1,43	74,64	8,3
	RoBERTa Twitter	15,41	1,46	74,78	8,34
	RoBERTa Normal	14,14	0,79	76,53	8,55
	RoBERTa Cardiff	14,74	1,33	75,08	8,85
	BERTIN Twitter	9,55	0	81,35	9,10
	BERTIN Normal	13,46	0	78,35	8,19
Train + Validación	BETO Twitter	13,75	1,25	75,20	9,8
	BETO Normal	14,03	1,31	75,06	9,61
	RoBERTa Twitter	19,45	1,82	70,42	8,31
	RoBERTa Normal	14,15	0,5	76,51	8,84
	RoBERTa Cardiff	15,23	1,44	74,2	9,14
	BERTIN Twitter	12,08	0	78,94	8,98
	BERTIN Normal	13,25	0	77,85	8,9

Tabla 3.8 Tabla Verificación Con Metadatos Antes del Clasificador y Textos Preprocesados (Estrategia MCP)

En las tablas superiores (Tablas de la 3.3 a la 3.8) vemos que todos los modelos generan respuestas para todas las clases y que la clase minoritaria (OFG) es la menos seleccionada por algunos modelos. Por ejemplo, vemos que los modelos que siguen una arquitectura BERTIN no llegan a clasificar ningún documento en esta clase.

3.7.2 Pruebas de validación del sistema

En cuanto a la validación del sistema, analizamos el grado de éxito de los modelos para generar predicciones acordes con las clases esperadas en el conjunto de test. Es decir, el grado de acierto en la clasificación de los comentarios.

Conjunto Entrenamiento	Modelo	%OFP	%OFG	%NO	%NOM
Train	BETO Twitter	62,69	57,35	97,31	70,16
	BETO Normal	72,01	57,35	96,37	66,67
	RoBERTa Twitter	72,78	66,82	96,48	75
	RoBERTa Normal	66,88	2,84	97,69	66,38
	RoBERTa Cardiff	70,85	64,45	96,39	71,79
	BERTIN Twitter	45,94	0	96,22	65,74
	BERTIN Normal	54,27	0	97,67	58,05
Train + Validación	BETO Twitter	70,73	50,71	96,69	61,75
	BETO Normal	69,32	55,92	96,46	72,44
	RoBERTa Twitter	78,46	69,67	95,77	72,65
	RoBERTa Normal	68,25	24,64	97,38	67,81

	RoBERTa Cardiff	72,52	57,82	95,84	73,29
	BERTIN Twitter	43,63	0	97,52	61,54
	BERTIN Normal	51,71	0	97,63	60,26

Tabla 3.9 Tabla Validación Sin Metadatos (Estrategia Base)

Conjunto Entrenamiento	Modelo	%OFP	%OFG	%NO	%NOM
Train	BETO Twitter	65,51	59,24	97,03	72,01
	BETO Normal	71,75	53,55	96,63	66,10
	RoBERTa Twitter	75,81	66,82	96,32	70,73
	RoBERTa Normal	65,34	25,59	97,68	68,09
	RoBERTa Cardiff	73,80	58,77	95,68	71,79
	BERTIN Twitter	54,40	0	94,76	62,11
	BERTIN Normal	48,08	0	97,51	60,04
Train + Validación	BETO Twitter	66,79	46,45	96,53	71,94
	BETO Normal	66,37	54,98	96,88	73,22
	RoBERTa Twitter	77,44	63,51	95,91	74,15
	RoBERTa Normal	67,39	12,8	96,92	72,86
	RoBERTa Cardiff	71,71	59,72	95,67	75,71
	BERTIN Twitter	44,66	0	97,42	62,82

	BERTIN Normal	53,29	0	95,93	59,90

Tabla 3.10 Tabla Validación Metadatos incorporados en el Texto (Estrategia MT)

Conjunto Entrenamiento	Modelo	%OFP	%OFG	%NO	%NOM
Train	BETO Twitter	66,32	44,08	96,28	74,64
	BETO Normal	69,1	54,98	96,77	67,17
	RoBERTa Twitter	76,07	68,25	95,79	75,5
	RoBERTa Normal	67,09	3,32	97,75	67,74
	RoBERTa Cardiff	69,91	54,5	96,31	74,72
	BERTIN Twitter	50,77	0	96,31	58,97
	BERTIN Normal	54,57	0	97,59	58,33
Train + Validación	BETO Twitter	69,7	60,66	96,79	59,83
	BETO Normal	69,53	56,4	96,43	72,22
	RoBERTa Twitter	81,67	69,19	94,24	75,5
	RoBERTa Normal	68,03	26,07	97,32	68,16
	RoBERTa Cardiff	72,14	58,77	95,64	75,28
	BERTIN Twitter	43,93	0	97,51	61,47
	BERTIN Normal	51,2	0	97,94	59,90

Tabla 3.11 Tabla Validación Sin Metadatos Preprocesado (Estrategia Base Preprocesado)

Conjunto Entrenamiento	Modelo	%OFP	%OFG	%NO	%NOM
Train	BETO Twitter	66,37	54,50	96,44	70,44
	BETO Normal	71,20	54,50	96,78	66,17
	RoBERTa Twitter	73,63	61,61	97,14	71,30
	RoBERTa Normal	65,81	19,91	97,59	67,88
	RoBERTa Cardiff	73,33	56,40	95,63	72,51
	BERTIN Twitter	54,02	0	94,76	62,82
	BERTIN Normal	48,42	0	97,80	60,83
Train + Validación	BETO Twitter	65,34	48,82	96,22	70,09
	BETO Normal	66,45	55,45	96,85	73,08
	RoBERTa Twitter	79,70	68,25	95,20	72,93
	RoBERTa Normal	66,15	23,70	97,27	69,37
	RoBERTa Cardiff	75,43	60,66	95,60	70,73
	BERTIN Twitter	43,93	0	97,58	62,32
	BERTIN Normal	51,41	0	96,97	59,62

Tabla 3.12 Tabla Validación Metadatos incrustados en el Texto Preprocesado (Estrategia MTP)

Conjunto Entrenamiento	Modelo	%OFP	%OFG	%NO	%NOM

Train	BETO Twitter	61,24	50,71	97,54	73,15
	BETO Normal	72,44	59,24	96,37	66,45
	RoBERTa Twitter	76,07	63,51	96,93	69,94
	RoBERTa Normal	66,92	37,91	97,43	68,45
	RoBERTa Cardiff	71,58	60,66	96,63	71,37
	BERTIN Twitter	41,07	0	97,78	62,61
	BERTIN Normal	54,91	0	97,42	58,33
Train + Validación	BETO Twitter	73,89	53,55	96,18	61,54
	BETO Normal	67,95	55,92	96,54	73,43
	RoBERTa Twitter	83,25	73,46	94,28	71,37
	RoBERTa Normal	67,01	12,80	97,32	67,31
	RoBERTa Cardiff	71,5	55,92	96,82	68,45
	BERTIN Twitter	48,85	0	95,22	60,11
	BERTIN Normal	54,91	0	97,38	61,75

Tabla 3.13 Tabla Validación Metadatos antes del Clasificador (Estrategia MC)

Conjunto Entrenamiento	Modelo	%OFP	%OFG	%NO	%NOM
Train	BETO Twitter	62,82	48,82	97,53	72,15
	BETO Normal	72,69	58,77	96,33	66,38

	RoBERTa Twitter	76,67	63,98	97,07	68,59
	RoBERTa Normal	66,67	27,49	97,34	66,81
	RoBERTa Cardiff	71,24	56,4	96,63	69,87
	BERTIN Twitter	41,28	0	97,76	58,55
	BERTIN Normal	55,00	0	97,35	58,55
Train + Validación	BETO Twitter	66,58	51,18	96,34	71,65
	BETO Normal	68,03	55,45	96,60	72,86
	RoBERTa Twitter	84,91	68,72	93,89	69,09
	RoBERTa Normal	66,79	22,27	97,27	68,02
	RoBERTa Cardiff	72,31	57,82	96,36	72,08
	BERTIN Twitter	46,67	0	96,16	61,75
	BERTIN Normal	55,17	0	97,22	62,11

Tabla 3.14 Tabla Validación Metadatos antes del Clasificador con Textos Preprocesados (Estrategia MCP)

En las tablas superiores (Tablas de la 3.9 hasta 3.14) vemos qué porcentaje de acierto tiene cada modelo en cada una de las etiquetas, y podemos llegar a la conclusión de que la clase menos representada, es en la que más les cuesta a los modelos acertar, habiendo modelos que no llegan a clasificar nunca en estas clases. Este es un problema habitual en tareas de clasificación sobre datos fuertemente desbalanceados, como es nuestro caso.

4 Experimentación, resultados y discusión

4.1 Experimentaciones y pruebas

Una vez que elegimos los modelos (véase Sección 1.8), se plantean las siguientes pruebas:

- Modelos entrenados con la división de los datos “Train” y evaluados con el subconjunto de “Test”. Los modelos que hayan sido entrenados con este conjunto de datos, llevarán a su lado la etiqueta “-T”. Las estrategias seguidas son:
  - Los modelos se entrenan con datos que no contienen información contextual. Aquí solo se usa la columna “comment” como entrada para el sistema. En las gráficas corresponderá con la palabra Base, al tratarse de la forma base de entrenamiento. En la Ilustración 4.1, se puede ver un ejemplo esquemático de la estrategia a seguir:

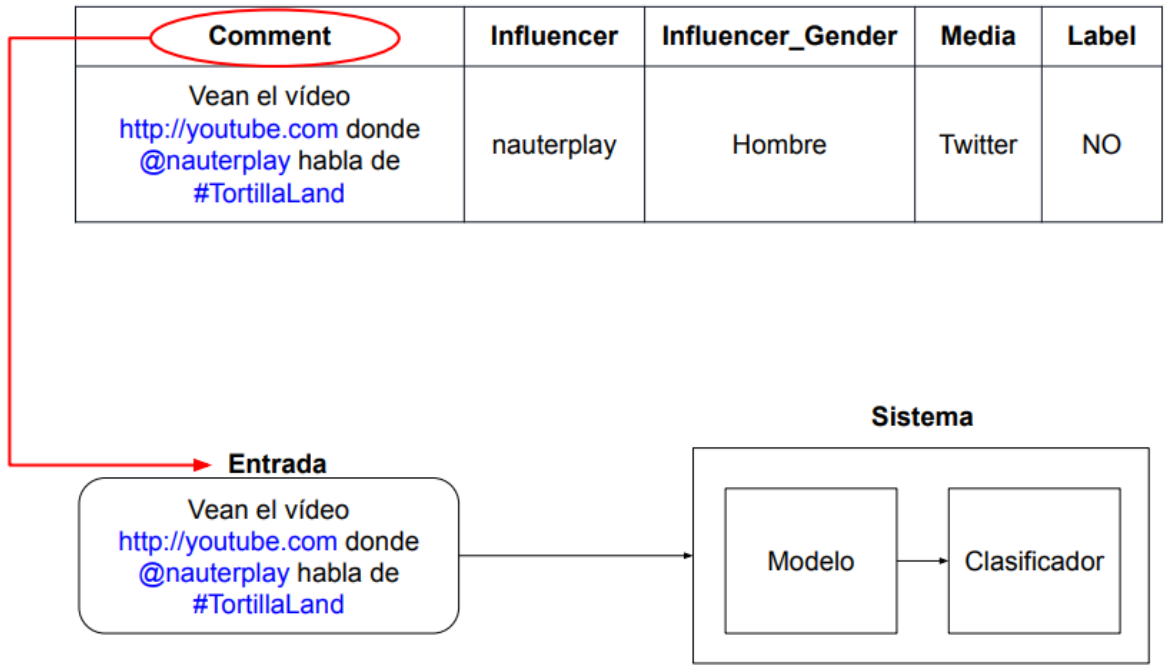

Ilustración 4.1 Representación esquemática estrategia Base

- Los modelos se entrenan con datos que contienen información contextual integrada en el texto. En este caso, la entrada de texto del modelo sería "*influencer influencer_gender media comment*". En las gráficas y tablas siguientes se llamará MT. En la Ilustración 4.2, se puede ver un esquema de la estrategia a seguir:

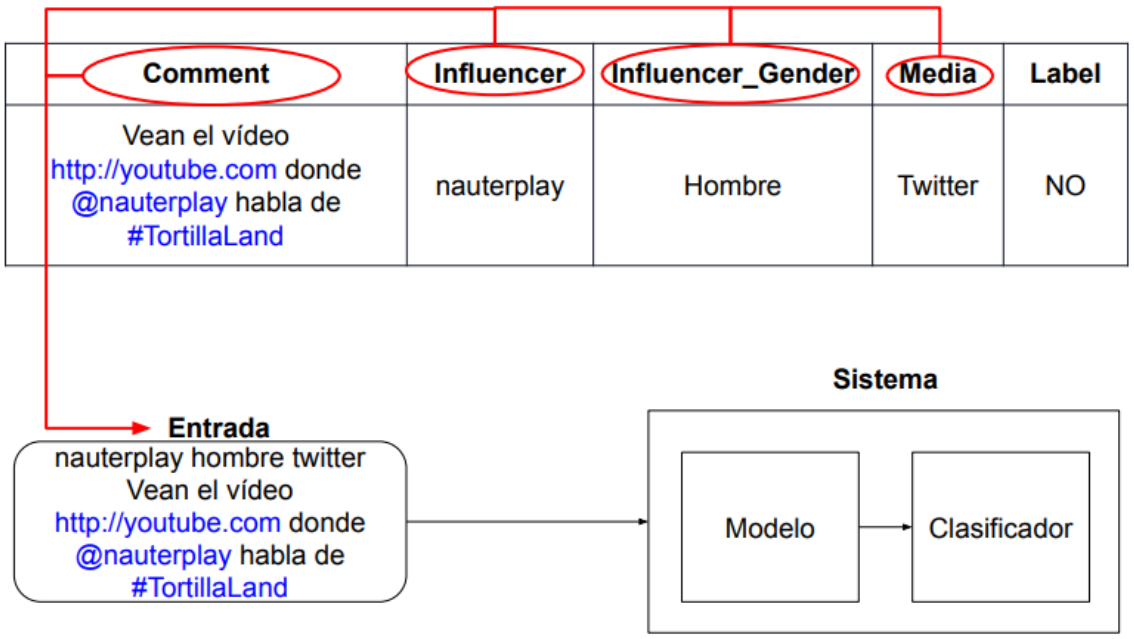

Ilustración 4.2 Representación esquemática estrategia integración de Metadatos en los textos (MT)

- Los modelos se entrenan con datos que no contienen metadatos, como en el primer caso, con la diferencia de que aquí los textos están preprocesados, es decir, a los hashtags se le quitan las almohadillas, las url del estilo http·… son sustituidas por una etiqueta "url" y las menciones, @usuario123, son reemplazadas por @user. En las gráficas y tablas, este tipo de entrenamiento se nombrará como P. En la Ilustración 4.3, se puede ver un esquema de la estrategia a seguir:

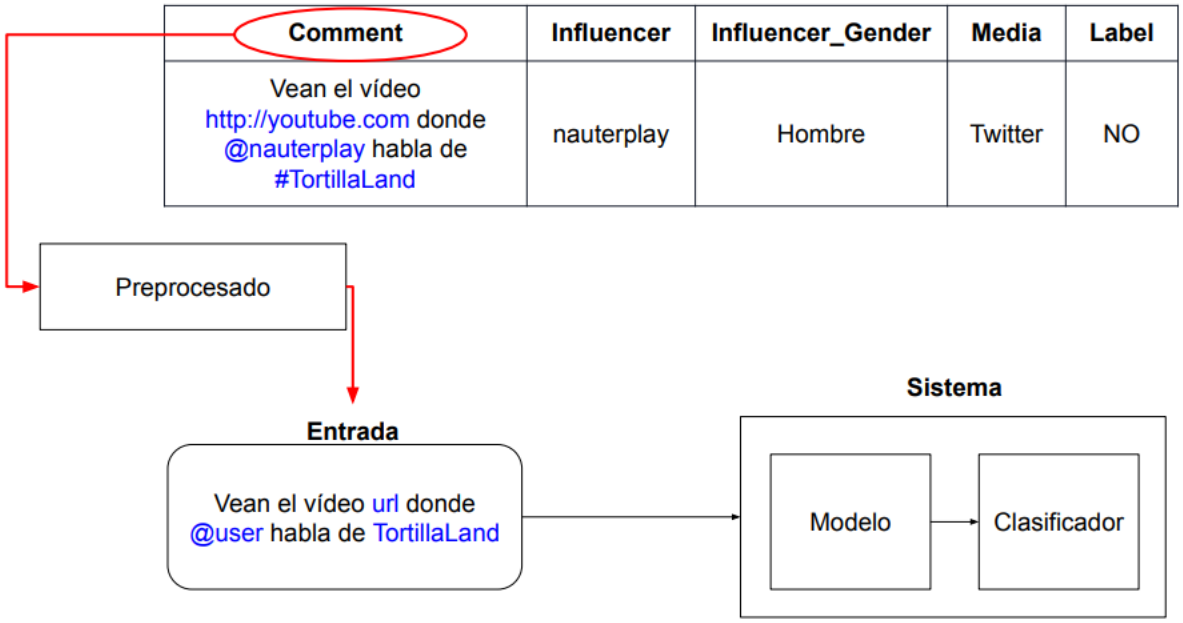

Ilustración 4.3 Representación esquemática estrategia base Preprocesada (P)

- Los modelos reciben como entrada de texto, los metadatos integrados en el comentario, al igual que en la segunda prueba. En este caso, los comentarios están preprocesados. A partir de ahora se llamará MTP. En la Ilustración 4.4, se puede ver un esquema de la estrategia a seguir:

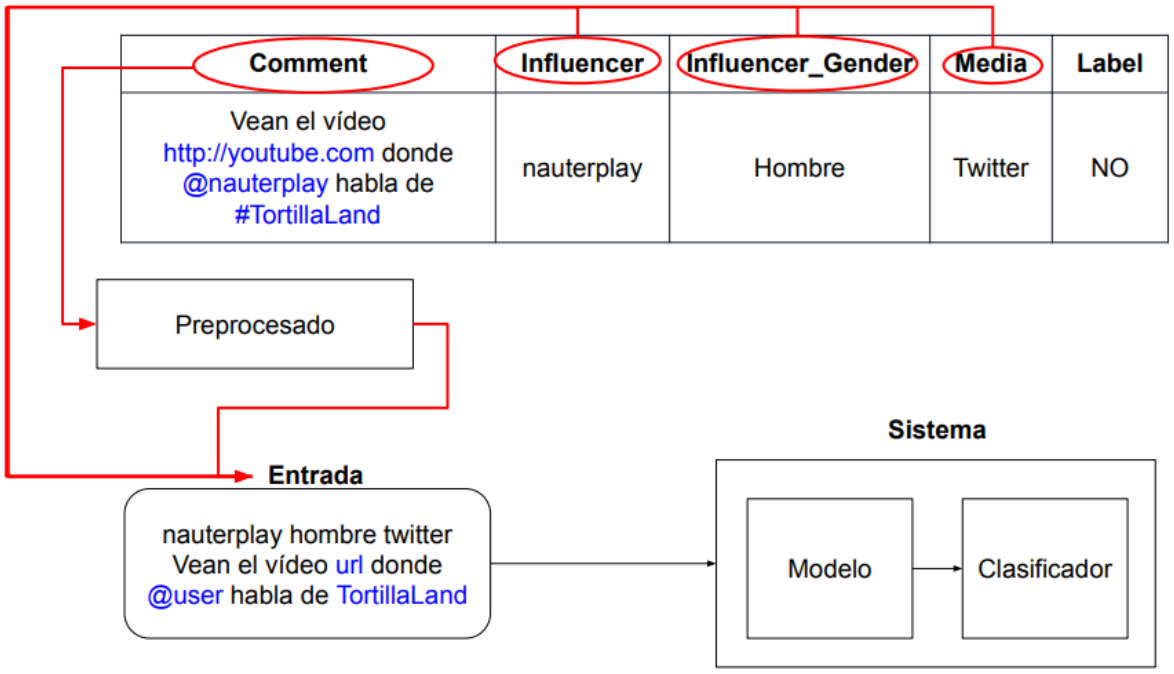

Ilustración 4.4 Representación esquemática estrategia integración de Metadatos en textos Preprocesados (MTP)

- El modelo recibe la información contextual antes de pasar por el clasificador. En este caso el modelo se entrena con la columna comment del dataset y antes de pasar al clasificador los metadatos se añaden al token de clasificación devuelto por el modelo en la última capa. La forma de nombrar a esta estrategia será MC. En la imagen 4.5, se puede ver un esquema de la estrategia a seguir:

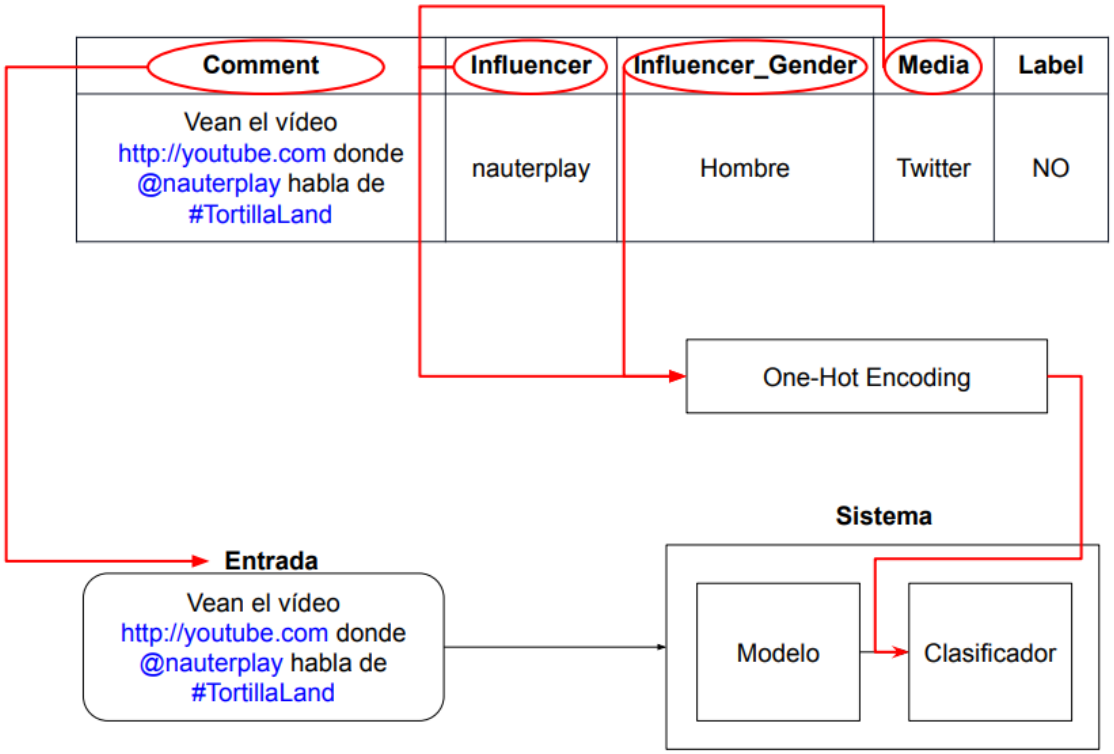

Ilustración 4.5 Representación esquemática estrategia inserción de Metadatos antes del Clasificador (MC)

- La última prueba es igual que la anterior, con la diferencia de que aquí los comentarios están preprocesados. Su nomenclatura a partir de ahora será MCP. En la Ilustración 4.6, se puede ver un esquema de la estrategia a seguir:

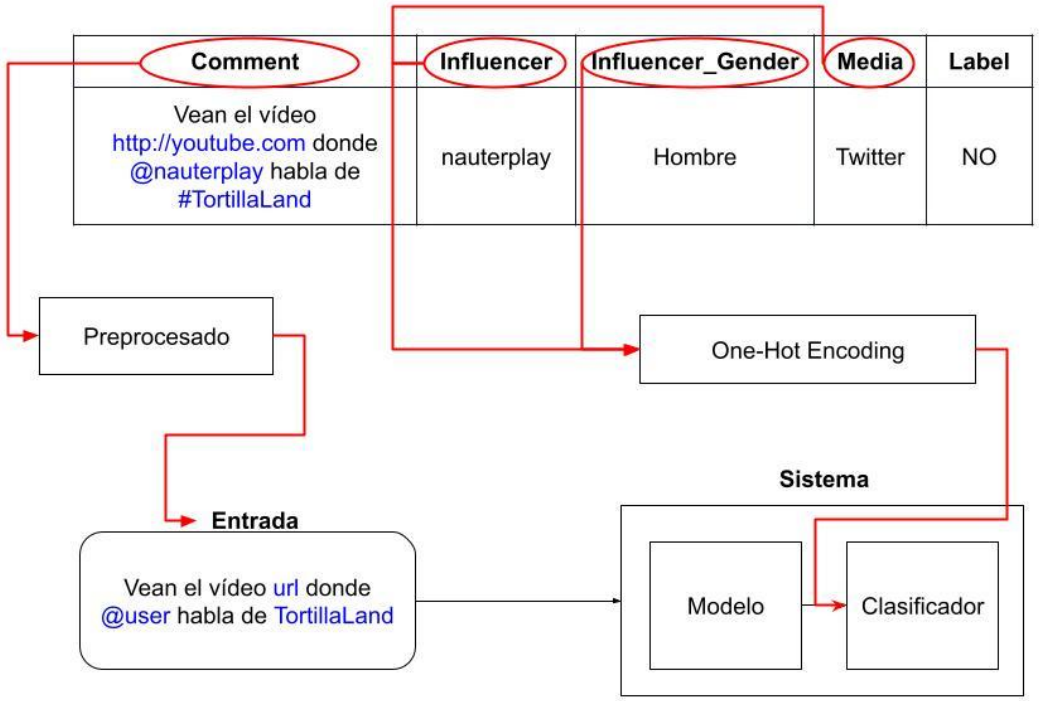

Ilustración 4.6 Representación esquemática estrategia inserción de Metadatos antes del Clasificador con textos Preprocesados (MCP)

- Modelos entrenados con la división del conjunto de datos "Train" y "Validation", además de ser evaluados con la división de "Test". Para estos modelos, se realizan las mismas pruebas que para los modelos que solo están entrenando con los datos de "Train". En este caso, los modelos entrenados con estos dos subconjuntos de datos irán acompañados de "-TV".

A modo de resumen, en la siguiente Ilustración 4.7 podemos ver un esquema de los experimentos planteados.

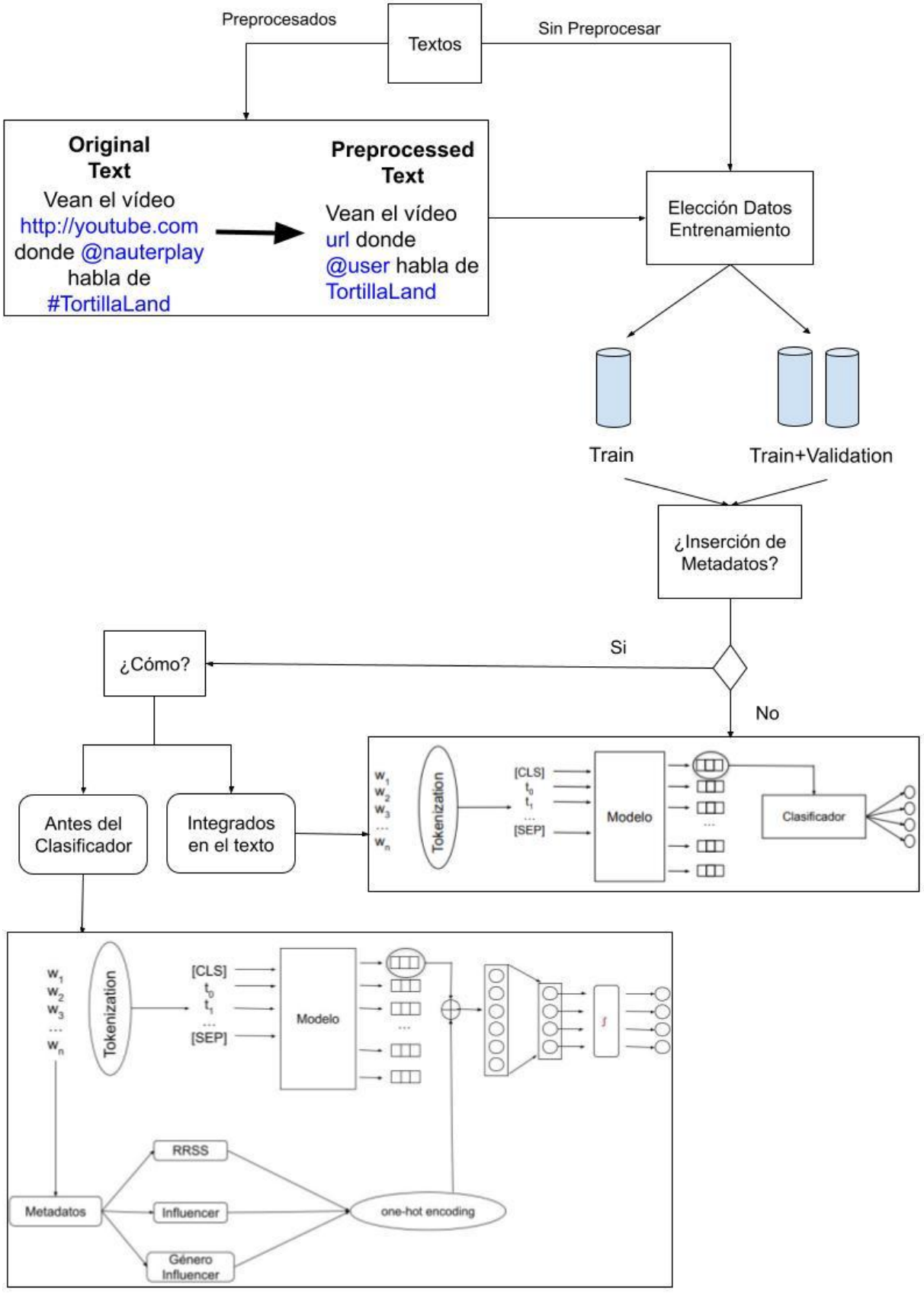
Preprocesados
Textos
Sin Preprocesar
Original Text
Vean el vídeo http://youtube.com donde @nauterplay habla de #TortillaLand
Preprocessed Text
Vean el vídeo url donde @user habla de TortillaLand
Elección Datos Entrenamiento
Train
Train+Validation
¿Inserción de Metadatos?
Si
No
¿Cómo?
Antes del Clasificador
Integrados en el texto
Tokenization
[CLS]
[SEP]
Modelo
Clasificador
RRSS
Metadatos
Influencer
Género Influencer
one-hot encoding

Ilustración 4.7 Representación esquemática experimentos planteados

La finalidad de todas estas pruebas es poder experimentar y concluir:

1. Si insertar más datos en el entrenamiento hace que los modelos funcionen mejor.
2. Si el preprocesado de los datos ayuda a eliminar ruido de los textos y conseguir que los modelos clasifiquen mejor.
3. Si añadir datos contextuales facilita la detección de comentarios ofensivos.
4. Qué forma de añadir información contextual puede ser más efectiva para cada modelo.
5. Si los modelos seleccionados que incorporan adaptación de dominio clasifican mejor que aquellos que son más generales.

4.2 Resultados y discusión

Analicemos ahora los resultados obtenidos tras la ejecución de todos los experimentos. Nos basaremos en ver los resultados de los modelos según el macro f1-score obtenido, pues como dijimos al inicio de este documento, nos da una mayor representación de cómo se clasifica de bien en las cuatro clases de este problema En caso de querer ver de forma detallada los resultados de todos los experimentos realizados, ver apartado 7.2.

Si nos fijamos en las gráficas inferiores (Ilustración 4.8 y 4.9) podemos llegar a las siguientes conclusiones:

1. El modelo que mejor ha funcionado en todas las estrategias ha sido RoBERTa Twitter y, según hemos podido ver, la adaptación de dominio de un modelo es relevante pero sólo para producir pequeñas mejoras

individuales, ya que no todos los modelos han funcionado mejor con esta adaptación de dominio. Los modelos RoBERTa Normal, BERTIN Twitter y BERTIN Normal, no llegan a superar en ningún experimento el mejor resultado de la competición.

2. La incorporación de metadatos, es decir, de un contexto, a los comentarios, es muy importante, ya que ayudan a los modelos a mejorar la clasificación y es fundamental ver de qué forma añadimos estos, ya que, aunque la estrategia a seguir que mejor ha funcionado para la mayor parte de los modelos es añadirlos antes del clasificador, había veces que era más interesante integrarlos en el texto, como ocurre con los modelos RoBERTa Cardiff o BERTIN Twitter.

3. El procesamiento de los datos en este caso, no aporta una mejora notable en las estrategias seguidas. Esto se debe a que muchos de los modelos ya han sido entrenados con los datos tal cual se obtienen, por lo que ya saben cómo tratar aquellos datos que hemos limpiado, como los hashtags, enlaces a sitios web o menciones a otros usuarios de la red social.

4. Los modelos que tienen un tamaño mayor de *batch-size* son los que obtienen mejores resultados en las estrategias seguidas. Podemos ver como RoBERTa Twitter y Beto Normal están primero y tercero respectivamente, en cuanto a los mejores resultados obtenidos.

5. Otra cosa que podemos observar es que entrenar los modelos en un conjunto de datos mayor aunque este incremento sea pequeño, nos ayuda a mejorar un poco los resultados obtenidos tras dicho entrenamiento.

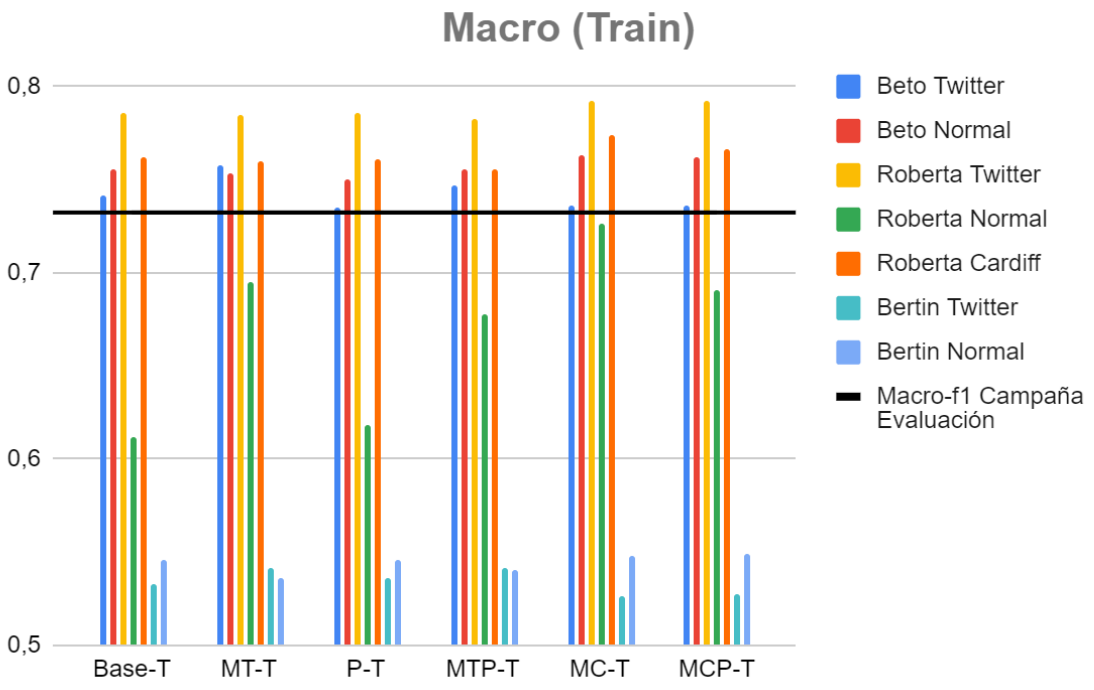

Ilustración 4.8 Valores Macro f1-score para los modelos entrenados con Train

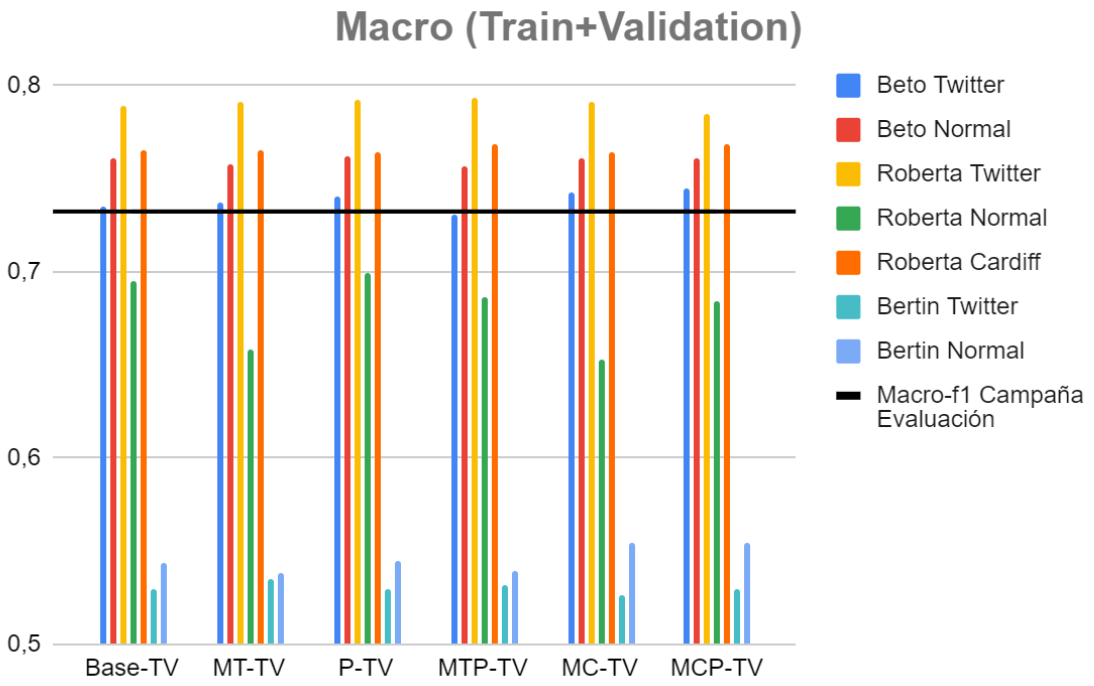

Ilustración 4.9 Valores Macro f1-score para los modelos entrenados con Train y Validation

Finalmente, como partimos de una campaña de evaluación MeOffendEs, que trataba de elaborar sistemas que detecten la ofensividad en comentarios de un dataset proporcionado a los participantes, consideramos importante hacer una pequeña comparativa del mejor resultado obtenido en dicha competición con nuestros sistemas. Visualizando las dos ilustraciones superiores (Ilustración 4.8 y 4.9), podemos ver una línea negra que representa el resultado del mejor equipo de la campaña de evaluación [1], el cual se corresponde al valor 0,7324. Si comparamos este resultado con el de nuestros sistemas, podemos ver como RoBERTa Twitter consigue superarlo con bastante diferencia en todos los experimentos realizados (0,0607 en términos de macro F1-score), seguido por RoBERTA Cardiff y Beto Normal. Beto Twitter consigue resultados más cercanos a este valor quedando en todos los experimentos por encima de este valor, excepto en el que integra los metadatos en el texto preprocesado con el modelo entrenado con los subconjuntos Train y Validación. El resto de modelos con los que hemos experimentado (RoBERTa Normal, BERTIN Normal y BERTIN Twitter) no llegan a alcanzar este valor, quedándose por debajo de 0,7.

5 TRANSFERENCIA TECNOLÓGICA DEL RESULTADO

En este apartado, explicaremos la aplicación que podría tener toda esta experimentación en los sistemas que usamos en nuestro día a día.

Como hemos dicho al inicio de este trabajo y podemos ver a diario, vivimos en un mundo cada vez más digitalizado, donde el uso de las redes sociales está en aumento, lo que provoca que en estos sitios se mueva una gran cantidad de información y se mantengan multitud de conversaciones a la vez. Vamos a ver en qué sitios y por qué puede ser interesante el uso de este sistema de detección de ofensividad:

- En las diferentes redes sociales: cada vez aumentan más los casos de acoso en estos medios y si tenemos un sistema que es capaz de

reconocer estos mensajes de odio hacia una persona o un colectivo, se pueden bloquear estos mensajes y empezar a activar las medidas necesarias para frenarlo.

- En la prensa digital: para controlar que las noticias que se redactan no ofenda a nadie, ya que muchas veces, al redactar lo que ha ocurrido podemos introducir pequeños matices de subjetividad por los que haya gente que se pueda sentir ofendida. Con un sistema que reconozca la ofensividad, podemos controlar que nuestras noticias lleguen a más gente, evitando malentendidos.
- En cualquier página web que permita realizar comentarios: ya que en multitud de ocasiones cuando opinamos sobre algo, no medimos el alcance de nuestras palabras y quizás estemos ofendiendo a alguien, o expandiendo nuestro odio hacia algo.

6 Conclusiones y trabajos futuros

6.1 Conclusiones

Finalmente me gustaría reflejar aquí una serie de conclusiones obtenidas de la realización de este proyecto. Según hemos podido comprobar, los modelos Transformers nos ayudan a realizar diferentes tareas dentro del PLN con resultados realmente asombrosos y sin necesitar siempre de un preprocesamiento de los datos, aunque este a veces es útil para ayudar a los modelos a reconocer los patrones ocultos de nuestro lenguaje.

En cuanto a lo que motivó la realización del trabajo, hemos podido conocer diversos modelos del lenguaje, cada uno con su arquitectura, diferentes formas de añadir datos contextuales y ver cómo la estrategia de inserción de metadatos afectaba a cada modelo. Además, hemos superado el resultado del mejor equipo que participaba en la competición oficial inicial denominada MeOffendES, en la que nos basamos para realizar este proyecto.. Este proyecto, por tanto, establece un

nuevo estado de la cuestión en la detección de la ofensividad para el idioma español.

Por lo que podemos observar, estos modelos no son perfectos pero se asemejan bastante a la realidad, y aunque aún quedan muchas cosas por descubrir y mejorar en este tipo de sistemas, cada día vamos viendo una evolución y avances sorprendentes de esta tecnología.

Durante la realización de este trabajo he podido aprender acerca de los modelos del lenguaje, su arquitectura y funcionamiento, que muchas veces se ven como cajas negras a los que les pasas una serie de datos y ellos te dan una salida con los resultados obtenidos de la tarea a realizar. También he aprendido a realizar búsquedas de hiper-parámetros para obtener los mejores valores de los parámetros establecidos al inicio del entrenamiento de los modelos. He entrado a un nivel de más detalle, modificando la arquitectura del clasificador y logrando entender cómo funcionaban. Además de todo lo detallado con anterioridad, he podido gestionar este proyecto desde dentro, intentando solucionar aquellos problemas que podían hacer que la entrega de este trabajo se retrasase, para intentar ajustarme a la planificación lo máximo posible.

6.2 Trabajos futuros

Como hemos visto a lo largo del desarrollo, se han usado distintas estrategias de integración de metadatos en los modelos. Esto abre camino a investigaciones más profundas que nos digan qué forma de integración de estos datos es mejor. Cabe remarcar la información que estos metadatos aportan, pues nos proporcionan un contexto útil para ayudarnos a clasificar un texto, por lo que, si sabemos integrarlos bien, pueden ser muy útiles.

Además, se podría estudiar cómo afectaría añadir más capas al clasificador, o en vez de utilizar el token de clasificación de la última capa, realizar una

combinación del token de clasificación de las últimas *n* capas. O bien, mezclar las dos estrategias anteriores.

Por otro lado, sabemos que los sentimientos y las emociones están muy relacionados con la ofensividad, por lo que sería interesante comprobar si los modelos que han sido entrenados para reconocer estas emociones y sentimientos son más eficaces a la hora de reconocer la ofensividad. Otra idea sería estudiar cómo podemos integrar datos relacionados con las emociones, el contexto en el que se escribe el mensaje, la forma de ser de la persona, a quién está el comentario dirigido, ya que no hablamos igual con personas a las que no conocemos que con personas con las que tenemos confianza. El objetivo de este estudio estaría en ver cómo proporcionarle al modelo más información que nos dé pistas sobre la intención del mensaje a analizar, y así poder ayudar al sistema a reconocer si verdaderamente se trata de un comentario ofensivo.

También hemos podido observar que con más datos para entrenar un modelo, aunque sean pocos, los modelos suelen tener una mejoría. Por tanto, sería buena idea para continuar en el futuro el explorar técnicas de aumento de datos, como tener los mismos datos con traducciones a otros idiomas y volver a traducir estos comentarios en español, para así tener textos similares con distintas variaciones.

Otro trabajo a realizar en un futuro es tratar este problema como un problema de regresión, ya que la ofensividad es algo subjetivo y que depende de la persona que recibe el comentario. Un comentario me puede parecer ofensivo y el mismo comentario para otra persona puede no serlo. En la campaña de evaluación, que supuso un punto de partida para este trabajo, los comentarios tenían una probabilidad de pertenencia a cada clase. Esta probabilidad se establecía realizando una media entre el número de anotadores que elegían la clase a la que pertenecía entre el total de anotadores. Entonces, podemos utilizar esos datos que incluyen la probabilidad de pertenencia del comentario a cada clase (nivel de confianza) para realizar un problema regresión y comprobar si así los modelos obtienen mejores resultados.

7 Apéndices

7.1 Guía original del Trabajo Fin de Título

(Cód.: 21/22-3631) Detección de la ofensividad en redes sociales mediante lenguaje natural e información contextual

Tutor del TFG: **ARTURO MONTEJO RAEZ**
Modalidad: Trabajo Teórico/Experimental | Tipo: TFG Específico
Número máximo de estudiantes: 1 (1 asignados)
Idioma: Castellano
Asignado al alumno con DNI:26053388T
Segundo tutor del TFG: **FLOR MIRIAM PLAZA DEL ARCO**

La detección de la ofensividad se ha centrado en el análisis de textos mediante tecnologías del lenguaje. El presente proyecto pretende abordar la integración de información contextual (perfil del usuario, historial de publicaciones, etc.) para mejorar los sistemas de clasificación tradicionales.

Conocimientos Previos

Haber cursado la asignatura de "procesamiento del lenguaje natural".

Objetivos del TFG

1. Desarrollar un sistema de detección de lenguaje ofensivo con tecnologías del lenguaje
2. Estudiar características adicionales al contenido del texto a clasificar que doten de contexto al mensaje
3. Realizar diversos experimentos para evaluar las características exploradas.

Metodología a Desarrollar

1. Estudio del estado de la cuestión
2. Diseño de los algoritmos
3. Preparación de los datos
4. Diseño de los experimentos
5. Evaluación y análisis

Documentos y Formatos de Entrega

1. Memoria del proyecto en PDF
2. Repositorio GitHub con el código desarrollado

7.2 Anexo con los resultados obtenidos en la experimentación

En este anexo se muestran unas tablas con los resultados obtenidos para todos los experimentos realizados. Podemos ver en verde aquellos resultados que mejoran al equipo que quedó primero en la campaña de evaluación desde la que parte este proyecto y de rojo para aquellos resultados que no logran superar al equipo con los mejores resultados.

7.2.1 Resultados de los Experimentos entrenados con *Train.*

Sin Metadatos									
Model	macro-f1	macro-precision	macro-recall	mIcro-f1	mIcro-precision	mIcro-recall	weighted-f1	weighted-precision	weighted-recall
Beto Twitter	0,7412	0,7760	0,7188	0,8793	0,8793	0,8793	0,8735	0,8774	0,8793
Beto Normal	0,7560	0,7870	0,7310	0,8851	0,8851	0,8851	0,8814	0,8812	0,8851
Roberta Twitter	0,7858	0,7984	0,7777	0,8973	0,8973	0,8973	0,8948	0,8953	0,8973
Roberta Normal	0,6123	0,7775	0,5845	0,8769	0,8769	0,8769	0,8655	0,8682	0,8769
Roberta Cardiff	0,7621	0,7726	0,7587	0,8897	0,8897	0,8897	0,8870	0,8878	0,8897
Bertin Twitter	0,5333	0,5587	0,5197	0,8293	0,8293	0,8293	0,8116	0,8044	0,8293
Bertin Normal	0,5457	0,5761	0,5250	0,8460	0,8460	0,8460	0,8301	0,8218	0,8460

Con Metadatos Integrados en el Texto									
Model	macro-f1	macro-precision	macro-recall	mIcro-f1	mIcro-precision	mIcro-recall	weighted-f1	weighted-precision	weighted-recall
Beto Twitter	0,7576	0,7894	0,7345	0,8844	0,8844	0,8844	0,8795	0,8815	0,8844
Beto Normal	0,7532	0,7946	0,7201	0,8853	0,8853	0,8853	0,8811	0,8809	0,8853
Roberta Twitter	0,7844	0,7994	0,7742	0,8970	0,8970	0,8970	0,8946	0,8948	0,8970
Roberta Normal	0,6946	0,8111	0,6418	0,8795	0,8795	0,8795	0,8714	0,8738	0,8795
Roberta Cardiff	0,7601	0,7721	0,7501	0,8888	0,8888	0,8888	0,8867	0,8861	0,8888
Bertin Twitter	0,5417	0,5597	0,5282	0,8298	0,8298	0,8298	0,8169	0,8078	0,8298
Bertin Normal	0,5359	0,5714	0,5141	0,8363	0,8363	0,8363	0,8177	0,8108	0,8363

Sin Metadatos Preprocesado									
Model	macro-f1	macro-precision	macro-recall	mIcro-f1	mIcro-precision	mIcro-recall	weighted-f1	weighted-precision	weighted-recall
Beto Twitter	0,7352	0,7794	0,7033	0,8809	0,8809	0,8809	0,8763	0,8768	0,8809
Beto Normal	0,7503	0,7879	0,7200	0,8831	0,8831	0,8831	0,8785	0,8785	0,8831
Roberta Twitter	0,7861	0,7872	0,7890	0,8988	0,8988	0,8988	0,8975	0,8977	0,8988
Roberta Normal	0,6178	0,8481	0,5897	0,8792	0,8792	0,8792	0,8679	0,8750	0,8792
Roberta Cardiff	0,7606	0,7878	0,7386	0,8889	0,8889	0,8889	0,8858	0,8859	0,8889
Bertin Twitter	0,5359	0,5674	0,5151	0,8313	0,8313	0,8313	0,8146	0,8069	0,8313
Bertin Normal	0,5462	0,5752	0,5262	0,8462	0,8462	0,8462	0,8306	0,8222	0,8462

Con Metadatos Integrados en el Texto Preprocesado									
Model	macro-f1	macro-precision	macro-recall	mIcro-f1	mIcro-precision	mIcro-recall	weighted-f1	weighted-precision	weighted-recall
Beto Twitter	0,7466	0,7812	0,7194	0,8793	0,8793	0,8793	0,8747	0,8754	0,8793
Beto Normal	0,7553	0,7977	0,7216	0,8856	0,8856	0,8856	0,8812	0,8812	0,8856
Roberta Twitter	0,7827	0,8113	0,7592	0,8988	0,8988	0,8988	0,8956	0,8958	0,8988
Roberta Normal	0,6782	0,8160	0,6280	0,8785	0,8785	0,8785	0,8699	0,8726	0,8785
Roberta Cardiff	0,7560	0,7693	0,7447	0,8880	0,8880	0,8880	0,8859	0,8852	0,8880
Bertin Twitter	0,5420	0,5592	0,5290	0,8299	0,8299	0,8299	0,8169	0,8078	0,8299
Bertin Normal	0,5406	0,5787	0,5176	0,8398	0,8398	0,8398	0,8209	0,8151	0,8398

Con Metadatos Antes del Clasificador									
Model	macro-f1	macro-precision	macro-recall	mIcro-f1	mIcro-precision	mIcro-recall	weighted-f1	weighted-precision	weighted-recall
Beto Twitter	0,7366	0,7806	0,7066	0,8806	0,8806	0,8806	0,8741	0,8784	0,8806
Beto Normal	0,7634	0,7967	0,7363	0,8859	0,8859	0,8859	0,8822	0,8819	0,8859
Roberta Twitter	0,7920	0,8230	0,7661	0,9004	0,9004	0,9004	0,8974	0,8975	0,9004
Roberta Normal	0,7262	0,8045	0,6768	0,8827	0,8827	0,8827	0,8764	0,8770	0,8827
Roberta Cardiff	0,7737	0,8013	0,7506	0,8916	0,8916	0,8916	0,8882	0,8881	0,8916
Bertin Twitter	0,5266	0,5771	0,5036	0,8288	0,8288	0,8288	0,8056	0,8055	0,8288
Bertin Normal	0,5485	0,5809	0,5267	0,8457	0,8457	0,8457	0,8297	0,8222	0,8457

Con Metadatos Antes del Clasificador Preprocesado									
Model	macro-f1	macro-precision	macro-recall	mIcro-f1	mIcro-precision	mIcro-recall	weighted-f1	weighted-precision	weighted-recall
Beto Twitter	0,7357	0,7808	0,7033	0,8819	0,8819	0,8819	0,8757	0,8787	0,8819
Beto Normal	0,7621	0,7949	0,7354	0,8859	0,8859	0,8859	0,8823	0,8820	0,8859
Roberta Twitter	0,7927	0,8258	0,7658	0,9011	0,9011	0,9011	0,8979	0,8983	0,9011
Roberta Normal	0,6909	0,7654	0,6458	0,8783	0,8783	0,8783	0,8712	0,8710	0,8783
Roberta Cardiff	0,7664	0,8041	0,7354	0,8888	0,8888	0,8888	0,8850	0,8848	0,8888
Bertin Twitter	0,5277	0,5775	0,5050	0,8294	0,8294	0,8294	0,8063	0,8062	0,8294
Bertin Normal	0,5487	0,5804	0,5272	0,8455	0,8455	0,8455	0,8297	0,8221	0,8455

7.2.2 Resultados de los Experimentos entrenados con *Train* y *Validation*.

Sin Metadatos									
Model	macro-f1	macro-precision	macro-recall	mIcro-f1	mIcro-precision	mIcro-recall	weighted-f1	weighted-precision	weighted-recall
Beto Twitter	0,7352	0,7825	0,6997	0,8791	0,8791	0,8791	0,8740	0,8744	0,8791
Beto Normal	0,7614	0,7929	0,7353	0,8868	0,8868	0,8868	0,8831	0,8830	0,8868
Roberta Twitter	0,7894	0,7928	0,7914	0,9000	0,9000	0,9000	0,8988	0,8990	0,9000
Roberta Normal	0,6954	0,8051	0,6452	0,8819	0,8819	0,8819	0,8746	0,8757	0,8819
Roberta Cardiff	0,7653	0,7845	0,7487	0,8892	0,8892	0,8892	0,8867	0,8861	0,8892
Bertin Twitter	0,5293	0,5743	0,5067	0,8303	0,8303	0,8303	0,8088	0,8065	0,8303
Bertin Normal	0,5442	0,5750	0,5240	0,8436	0,8436	0,8436	0,8267	0,8191	0,8436

Con Metadatos Integrados en el Texto									
Model	macro-f1	macro-precision	macro-recall	mIcro-f1	mIcro-precision	mIcro-recall	weighted-f1	weighted-precision	weighted-recall
Beto Twitter	0,7376	0,7811	0,7043	0,8810	0,8810	0,8810	0,8762	0,8767	0,8810
Beto Normal	0,7580	0,7958	0,7286	0,8854	0,8854	0,8854	0,8808	0,8817	0,8854
Roberta Twitter	0,7911	0,8063	0,7775	0,8998	0,8998	0,8998	0,8982	0,8975	0,8998
Roberta Normal	0,6585	0,7820	0,6249	0,8806	0,8806	0,8806	0,8727	0,8738	0,8806
Roberta Cardiff	0,7657	0,7767	0,7570	0,8893	0,8893	0,8893	0,8873	0,8872	0,8893
Bertin Twitter	0,5346	0,5771	0,5122	0,8326	0,8326	0,8326	0,8120	0,8087	0,8326
Bertin Normal	0,5383	0,5595	0,5228	0,8339	0,8339	0,8339	0,8198	0,8105	0,8339

Sin Metadatos Preprocesado									
Model	macro-f1	macro-precision	macro-recall	mIcro-f1	mIcro-precision	mIcro-recall	weighted-f1	weighted-precision	weighted-recall
Beto Twitter	0,7404	0,7757	0,7175	0,8776	0,8776	0,8776	0,8724	0,8736	0,8776
Beto Normal	0,7624	0,7937	0,7364	0,8868	0,8868	0,8868	0,8831	0,8829	0,8868
Roberta Twitter	0,7929	0,7866	0,8015	0,8975	0,8975	0,8975	0,8977	0,8983	0,8975
Roberta Normal	0,6996	0,8077	0,6489	0,8817	0,8817	0,8817	0,8746	0,8756	0,8817
Roberta Cardiff	0,7638	0,7752	0,7546	0,8892	0,8892	0,8892	0,8873	0,8871	0,8892
Bertin Twitter	0,5297	0,5746	0,5073	0,8307	0,8307	0,8307	0,8093	0,8072	0,8307
Bertin Normal	0,5453	0,5812	0,5226	0,8446	0,8446	0,8446	0,8269	0,8203	0,8446

Con Metadatos Integrados en el Texto Preprocesado									
Model	macro-f1	macro-precision	macro-recall	micro-f1	micro-precision	micro-recall	weighted-f1	weighted-precision	weighted-recall
Beto Twitter	0,7304	0,7679	0,7012	0,8748	0,8748	0,8748	0,8697	0,8702	0,8748
Beto Normal	0,7570	0,7921	0,7296	0,8853	0,8853	0,8853	0,8807	0,8816	0,8853
Roberta Twitter	0,7931	0,7978	0,7902	0,8982	0,8982	0,8982	0,8973	0,8970	0,8982
Roberta Normal	0,6865	0,7803	0,6412	0,8790	0,8790	0,8790	0,8716	0,8722	0,8790
Roberta Cardiff	0,7680	0,7823	0,7560	0,8902	0,8902	0,8902	0,8883	0,8877	0,8902
Bertin Twitter	0,5322	0,5767	0,5096	0,8320	0,8320	0,8320	0,8107	0,8082	0,8320
Bertin Normal	0,5397	0,5689	0,5200	0,8378	0,8378	0,8378	0,8214	0,8130	0,8378

Con Metadatos Antes del Clasificador									
Model	macro-f1	macro-precision	macro-recall	micro-f1	micro-precision	micro-recall	weighted-f1	weighted-precision	weighted-recall
Beto Twitter	0,7421	0,7814	0,7129	0,8811	0,8811	0,8811	0,8772	0,8776	0,8811
Beto Normal	0,7613	0,7945	0,7346	0,8861	0,8861	0,8861	0,8821	0,8822	0,8861
Roberta Twitter	0,7911	0,7843	0,8059	0,8970	0,8970	0,8970	0,8971	0,8991	0,8970
Roberta Normal	0,6531	0,8114	0,6111	0,8770	0,8770	0,8770	0,8679	0,8704	0,8770
Roberta Cardiff	0,7645	0,8049	0,7317	0,8890	0,8890	0,8890	0,8849	0,8850	0,8890
Bertin Twitter	0,5263	0,5493	0,5105	0,8215	0,8215	0,8215	0,8057	0,7964	0,8215
Bertin Normal	0,5544	0,5828	0,5351	0,8489	0,8489	0,8489	0,8335	0,8260	0,8489

Con Metadatos Antes del Clasificador Preprocesado									
Model	macro-f1	macro-precision	macro-recall	micro-f1	micro-precision	micro-recall	weighted-f1	weighted-precision	weighted-recall
Beto Twitter	0,7443	0,7829	0,7144	0,8798	0,8798	0,8798	0,8753	0,8755	0,8798
Beto Normal	0,7611	0,7967	0,7324	0,8860	0,8860	0,8860	0,8819	0,8820	0,8860
Roberta Twitter	0,7844	0,7853	0,7915	0,8939	0,8939	0,8939	0,8942	0,8975	0,8939
Roberta Normal	0,6846	0,7997	0,6359	0,8785	0,8785	0,8785	0,8708	0,8720	0,8785
Roberta Cardiff	0,7683	0,7936	0,7464	0,8912	0,8912	0,8912	0,8883	0,8876	0,8912
Bertin Twitter	0,5297	0,5595	0,5114	0,8260	0,8260	0,8260	0,8080	0,8004	0,8260
Bertin Normal	0,5543	0,5805	0,5363	0,8486	0,8486	0,8486	0,8336	0,8258	0,8486

7.2.3 Tabla resumen de los resultados obtenidos para las métricas evaluadas.

Macro-f1												
Model	Base-T	Base-TV	MT-T	MT-TV	P-T	P-TV	MTP-T	MTP-TV	MC-T	MC-TV	MCP-T	MCP-TV
Beto Twitter	0,7412	0,7352	0,7576	0,7376	0,7352	0,7404	0,7466	0,7304	0,7366	0,7421	0,7357	0,7443
Beto Normal	0,7560	0,7614	0,7532	0,7580	0,7503	0,7624	0,7553	0,7570	0,7634	0,7613	0,7621	0,7611
Roberta Twitter	0,7858	0,7894	0,7844	0,7911	0,7861	0,7929	0,7827	0,7931	0,7920	0,7911	0,7927	0,7844
Roberta Normal	0,6123	0,6954	0,6946	0,6585	0,6178	0,6996	0,6782	0,6865	0,7262	0,6531	0,6909	0,6846
Roberta Cardiff	0,7621	0,7653	0,7601	0,7657	0,7606	0,7638	0,7560	0,7680	0,7737	0,7645	0,7664	0,7683
Bertin Twitter	0,5333	0,5293	0,5417	0,5346	0,5359	0,5297	0,5420	0,5322	0,5266	0,5263	0,5277	0,5297
Bertin Normal	0,5457	0,5442	0,5359	0,5383	0,5462	0,5453	0,5406	0,5397	0,5485	0,5544	0,5487	0,5543
Macro-f1 Campaña Evaluación	0,7324	0,7324	0,7324	0,7324	0,7324	0,7324	0,7324	0,7324	0,7324	0,7324	0,7324	0,7324

Micro-f1												
Model	Base-T	Base-TV	MT-T	MT-TV	P-T	P-TV	MTP-T	MTP-TV	MC-T	MC-TV	MCP-T	MCP-TV
Beto Twitter	0,8793	0,8791	0,8844	0,8810	0,8809	0,8776	0,8793	0,8748	0,8806	0,8811	0,8819	0,8798
Beto Normal	0,8851	0,8868	0,8853	0,8854	0,8831	0,8868	0,8856	0,8853	0,8859	0,8861	0,8859	0,8860
Roberta Twitter	0,8973	0,9000	0,8970	0,8998	0,8988	0,8975	0,8988	0,8982	0,9004	0,8970	0,9011	0,8939
Roberta Normal	0,8769	0,8819	0,8795	0,8806	0,8792	0,8817	0,8785	0,8790	0,8827	0,8770	0,8783	0,8785
Roberta Cardiff	0,8897	0,8892	0,8888	0,8893	0,8889	0,8892	0,8880	0,8902	0,8916	0,8890	0,8888	0,8912
Bertin Twitter	0,8293	0,8303	0,8298	0,8326	0,8313	0,8307	0,8299	0,8320	0,8288	0,8215	0,8294	0,8260
Bertin Normal	0,8460	0,8436	0,8363	0,8339	0,8462	0,8446	0,8398	0,8378	0,8457	0,8489	0,8455	0,8486
Micro-f1 Campaña Evaluación	0,8815	0,8815	0,8815	0,8815	0,8815	0,8815	0,8815	0,8815	0,8815	0,8815	0,8815	0,8815

8 DEFINICIONES Y ABREVIATURAS

Las abreviaturas y su correspondiente significado que nos encontramos en este proyecto son:

- ALBERT: A Little BERT (Bidirectional Encoder Representation from Transformers), es un modelo del lenguaje derivado del modelo BERT.
- API: Interfaz de Programación de Aplicaciones
- BART: es otro modelo de lenguaje que elimina el ruido de los pre-entrenamientos de los modelos secuencia a secuencia para generación del lenguaje, traducción y comprensión.
- Batch-size: establece el tamaño de lote que va a entrenar a la red, es decir, el número de comentarios que van a entrar a la vez para entrenar un modelo. Esto se hace porque si tenemos millones de textos, necesitaríamos cargar estos datos en memoria y a veces no tenemos tanta memoria, entonces la mejor solución es hacer pequeños conjuntos de datos y así lograr que estos textos pasen poco a poco por la red neuronal en su entrenamiento, reduciendo la cantidad de memoria necesitada.
- BERT: Bidirectional Encoder Representation from Transformers

- BERTIN: Modelo de lenguaje muy similar a BERT con pequeñas modificaciones.
- CLS: significa clasificación, y es el primer token que asina un tokenizador al texto que este debe tokenizar.
- DGT: Dirección General de Tráfico.
- Época: es el número de ciclos que se deben completar en el entrenamiento de los modelos. Una época se completa cuando todos los datos de entrenamiento han pasado ya por el modelo.
- GPT: Sus siglas significan Generative Pre-Trained Transformers, y es un modelo de lenguaje.
- GPU: Graphics Processing Unit, en inglés. En español sería Unidad de Procesamiento Gráfico.
- Hiper-parámetro:
- IberLEF: Iberian Languages Evaluation Forum
- Iteración: una iteración dentro de una época, indica cuántos lotes de datos deben de entrar por la red neuronal para completar un ciclo, de forma que en un ciclo todos los lotes hayan entrado a la red neuronal.
- Learning rate: es un parámetro que se puede establecer dentro de los entrenamientos de los modelos. El learning rate se encarga de actualizar los pesos del modelo según el valor de error estimado obtenido en su entrenamiento. Un tamaño grande de learning rate significa que los pesos de la red neuronal van a cambiar mucho, logrando así que el ajuste de estos pesos no se estabilice, mientras que si este valor es muy pequeño, podríamos no actualizar los pesos y quedarnos atascados en el entrenamiento de la red neuronal.
- LOC: Lines of Code, en español conocido por Líneas de Código.
- mC4: Conjunto de datos Multilingüe Common Crawl.
- MC: Abreviatura de la estrategia aplicada a los modelos que llevan introducen los Metadatos antes del Clasificador.
- MCP: Abreviatura de la estrategia aplicada a los modelos que llevan introducen los Metadatos antes del Clasificador con textos Preprocesados.
- Modelo: Un modelo del lenguaje es un sistema, basado en redes neuronales profundas, que se encarga de analizar una gran cantidad de

texto y así poder extraer los patrones que estos textos contienen y lograr aprender del lenguaje contenido en los textos a analizar.

- MT: Abreviatura de la estrategia aplicada a los modelos que llevan incorporados los Metadatos con el Texto.
- MTP: Abreviatura de la estrategia aplicada a los modelos que llevan incorporados los Metadatos con el Texto Preprocesado.
- NO: Etiqueta asignada a comentarios No Ofensivos.
- NOM: Etiqueta asignada a comentarios No Ofensivos Malsonantes
- OFP: Etiqueta asignada a comentarios OFensivos dirigidos a una Persona.
- OFG: Etiqueta asignada a comentarios OFensivos dirigidos a un Grupo o colectivo.
- P: Abreviatura de la estrategia aplicada a los modelos que son entrenados con los textos preprocesados
- PBT: Se refiere a los algoritmos de entrenamiento basados en población, proviene del inglés (Population Based Training)
- PLN: Procesamiento del Lenguaje Natural.
- POS: Part of Speech. Son las etiquetas que indican la categoría gramatical de las palabras que forman un texto.
- RAM: Del inglés, Random Access Memory. Es la memoria de corto plazo que tienen los ordenadores.
- RoBERTa: Robustly Optimized BERT Pretraining Approach
- SEP: Un token de separación que introducen los tokenizadores para separar dos o más textos de entrada o al final de un texto.
- SVM: Support Vector Machine
- SEPLN: Sociedad Española de Procesamiento del Lenguaje Natural
- T5: Text-To-Text Transfer Transformers, un modelo de lenguaje.
- TFG: Trabajo Fin de Grado.
- Train: división del corpus, necesaria para entrenar los modelos y ajustarlos a la tarea de clasificación que deben realizar. El número total de textos que nos encontramos en este subconjunto es de dieciséis mil setecientos once.

- Test: división del conjunto de datos utilizada para evaluar los sistemas finales. Está formada por trece mil seiscientos seis textos.
- Validation: división del conjunto de datos utilizada para ajustar los hiper-parámetros de los modelos y en algunos experimentos para tener más datos con los que entrenar a los modelos. Esta división está formada por cien comentarios.

9 BIBLIOGRAFÍA

[1] Flor Miriam Plaza-del-Arco, Marco Casavantes, Hugo Jair Escalante, M. Teresa Martín-Valdivia, Arturo Montejo-Ráez, Manuel Montes-y-Gómez, Horacio Jarquín-Vásquez, Luis Villaseñor-Pineda. Overview of MeOffendEs at IberLEF 2021: Offensive Language Detection in Spanish Variants. http://journal.sepln.org/sepln/ojs/ojs/index.php/pln/article/view/6388

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. ArXiv, abs/1810.04805, 2018.

[3] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach. ArXiv, abs/1907.11692, 2019.

[4] Javier de la Rosa, Eduardo G. Ponferrada, Paulo Villegas, Pablo Gonzalez de Prado Salas, Manu Romero, Maria Grandury. BERTIN: Efficient Pre-Training of a Spanish Language Model using Perplexity Sampling. ArXiv, abs/2207.06814, 2022.

[5] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, Luke Zettlemoyer. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation and Comprehension. ArXiv, abs/1910.13461, 2019.

[6] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, Radu Soricut. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. ArXiv, abs/1909.11942, 2019

[7] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J. Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. ArXiv, abs/1910.10683, 2019.

[8] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, Veselin Stoyanov. Unsupervised Cross-lingual Representation Learning at Scale. ArXiv, abs/1911.02116, 2019.

[9] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, Dario Amodei. Language Models are Few-Shot Learners. ArXiv, abs/2005.14165 ,2020

[10] OpenAI. GPT-3. URL: https://beta.openai.com/docs/models/gpt-3, .Comprobado el 23-09-2022

[11] Flor Miriam Plaza-del-Arco, Arturo Montejo-Ráez, L. Alfonso Ureña-López, María-Teresa Martín-Valdivia. OffendES: A New Corpus in Spanish for Offensive Language Research, 2021. URL: https://aclanthology.org/2021.ranlp-1.123.pdf

[12] José Cañete, Gabriel Chaperon, Rodrigo Fuentes, Jou-Hui Ho, Hojin Kang, Jorge Perez. Spanish pre-trained BERT model and evaluation data, 2020. PML4DC, ICLR 2020. URL: https://users.dcc.uchile.cl/~jperez/papers/pml4dc2020.pdf

[13] The pandas development team. Pandas, 2020 pandas - Biblioteca de análisis de datos de Python (pydata.org)

[14] Hunter, J. D. Matplotlib: A 2D graphics environment, 2007. URL: Matplotlib — Visualization with Python

[15] Steven Bird, Ewan Klein, and Edward Loper. Natural Language Toolkit, 2009. URL: https://www.nltk.org/

[16] Fabian Pedregosa *et al*. Scikit-learn: Machine Learning in Python, 2011. URL:https://scikit-learn.org/stable/

[17] NumPy Community. NumPy. URL:https://numpy.org/doc/stable/

[18] MLflow Community. MLflow. URL: https://mlflow.org/

[19] Hugging Face. Datasets. URL: https://huggingface.co/docs/datasets/index

[20] Google. Google Colaboratory. URL: https://colab.research.google.com/?utm_source=scs-index

[21] ngrok, Inc. Ngrok. URL: https://ngrok.com/

[22] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta,and Masanori Koyama. Optuna: A Next-generation Hyperparameter Optimization Framework, 2019. URL: https://optuna.org/

[23] Andreas Mueller. WordCloud. URL: https://pypi.org/project/wordcloud/. Comprobado el 23-09-2022 .

[24] Optuna Contributors. Efficient Optimization Algorithms. URL: https://optuna.readthedocs.io/en/stable/tutorial/10_key_features/003_efficient_optimization_algorithms.html

[25] El blog de investigación de inteligencia artificial de Berkeley. La importancia de la optimización de hiperparámetros para el aprendizaje por refuerzo basado en modelos. URL: https://topbigdata.es/la-importancia-de-la-optimizacion-de-hiperparametros-para-el-aprendizaje-por-refuerzo-basado-en-modelos-el-blog-de-investigacion-de-inteligencia-artificial-de-berkeley/. Comprobado el 23-09-2022

[26] Li Yang, Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice, 2020. URL: https://www.sciencedirect.com/science/article/pii/S0925231220311693?casa_token=XiucaTovXy8AAAAA:H1plyba2bwlrD1o1EgP6leNaNeWfBbsuCupQWx8DZNK6oJ4xPq-eJsPSzZaRv6uFl6MD58gDnQ Comprobado el 23-09-2022

[27] Sharon Licari. GPT3: la inteligencia artificial y su posible impacto en las empresas, 2022. URL: https://blog.hubspot.es/marketing/que-es-gpt3. Comprobado el 23-09-2022

[28] Luís Leão. La guía máxima para el modelo de lenguaje GPT-3 de OpenAI, 2021. https://www.twilio.com/blog/la-guia-maxima-para-el-modelo-de-lenguaje-gpt-3-de-openai. Comprobado el 23-09-2022

[29] Rafael Lucca. ¿Qué es BERT y cómo funciona?. URL: https://dxmedia.net/algoritmo-bert-google/ Comprobado el 23-09-2022

[30] Adam Roberts. Exploring Transfer Learning with T5: the Text-To-Text Transfer Transformer, 2020. URL: https://ai.googleblog.com/2020/02/exploring-transfer-learning-with-t5.html. Comprobado el 23-09-2022

[31] José Martinez Heras. Precisión, Recall,F1, Accuracy en clasificación, 2020. URL https://www.iartificial.net/precision-recall-f1-accuracy-en-clasificacion/. Comprobado el 23-09-2022

[32] Kenneth Leung. Micro,Macro & Weighted Averages of F1 Score, Clearly Explained, 2022. URL: https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f. Comprobado el 23-09-2022

[33] Jaime Collado, Estrella Vallecillo. TextFlow: A text analysis library for Python, 2022. URL: https://gitlab.ujaen.es/jcollado/textflow.git. Comprobado el 23-09-2022.

[34] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E. Gonzalez, Ion Stoica. Tune: A Research Platform for Distributed Model Selection and Training. ArXiv, abs/1807.05118, 2018.

[35] Juan Manuel Pérez, Juan Carlos Giudici, Franco Luque. Pysentimiento: A python Toolkit for Sentiment Analysis and SocialNLP task. ArXiv, abs/2106.09462, 2021.

[36] Asier Gutiérrez-Fandiño, Jordi Armengol-Estapé, Marc Pàmies, Joan Llop-Palao, Joaquin Silveira-Ocampo, Casimiro Pio Carrino, Carme Armentano-Oller, Carlos Rodriguez-Penagos, Aitor Gonzalez-Agirre, Marta Villegas. MarIA: Spanish Language Models,2022.
URL:http://journal.sepln.org/sepln/ojs/ojs/index.php/pln/article/view/6405.
Comprobado el 23-09-2022.

[37] Juan Manuel Pérez, Damián A. Furman, Laura Alonso Alemany, Franco Luque. RoBERTuito: a pre-trained language model for social media text in Spanish. ArXiv, abs/2111.09453, 2021.

[38] Francesco Barbieri, Luis Espinosa Anke, Jose Camacho-Collados. XLM-T: Multilingual Language Models in Twitter for Sentiment Analysis and Beyond. ArXiv, abs/2104.12250, 2021.

[39] Hugging Face. Modelo finiteautomata/beto-sentiment-analysis. URL: https://huggingface.co/finiteautomata/beto-sentiment-analysis. Comprobado el 23-09-2022.

[40] Hugging Face. Modelo dccuchile/bert-base-spanish-wwm-cased. URL: https://huggingface.co/dccuchile/bert-base-spanish-wwm-cased. Comprobado el 23-09-2022.

[41] Hugging Face. Modelo RoBERTa spanish PlanTL-GOB-ES/roberta-base-bne. URL: https://huggingface.co/PlanTL-GOB-ES/roberta-base-bne. Comprobado el 23-09-2022.

[42] Hugging Face. Modelo pysentimiento/robertuito-base-cased. URL: https://huggingface.co/pysentimiento/robertuito-base-cased. Comprobado el 23-09-2022.

[43] Hugging Face. Modelo bertin-project/bertin-roberta-base-spanish. URL: https://huggingface.co/bertin-project/bertin-roberta-base-spanish. Comprobado el 23-09-2022.

[44] Hugging Face. Modelo maxpe/bertin-roberta-base-spanish_semeval18_emodetection. URL: https://huggingface.co/maxpe/bertin-roberta-base-spanish_semeval18_emodetection .Comprobado el 23-09-2022.

[45] Hugging Face. Modelo RoBerta XML Cardiff cardiffnlp/twitter-xlm-roberta-base-sentiment. URL: https://huggingface.co/cardiffnlp/twitter-xlm-roberta-base-sentiment. Comprobado el 23-09-2022.

[46] Hugging Face. Transformers. URL: https://huggingface.co/docs/transformers/index. Comprobado el 23-09-2022.

[47] CEATIC. Clúster ADA. URL: https://www.ujaen.es/centros/ceatic/servicios/supercomputacion/cluster-ada. cOMPROBADO EL 23-09-2022.

[48] Jason Brownlee. A Gentle Introduction to Dropout for Regularizing Deep Neural Networks, 2018. URL: https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/. Comprobado el 23-09-2022

[49] Hugging Face. URL: https://huggingface.co/. Comprobado el 23-09-2022.

[50] Jason Brownlee. Understand the Impact of Learning Rate on Neural Network Performance, 2019. URL: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/. Comprobado el 23-09-2022

[51] Vicente Rodríguez. Conceptos básicos sobre redes neuronales, 2018. URL: https://vincentblog.xyz/posts/conceptos-basicos-sobre-redes-neuronales. Comprobado el 23-09-2022

[52] Sebastian, Weight Decay in Neural Networks, 2021. URL: https://programmathically.com/weight-decay-in-neural-networks/ Comprobado el 23-09-2022

[53] Miguel Sotaquirá BERT: el inicio de una nueva era en el Natural Language Processing, 2020. URL: https://www.codificandobits.com/blog/bert-en-el-natural-language-processing/. Comprobado el 23-09-2022

[54] Fernando López. OPTUNA: A Flexible, Efficient and Scalable Hyperparameter Optimization Framework, 2021. URL: https://towardsdatascience.com/optuna-a-flexible-efficient-and-scalable-hyperparameter-optimization-framework-d26bc7a23fff. Comprobado 23-09-2022.

[55] Sreenath S. Hyperparameter Tuning using Optuna, 2020. URL: https://www.analyticsvidhya.com/blog/2020/11/hyperparameter-tuning-using-optuna/. Comprobado 23-09-2022.

[56] Hugging Face. How do Transformers work? URL:https://huggingface.co/course/chapter1/4?fw=pt. Comprobado el 23-09-2022

[57] Ricardo Geek. ¿Qué son las redes neuronales artificiales?, 2018. URL: https://ricardogeek.com/que-son-redes-neuronales-artificiales/. Comprobado el 23-09-2022

[58] Wikipedia. Perceptrón. URL: https://es.wikipedia.org/wiki/Perceptr%C3%B3n. Comprobado el 23-09-2022

[59] Rani Horev. BERT Explained: State of the art language model for NLP, 2018. URL: https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270 Comprobado el 23-09-2022

[60] Jay Alammar. The Illustrated BERT, ELMo and co.(How NLP Cracked Transfer Learning), 2021. URL: https://jalammar.github.io/illustrated-bert/. Comprobado el 23-09-2022

[61] Ureña, L. A. Apuntes de la asignatura fundamentos de ingeniería del software,2019. Universidad de Jaén.